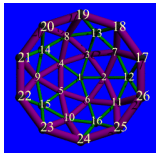


Buckminster | Component Assembly Project

Kenneth Ölwing
Thomas Hallgren
Henrik Lindberg
Mitch Sonies

A subproject of the Eclipse Technology Project

Goals

- **To unify the way components are described in terms of versions and dependencies**
- **To unify the way components are obtained from a variety of repositories**
- **To become a state-of-the-art component assembly tool, providing elegant and extendable solutions to very complex problems**

So what's the problem in simple terms?

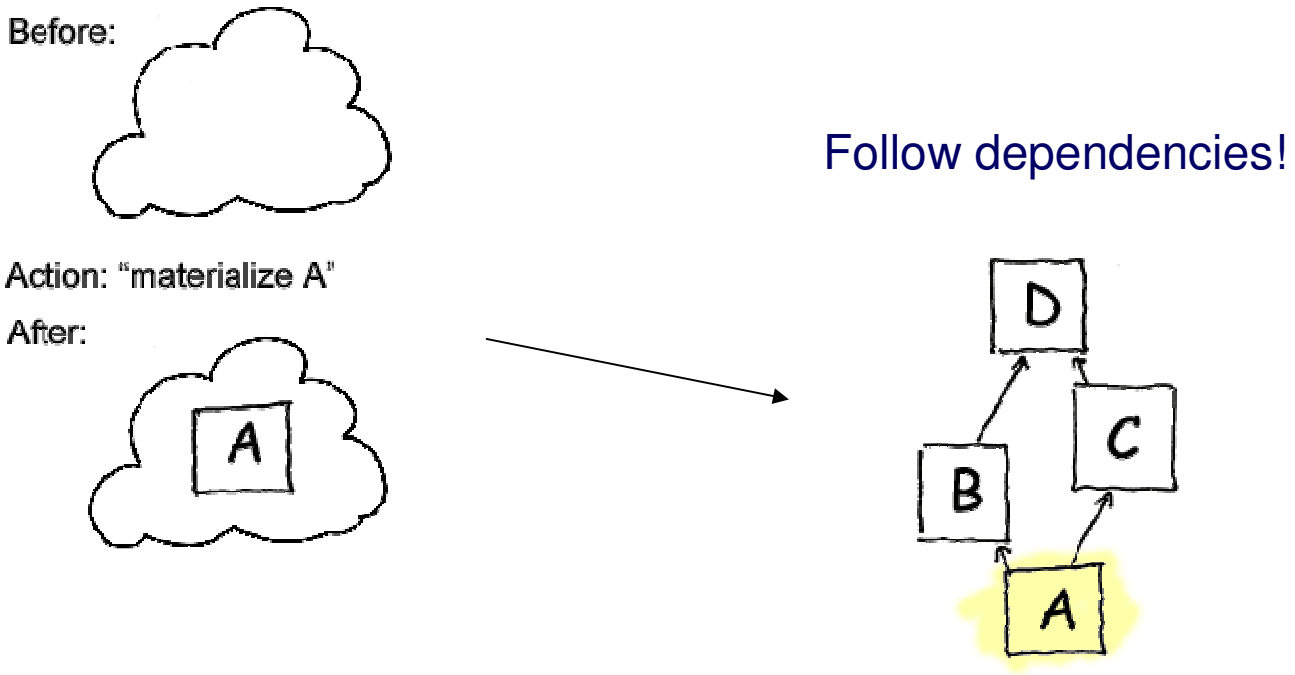
You've all been exposed to this…

- Your colleague say "ok, project Overland is ready for you to work on - just check out com.megacorp.overland from CVS and get started.
- You checkout com.megacorp.overland and discover that it has unresolved project dependencies.
- "Oh yeah," he says, "you have to checkout project overwater as well". Of course that doesn't completely solve the problem...
- "Also you have to get the latest undersea.jar from the central FTP server."
- Eventually, after a few rounds of this, you have the whole set of resources and you are ready to go to work.

Actually, there's a lot more to it – move away from monoliths, improve build times, scale up development etc etc.

