Language Features As A Library

Using Gosu's Open Type System With External DSLs

Carson Gross
Languages
Amalgamated Code
6 14 2011

Gosu

Warning: This talk will not involve method handles or JSR 292

Gosu

- Yet Another Statically Typed JVM Language
- The usual good things: closures, type inference, properties, blah, blah, blah
- But one unusual cool thing: The Open Type System

The Open Type System

- A formal API for plugging new types into the Gosu compiler
- Makes metaprogramming a first class citizen in the language
- It is used, for example, to expose java properties files in a type safe manner to the Gosu compiler.

The Open Type System

/example/MyProps.properties

```
Prop1 = A Prop
Prop2 = Another Prop
```


/example/MyClass.gs

```
package example
class MyClass {
  function test() {
    print(MyProps.Prop1)
    print(MyProps.Prop2)
  }
}
```

Properties As An External DSL

- In this case, we can think of the Java Properties syntax as an external DSL that has been adapted for Gosu.
- The Open Type System provides the bridge between this external DSL and Gosu code.
- There are quite a few interesting type loaders that exist for Gosu today: WSDL, XSD, SQL, etc.

But Is This Just Code Gen?

But Is This Just Code Gen?

- **No! Since the Gosu compiler loads types on demand, a much larger potential type space can be supported as long as only reasonable subset is actually used.**

But Is This Just Code Gen?

- No! Since the Gosu compiler loads types on demand, a much larger potential type space can be supported as long as only reasonable subset is actually used.
- No! The Open Type System gives you more control over the semantics of your type than regular code generation.

Language Features As A Library

- **Structural Typing: duck typing for statically typed languages**
- **“If it looks like a programmer, and complains like a programmer, it’s a programmer.”**
- **Like duck typing, but with static verification**

Language Features As A Library

- Gosu does not have support for structural typing
  - A future release *may* include it
- However, we can implement structural typing support today by using the Open Type System

Introducing Protocols

- **Protocols use a simple, Gosu interface-like syntax:**

/example/ProgrammerProtocol.proto

```
package example
protocol ProgrammerProtocol {
  function complain()
}
```

Introducing Protocols

- The ProtocolTypeLoader creates types based on the .proto files it finds on the source path
- Crucially, the Open Type System allows us to adopt *structural assignment semantics*

Lets Look At The Code…

- ITypeLoader → ProtocolsTypeLoader
  - getType(String) → IType
- IType → ProtocolType
  - getTypeInfo() → ITypeInfo
- ITypeInfo → ProtocolTypeInfo
  - getMethods() -> List<IMethodInfo>
  - getProperties() -> List<IPropertyInfo>
- IMethodInfo → ProtocolMethodInfo
- IPropertyInfo → ProtocolPropertyInfo

Using A Protocol

```
uses example.ProgrammerProtocol

class DynamicLanguageProgrammer {
  function complain() { print( “Static typing sucks!” ) }
}
class FunctionalLanguageProgrammer {
  function complain() { print( “Covariance isn’t safe!” ) }
}
var programmer : ProgrammerProtcol
programmer = new DynamicLanguageProgrammer()
programmer.complain()
programmer = new FunctionalLanguageProgrammer()
programmer.complain()
```

Language Features As A Library

- We have introduced structural typing to Gosu without any changes to the underlying language
- The Open Type System takes pressure off of the core language
- Microsoft is exploring a similar technology, which they call Type Providers

Info

- **Gosu:**

  http://gosu-lang.org

- **Protocols Type Loader:**

  http://protocols.github.com/

- **Me:**

  carsongross@gmail.com

  @carson_gross