OBJECT ORIENTED VERSION PROGRMAMING

Luthfan Anshar Lubis, Yudai Tanabe,
Tomoyuki Aotani, Hidehiko Masuhara
Tokyo Institute of Technology

東京工業大学
Tokyo Institute of Technology

Motivation

- Dependency relation between programs is convoluted: conflicts among dependencies upon updates
- Version is a common identifier used in distinguishing programs: use in a type-safe system to increase flexibility.
- LambdaVL[1], functional programming language with versioned type: apply to OOPL.

Related Work

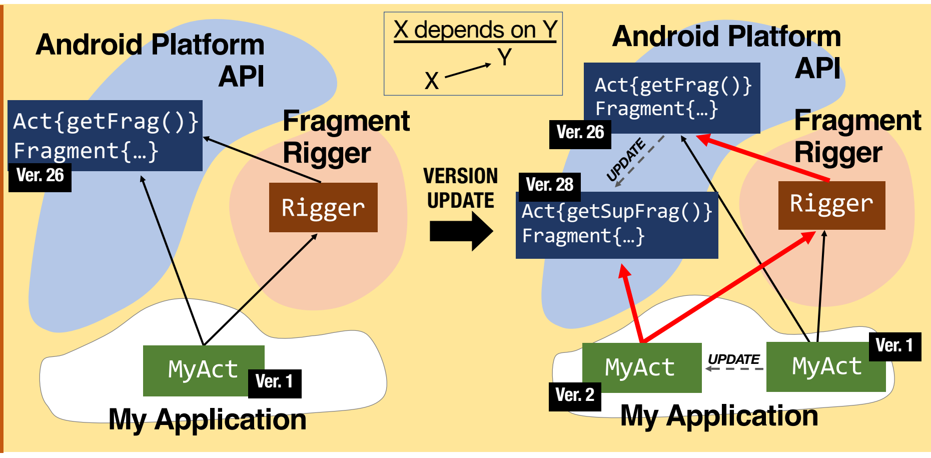
- The compiler is built by using an extensible compiler called **ExtendJ**[3].
- The core calculus is built on Featherweight Java[2], a minimal Java language.

[1] Yudai Tanabe, Tomoyuki Aotani, Hidehiko Masuhara.A Context-Oriented Programming Approach to Dependency Hell. COP 2018
[2] Atsushi Igarashi, et. al. Featherweight Java: a minimal core calculus for Java and GJ. 2001.
[3] Jesper Öqvist. ExtendJ: extensible Java compiler. 2018.

Proposal

- **BatakJava**, Java language extended with versions.
- **FBJ**, a core calculus ensuring the existence of necessary versions of classes.

Dependency Conflict's Example



- Update deprecates method `getFrag()` in ver.28.
- **Fragment Rigger** uses ver.26, while **My Application** is updated with ver.28.

- Rigger depends on **old API**
- MyAct ver.2 depends on **new API**

⬇

UNRESOLVED CONFLICT

Version Programming

= programming using versions explicitly in typing

Contextual Class

- Class declarations are annotated with **contexts**, e.g. `{A26}`
- **Contexts** consist of **version tags**, e.g. `A26` in `{A26}`

Ver. 26
```
class Act#{A26}
class Fragment#{A26}
```
Ver. 28
```
class Act#{A28}
class Fragment#{A28}
```

Overview Class

- Interface for each class where signature info are collected.
- Contains **constructor and method signature** and their available contexts.

```
overview of Act {
  Act(…) in {A26},{A28}
}
```
```
overview of Fragment {
 Fragment(…) in {A26},{A28}
}
```

Inheritance

- `extends` is declared in overview.
- Context keeps track of necessary versions.

```
overview of MyAct extends Act{
  MyAct(…) in {A26,M1},{A28,M2}
}
```
`MyAct#{M1,A26}<:Act{A26}`

||

MyAct ver.1 <: Act ver.26

Contextual Objects

Contextually Specific Objects

- Refer to a specific contextual class. Similar to Java.

```
MyAct#{A26,M1} act =
  new MyAct#{A26,M1}(...);
```

Contextually Polymorphic Objects

- Refer to signature information obtained from overview class.

```
MyAct act = new MyAct(...)
```

- Method invocation infers callable methods by checking overview and context of the object.
- Users can manually restrict context.

```
act.{A26}.getFrag()    METHOD NOT FOUND
act.{A28}.getSupFrag()    METHOD FOUND
```

Limitations

To allow working with multiple versions simultaneously.

- Can't change constructor in newer versions
- Can't change its superclass in new versions

Relaxing these is left as future work

Application

```
class MyActivity#{M2,A28} {
 void main(String[] args) {
  Rigger@ rig = new Rigger@();
  rig.getRigger(this).startFragment(frag1);
  this.getSupportFrag().replace(frag2);
 }
}
```

uses ver.26 method

uses ver.28 method