Software Architecture Recovery from Build Processes

Bram ADAMS
Ghislain Hoffman 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
1. Why Look At Build Systems?

Case study with Aspicere:

- Source code
- Makefiles
- Application
- Trace generating

More general:
- How to easily modify a build system?
- How to gain quick insight into build process?
- How to assess general software architecture?

Reverse-engineering

© 2006, Ghislain Hoffman Software Engineering Lab. All rights reserved.
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

Directed Acyclic Graph (DAG)

Makefile

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

make$(EXEEXT): $(make_OBJECTS)
@rm -f make$(EXEEXT)
$(LINK) $(make_LDFLAGS) \ 
$(make_OBJECTS) \ 
$(make_LDADD) $(LIBS)
```

⇒ de facto build tool/process model!
4. MAKAO

Makefile Architecture
Kernel for AO

legend

Gython console

Prolog

graph

hull
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 $\leftrightarrow$ [Bowman99]
- propagate clean-up passes back to build (configuration?) system
Lose the FORCE, Luke!

1 node
114 edges
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

\[ \begin{align*}
&\text{b} \\
&\text{c} \\
&\text{e} \\
\end{align*} \]

\[ \begin{align*}
&\text{b} \\
&\text{c} \\
\end{align*} \]

- 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
944 edges
6. General Rules (5)

Raising level of abstraction:

- sandwich rule

![Diagram](a \rightarrow b \rightarrow c) \quad \Rightarrow \quad a \rightarrow c

- no regular file

abstraction

- rules influenced by style of build scripts
  - some build systems have more/less architectural info
  - lose architectural info?
AFTER

system-wide headers

built-in modules

kernel image
kernel image
composite object
7. Application-Specific Rules (1)

- composite object files

897 nodes
1056 edges

highly effective
target __build

kernel image
built-in drivers

network

target __build
7. Application-Specific Rules (2)

- unchaining redundant cycles

\[\begin{align*}
\text{___build} & \quad \rightarrow \\
\text{a} & \quad \text{b} & \quad \text{c} \\
\text{object} & \quad \text{directory}
\end{align*}\]

0 nodes

174 edges

decouples tangled clusters

what does this construct mean?
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?
• ...

...
References