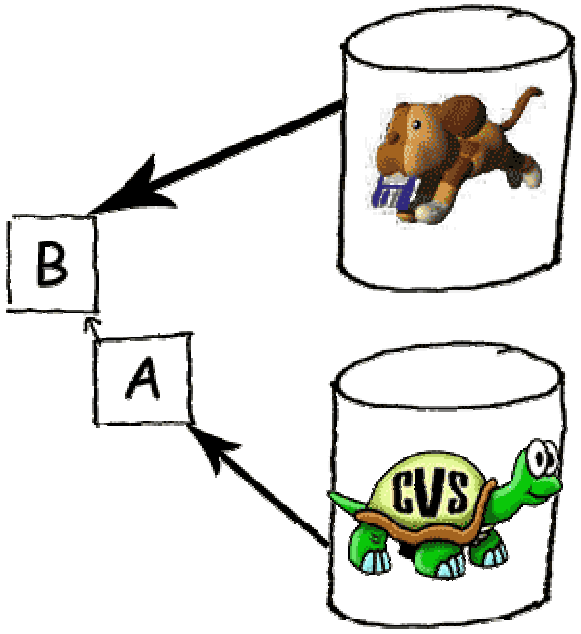
Ok. So then what?

Help the user with ’materialization’ of components

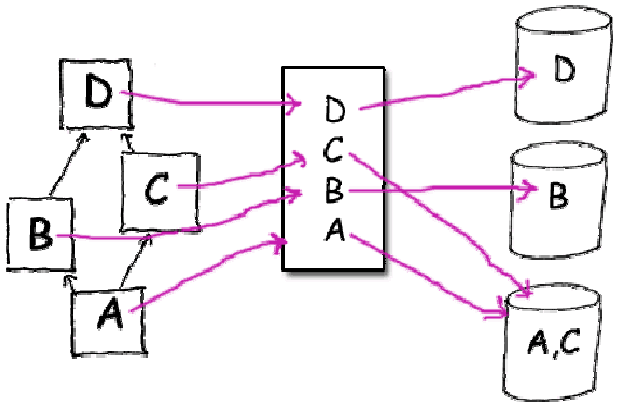


Whoa! Where does it get all those things?

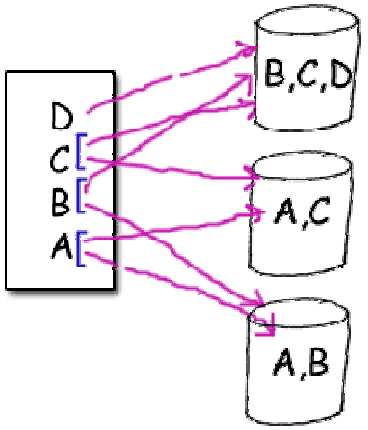
From ’repositories’ – CVS is one kind, but there are many possibilities (including beyond regular version control systems)



Hmm, how does it find all these repositories?

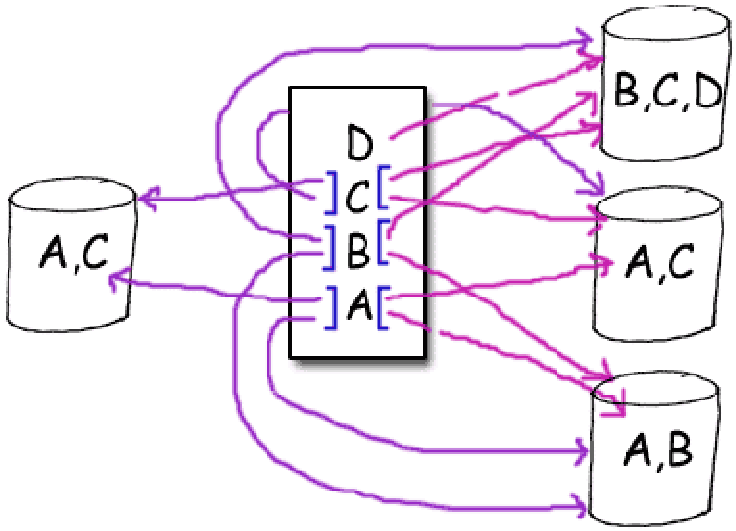


It uses a 'resource map' (rmap) to find them indirectly



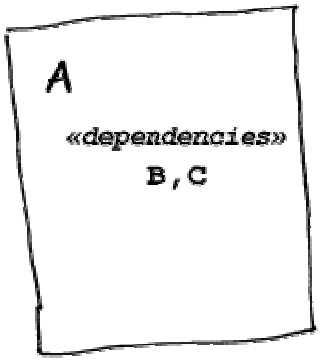
It can easily map families to one location, as well as handle categories...

