MAKAO: Dealing with Legacy Build Systems

Bram Adams

Bram.Adams@UGent.be
http://users.ugent.be/~badams/
Ghislain Hoffman Software Engineering Lab
INTEC, Ghent University, Belgium

ABSTRACT

Legacy software systems do not solely consist of source code: various other types of artifacts play a role, notably the build system. As each considerable change to the software system potentially demands modifications of build-related files, maintainers and developers alike need to find their way around quickly. The MAKAO-framework aims to be a suitable visualisation and reverse-engineering framework for build systems, facilitating research regarding the need for aspect-oriented mechanisms when modularizing builds.

Keywords

AOP, legacy build systems, visualisation, reverse-engineering

1. CONTEXT

During a reverse-engineering experiment [2] using Aspicere, our aspect language for C, on an industrial legacy application, we encountered a paradox. On the one hand, Aspicere's logic-based pointcuts and template mechanism allowed us to write two concise, generic advices to trace the whole source code system. But here is the catch: as Aspicere's weaver basically is a sourcecode preprocessor, this meant that every source file in the system needed to be preprocessed prior to compiling. Aspicere obviously had to be integrated into the existing build process in two phases.

First, the weaver had to be executed at the right time during the build. Ideally, this would just mean making the compiler commands and other tools point to a wrapper script around the aspect weaver. A (minor) nuisance here, is finding out all commands used and writing suitable wrappers. A subtle issue arises when two or more commands invoke each other, as weaving already woven code would not do what we intended.

Second, we needed to link a file generated by Aspicere with every library or executable, but the lack of proper tool support prevented this. Legacy build systems are very complex processes composed of compiling, linking, preprocessing, populating databases, ... Moreover, there are likely various paths for separate architectures and platforms. Whenever one needs to perform a non-trivial change to some software system, chances are high that the build system also needs to change. The impact of these modifications currently can only be verified by trial-and-error.

2. PROPOSAL

These observations made us think about an extensible visualisation framework for build systems, like Ant Explorer[1], but more powerful. At its core, it should be capable of showing the (static) structure of a build system, indicating the various types of build artifacts and maintaining links to the actual build files. Zooming in on regions producing the particular artifact type one is interested in, is another prerequisite. Finally, one should be able to correlate the behavior of a build run to the static build structure.

All this should be tested on several industrial and open source projects, from small to large scale to gauge how the average build system structure compares to modularity metrics. We want to find out whether build systems would benefit from aspect-oriented techniques. In that case, the framework will be extended with aspect weaving capabilities, resulting in the Makefile Architecture Kernel for Aspect Orientation (MAKAO[2]).

3. FIRST STEPS

As the underlying model of build systems like "make" and "ant" is based on a Directed Acyclic Graph (DAG) [1], we are currently experimenting with some graph manipulation tools. An initial prototype extracts the dependency graph generated during a concrete "make" run and imports it into GUESS[3], a graph exploration tool with an embedded Jython-based scripting engine. Metadata can be attributed to each node of the graph, which allows e.g. to color them or to paint convex hulls around all nodes of the same makefile.

Statically parsing makefiles will be required to visualize architecture-specific variants. Finally, graph transformation techniques could be used to implement weaving functionality.

4. CONCLUSION

Build systems are inherently part of and tied to (legacy) software systems. To facilitate quick and easy understanding, MAKAO offers visualisation of both build structure and behavior. Metrics will indicate the existence or lack of modularity, in which case aspect-oriented techniques can be evaluated and applied.

Acknowledgements

The author wants to thank Kris De Schutter, Andy Zaidman and Herman Tromp for their support and cumulative wisdom.

5. BIBLIOGRAPHY

[1] S. I. Feldman. Make-a program for maintaining computer programs. *Software - Practice and Experience*, 1979.

[2] A. Zaidman, B. Adams, K. De Schutter, S. Demeyer, G. Hoffman, and B. De Ruyck. Regaining lost knowledge through dynamic analysis and Aspect Orientation - an industrial experience report. In *CSMR*, 2006.

[1] `http://www.yworks.com/en/products_antexplorer_about.htm`

[2] Makao (or Mau-Mau) is also the name of a popular Polish game.

[3] `http://graphexploration.cond.org/`