Eclipse b3

Eclipse Project Creation Review

November, 2009

Henrik Lindberg, Cloudsmith Inc

b3 objectives

…to develop a new generation of Eclipse technology to simplify software build and assembly.

- The work of the project will be a complement and potential successor to the PDE/Build component.
- The project will have three key objectives:
  - Making build processes simpler, repeatable, reproducible, adaptable and possible to debug
  - Bridging discontinuities between building and provisioning software
  - Aligning potentially overlapping Eclipse build technologies

b3 ambitions…

- execute in the exact same way when run by anyone, anywhere in the develop-test-deploy cycle
- ensure that when a build fails, it fails for reasons that can be discovered, are clear and reproducible.
- Developing a declarative, model-driven approach to build definition and execution will be essential to realizing this ambition. Clear separations will be created between concerns of build definition and build execution. Build definitions will describe build input, tooling, platform constraints and processes in ways that are amenable to transformation and manipulation. Where possible, build models will be automatically generated from project structure.

Key Technologies

- Key technologies to be developed
  - models describing various aspects of the build domain as well as the relationship between build and provisioning
  - engines that interpret and act on these models
  - a build execution debugger
  - an easy to use, compact and powerful syntax to drive builds
- Key technologies used
  - EMF
    - Xtext
    - CDO
    - Potential integration with MoDisco
  - p2, PDE, JDT, Team Support (SVN, CVS)

Key Technologies

- Leverage and support of established Eclipse
  - Buckminster and PDE Build are directly relevant to the work of this project, and we intend to combine their respective strengths and capabilities
  - Creating a common successor technology, or folding one technology into the other, will be within scope of this project.
  - Possibility to integrate/collaborate with future Xtext technologies for common expression language, execution and debugging of DSL.

Initial Focus

- Next generation PDE/Build & Buckminster
  - Initial release will focus on the functional scope of PDE/Build, and Buckminster.
  - Future versions are intended to also work in other domains.
- CI-integration
  - The goal is to integrate well with CI systems (Hudson, Bamboo, Cruise Control etc.).
  - Later versions may contain more CI functionality, if there are technical advantages, and demand from the community.
- Part of an overall process
  - It is not an initial goal to describe an organization's various development streams, catalog of such streams, auxiliary information about development streams, or overall orchestration of such streams.
  - It is b3's ambition to fit well in such an overall process
  - Later versions of b3 may expand the support in this area.

