A Graph-based Approach to API Usage Adaptation

Software Engineering Paper Presentation

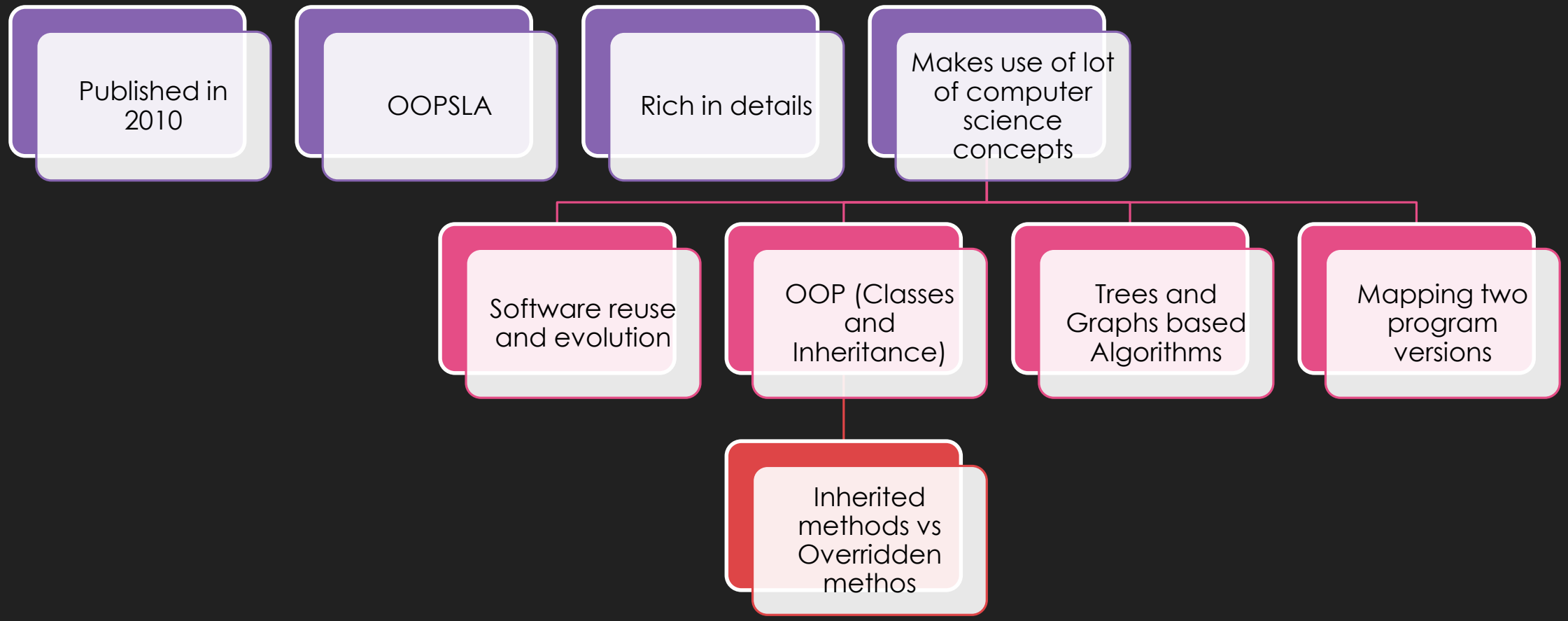
- Kanagaraj Nachimuthu Nallasamy

Outline

- Introduction
- Motivation
- Problem
- Solution
- LIBSYNC (Components)
- Evaluation
- Conclusion
- Discussion

Introduction

Introduction: Some background info



Motivation Scenario

- Sorting Algorithm in Java

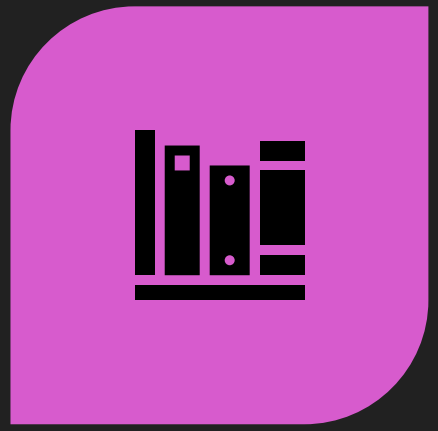
Why Re-use Software?

SOFTWARE DEVELOPMENT COST

TESTING COST

MAINTENANCE COST

Libraries and Clients

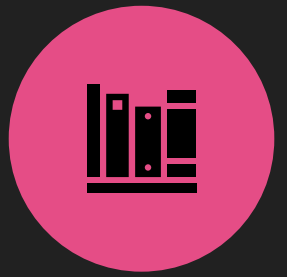


LIBRARY

CLIENTS

Library Changes (Evolves)

Why Library changes?



What happens when library changes?

Evolution of Libraries

NEW FEATURE REQUESTS

BUG FIXES

SECURITY FIXES

MEET NEW STANDARDS

Clients => API Usage

Correct names of APIs

Passing right number of arguments

Correct handling of return type of APIs

Order of API invocations

Motivating Example 1

Client

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

Figure 1. API usage adaptation in JBoss caused by the evolution of JFreeChart

Motivating Example 2

Client

```
SnmpPeer peer=new SnmpPeer(this.address
                          ,this.port, this.localAddress, this.localPort );
peer.setPort(this.port);
peer.setServerPort(this.localPort);
```

