Using Aspects for Language Portability

Lennart Kats    Eelco Visser
DSLs
Stratego
SDF
Spoofax
DSL compilers (code generators)
Backend targets the platform

DSL → Frontend → Backend → Java

Java logo
Backend targets the platforms

DSL

Frontend

Backend

Backend

Backend

Java

php

JavaScipt
“So switching to another platform is just a little matter of switching the backend, right?”

(wrong)
Why not?

• platform-exclusive libraries
• platform escapes and native calls
• interoperability and integration with platform applications
• performance and stack behavior
Then what?

Use aspect weaving to address portability issues in programs and libraries!
Our case study

Frontend

Backend

C/POSIX

Backend

Stratego/XT

Java
Why Java?
1. Glue code aspects

Override functions and library invocations to work with platform-specific libraries

- SGLR vs. JSGLR
- ORM
- communication
- etc.
2. Migration aspects

Because we cannot solve all portability problems (right away)

• There may be no alternative for a library
• Primitives may make assumptions about the platform (e.g., POSIX vs. Java)
2. Migration aspects

Warn developers about unportable code

Perform “next best” operation
3. Integration aspects

Enhance platform integration:

- error handling (exceptions, console vs. GUI)
- logging
- hooks
- user interaction (console vs. GUI vs. web)
4. Optimization aspects

Address platform performance issues of...

...expensive operations
...common operations (bottlenecks)

by using platform-specific code
or by using code more suited for the platform
Summary

Many additional portability issues

- replacing the backend is not enough!

AOP elegantly addresses them

- four classes of portability aspects
- encapsulate platform concerns in separate libraries