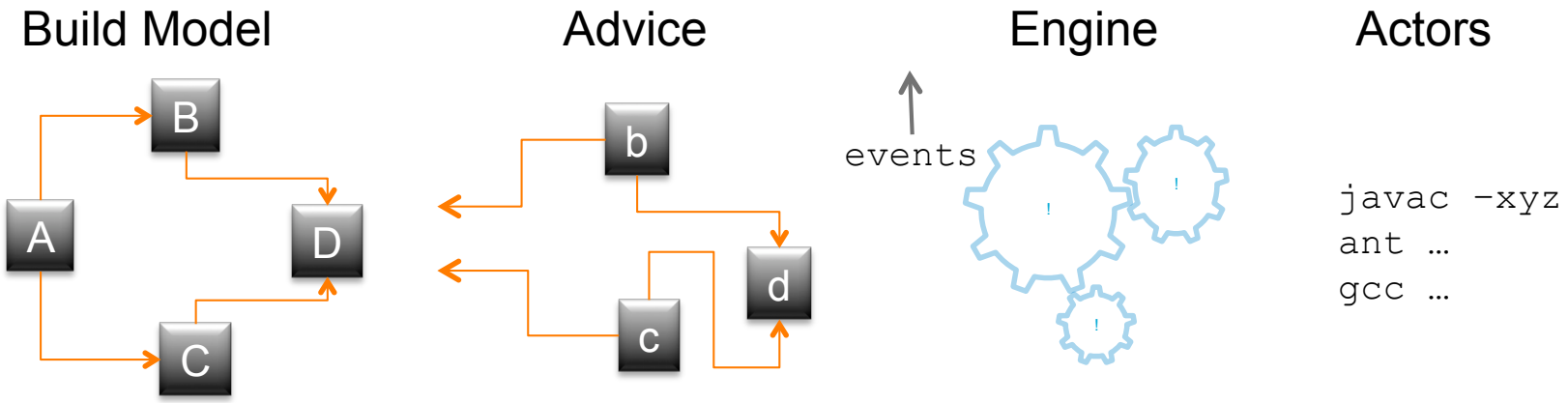
Initial Comitters

- Ed Merks (Project Co-Lead)
- Thomas Hallgren (Project Co-Lead)
- Henrik Lindberg
- Karel Brezina
- Stefan Daume
- Bjorn Freeman-Benson
- Oisin Hurley
- Eike Stepper
- Chris Aniszczyk
- Andrew Niefer
- Ian Bull
- Filip Hrbek

Each initial committer is currently an active Eclipse committer on other projects. There are 13 additional “interested parties” currently listed.