Figure 2. API usage adaptation in JBoss caused by the evolution of OpenNMS

Motivating Example 3

Library API

Client

Change in Apache Axis API

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

Change in JBoss

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

class ObjectNameSerializer extends Serializer {
  public boolean Element writeSchema( Class clazz, Types types)...
  ...
```

Figure 3. API usage adaptation in JBoss caused by the evolution of Axis

Motivating Example 4

Library API

Client

Change in Apache Axis API

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

Change in JBoss

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

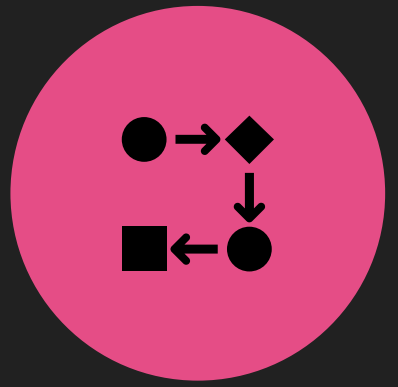
Figure 4. API usage adaptation in JBoss caused by the evolution of Axis

Observations from Examples

- In OOP, two ways to use API functionality
- Method invocation
  - Directly calling to API methods or creating objects of API classes
- Inheritance
  - Declaring classes in client code that inherit from the API classes and override their methods
- Client code must follow order of API method calls
- API usage and adaptation model should capture complex context surrounding API usages
  - Data and Ordering dependencies
  - Control structures around API usages
  - Interaction among multiple objects of different types

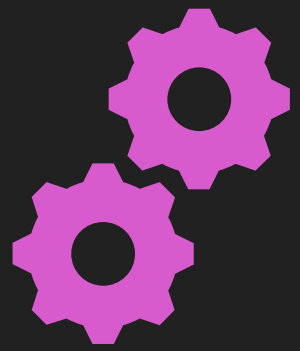
Limitations of Existing API usage Adaptation

Manually write expected adaptations

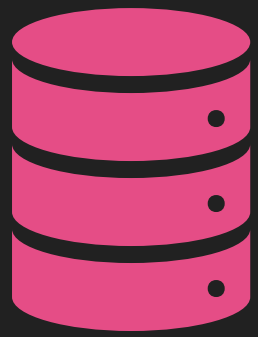


Not capturing complex control and data dependencies

Problem

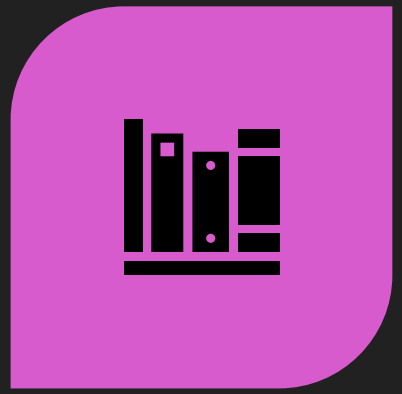


Handling complex API usage adaptations



Capturing complex Control and Data dependencies

Libraries, Clients and Researchers



Library

Clients

Software Engineering Researchers / Problem Solvers

Solution: LIBSYNC

Learn complex API usage adaptation patterns

From other clients

From Library's test code

Recommend to the developers

Solution (High Level View)

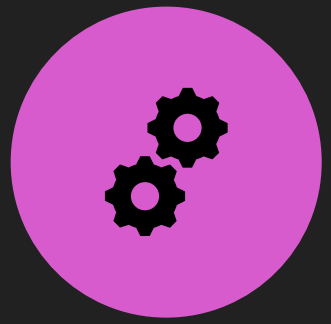
Identify changes to API declarations by comparing two library versions

Extract associated API usage skeletons before and after library migration

Compare the extracted API usage skeletons to recover API usage adaptation patterns

