Ant Build Maintenance with Formiga

R. Hardt and E. V. Munson
University of Wisconsin-Milwaukee, USA

Research Problem and Motivation

- **Build maintenance** refers to changes made to the build system as a software project evolves over time
- Adams et. al showed that the build system:
  - grows in size and complexity as the source code does
  - needs to evolve in parallel with the source code
- Build system maintenance imposes a 12%-36% overhead on the overall development of a software project [Kumfert and Epperly]
- Up to 27% of the work items involving production source code changes require accompanying build maintenance [McIntosh et. al]
- Despite these facts, little support for build maintenance exists

Background and Related Work

- **SYMake** [Tamrawi et. al]
  - Produces a **symbolic dependency graph** from a makefile and is used in a tool that supports build refactoring and error identification
  - Uses **static build analysis** to analyze the build files allowing it to discover information about all build configurations
- **MAKAO** [Adams et. al]
  - Constructs a makefile's build dependency graph that supports querying and filtering and allows for build refactoring and validation
  - Uses **dynamic build analysis** to analyze an execution of the build for a particular configuration

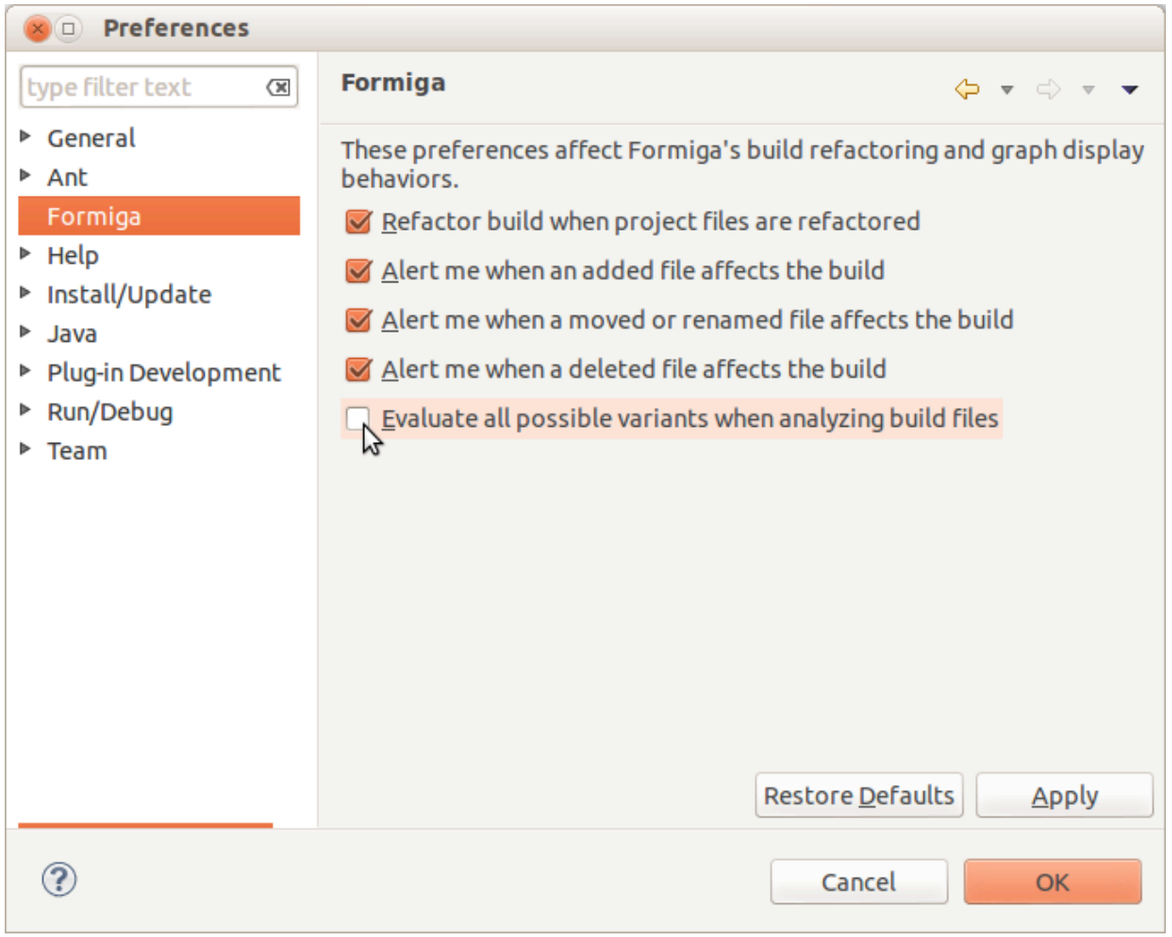
Approach and Uniqueness

- Formiga is a build maintenance and dependency discovery tool implemented as an Eclipse plugin for use with software projects using the Ant build system
- Primary uses:
  - Build maintenance caused by source code refactoring
  - Build system refactoring or fine-tuning
  - Identification of build dependencies in a software project
  - Understanding differences between build file versions (Future Research)