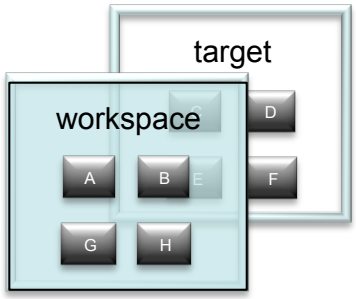
Drilldown - How b3 is intended to work

b3 on one slide

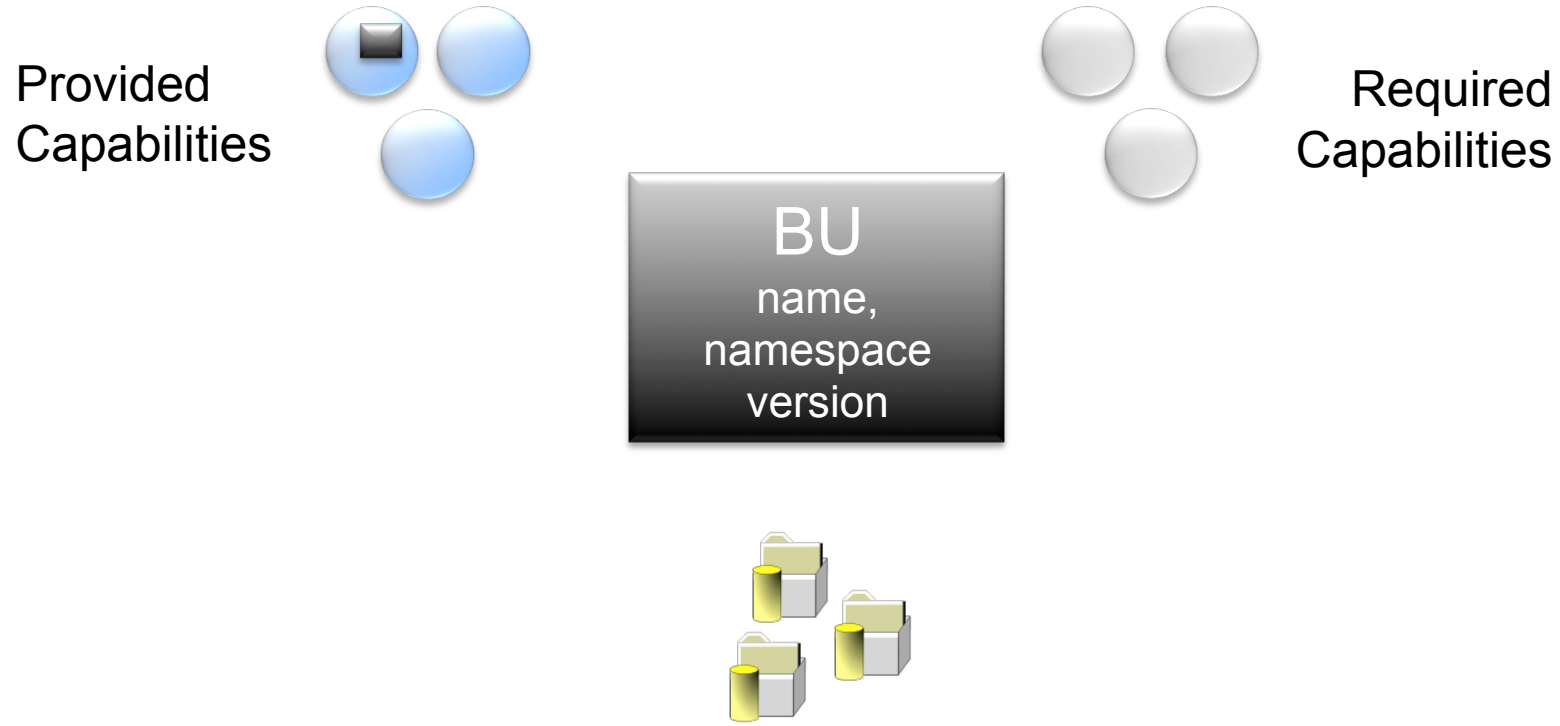