Using the learned adaptation patterns, recommend the locations and the edit operations for adapting API usages

Solution: LIBSYNC (Components)



1) Origin Analysis Tool (OAT):

Tree-based analysis technique

2) Client API Usage Extractor (CUE):

Graph-based representation

3) Usage Adaptation Miner (SAM):

Graph alignment algorithm

4) LIBSYNC:

Recommends locations and edit operations

Origin Analysis Tool (OAT)

Origin Analysis Tool (OAT)

Map corresponding code elements between two program versions

- Packages
- Classes
- Methods

Two different purposes

- Maps two Library versions
- Maps two Client versions (API usage changes)

OAT

Project Tree: T(P)

Node:

- Package, Class, Interface, Method

Attributes:

- declare(u): name, parameter types, all modifiers, return type
- parent(u): node's container element
- content(u): a set of immediate descendant

Transformations

Add	add(u)
Delete	delete(u)
Move	move(u)
Update	update(u)

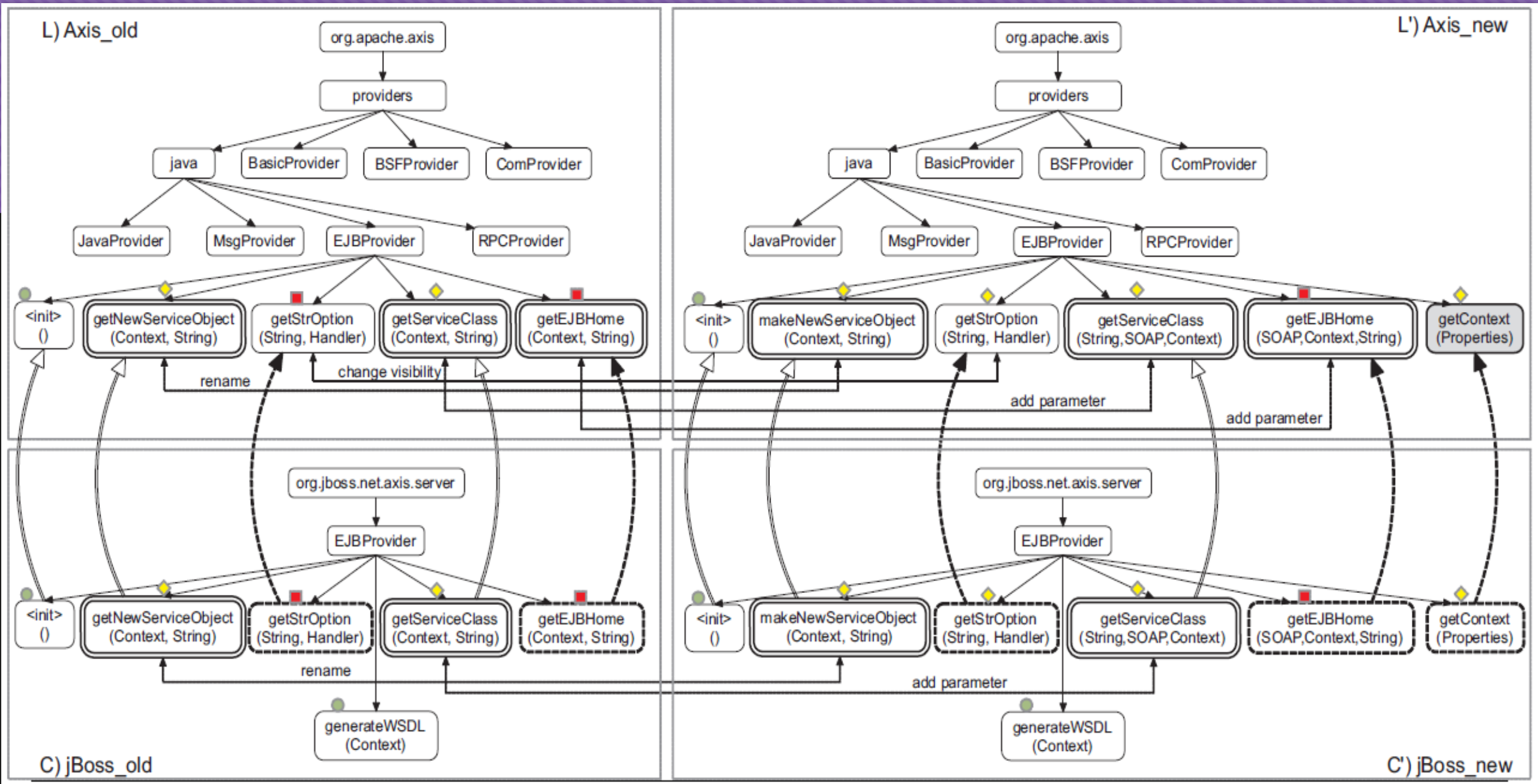


Figure 8. API x-Usage models in JBoss before and after migration to a new Axis library version

Similarity Measures (Method Level)

- Declaration attribute similarity

$$s_d(u, u') = 0.25 * strSim(\text{returntype, returntype'}) + 0.5 * seqSim(\text{name, name'}) + 0.25 * seqSim(\text{parameters, parameters'})$$

$$s_d(u, u') = 0.25 * strSim(\text{Object,Object}) + .5 * seqSim(\text{getNewServiceObject,makeNewServiceObject}) + .25 * seqSim(\text{[Context,String],[Context,String]}) = 0.25 * (1/1) + 0.5 * (3/4) + 0.25 * (2/2) = 0.875$$

Similarity Measures (Method Level)

- Content attribute similarity
- v(u): vector representation of method content

$$s_c(u, u') = \frac{2 * ||Common(v(u), v(u'))||_1}{||v(u)||_1 + ||v(u')||_1}$$

Similarity Measures (Class and Package Level)

- Declaration similarity is same
- Content similarity is based on how many of their children can be mapped

$$s_c(C, C') = \frac{2 * |MaxMatch(content(C), content(C'), sim)|}{|content(C)| + |content(C')|}$$

Mapping Algorithm of OAT

INPUT: 2 project trees

Maps nodes in TOP-DOWN order

3 sets

- AM: Node Already Mapped
- PM: Parent Mapped
- UM: Node and Parent Not Mapped

Hash-based optimizations

