UNIVERSITY OF CAMBRIDGE

raoul.urma@cl.cam.ac.uk
Raoul-Gabriel Urma

#cprg

Programming Language Maintenance & Evolution

Alan Mycroft
alan.mycroft@cl.cam.ac.uk

Programming languages evolve over time. New features are added to match new needs of developers and to make programming simpler. However, developers do not always adopt such features at the same rate. At the same time, some features become obsolete because newer features make them redundant. However, they often remain in future versions of the language in order to maintain backward compatibility. As a result, language designers are left with a programming language that is increasingly more complex to maintain and understand.

Projects

1 A Platform for Studying Language Features

Empirical studies are critical to understand how programming language features are used in practice. They provide answers to questions that help programming languages to evolve.

However, conducting such studies can be difficult and time consuming. For example, analyzing the usage frequency of a specific language feature or idiom requires writing complex static analysis and reporting tools.

We are working on a new platform that automates the analysis of language features in Java. It comprises a corpus of open-source software, a source code query language as well as an automated reporting tool.

In the long run we aim to extend this platform to be language agnostic.

2 Discovering Language Idioms

Java introduced a more compact loop form to help programmers iterate over collections. This is an example of a language idiom that was built in the language to help code readability and make life simpler for programmer.

We intend to develop a technique that discovers such idioms. This way contributing to evolving programming languages closer to programmers' needs.

```
for(int i = 0; i < array.length; i++)
{
    Object o = array[i];
}
```

Java 1.4

```
for(Object o : array)
{

}
```

Java 5.0

3 Automated Adapter Generations

Migrations between classes is difficult. In fact, a legacy class may not be quite compatible with its replacement. Consequently, the operations of the legacy class can differ from its replacement and their properties can be different. As a result, such refactoring is time consuming and require programmers to be extremely careful in order to preserve semantic behaviour of the transformed program.

We are working on a method that dynamically extracts common properties and highlights differences between a legacy class and its replacement in order to help such migrations.

```
Map.put(k,v1);
Map.put(k,v2);
Map.get(k); // v2
```

```
Multimap.put(k,v1);
Multimap.put(k,v2);
Multimap.get(k); // {v1, v2}
```

```
Map.put(k,v)
=
Multimap.removeAll(k);
Multimap.put(k,v);
```

Current Work

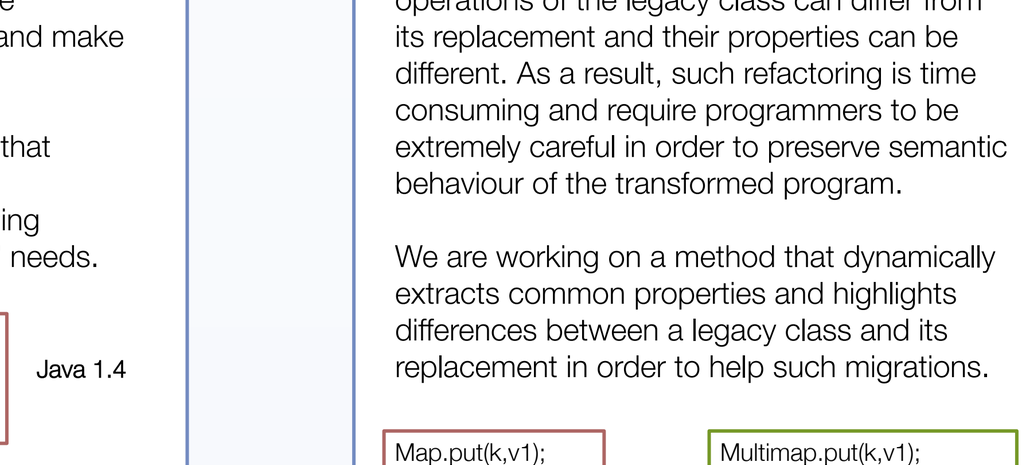
An Empirical Study of Variance in Object-Oriented languages

Variance constructs were introduced to increase the flexibility of object-oriented programming languages supporting generics. There are two approaches to specifying variance: declaration-site variance, which is simple but restrictive, and use-site variance, which is more flexible but more complex. However, it remains unclear how programmers use the flexibility provided by variance, and whether they use it at all.

We undertake three studies to understand how programmers use variance in real programs:

- Investigation of Covariant Arrays in Java
- Investigation of wildcards (use-site variance) in Java
- Investigation of declaration-site variance in C# and Scala

Java Corpus Tools



We are working on a prototype of our project to create a platform for studying language features. It is based on a corpus (5M loc) of Java software, an annotation processor to analyse the AST of the software and a query language that reports how Java features are used.

In further work, we intend to pre-index the source code of the software ahead of time for fast query response.