import
(materializers)

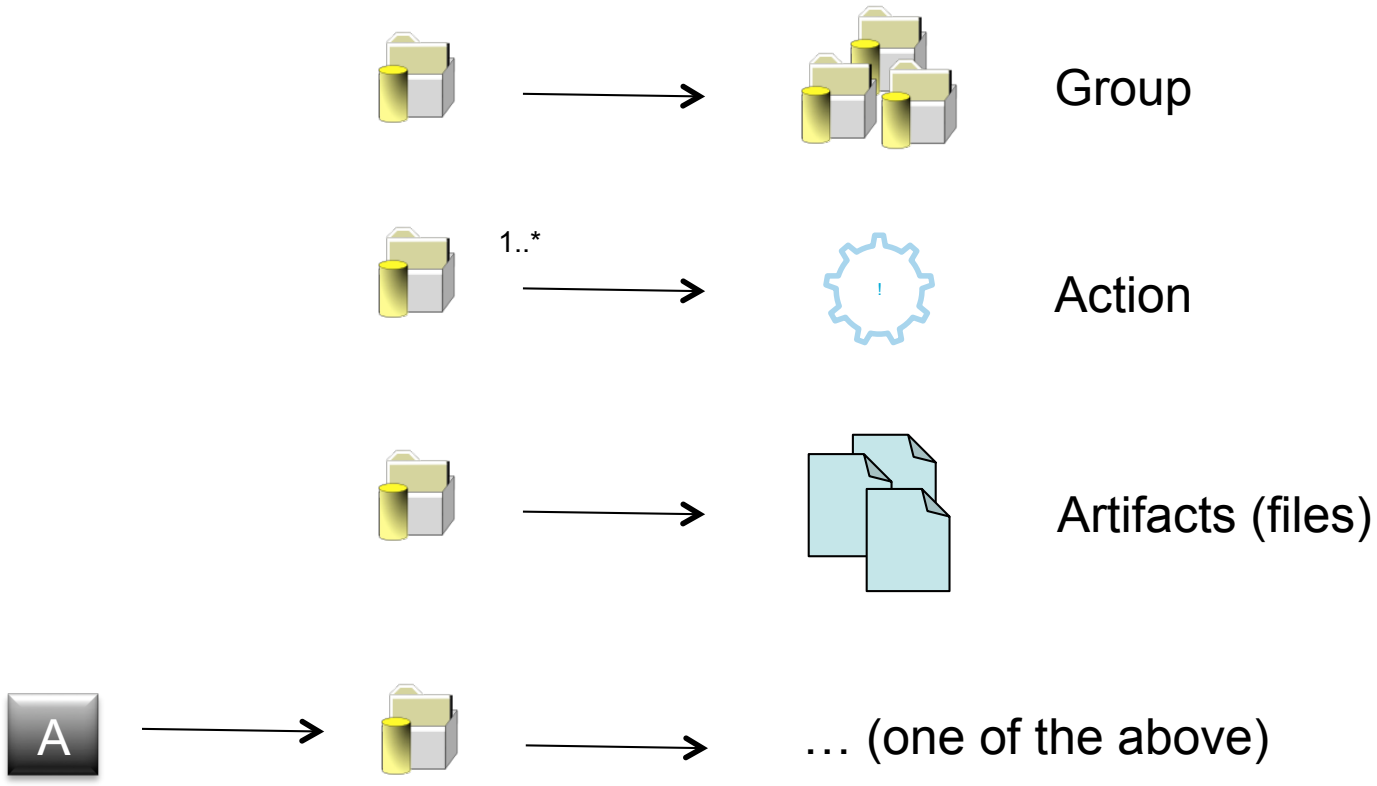
resolvers
meta data translators
connectors