```
1  function Map(T, T′) // find mapped nodes and change operations
2   UM.addAll(T, T′)
3   for packages p ∈ T, p′ ∈ T′ // map on exact location
4    if location of u and u′ is identical then Map(p, p′)
5   for packages p ∈ T ∩ UM, p′ ∈ T′ ∩ UM // unmapped pkgs
6    if Sim(p, p′) ≥ δ then SetMap(p, p′) // map on similarity
7   for each mapped pairs of packages (p, p′) ∈ M
8    MapSets(Children(p), Children(p′))) // map parent−mapped
         classes
9   for classes C ∈ T ∩ UM, C′ ∈ T′ ∩ UM // unmapped classes
10   if (C and C′ are in a text−based/LSH−based filtered subset
11   and sim(C, C′) ≥ δ) then SetMap(C, C′) // map on similarity
12  for each mapped pairs of classes (C, C′) ∈ M
13   MapSets(Children(C), Children(C′))) // parent−mapped meths
14  for methods m ∈ T ∩ UM, m′ ∈ T′ ∩ UM // unmapped meths
15   if (m and m′ are in a text−based or LSH−based filtered subset
16   and sim(m, m′) ≥ δ and dsim(m, m′) ≥ μ then
17    SetMap(m, m′) // map on similarity
18  Op = ChangeOperation(M)
19  return M, Op
20
21 function SetMap(u, u′) // map two nodes
22  M.add((u, u′))
23  UM.remove(u, u′)
24  PM.add(content(u), content(u′))
25
26 function MapSets(S, S′) // map two sets of nodes
27  M2 = MaxMatch(S, S′, sim) // use greedy matching
28  for each (u, u′) ∈ M2
29   SetMap(u, u′)
```

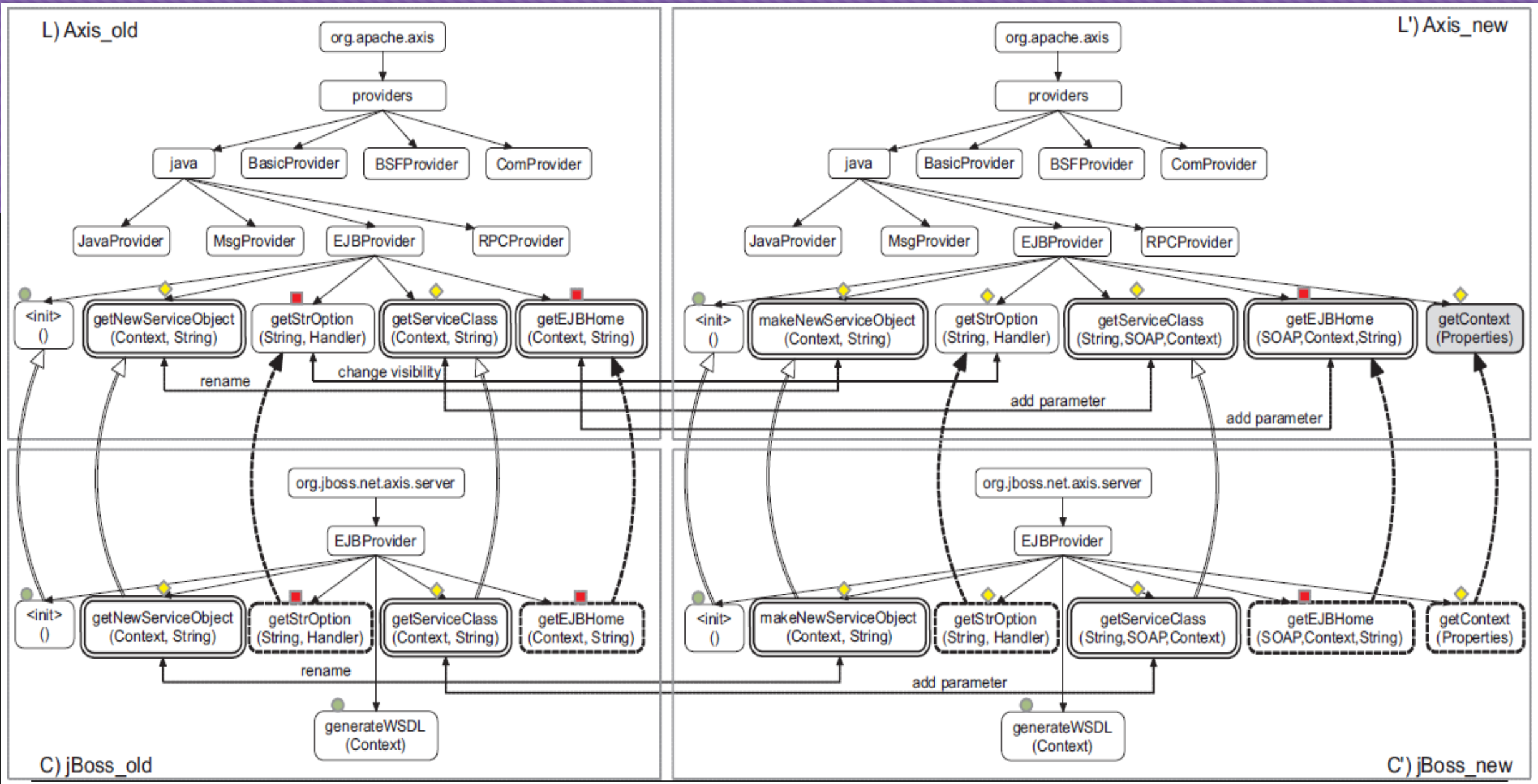


Figure 8. API x-Usage models in JBoss before and after migration to a new Axis library version

Client API Usage Extractor (CUE)

Client API Usage Extractor (CUE)

- API Usage via Invocation
- API Usage via Inheritance

API Usage via Invocation (Graph Model)

CUE represents API i-usages in clients via iGROUM

- Invocation based GRaph based Object Usage Model

Each usage is represented as a labeled, directed, acyclic graph

Usages: Nodes

- Action node: method calls
- Data node: variables (objects)
- Control node: branching point (for, if, while)

Dependencies: Edges

- Action node => Action node: control and data dependencies
- Data node => Action node: object is used as an input
- Action node => Data node: object is return as output

Motivating Example 1

Client

