On the relationship between weavers and the build system

Bram Adams
GH-SEL, Ghent University

Kris De Schutter
PROG, VUB
Everything has a Makefile!
How do you bring AOP into your build system?
How to bring AOP in your build system?

- Why a potential problem/challenge?
  - source code modularity vs. build system dependencies
  - existing build system
  - weaver acceleration using build level-tricks

- But: Is there really a problem?
1. Problem Statement

2. Approach

3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation

4. Conclusion
# 33 surveyed languages

<table>
<thead>
<tr>
<th>preprocessed</th>
<th>Cobble, AspectC, AspectC++, XWeaver, Aspicere, C4, WeaveC, ACC, CaesarJ, Apostle, AHEAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>link-time</td>
<td>Aspicere2</td>
</tr>
<tr>
<td>load-time</td>
<td>AspectJ, AspectWerkz2, Weave.NET</td>
</tr>
</tbody>
</table>
Comparison of AOP approaches

- discover prominent build system issues:
  - effort
  - risks
  - workarounds
- derive lessons learnt for stakeholders
- distill requirements for AOP-aware build system?
Build System ⇒ 5 Issues

- a. Resolve Platform-dependencies
- b. Module Configuration
- c. Build Integration
- d. Build Order
- e. Incremental Compilation

```
somelib.a
```

```
a.h  config.h  b.h
```

```
a.c
```

```
b.c
```

```
c.c
```

myapp

configuration layer

build layer
1. Problem Statement

2. Approach

3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation

4. Conclusion
a. Resolving platform-dependencies

- Configuration layer parameterises:
  - source code:
    - base
    - aspects
  - build layer

- no real problem ⇔ do better than now?
# Platform Dependency Issues

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing</th>
<th>Compile-time</th>
<th>Link-time</th>
<th>Run-time</th>
<th>Load-time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tool</strong></td>
<td></td>
<td>conditional compilation</td>
<td>AOP in build</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>User</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- versioned modules/libraries?
- other language support?

**vrijdag 14 december 2007**
1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion
b. Module Configuration

• Configuration layer selects:
  • which aspects?
  • onto which base modules are aspects applied?
    • explicit mapping
    • implicit mapping (weaving time, ...)
  • aspect pluggability ↔ implicit dependencies
  • fine-grained control ↔ keep it manageable
System-wide Configuration vs. Build Decomposition
Conventions

- ACC:
  - which aspects? local ↔ global
  - scope?

- current-and-sibling rule
## Module Configuration Issues

<table>
<thead>
<tr>
<th></th>
<th>preprocessor</th>
<th>compile-time</th>
<th>link-time</th>
<th>load-time</th>
<th>run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>language</strong></td>
<td>no control</td>
<td></td>
<td></td>
<td></td>
<td>programmatic</td>
</tr>
<tr>
<td></td>
<td>programmatic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>tool</strong></td>
<td>easily enforceable</td>
<td>weaver</td>
<td>complexity</td>
<td>deployment tools</td>
<td>decompose build</td>
</tr>
<tr>
<td></td>
<td>command line switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>user</strong></td>
<td>conventions</td>
<td></td>
<td></td>
<td></td>
<td>product line</td>
</tr>
</tbody>
</table>

- **Application-wide textual list**
- **Decompose build**

**How to manage dependencies?**
1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion
c. Build integration

- **pre-AOP:**
  - file-level composition

- **AOP:**
  - subfile-level, dynamic composition
  - whole-program view needed

current build systems

AOP-aware build systems
Repository

alternative: IDE/all-in-one
Support code

how to configure?

duplicate symbols?
Switching between 2 or more weavers

weaving time: same or different
Wrappers

undocumented interactions between wrappees?