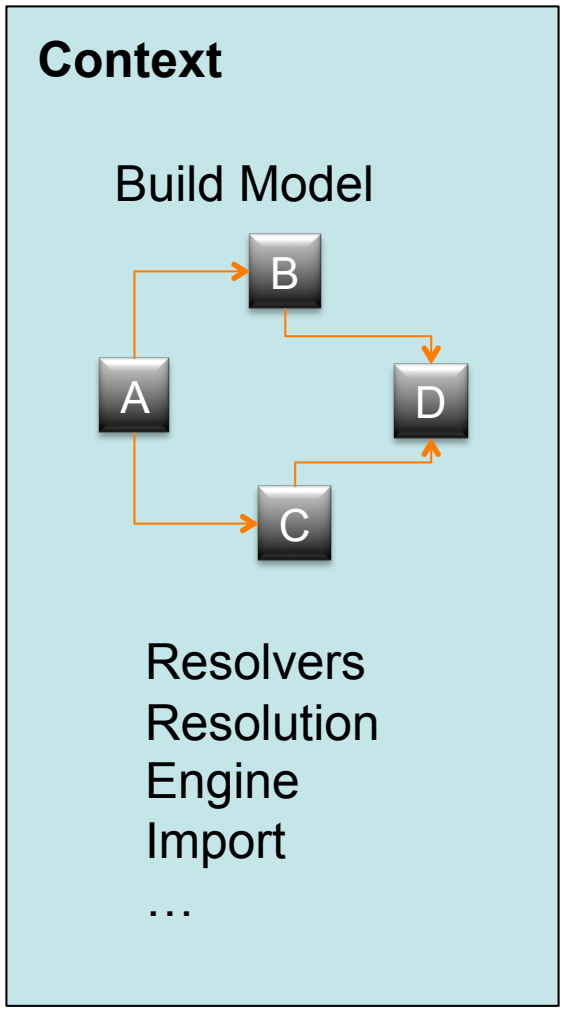
Anatomy of an Build Unit

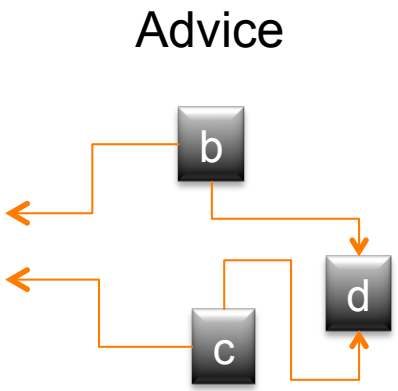


Build Parts

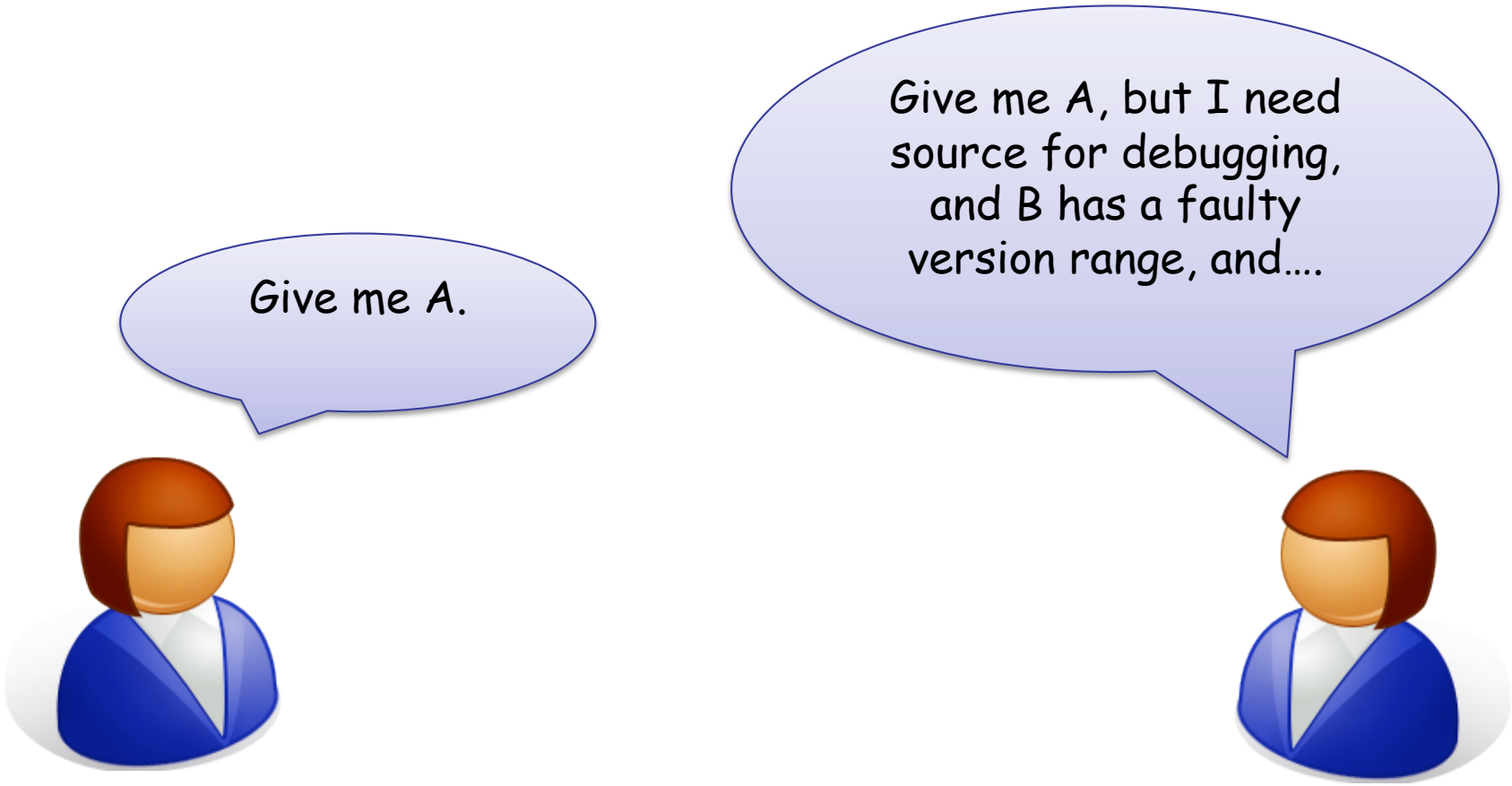


Advice





Make it as simple as possible…



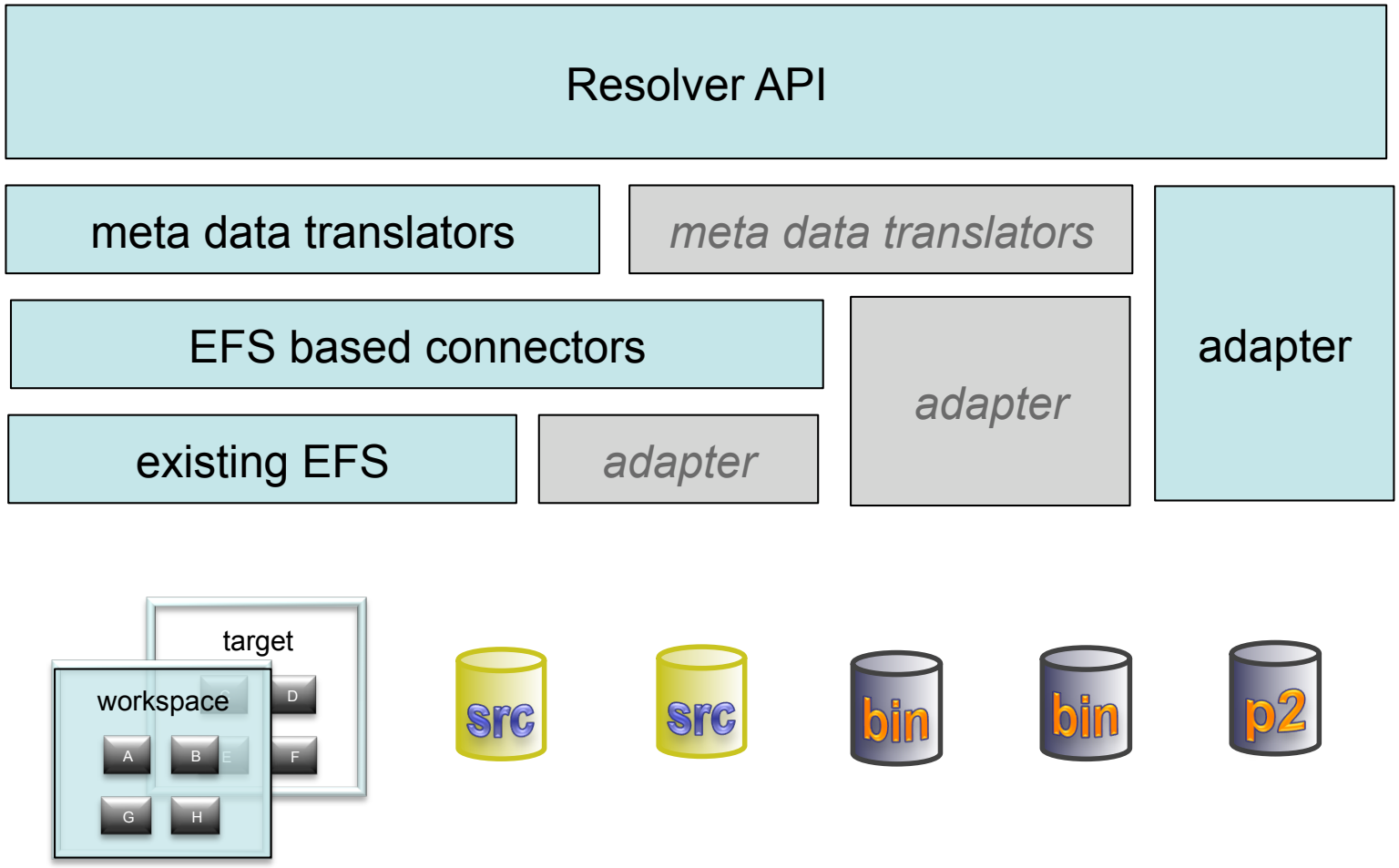
“build file”

- A b3 build file contains information how a context should be set up and how the model should be advised.
  - Other implementations are possible, b3 can be driven programmatically, other syntax is possible etc.
- The b3 build file is XText based and is a compact and powerful way to specify the details of a build.
  - In comparison - Buckminster has CSPEX, CQUERY, RMAP, and MSPEC XML artifacts to cover the same area and is much more verbose (due to XML), less powerful, and is not based on a single concept (i.e. advice).