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

Figure 1. API usage adaptation in JBoss caused by the evolution of JFreeChart

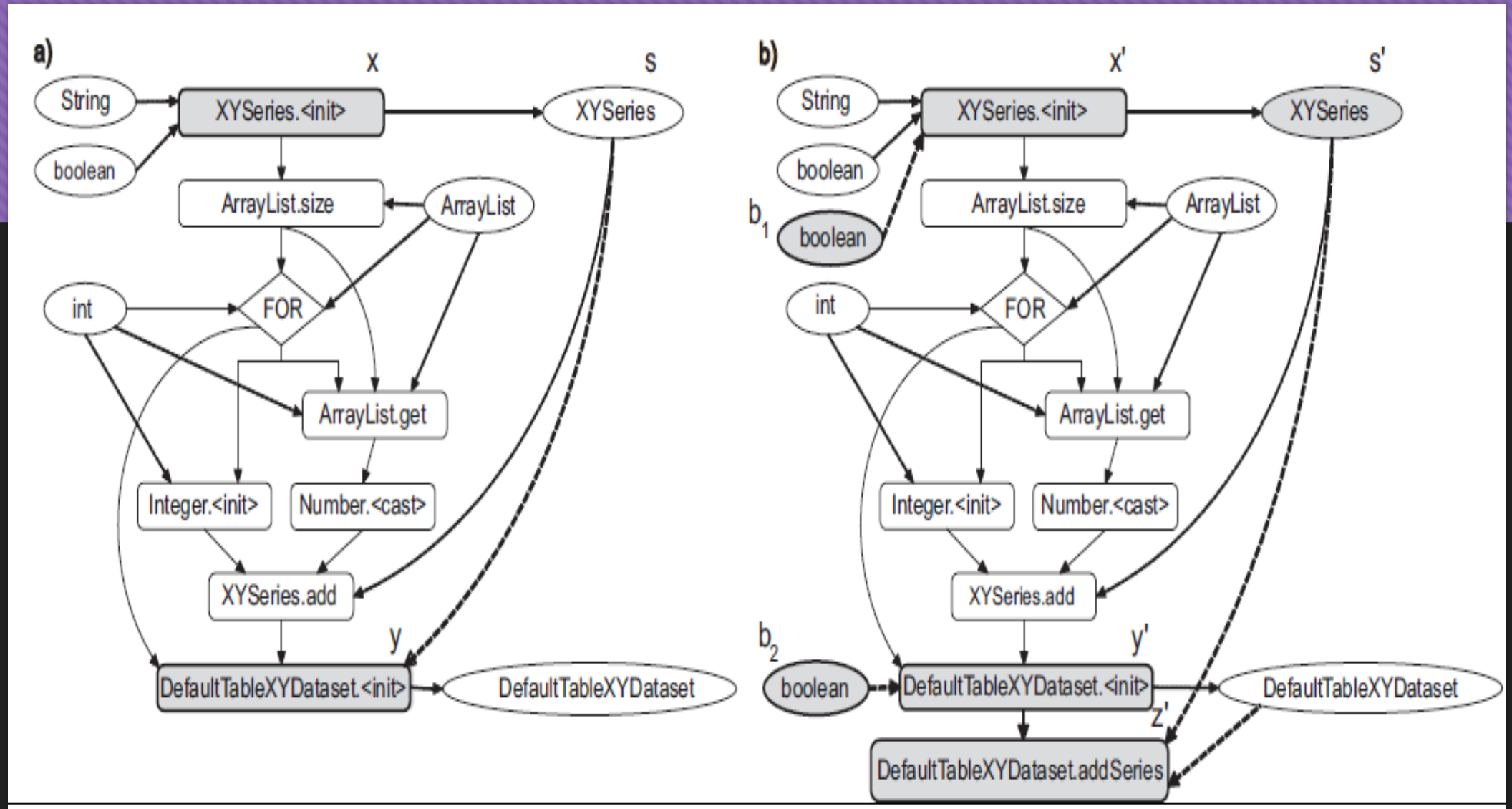


Figure 7. API i-Usage models in JBoss before and after migration to a new JFreeChart library version

i-Usage Extraction

Build API usage model from each method in client

Source code (method) => AST => Build Graph

Traverses the AST tree to analyze the nodes of interest

- Method invocations
- Object declarations
- Object initializations
- Control structures

Build corresponding action, data and control nodes

Removes unrelated nodes after extraction

Finds the subgraph of the original graph model

Performing **Program Slicing** from the API usage nodes via control and data dependency edges

API Usage via Inheritance (Inherits vs Overrides

API Class: A => method: m(A.m)

Client Class: C => method: m (C.m)

Inherits: C.m is directly inherited from A.m

- C.m is not explicitly declared in the body of Class C

Overrides: If C.m is overriding A.m

- C.m is explicitly declared in Class C
- C.m has the same signature as A.m

x-Usage model (Graph model)

xGROUM:

- Extension-based GRaph-based Object Usage Model

Directed Labeled Acyclic Graph

Nodes (both in clients and libraries):

- Class
- Method

Edges (Sub-typing relationships)

- o-edge : Overriding edge
- i-edge : Inheriting edge

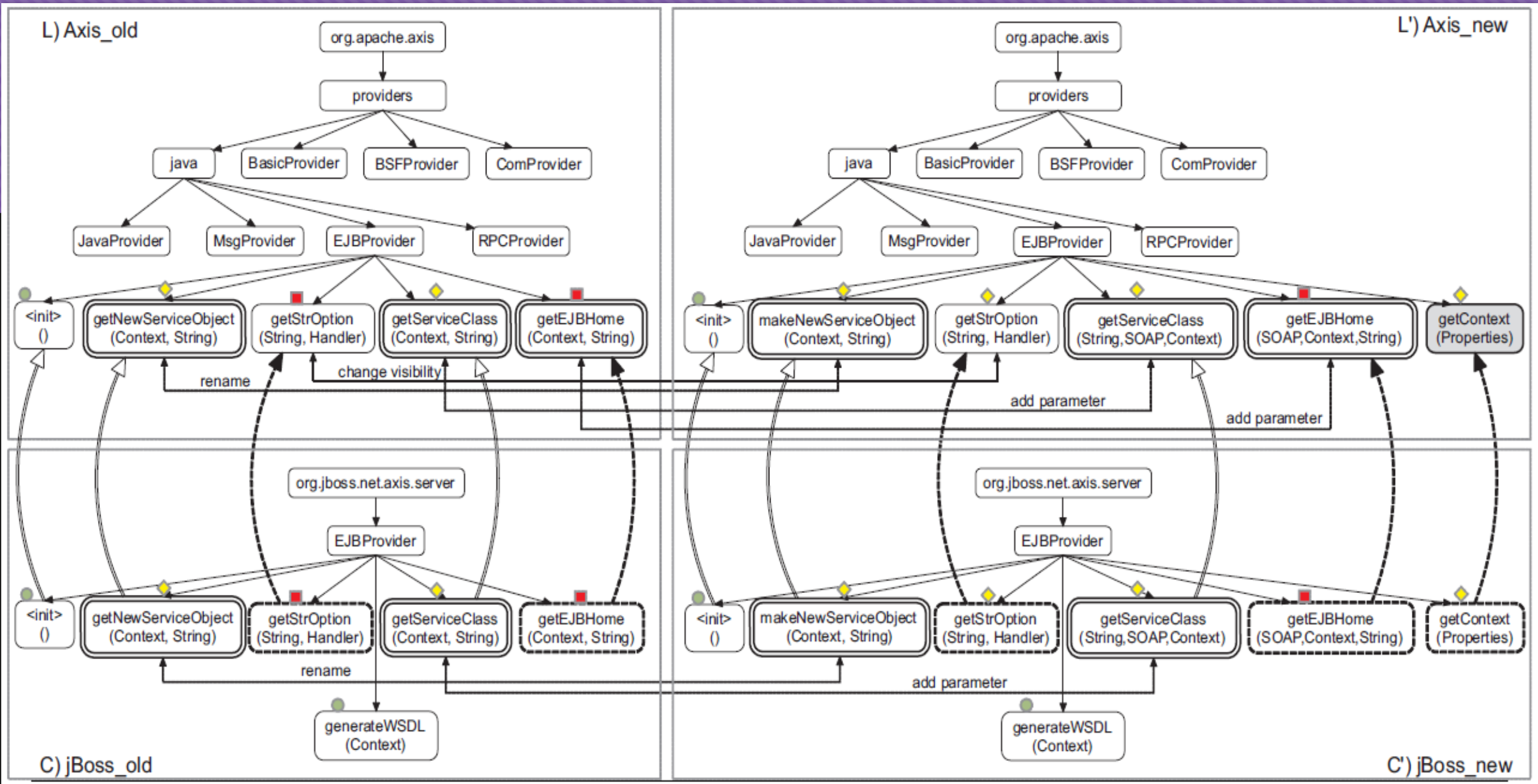


Figure 8. API x-Usage models in JBoss before and after migration to a new Axis library version

Usage Adaptation Miner (SAM)

Usage Adaptation Miner (SAM)

SAM uses iGROUMs to represent API i-usages in both Client and Library test codes

Adaptation of API usages:

- Modeled as a generalization of changes to the corresponding individual iGROUMs

Graph alignment algorithm

i-Usage Change Detection

- LIBSYNC uses OAT to derive ΔL and ΔP
- ΔL => Changed entities of library versions
- ΔP => Changed entities of client versions
- For each method, LIBSYNC builds two iGROUMs in two corresponding versions
- Uses **GroumDiff** : Graph-based alignment and differencing algorithm
  - To find changes between corresponding versions

