WG4: Tool Integration

Joe Kiniry and Erik Poll

COST Action 0701 Working Group Meeting, Gothenburg, Sweden, June 2008
Why Build Tools?

• tools are the fulcrum to getting FMs adopted in industry and more people involved in our field

• provide evidence that theory has utility

• and perhaps...
  generate intellectual property
Tool Integration

• What does integration mean? How do we “integrate” tools today? Tomorrow?

• APIs, common languages, pipes and components

• ad hoc Frankenstein Monsters

• ?
Potential WG4 Goals

• document and foster best practices and “gold standards” in tool construction, community building, and tool integration

• integrate with mainstream, real-world software development practices

• overcome fragmentation, avoid duplicate development, ensure coverage

• catalog and experiment with case studies for other working groups
Where to Focus Tool Work?
Industrial Relevance

- industry programmers are our stakeholders
- we must understand what they want/need
- a new formal method only is adopted if there is excellent tool support
- going “mainstream” only happens if a tool has the right non-functional properties for adoption
Theory/Practice Duality

- theoreticians must inform tool developers of most promising new theory
- tool developers and users must inform theoreticians of their critical problems
Tools of the Trade
Tool Classes

• Verification Tools: Boogie, Jive, the JML tool suite, KeY, Krakatoa, LOOP, Mobius PVE

• Static Checkers: CheckStyle, ESC/Java, FindBugs, PMD

• Model Checkers: Blast, Bogor, Java Pathfinder

• System Specification Systems: RODIN (B), Overture (VDM), Z/Eves

(tools in italics have been used in our research at UCD)
Tool Classes II

- Logical Frameworks: Coq, Isabelle, PVS
- Provers: CVC3, Fx7, Simplify, Yices, Z3
- Specification Languages: JML, OCL
- Intermediate Representations: BoogiePL, ESC-GC, Simplify, SMT-LIB, TPTP, Why
Tool Builder Behavior

I *almost* wrote a behavioral specification of the typical tool developer....
Best Practices

• know what the best tools are today
  • must understand what “best” means
• let someone else inventory tools (FME)
• coordinate with other efforts
  • JML Reloaded, MSR, SMT-LIB, SRI
• communicate best practices in tool sales and marketing (i.e., adoption and evolution)
Anti-Practices

• no release? the tool does not exist
• lack of documentation, support
• scientific responsibility/good citizenship
COST Actions
Missing Tool Artifacts

- best practices in process and coordination
- clearly standardized interfaces
- assessment with common benchmarks
- textbooks that incorporate FM tool use
- pedagogical materials for tool training
- accurate characterization of tools
Concrete Action Items

• programming contest involvement (i.e., SCORE, ACM, TopCoder, Imagine Cup)
• tool competitions (i.e., SAT, SMT, etc.)
• look at one tool in depth per meeting
• tools in training schools
• STSM to facilitate tool integration and dissemination of best practices
• coordinate with ongoing work in GC6

• Mondex, pacemaker, flash file store, Linux kernel, Microsoft Hypervisor, e-voting
UCD Tools Summaries
- **name**: ESC/Java2

- **author/institution**: dozens of authors and institutions coordinated by Kiniry

- **description**: statically checker for common errors/lightweight verifier

- **input/output**: JML-annotated Java → human readable warning messages

- **implementation**: written in JML-annotated Java, has thousands of system and unit tests, has a high-level informal architecture description

- **license**: Hewlett-Packard Open Source License

- **availability**: source and binary packages freely available from UCD

- **platforms**: runs on Linux, Mac OS X, Windows, and some misc UNIXen

- **status**: beta (some incomplete advanced features & hundreds of pages of documentation but quite a bit of docs are still missing)

- **usage**: 10Ks of downloads, used in dozens of universities, research groups, and companies for teaching and research in verification

- **integration**: several public APIs that are also shipped as Eclipse plugins

- **example artifacts**: dozens of case studies, thousands of tests, all open source