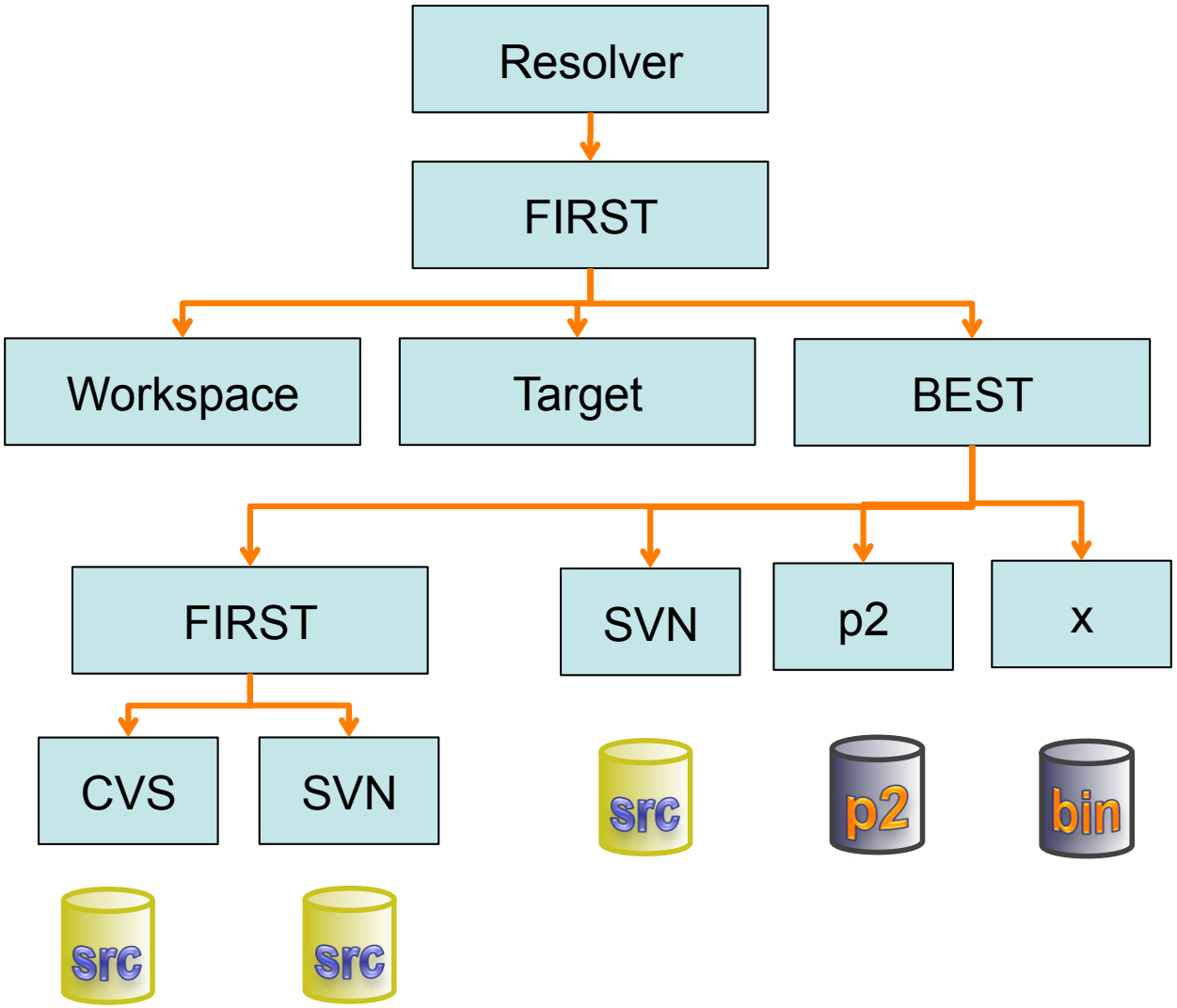
Build File Example

```
unit myBuildUnit {
repositories{
    platform:/plugin:/;
    platform:/resource:/;
    p2:http://www.someplace.good/updates-3.5;
    svn:svn+ssh://org.myorg.repo/productx;
}
builder buildAll {
    input {
        eclipse.feature/org.eclipse.myproj.site/1.0#site.p2;
        eclipse.feature/org.eclipse.myproj.packaging;
        }
    output { $outputDir }
}
required { /* required capabilities */ }
provided { /* provided capabilities */ }
//. . .
}
```

