B3 | Modeling Unbreakable Builds

Eclipse Modeling Day
Nov 16-19, 2009

Ed Merks, Bjorn Freeman-Benson, Miles Daffin

agenda

1. usage scenario: eclipse release assembly
2. b3 project: overview
3. usage scenario: enterprise provisioning

1. usage scenario: eclipse release assembly

eclipse as singleton

- Eclipse starts as one project: Java IDE.
  - Just one .zip file to download
  - Just one update site for updates
  - Extra features added manually through complex update menus and wizards
  - Java IDE built using custom scripts and headless instances from IBM .

eclipse as a small collection

- Eclipse expands to a few core projects
  - Each still responsible for own builds
  - Some copy the complex Java IDE build; others use simpler custom, scripts
  - Each has its own downloads and update site(s)
  - Each has its own release schedule and versioning scheme
  - Consumers struggle with version compatibility and … (worse) … assembling the pieces for a working installation

assembling the pieces

- Assembly has been a problem since the start of the software industry
- Assembly = “which source files are compiled and linked into an .exe” or "which versions of which components are packaged into coherent whole“
- Problem grew as software shifted from a single binary executable into multiple source files/ executables and then multiple libraries

many solutions

- Multiple generations of tools: make, autoconf, MSBuild, cook, ant, maven, and even winzip and msi.
- Commonality: assembles result X from inputs A, B, C and dependencies.
- Result X may be:
  - a compiled binary
  - an installed program...
  - a library of compatible components

callisto and europa (2006/7)

- Need: consumers wanted single Eclipse release with all the correct pieces
  - Solved by internalizing problem assembling a distro: the Callisto coordinated release.
- Consumers happy but assembling the single update site was painful
  - Committer Dave Williams (IBM) wrote set of perl, grep, awk, and ant scripts to get the bits & package them into an update site.
  - It broke more often then it worked!

new problems

- It broke due to:
  - New hidden dependencies
  - New explicit dependencies people forgot to declare in the xml files
  - Version numbers changed
  - Included old & unused (thus broken) code
  - Conflicting dependencies
- It worked by:
  - loading everything into an Eclipse image
  - checking to see if it worked
  - slow and inherently error prone

ganymede and galileo (2008/09)

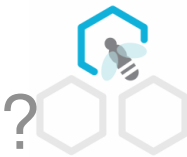
- "Dave-o-matic"
  - Committers Thomas Hallgren and Henrik Lindberg (Cloudsmith) created new system to replace the custom assembly scripts
  - Worked better because it used the (powerful) Eclipse/Buckminster technology to fetch components and resolve dependencies.
  - Changing output format from the old update site (Ganymede) to new p2 site (Galileo) relatively trouble-free because descriptions stayed constant when the tool changed
  - Still not perfect: custom external scripts and incomplete XML configuration files

what next?

- Simultaneous release requires same things as anyone working on complex component-based software
- Formal descriptions of.
  - components
  - dependencies
  - processing
  - “the result”
- Execution engine that understands all the different component metadata and file formats.
- In other words, modeling!

2. b3: project overview

target problem: why are builds so #$%~! complicated?



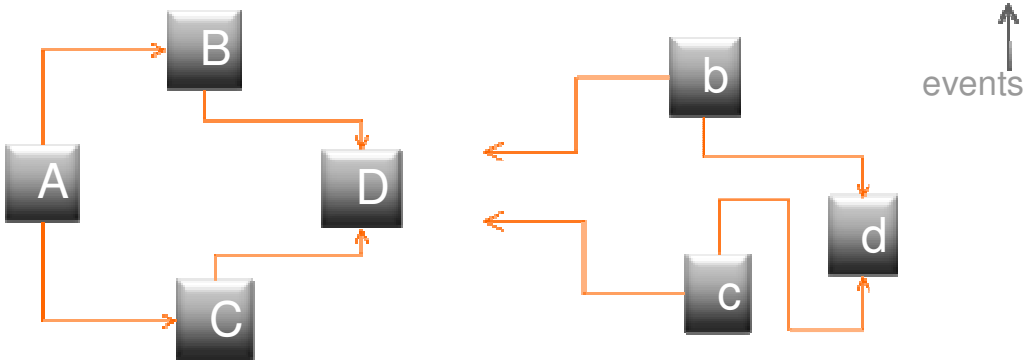
- Software is developed with the latest technology…
  separation of concerns, dependency injection, declarative services, OSGi, modeling, etc.
- Is built & assembled using the oldest technology...
  hand-coded and monolithic scripts created through successive hacks
- So it's no surprise everything breaks whenever something changes or someone else tries to run it

b3 project objective

- New generation of Eclipse technology to supporting build & assembly processes that are simple, repeatable, and adaptable:
  - declarative, model-driven, DSL-based process definition
  - support for discoverable, reusable build/assembly actions
  - support for composable build/assembly processes
  - separation between process definition and execution (who/where/when)
  - debuggable processes
- Project will deliver a successor to current PDE/Build and consolidate/clarify related technologies (p2, Buckminster, Athena, etc.)