```
1  function GroumDiff(U, U') // align and differ two usage models
2   for all u ∈ U, v ∈ U' // calculate similarity between u and v
          based on label and structure
3    sim(u, v) = α • lsim(u, v) + β • nsim(u, v)
4   M =MaximumWeightedMatching(U, U', sim) // matching
5   for each (u, v) ∈ M:
6    if sim(u, v) < λ then M.remove((u, v)) //remove low matches
7    else switch // derive change operations on nodes
8     case Attr(u) ≠ Attr(v): Op(u) = Op(v) = ''replaced''
9     case Attr(u) = Attr(v), nsim(u, v) < 1: Op(u)=''updated''
10    default: Op(u) = ''unchanged''
11  for each u ∈ U, u ∉ M: Op(u) = ''deleted'' // unaligned nodes
12  for each v ∈ U', v ∉ M: Op(v) = ''added'' // are deleted/added
13  Ed = EditScript(Op)
14 return M, Op, Ed
```

Figure 9. API Usage Graph Alignment Algorithm

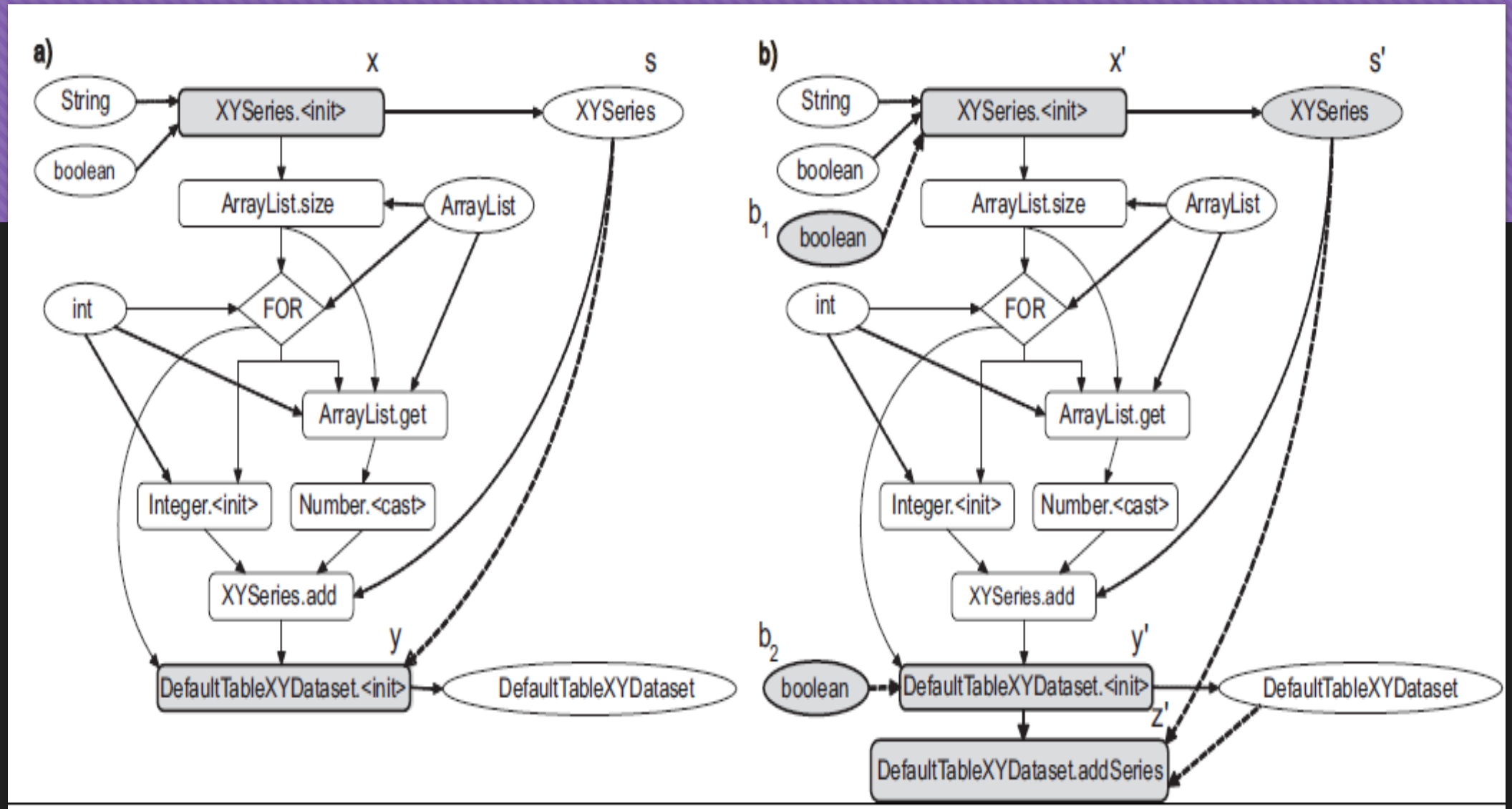


Figure 7. API i-Usage models in JBoss before and after migration to a new JFreeChart library version

GroumDiff: Derived Edit Script

