**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.

**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}

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

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

**Proposal**
- 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

**Contextual Objects**
- Refer to a specific contextual class. Similar to Java.
  
  ```java
  MyAct#{A26,M1} act = new MyAct#{A26,M1}();
  ```

**Contextually Specific Objects**
- Refer to signature information obtained from overview class.
  
  ```java
  MyAct act = new MyAct(
  ```

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

**Application**
- uses ver.26 method
- uses ver.28 method