b3 functional overview

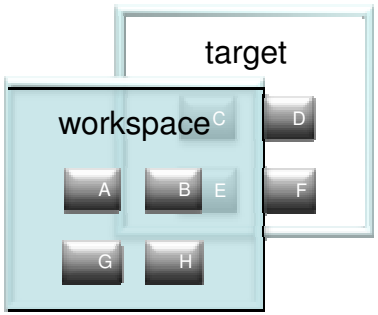
build model | **“advice”** | **engine** | **actors**



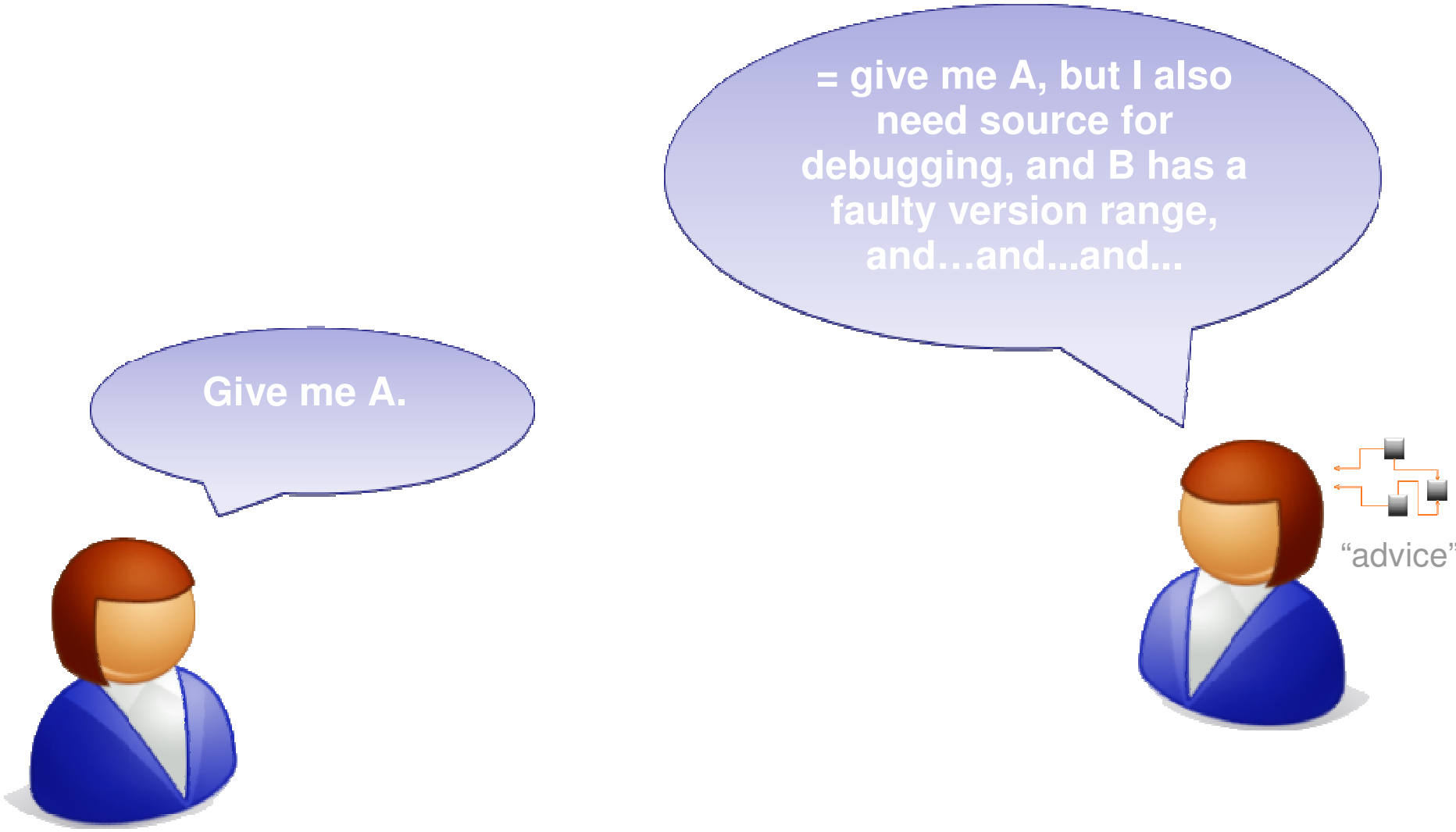
for(;;) …javac –xyz
ant …
gcc …

import
(materializers)

resolvers
meta data translators
connectors



goal: maximal simplicity



drill-down: b3 DSL

- B3 build files contain info about:
  - how the environment should be set up
  - how the model should be advised
  - what actions can be performed on the build file