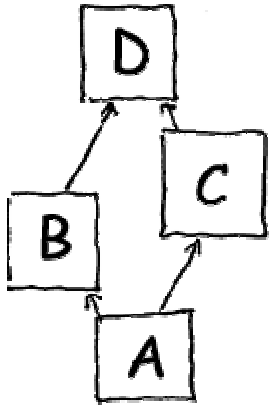
...as well as handle caching and distributed repositories



What are ’components’ then? Overloaded term, yeah?

A component is anything from a single file, to a collection of files (in a tree). Typically a component is not alone – it depends on other components.

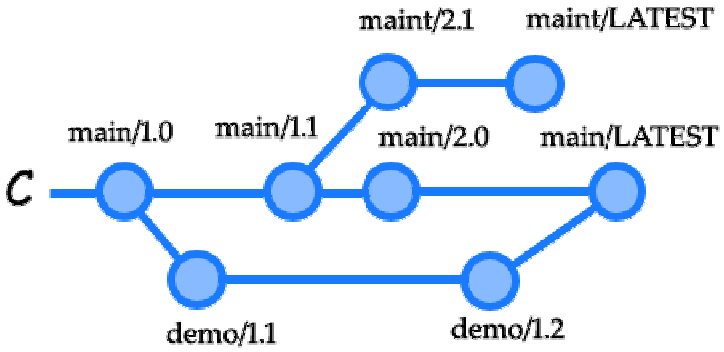


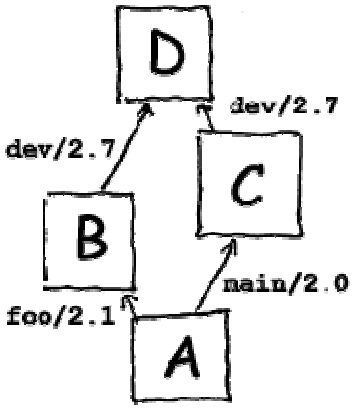


But the world isn’t that simple – we must also care about versions...

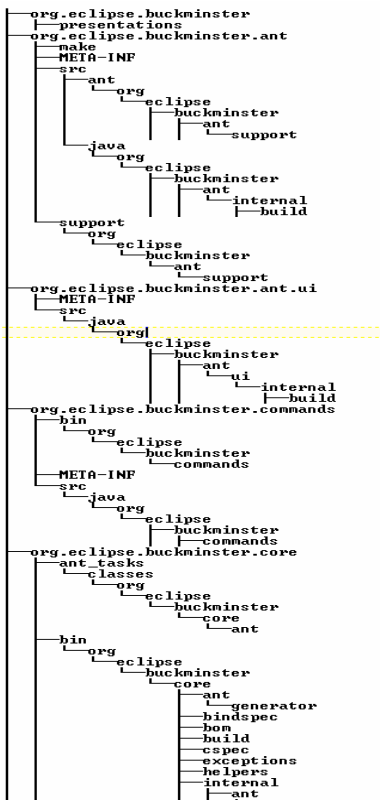
Umm, versions might complicate things...

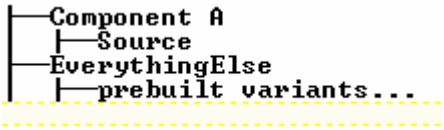
Most certainly. Buckminster will help you resolve all this by rich declarative means so that you can combine development time flexibility (goofing around with configurations are important) with QA time rigidity (strictness and reproducibility are key concepts!)





There's more to the resolvement story...if you want to work on component A, do you really wish to clutter your workspace with all it's dependencies in source form?