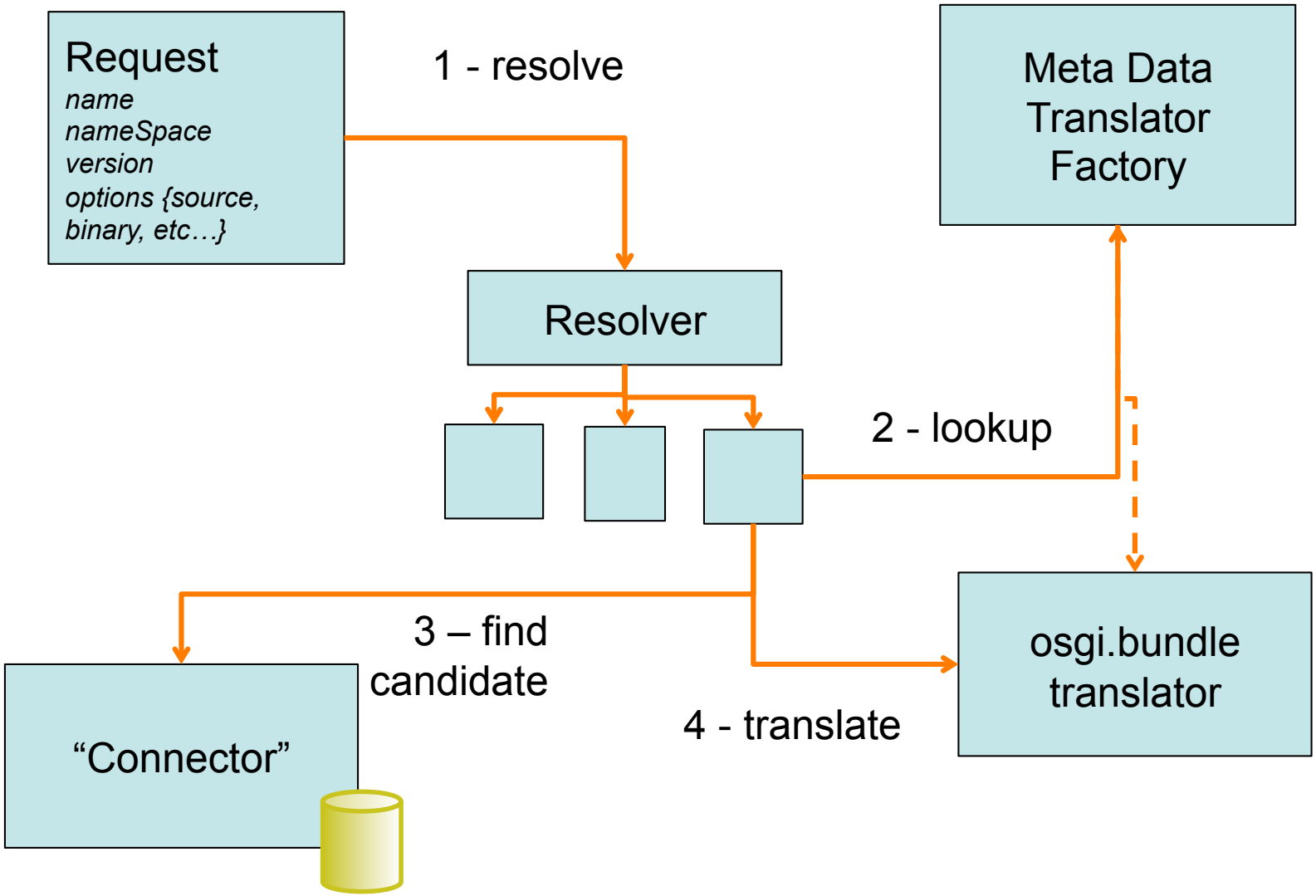
Easy to extend - Resolvers and Meta Data Translators



Resolver network / map - example



Resolver network / map - example



Implementation

- Implementation based on EMF
  - The BuildUnit and Context/Advice are Ecore model instances
  - By default the model is “executed”…
  - Other implementations possible – generate scripts…
- Based on interfaces rather than document formats
  - Easy to create different type of drivers and interactions
  - From different types of files, headless commands, etc.
  - Easy to drive builds programmatically without the need to create external documents.
- Reference implementation of an Xtext based advice/ invocation document format
  - For human interaction
  - Compact and easy to edit with just a plain text editor

Ease of Use

- Automatic translation into build model
- Most common case by default
- Example – producing a p2 update site from a feature should require no additional authoring
- When things must be specified – they should be specified as simple as possible. Builds are often executed on remote servers, and tooling must work well with just the command line and a simple text editor
- Debugging should be available from the command line as well – even if b3 runs the same build locally as on servers, differences in the environment may lead to non reproducible failures.

Build Debugger and Asserts

- It is important to be able to set breakpoints, and step through a build, inspect input and output files, see property values, etc.
- The b3 engine will provide hooks for such a debugger
- Inspection is made easy as the runtime state is maintained in the context which is implemented using Ecore.

- Asserts are important to put guards in place that warn/fail the build at the point where a problem is detected. (Useful for checking faulty output – zero length files etc. that subsequent steps can not determine if they are correct or not).

VISIT US AT

(http://www.eclipse.org/b3 – TBD)

http://wiki.eclipse.org/b3

nntp:news://news.eclipse.org/eclipse.b3

irc: irc://irc.freenode.net/#eclipse-b3