Approach and Uniqueness

Build Maintenance Caused By Source Code Refactoring

- Formiga is able to update build files when project resources are moved, renamed, or deleted using the IDE
- Developers use the standard Eclipse refactoring operations to refactor project resources
- Corresponding build updates are made automatically, but users may choose to require confirmation before each update
  - Confirmation displays the affected target, task, attribute, and old and new attribute values



Approach and Uniqueness
Build Maintenance Caused By Source Code Refactoring

- References to project resources in an Ant build file are updated based on:
  - The type of refactoring operation (move, rename, delete, or add)
  - The type of reference
    - **Indirect**: a reference that includes a wildcard pattern
    - **Direct**: a reference that does not include a wildcard pattern
  - When references are updated, Formiga uses existing property references whenever possible

Approach and Uniqueness
Build Maintenance Caused By Source Code Refactoring

- **Moving or renaming a file**
  - If referenced directly, that reference will be updated to reflect the new path
  - If referenced indirectly
    - If the existing reference still refers to the file's new path, no changes will be made
    - If the existing reference no longer refers to the file's new path, then a new reference will be appended to the existing reference
  - Moving a file may imply that it should no longer be treated the same way as files in its previous directory, in which case the user can reject the update

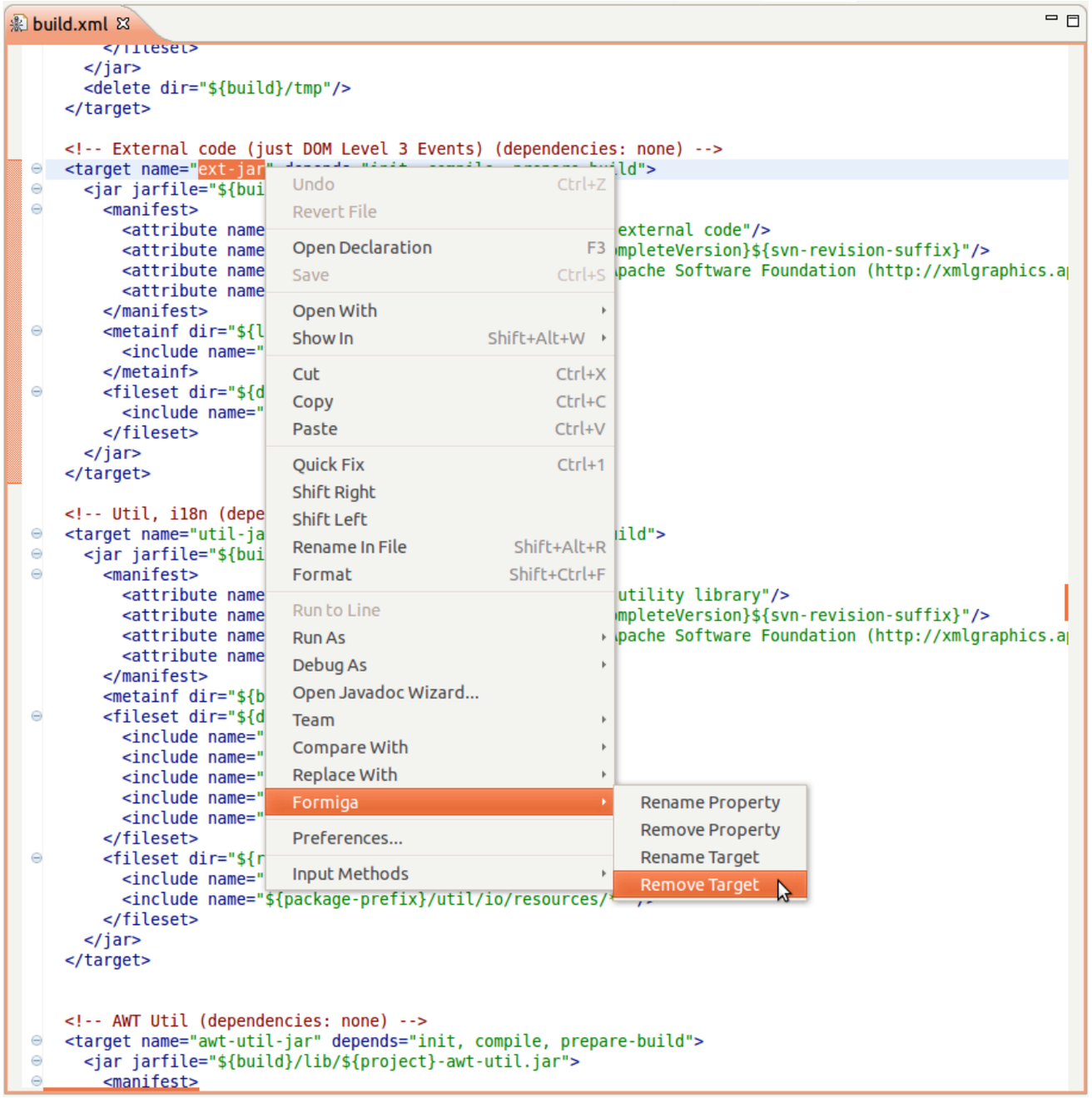
Approach and Uniqueness
Build Maintenance Caused By Source Code Refactoring