Buckminster can help you easily shift to one or the other (and the components none the wiser)

- Browse source ⇒ no dependencies
- Build ⇒ interface dependencies
- Deploy ⇒ binary dependencies

Getting Started:
When Someone Else Did The Hard Work

- Install Buckminster
  http://download.eclipse.org/technology/buckminster/updates
- Run the ***cquery*** that is handed to you (usually as a URL)
- You're all set

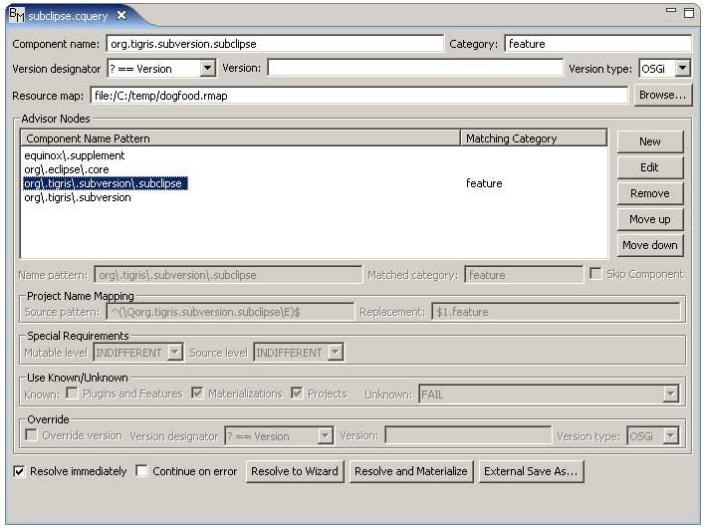
Getting Started With Your Own Projects

cspec: for most projects, it's automatic

- Plugins, Features, Fragments, Update sites, Maven, Ivy, JDT
- For other types, create a cspec XML file
  (we suggest using the XML editor from WTP)

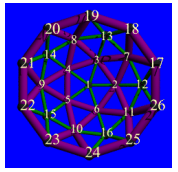
rmap: reuse an existing one or write one (another XML file with a well-defined XML schema).

cquery: create a ***cquery*** using the cquery editor and place it at an accessible location for others to use



Thank You

http://www.eclipse.org/buckminster



Buckminster compared

What differentiates Buckminster from other tools in its domain?

Several tools have functionality that overlaps with Buckminster.
Maven, Ivy, the Eclipse update manager, and the Eclipse Team Project Sets to name a few

The following slides will highlight some key points where Buckminster stands out. Some keypoints may be implemented in whole or partially by the other tools

Buckminster Compared, keypoints

- Buckminster does not stipulate the presence of specific component descriptor files
- Component Types, capable of recognizing the Eclipse Plugin Environment (PDE) components (update-sites, features, plugins, and fragments) and Maven projects are bundled with Buckminster
- Buckminster does not stipulate a specific versioning scheme
- Buckminster use a two phase resovelment scheme. The first phase results in a Bill Of Materials that uniquely identifies the components that are to be materialized

Buckminster Compared, keypoints continued...

- Buckminster does not stipulate how a component repository is constituted
- The Buckminster rmap allows different search paths for different distributed development sites
- Buckminster is language agnostic
- When Buckminster performs its dependency resolvement, it will make use of several threads that executes in parallel. This significantly shortens the time spent
- Buckminster is implemented using the Eclipse PDE. As a consequence, it reuses a lot of existing Eclipse functionality

