Towards a flexible Pharo Compiler

Clement Bera and Marcus Denker
Three Problems

• Architecture is not reusable
• Compiler can not be parametrized
• The mapping between source code and bytecode is overly complex.
Reusability

- AST is special for the Compiler
- Tools use own AST (RB)
- AST is destroyed when compiling
- No reusable backend/parser...
No Parametrization

- No pluggable architecture
- Parser, code generator fixed
- No infrastructure for compiler options
Mapping bc2source

- For the Debugger, we need to map bytecode to source offsets.
- With closures, we need to map temp offsets to real temps.

Old Compiler: Encoder builds complex table structure
Solution: OPAL

- New compiler framework for Pharo
- Default compiler in Pharo3
  - Old Compiler will be a loadable package
Design

- RB AST
- Visitors
- Bytecode level IR
Reusability

- AST is unchanged
- Backend independent

AST Interpreter

Node navigation

Reflectivity

Metalinks

Oz/Hazelnut

Smart suggestions

Class Builder
Parametrization

- Explicit compiler context
- All visitors are pluggable
Compiler Options

- Turn off inlining of ifTrue: and friends

MyClass>>foo
<compilerOptions: - optionInlineIf>

^ #myNonBooleanObject
  ifTrue: [ 1 ]
  ifFalse: [ 0 ]
Mapping

- Mapping uses AST directly
Performance

- Visitors and IR do cost a bit of speed
- But not much

<table>
<thead>
<tr>
<th></th>
<th>Recompile</th>
<th>Opal Compiler</th>
<th>Old Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object class (ms)</td>
<td>296.66 ± 0.98</td>
<td>222.9 ± 2.4</td>
<td></td>
</tr>
<tr>
<td>Whole image (ms)</td>
<td>72120 ± 189</td>
<td>49908 ± 240</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Opal solves the problems of the old compiler
- Important basis for many features you will see in Pharo3 and Pharo4
Conclusion

- Opal solves the problems of the old compiler
- Important basis for many features you will see in Pharo3 and Pharo4

Questions?