- **Deleting a file**
  - If referenced directly, that reference will be removed
  - If reference indirectly, no changes will be made
    - The reference may still refer to existing files or to a path that will later be populated with files relevant to the task
- **Adding a file**
  - Formiga does not update the build system but does report affected targets and tasks

Approach and Uniqueness

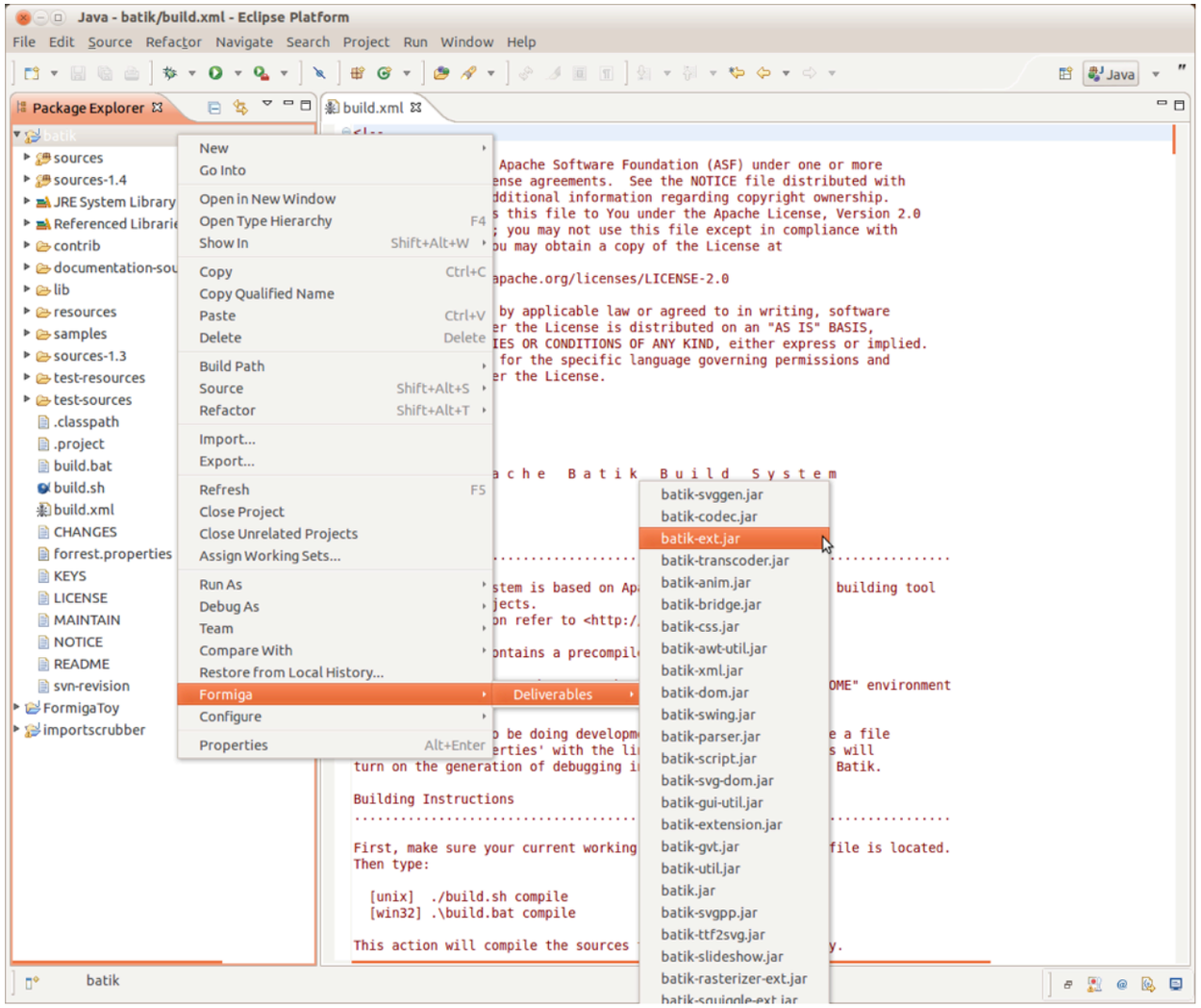
Build System Refactoring or Fine-Tuning

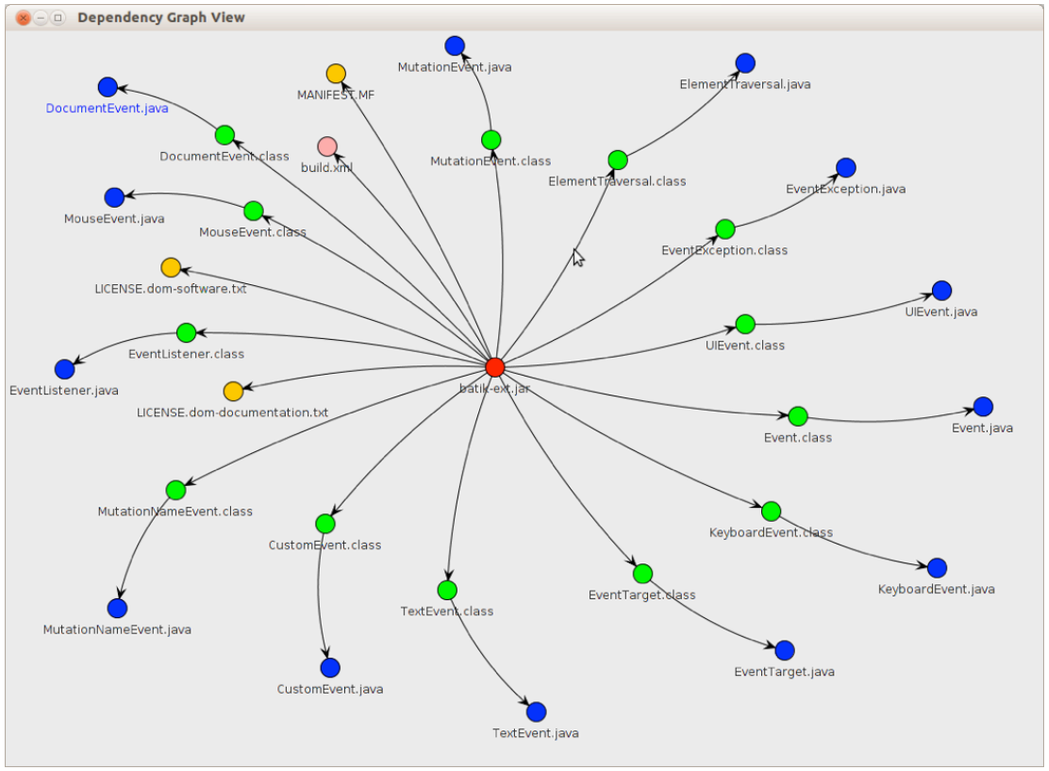
- Target and property renaming and removal
  - Renaming a target or property updates its declaration and all references with the new name
  - Removing a target deletes it and will ask the user if any now unused targets should also be removed
  - Removing a property will replace its references with its (previously) specified value



Approach and Uniqueness

Identifying Build Dependencies





Approach and Uniqueness
Identifying Build Dependencies

- Dependencies are identified using a modified version of Ant
  - Formiga doesn’t execute Ant tasks that read/write to the filesystem
  - Instead, Formiga keeps track of the accessed files in its **filespace**
    - The **filespace** is a virtual filesystem maintained in memory
    - Its file models keep track of locations and dependencies
  - When the target “execution” has finished, filespace files are written to an embedded database
  - Mostly dynamic approach with static handling of tools

Approach and Uniqueness
Identifying Build Dependencies

- Configuration handling
  - Formiga supports configurations that are implemented using **conditionally set properties** (**CSPs**)
    - **CSPs** are properties whose value (or instantiation) is set based on the result of some condition
    - CSPs are created in Ant using the `condition`, `available`, and `uptodate` tasks

```
<condition property="app-extension" value="app" else="exe">
    <os name="Mac OS X"/>
</condition>
```

  - When a CSP is referenced within a target, Formiga will “execute” that target and all remaining targets twice (once for each CSP value)

Constraints

- Tasks not packaged with Ant and **arbitrary execution tasks** (**AETs**) cannot be processed in the same way as tasks packaged with Ant
  - **AETs** are Ant tasks that execute a specified command or Java class
  - Formiga could support AETs by parsing structured comments that describe the input and output files read and written by the task
  - Comments could allow Ant property references and wildcard patterns

Contributions

- Formiga's implementation as an Eclipse plugin allows it to automatically update the build files when the project resources are refactored
  - This implementation also facilitates ease of use since build maintenance operations are similar to source code maintenance operations
- Formiga's analysis approach allows dynamic analysis benefits without all of the costs
  - Runs more quickly than a strict dynamic approach as it doesn't execute every tool
  - Won't produce any undesirable side-effects caused by destructive build operations

Thank you!