GCC 3.0

weaver

GCC 4.0

vrijdag 14 december 2007
Modes

• AspectC++:
  • whole-program ↔ single-translation unit

• ACC:
  • batch ↔ sequential weaving

<table>
<thead>
<tr>
<th></th>
<th>simple</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td>accmake</td>
<td>tacc (aspects in base directory)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manually</td>
</tr>
<tr>
<td>local</td>
<td></td>
<td>manually</td>
</tr>
<tr>
<td>language</td>
<td>preprocessing</td>
<td>compiling</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td>tool</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>require all input</td>
<td></td>
</tr>
<tr>
<td></td>
<td>repository</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weaver intelligence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weaver modes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all-in-one build/IDE</td>
<td>whole-program view (in theory)</td>
</tr>
<tr>
<td></td>
<td>MAKAO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>technical integration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wrappers</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>preprocessing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>intermediate files</td>
<td></td>
</tr>
<tr>
<td></td>
<td>library/support code</td>
<td></td>
</tr>
<tr>
<td></td>
<td>switch weavers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>test woven code</td>
<td></td>
</tr>
</tbody>
</table>
1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion
d. Build order

- **pre-AOP:**
  - separate compilation (Modula)
  - independent compilation (C/C++/Java)

- **AOP:** order of aspects
  - explicit control?
  - weaver implementation-dependent?
# Build Order

<table>
<thead>
<tr>
<th></th>
<th>preprocessing</th>
<th>compile-time</th>
<th>link-time</th>
<th>load-time</th>
<th>run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>language</strong></td>
<td>lexical and configuration-based conventions</td>
<td>language control (precedence)</td>
<td></td>
<td></td>
<td>programmatic</td>
</tr>
<tr>
<td><strong>tool</strong></td>
<td>CT’s (base code-independent)</td>
<td>GROOVE (whole-program view needed)</td>
<td>feature map</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bounded quantification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>user</strong></td>
<td>build configuration flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **tool**
  - CT’s (base code-independent)
  - GROOVE (whole-program view needed)
    - feature map
  - bounded quantification

- **user**
  - build configuration flow

- **language**
  - lexical and configuration-based conventions
  - language control (precedence)

- **run-time**
  - programmatic

- **preprocessing**
- **compile-time**
- **link-time**
- **load-time**
1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion
### e. Incremental Compilation

#### Problems:
- **no weaver from scratch**
- **whole-program reasoning**
- **traditional incremental compilation not applicable**

<table>
<thead>
<tr>
<th>WeaveC</th>
<th>X build time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 (→ 1.3)</td>
</tr>
<tr>
<td>Aspicere</td>
<td>70</td>
</tr>
<tr>
<td>C4</td>
<td>4 (→ 2)</td>
</tr>
<tr>
<td>AspectJ</td>
<td>“4”</td>
</tr>
<tr>
<td>abc</td>
<td>8</td>
</tr>
<tr>
<td>Compose*</td>
<td>→ &lt;2</td>
</tr>
</tbody>
</table>
Aspect configuration

• limit scope of aspects in build configuration:
  • extra/missing matches
  • implicit dependencies?
  • weaver-dependent

• partition base code (e.g. AspectJ)
  • according to non-interacting aspects
  • weave into binary form
Caching

- source-to-source weavers (AspectC++)
- compile-time (AspectJ)
# Incremental Compilation

<table>
<thead>
<tr>
<th></th>
<th>prepro-</th>
<th>compile-</th>
<th>link-time</th>
<th>load-time</th>
<th>run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>language</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>tool</strong></td>
<td></td>
<td><strong>caching</strong></td>
<td></td>
<td></td>
<td><strong>by design</strong></td>
</tr>
<tr>
<td><strong>user</strong></td>
<td></td>
<td><strong>explicit weaver support</strong></td>
<td></td>
<td><strong>aspect configuration</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>partitioning system</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

→ limit expressivity aspect language?
1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion
Conclusion

- **No structural solutions** for build integration
- Many open questions... ⇒ **opportunities for research** :-)
- **AOP-aware build system?**
QUESTIONS?