- B3 build files are an XText DSL
  - concise, powerful way to specify build details based on "advice"
  - comparison: XML-based approaches
  - Alternatives to the build file are also possible
    (i.e. programmatically using EMF, MoDisco, Xtext)

drill-down: b3 DSL

- Declarative syntax for
  - build unit (name, version, visibility, parallel or sequential execution)
  - build unit interfaces (compare to p2 BU namespace)
  - provided and required capabilities
  - build part structure
  - properties
  - advice
  - repositories (and more advanced resolution support)
- Script syntax for build part actions
  - javascript like syntax for imperative programming
  - functional programming additions
  - literal queries (selecting model features, files, results etc.)
  - set operations, select, reject, collect, exists, foreach

build file example

```
unit {
    requires {
        eclipse.feature/org.myorg.myproj/1.0;
    }

repositories {
    platform:plugin:/;
    platform:resource:/;
    p2:http://www.someplace.good/updates-3.5;
    svn:svn+ssh://org.myorg.repo/productx;
    }

main {
    input { org.myorg.myproj#siteP2;
        org.myorg.myproj.packaging;
        }
    }
}
```

broad implications

- “Building" is more than just producing a binary from source
  - Assembling components
  - Producing a single exe
  - Producing an Eclipse update site for features
  - Producing an msi installable unit
  - Installing an msi on a computer (!)
  - Producing an internal library of “blessed” & compatible components

3. usage scenario: enterprise provisioning

overview

- Enterprise constraints & consequences
- Enterprise provisioning challenges
- Past: delivering 3.2 - 3.4 with internal solution
- Present: delivering 3.5 using b3 predecessor
- Future: building on b3

enterprise constraints & consequences

- Constraints: cannot allow developers free access to external Eclipse environment
  - Against firm policy
  - Risky (malware,legal & licensing)
  - Inefficient, chaotic & hard to support
- Consequences: restrict users to private Eclipse inside the firewall
- Requirements:
  - Easy to keep up to date
  - “Clean” IP and safe to use
  - Provides transitive closure of all dependencies

provisioning challenges

- Objectives:
  - Provide everything our developers need.
  - Ensure internal environment is stable and reliable.
  - Reduce Eclipse TCO
- Requirements
  - Security & Legal Requirements
  - Repository aggregation & Mirroring Requirements
  - Installation Requirements

security & legal requirements (1)

- Primary requirement: restrict user access to internal repos for browsing, installs and updates
- Secondary requirement: purify repositories of malicious code and problematic IP

aggregation & mirroring requirements (2)

- Mirror Eclipse Platform + 'Jupiter Moon' repositories (core stuff)
- Shopping list of 3rd party software from: update sites; p2 repositories; downloads (update site, p2 repository, feature, plugin)
- Ensure all dependencies are met internally
- Add homegrown stuff to the mix
- Keep resulting configuration up to date
  - Reduce lag between external and internal updates
  - Ideally, get updates automatically
- Expose updated configuration to users in stages
- Edit the configuration easily (add/remove software)

installation requirements (3)

- Eclipse installer must:
  - install a version of eclipse based on a custom profile (e.g. a shopping list of different plug-ins sourced from internal repositories)
  - enable the install to be run in a custom environment (e.g. perforce executables on PATH so plugin will work)
  - install an arbitrary selection of “other stuff” with Eclipse install (e.g. JDKs)
  - create a “default” install configured with standard global preferences (e.g. java code format, installed JREs)
  - support desktop shortcuts, etc.
  - be reliable & easy to configure and maintain

past: internal solution for 3.2 - 3.4

- Internal solution was basically, a nightmare!
  - Mirroring: did it work? Gave up all Eclipse-based mirroring tools for Ganymede SR2 (used wget from unixheads...)
  - Proprietary mirroring tools not much help
  - Test process was manual and awkward (install everything into a vanilla Eclipse!)
  - Complex ant scripts needed to post-process mirrored repos
    - prevent Eclipse from “phoning home”
    - hard coded to unstable Eclipse implementation internals
    - generate p2 metadata
  - Installer was a hack
  - Four internal repositories to be kept in sycn
  - Could take 3 days to update the main 3rd party repo

present: b3 predecessor for 3.5

- B3 predecessor technologies (Buckminster Aggregator + P2 director) makes it manageable
  - Use Buckminster Aggregator for mirroring, verification, testing, etc.
  - Only two repositories needed to create releases: private Galileo base repo + public “mega“ repo aggregating everything else
  - Only maintain 2 build models needed: one for each repo.
  - Installer uses P2 director to install straight from the mega repo. Much simpler.
  - Can build new release of the mega repo in 2-3 hours instead of days

future: building on b3

- Use b3 models to drive/simplify entire Eclipse delivery process
  - Model build from revision control to source repository (input for aggregation)
  - Add heuristics for coping with aggregation build problems: missing repositories, IUs, artifacts.
  - Add heuristics for auto-update of pre-defined aggregation model
  - Model more of the installation (non-eclipse IUs, global preferences, etc.)
  - Intelligent error reporting: “this is what I found” … “this is what I fixed; this” … “ this is what I couldn’t fix and why”
  - ->Ultimately, just run it all headlessly, once a week.!
  - B3 IDE – easy-to-use tools for designing, running, testing and debugging b3 builds
- Extend into other build/assembly domains: Java; C++; .Net, etc..

VISIT US AT

http://www.eclipse.org/b3
http://wiki.eclipse.org/b3
nntp:news://news.eclipse.org/eclipse.b3
irc: irc://irc.freenode.net/#eclipse-b3