```
Replace XYSeries.<init>(..., boolean)
        XYSeries.<init>(..., boolean, boolean)
```

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

Add DefaultTableXYDataset.addSeries(XYSeries)
```

Mining Algorithm

```
1  function ChangePattern(ΔP_i, ΔL_i) //mine usage change patterns
2    for each (U, U') ∈ UsageChange(ΔP_i, ΔL_i) //compute changes
3      Add(GroumDiff(U, U')) into E // add to dataset of sets of ops
4    F = MaximalFrequentSet(E, σ) //mine maximal frequent subset
         of edit operations
5    for each f ∈ F:
6      Find U, U' : f ⊂ GroumDiff(U, U') //find usages changed by f
7      Extract (U_o(f), U'_o(f)) from (U, U') // extract ref models
8      Add (U_o(f), U'_o(f)) into Ref(f) // add to reference set for f
9  return F, Ref
```

Figure 11. Adaptation Pattern Mining Algorithm

Another Example

```
protected JFreeChart createXyLineChart() throws JRException {
 ChartFactory.setChartTheme(StandardChartTheme.createLegacyTheme());
JFreeChart jfreeChart=ChartFactory.createXYLineChart(..., getDataset(),...);
 ...
return jfreeChart
}
```

Figure 10. API Usage Changes in JasperReports

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

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

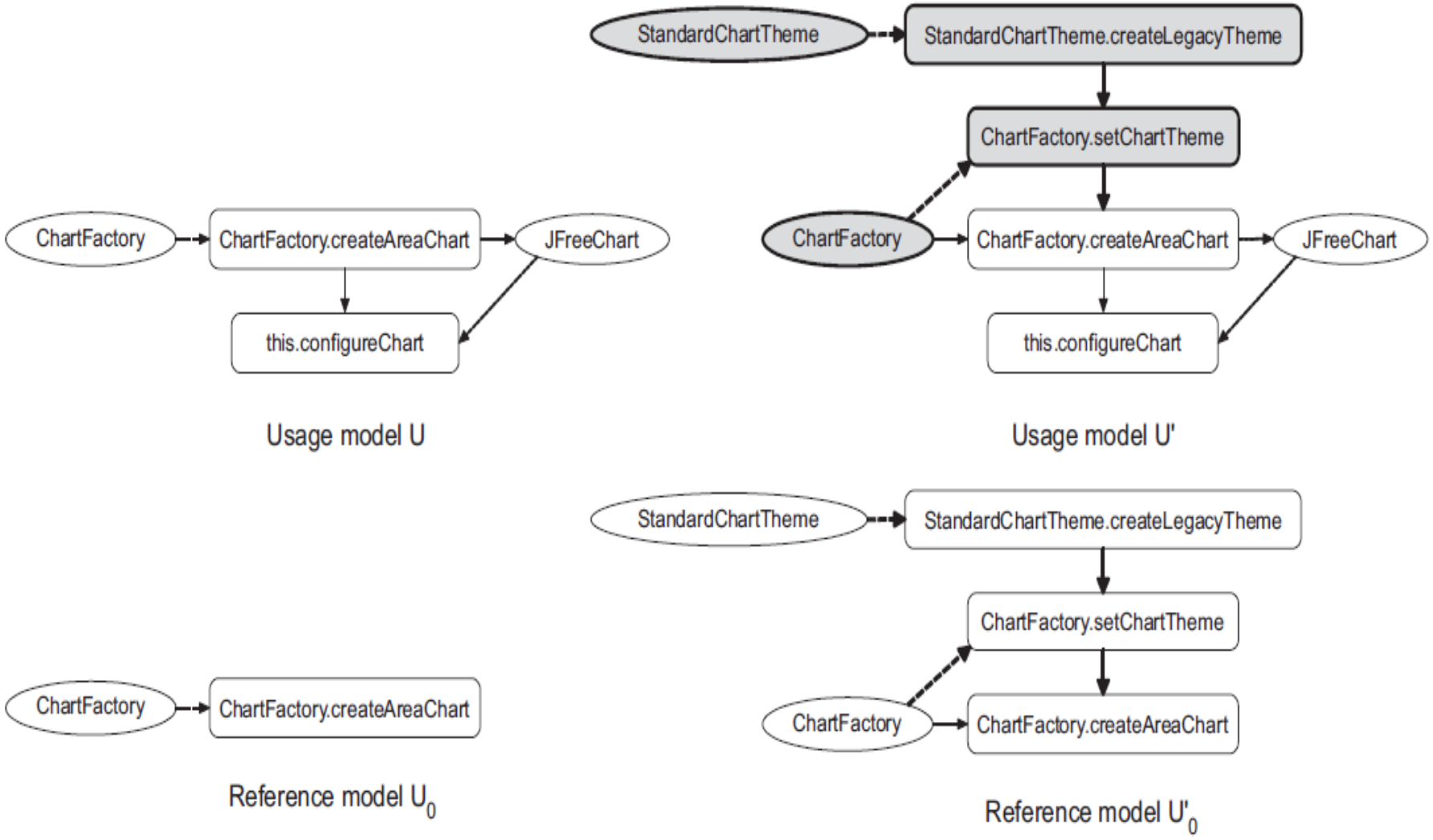


Figure 12. API Usage Change Patterns and Reference Models

Recommending Adaptations (LIBSYNC)

Recommending Adaptations (LIBSYNC)

