Software Architecture
Recovery
from Build Processes

Eram ADAMS
Ghent University Software Engineering Lab, INTEC, Ghent University
http://users.ugent.be/~badams

Kris DE SCHUTTER
Lab On REengineering, University of Antwerp
http://faramir.ugent.be/~kdschutt

Outline

1. Why Look At Build Systems?
2. Software Architecture Recovery
3. Make
4. MAKAO
5. Rule-Based Approach
6. General Rules
7. Application-Specific Rules
8. Conclusions and Future Work
Build systems

Some history:
- ...-1977: ad hoc build and install scripts
- 1977: make (Stuart Feldman), most influential build tool
- later:
  - various clones (GNU Make, ...) and alternatives
  - build configuration systems like `make` and GBS

1. Why Look At Build Systems?

Case study with Aspicere:

More general:
- how to easily modify a build system?
- how to gain quick insight into build process?
- how to assess general software architecture?
2. Software Architecture Recovery

Software architecture recovery:
- software and build system co-evolve
- assumptions:
  - correct makefiles
  - modular source files (no giant implementation files)

Related work:
- Build-Time Software Architecture View [Tu01]
- Dali (and Rigi) [Kazman99], Portable BookShelf [Finnigan97], and Desire [Biggerstaff89]
- [Bowman99] Linux kernel architecture
  - conceptual architecture ⇒ concrete architecture
  - tedious discovery and population of subsystems

3. Make

Makefile

```
make_OBJECTS = ar.o arscan.o \
commands.o dir.o ... hash.o

make$(EXEEXT): $(make_OBJECTS)
  @rm -f make$(EXEEXT)
  $(LINK) $(make_LIBFLAGS) \
  $(make_OBJECTS) \n  $(make_LDADD) $(LIBS)

... commands
```

Directed Acyclic Graph (DAG)
4. MAKAO

Makefile Architecture
Kernel for AO

legend

Gython console

hull

Prolog

graph

Linux 2.6.16.18 kernel
• 2787 nodes
• 7465 edges
5. Rule-Based Approach

Observations:
- previous slide looks like a mess, even after layouting
- too much detail

Possible solution:
- define rules to modify graph:
  - general vs. application-dependent [Kazman99]
  - semantics-preserving ("cleaning-up") or not
- challenge: don't touch the code ↔ [Bowman99]
  - propagate clean-up passes back to build (configuration?)
  - system
6. General Rules (2)

Redundant dependencies:
- simple transitivity
- extended transitivity

* semantics-preserving
* faster build
* lose architectural info?

0 nodes
108 edges

not applied
6. General Rules (3)

Redundant dependencies (cont.):
- obsoleteness

11 nodes
17 edges

\[ \begin{align*}
\text{no object} & \\
\text{source} & \\
\end{align*} \]

- semantics-preserving if no commands tied to source node
- faster build

BEFORE
6. General Rules (4)

Raising level of abstraction:
- pulling up source file relations
- abstracting away source files

0 nodes
0 edges

929 nodes
984 edges
6. General Rules (5)

Raising level of abstraction:

- sandwich rule

\[
\begin{align*}
\text{a} & \rightarrow \text{b} & \rightarrow \text{c} \\
& \rightarrow \text{no regular file}
\end{align*}
\]

\[14 \text{ nodes} \quad 14 \text{ edges}\]

😊 abstraction

- rules influenced by style of build scripts
- some build systems have more/less architectural info
- lose architectural info?
kernel image
7. Application-Specific Rules (1)

- composite object files

897 nodes
1056 edges

- object
  - object

 высокоэффективный
7. Application-Specific Rules (2)

- unchaining redundant cycles

0 nodes
174 edges

build

build

object

directory

decouples tangled clusters

what does this construct mean?

file system support

network

ACPI drivers

kernel image

built-in drivers
8. Conclusions and Future Work (1)

Conclusions:
- work in progress!
- lots of clean-up and abstraction rules necessary
- build system's knowledge varies per project

Rules' effectiveness:

8. Conclusions and Future Work (2)

Future work:
- working out dependencies of kernel image
- other cases (GCC, vim, KDE, ...)
- applying clustering techniques
- feed clean-up rules back to build scripts
- come up with new rules
- does order of rules play a role?
- ...

<table>
<thead>
<tr>
<th>Node reduction</th>
<th>Edge reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References