- **future**: going into maintenance-mode now, no new major developments by core team, future development and experiments will be based on JML4, but we expect ESC/Java2 to be used for a couple more years as all transition
- **name**: Mobius Logging Framework

- **author/institution**: Joe Kiniry/Caltech, Nijmegen, UCD

- **description**: a formally specified and verified API for logging program behavior

- **input/output**: used in Java programming via API → human readable warning messages

- **implementation**: written in JML-annotated Java, has hundreds of system and unit tests, has a high-level formal architecture description

- **license**: GPL

- **availability**: source and binary packages freely available from UCD

- **platforms**: runs on all platforms

- **status**: beta (new version is being verified now)

- **usage**: 100s of downloads, used in a handful of projects, influenced the design of the Java logging framework (java.util.logging) and Apache’s log4j

- **integration**: public JML-annotated API provided as an Eclipse plugin

- **example artifacts**: some examples, a handful of system tests, all open source

- **future**: ongoing work into integrating ideas into Java and Apache logging frameworks, used inside of the Mobius PVE, hope to influence future versions of aforementioned frameworks
name: JavaFE, the H.P. Java Front-end

author/institution: dozens of authors and institutions coordinated by Kiniry

description: scanner, parser, typechecker for JML-annotated Java

input/output: JML-annotated Java ➔ typed AST

implementation: written in JML-annotated Java, has thousands of system and unit tests, has a high-level informal architecture description

license: Hewlett-Packard Open Source License

availability: source and binary packages freely available from UCD

platforms: runs on Linux, Mac OS X, Windows, and some misc UNIXen

status: beta (some missing documentation, only works on Java 1.4)

usage: 10Ks of downloads, used in dozens of universities and research groups for teaching and research in verification

integration: several internal APIs that are also shipped as Eclipse plugins

documentation: dozens of case studies, thousands of tests, all open source

future: going into maintenance-mode now, no new major developments by core team, future development and experiments will be based on JML4, but we expect JavaFE to be used for a couple more years as all transition
• name: BONc, the BON compiler framework

• author/institution: Fintan Fairmichael at UCD

• description: scanner, parser, typechecker for the BON specification language

• input/output: textual BON specifications ➔ typed Java AST

• implementation: written in JML-annotated Java, has hundreds of system and unit tests, has a high-level formal architecture description

• license: BSD

• availability: source and binary packages freely available from UCD

• platforms: runs on all platforms

• status: beta

• usage: hundreds of downloads, used at UCD for teaching

• integration: public API (not yet available as an Eclipse plugin)

• example artifacts: handful case studies, hundreds of tests, all open source

• future: new development, will be used as the foundation for ongoing research in high-level specification with formal refinement, expect it to be used by a handful of universities in teaching this coming year, hope that it will become a popular platform for teaching and research into high-level specification
FreeBoogie, an FLOSS version of MSR’s Boogie

Radu Grigore at UCD

scanner, parser, typechecker for an extended version of the BoogiePL intermediate representation language for program verification

BoogiePL ➜ typed Java AST

written in Java, has hundreds of system and unit tests, has a high-level formal architecture description

MIT

source and binary packages freely available from UCD

runs on all platforms

alpha

dozens downloads

public API (not yet available as an Eclipse plugin)

handful case studies, hundreds of tests, all open source

new development, will be used as the foundation for ongoing research in efficient formal static checking, expect it to be used by a handful of universities in teaching this coming year, hope that it will become a popular platform for teaching and research into intermediate representations and verification
name: KOA, an FLOSS platform for e-voting research and experimentation

author/institution: LogicaCMG; Hubbers, Kiniry, Oostdijk at Nijmegen; Cochran, Fairmichael, Kiniry, Morkan at UCD

description: platform for e-voting research into, and performing, e-voting

input/output: ballots via internet → accurate tally

implementation: written in JML-annotated Java, has hundreds of system and unit tests, has a high-level informal architecture description

license: GPLv2

availability: source and binary packages freely available from UCD

platforms: runs on all platforms, primarily tested on Linux and Mac OS X

status: beta

usage: hundreds downloads

integration: public API (not yet available as an Eclipse plugin)

example artifacts: handful case studies, hundreds of tests, all open source

future: in maintenance-mode, no new development, used mainly in case studies in e-voting and verification, please do not use for elections that matter!
• **name**: Simplify