- Location Recommendation
- Edit Operations Recommendation

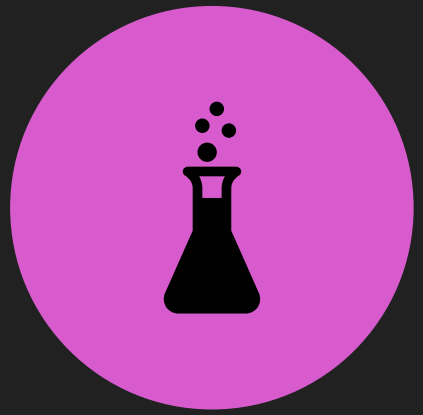
```
1  XYSeries set = new XYSeries(attribute,false, false );
2  for (int i = 0; i < data.size(); i++)
3    set.add(new Integer(i), (Number)data.get(i));
4  DefaultTableXYDataset dataset = new DefaultTableXYDataset(set false );
5  dataset.addSeries(set) ;
6  JFreeChart chart = ChartFactory.createXYLineChart(...,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

Figure 1. API usage adaptation in JBoss caused by the evolution of JFreeChart

Id	Label	Change
A	XYSeries	modified class
B	DefaultTableXYDataset	modified class
x	XYSeries.<init>(String,boolean)	deprecated
x'	XYSeries.<init>(String, boolean, boolean)	added
y	DefaultTableXYDataset.<init>(XYSeries)	deprecated
y'	DefaultTableXYDataset.<init>(boolean)	added

Evaluation

OAT Evaluation: Experiments



TWO EXPERIMENTS

FIRST: MANUAL CHECKING

SECOND: COMPARE WITH ***KIM'S*** API MATCHING RESULTS

OAT Evaluation: Experiment 1

- Quality of change detection in OAT
- Method level matches
- Four different version pairs of JHotDraw (Library)

Table 1. Precision of Origin Analysis Tool OAT

Version Pairs	Mapped	Checked	√	X	Precision
5.2-5.3	71	71	69	2	97%
5.3-5.4b1	70	70	68	2	97%
5.4b1-5.4b2	9	9	8	1	89%
5.4b2-6.0b1	3,250	100	100	0	100%

OAT Evaluation: Experiment 2

- Library 1: JFreeChart
- Library 2: JHotDraw
- Method level matches
- Common set of matches
- OAT – Kim
- Kim - OAT

OAT Evaluation: Experiment 2 (Contd.)

Table 2. Comparison of Origin Analysis Tools

JFreeChart																	
Pairs	OAT	Kim	∩	OAT - Kim						Kim - OAT							
				∑	√	X	?	TP	FP	∑	√	X	?	TP	FP		
0.9.5-0.9.6	5	5	5	0	0	0	0	100%	0%	0	0	0	0	100%	0%		
0.9.6-0.9.7	368	366	364	4	2	1	1	50%	25%	2	0	0	2	0%	0%		
0.9.7-0.9.8	3157	3158	3121	36	36	0	0	100%	0%	37	7	30	0	19%	81%		
0.9.9-0.9.10	144	159	130	14	3	10	1	21%	71%	29	14	2	13	48%	7%		
0.9.10-0.9.11	9	7	7	2	2	0	0	100%	0%	0	0	0	0	100%	0%		
0.9.11-0.9.12	66	66	35	31	12	10	9	39%	32%	31	19	6	6	61%	19%		
0.9.12-0.9.13	134	133	133	1	1	0	0	100%	0%	0	0	0	0	100%	0%		
0.9.13-0.9.14	84	96	74	10	6	3	1	60%	30%	22	12	6	4	55%	27%		
0.9.14-0.9.15	6	12	6	0	0	0	0	100%	0%	6	6	0	0	100%	0%		
0.9.15-0.9.16	79	75	65	14	13	0	1	93%	0%	10	2	4	4	20%	40%		
0.9.16-0.9.17	205	240	171	34	4	30	0	12%	88%	69	27	42	0	39%	61%		
0.9.17-0.9.18	36	45	36	0	0	0	0	100%	0%	9	0	9	0	0%	100%		
0.9.18-0.9.19	140	282	102	38	30	8	0	79%	21%	180	41	139	0	23%	77%		
Avg.	341.00	357.23	326.85	14.15	8.38	4.77	1.00	73%	21%	30.38	9.85	18.31	2.23	51%	32%		
JHotDraw																	
Pairs	OAT	Kim	∩	OAT - Kim						Kim - OAT							
				∑	√	X	?	TP	FP	∑	√	X	?	TP	FP		
5.2-5.3	71	77	66	5	3	2	0	60%	40%	11	2	4	5	18%	36%		
5.3-5.4b1	70	69	56	14	12	1	1	86%	7%	13	5	6	2	38%	46%		
5.4b1-5.4b2	9	13	8	1	1	0	0	100%	0%	5	3	1	1	60%	20%		
5.4b2-6.0b1	3,250	3,239	3,239	11	11	0	0	100%	0%	0	0	0	0	100%	0%		
Avg.	850	849.5	842.25	7.75	6.75	0.75	0.25	86%	12%	7.25	2.5	2.75	2	54%	26%		

Evaluation: Adaptation of i-Usage (LIBSYNC)

Table 3. Subject Systems

Client	Life Cycle	Releases	Methods	Used APIs
JBoss (JB)	10/2003 - 05/2009	47	10-40K	45-262
JasperReports (JR)	01/2004 - 02/2010	56	1-11K	7-47
Spring (SP)	12/2005 - 06/2008	29	10-18K	45-262

Precision of API Usage Change Detection

Table 4. Precision of API Usage Change Detection

Client	Changes	Libs	Operations		API	
			√	X	√	X
JasperReports	30	5	30	0	27	3
JBoss	40	17	38	2	38	2
Spring	30	15	30	0	30	0

Accuracy of i-Usage Location & Operations Recommendation

Table 5. Accuracy of i-Usage Location Recommendation

API - Client	Version	Rec.	√	Hint	X	Miss
JFree - Jasper	3.0.1 - 3.1.0	12	9	3	0	0
Mondrian - Jasper	1.3.4 - 2.0.0	3	3	0	0	0
Axis - JBoss	3.2.5 - 4.0.0	8	5	1	2	0
Hibernate - JBoss	4.2.0 - 4.2.1	29	25	0	3	1
JDO2 - Spring	2.0m1 - 2.0m2	8	8	0	0	0
JRuby - Spring	2.0.3 - 2.0.4	7	7	0	0	0

Table 6. Accuracy of i-Usage Operations Recommendation

Mine on	Adapt to	Usages	Rec.	√	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

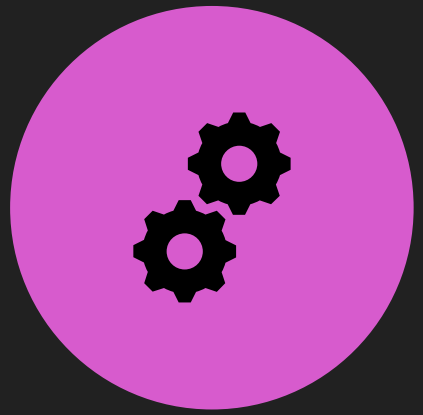
Accuracy of x-Usage Recommendation

Table 7. Accuracy of x-Usage Recommendation

	Rec.	√	X
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

Incorrect mapping result from OAT

Conclusion



HANDLES COMPLEX API USAGE ADAPTATIONS

USES SEVERAL GRAPH BASED APPROACHES

HIGHLY ACCURATE CHANGE DETECTION AND RECOMMENDATIONS

Discussion

How often API's evolve in a complex way?

Runtime complexity?

How to address the limitation of training data?

Need of clients who already migrated

Thank You!

Appendix:

Related Work

- Library Evolution and Client Adaptation
- Program Differencing and Origin Analysis
- API Usage Specification Extraction
- Empirical Studies of API Evolution