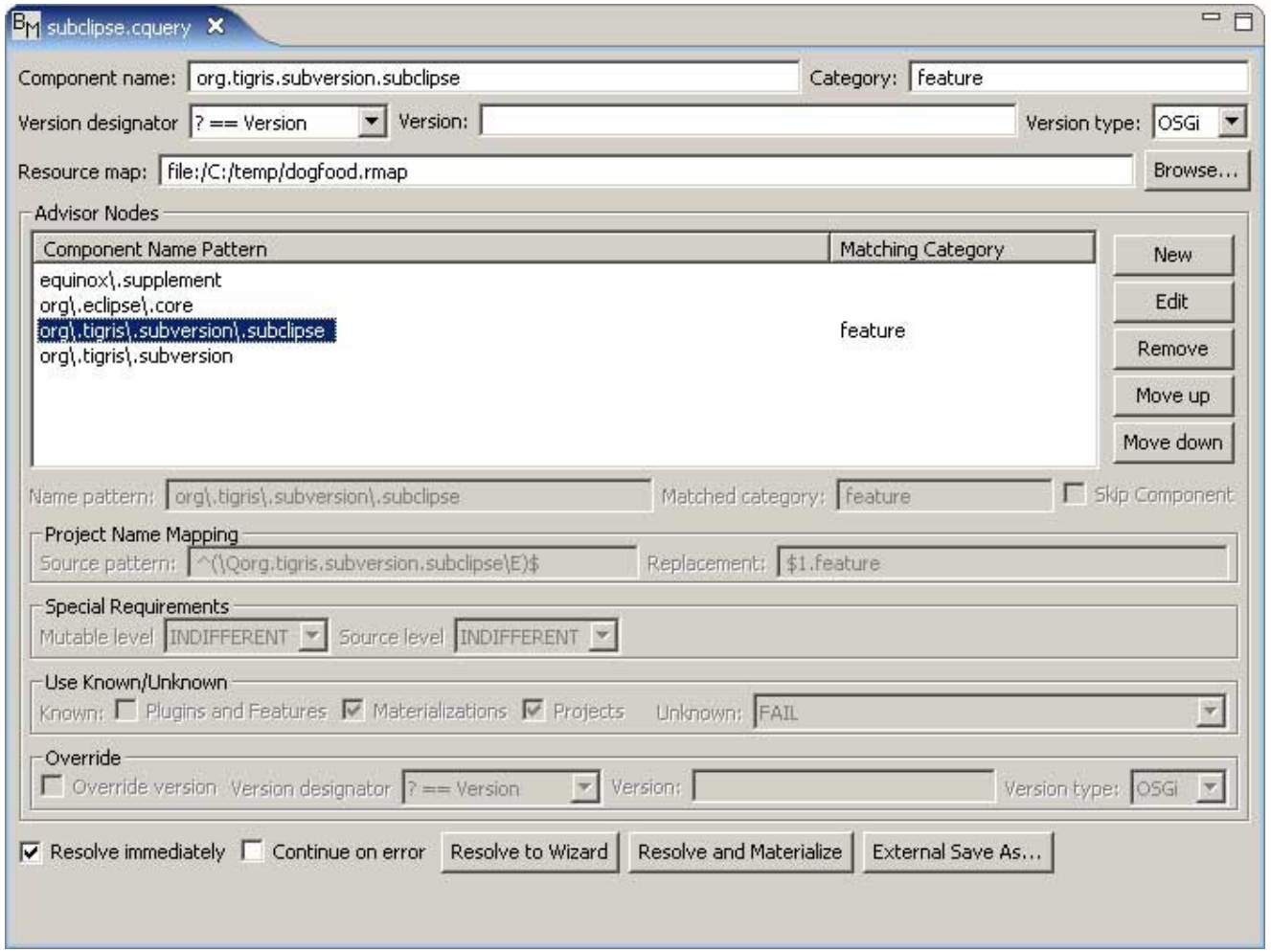
Artifacts used by Buckminster

- Component Specification (***cspec***): defines a component (only needed when the component type is unknown to Buckminster)
- Resource Map (***rmap***): defines a mapping from names to repository providers and locations
- Component Query (***cquery***): defines
  - which component to materialize
  - which *rmap* to use
  - rules to apply during transitive resolve and materialization

Installation

- Buckminster is installed using the Eclipse Update Manager
- The Buckminster update site is located at: http://download.eclipse.org/technology/buckminster/updates
- Aside from repository providers that are included in the core feature (such as CVS, Maven, and URL) you will also find providers for Perforce and Subversion.
- The headless feature is important if you plan to do unattended builds

The Component Query editor



Using features and plugins

Buckminster will immediately recognize both feature and plugin projects and generate the *cspec* for them on the fly.

Generated *cspecs* contain version information derived from manifest files and are used during transitive resolvement

Buckminster will honor OSGi version ranges during the resolve phase and make intelligent decisions to deliver the best version

Buckminster can download features and plugins from a variety of sources, including the target platform and remote update sites

Components that lack dependency information

Buckminster will recognize and generate *cspecs* for several component types in addition to plugins and features. New types can be added using Eclipse extension points

As a last resort, a *cspec* can be created manually and stored in a component

Buckminster will look for a cspec file when it encounters an Eclipse project with a specific buckminster nature