• **author/institution**: Nelson et al. at DEC SRC; Kiniry at Nijmegen/UCD

• **description**: automatic theorem prover for AUFLA

• **input/output**: FOL formulae in Simplify syntax ➔ valid/invalid+counterexample/timer

• **implementation**: written in Modula-III, has hundreds of system and unit tests, has a high-level informal architecture description

• **license**: Hewlett-Packard Open Source License

• **availability**: source and binary packages freely available from UCD

• **platforms**: runs on all major and some minor platforms

• **status**: stable

• **usage**: 10Ks of downloads, widely used as a default FOL theorem prover

• **integration**: used via pipes and available as Eclipse plugin

• **example artifacts**: dozens of case studies, hundreds of tests, all open source

• **future**: in maintenance-mode, no new development, used mainly in many research groups’ tools as primary FOL prover
• **name:** the Mobius Program Verification Environment

• **author/institution:** dozens of authors coordinated by Kiniry at UCD

• **description:** preconfigured Eclipse IDE for program verification

• **input/output:** JML-annotated Java ➔ classfiles, warnings/errors, unit tests, proofs

• **implementation:** based upon Eclipse, written nearly wholly in Java, has dozens of subsystems with thousands of system and unit tests, has a high-level informal architecture description

• **license:** a mixture of several FLOSS licenses, including the EPL

• **availability:** source and binary packages freely available from UCD

• **platforms:** runs on all major and some minor platforms

• **status:** beta

• **usage:** 100s of downloads, used in teaching at UCD and within Mobius

• **integration:** integrated platform that contains dozens of Eclipse plugins

• **example artifacts:** dozens of case studies, hundreds of tests, all open source

• **future:** under active development, hope for it to be used for years as the foundation of, or at least a case study in, the idea of a *comprehensive verification bus and platform*
• **name:** the JML2 Tool Suite

• **author/institution:** dozens of authors coordinated by Leavens at UCF

• **description:** typechecker, compiler, runtime assertion checker, documentation generator, and unit test generator for the JML specification language

• **input/output:** JML-annotated Java ➔ classfiles, unit tests, documentation

• **implementation:** written in Java with some JML annotations and Javadocs

• **license:** GPLv2

• **availability:** source and binary packages freely available from SourceForge

• **platforms:** runs on all platforms

• **status:** stable but under constant development and evolution

• **usage:** 1000s of downloads, used in teaching and research at dozens of universities and companies

• **integration:** command-line tool and wrapped in JML4 Eclipse plugin

• **example artifacts:** dozens of case studies, hundreds of tests, all open source

• **future:** under active development
• *name*: the JML4 Tool Suite

• *author/institution*: dozens of authors coordinated by Chalin at Concordia

• *description*: typechecker, compiler, runtime assertion checker, extended static checker, and full-functional verifier for the JML specification language

• *input/output*: JML-annotated Java ➔ classfiles, docs, theorems

• *implementation*: written in Java with some JML annotations and Javadocs

• *license*: GPLv2 and Eclipse license

• *availability*: source and binary packages freely available from SourceForge

• *platforms*: runs on all platforms

• *status*: pre-alpha release and under active development and evolution

• *usage*: first release has not yet taken place; we expect it to replace JML2, perhaps by the end of 2008

• *integration*: command-line tool and (set of) Eclipse plugin(s)

• *example artifacts*: several research papers, hundreds of tests, all open source

• *future*: very under active development