The Sisyphus Build System

Tijs van der Storm, Deliver meeting 19-1-2006

Overview

- Design overview
- Configuration model
- Software Knowledge Base
- Web frontend
- Observations, Summary, Future Work

Introduction

- “Builders” run concurrently on different machines
- Configuration of Sisyphus is versioned
- Build results stored in a database
- Web front end to view results

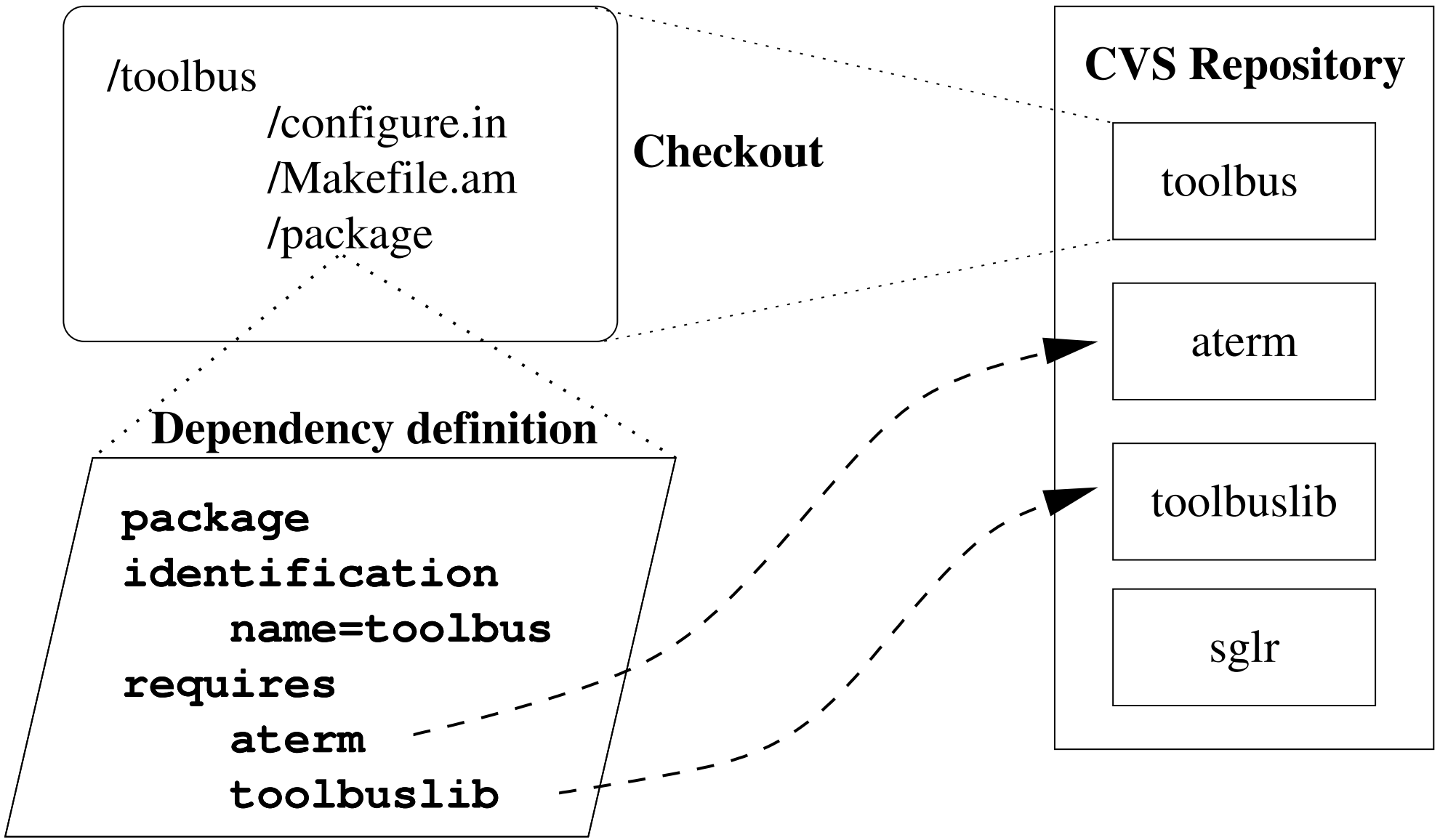
Decentralized Approach

- First attempt, push-approach:
  - central daemon distributes work
  - problem: threads, races, locks, fragile
- Now, pull-approach
  - builders check for work at fixed intervals
  - concurrency issues resolved through DB

What Builders Do

- check out all components
  - infer version
  - infer dependencies
  - if any has changed wrt the database
    - build it
    - build every co-dependency

Dependency Extraction



Sisyphus Configuration

- Source configuration
  - repositories + component locations
- Build script configuration
- Host specific profile
  - build and install directories
  - environment variables during build
- Stored in YAML file format (no XML;-)

Repositories

```
repositories:
  -
    type: cvs
    protocol: ssh
    location: cvs.cwi.nl/cvs
    components:
      - asc-support
      - asf
      - asf-library
      ...
```

Rationale

- Requirements:
  - Multiple types of VCS
  - Multiple locations
  - Migration of components

Build Actions

```
order: [configure,make,install]
templates:
 configure: ./configure
 --prefix=<%=prefix%>
 <%deps.each do |d|%>
 --with-<%=d.name%>=<%=d.prefix%>
 <%end%>
 make: make
 install: make install
```

Rationale

- Builds steps are identified separately
  - Failure is more localized
- Templates are restricted
  - Well formedness checking
- Independent of specific tooling

Builder Profile

```
user: daybuild
build_dir: /ufs/daybuild/build
install_dir:/ufs/daybuild/install
environment: |
  PATH=/usr/bin:/bin
  export PATH
world_version: 6
```

Rationale

- Environment enables build isolation
  - Traceability to compilers etc.
- World version to trigger builds
  - Cannot manage everything
  - Example: changing hardware

Versioned Configuration

- What if configuration parameters change?
  - Config files under Subversion
  - Versions of config files in bill of material
- Buiders checkout the most recent versions
- Allows:
  - Reproducibility (Go back in time)

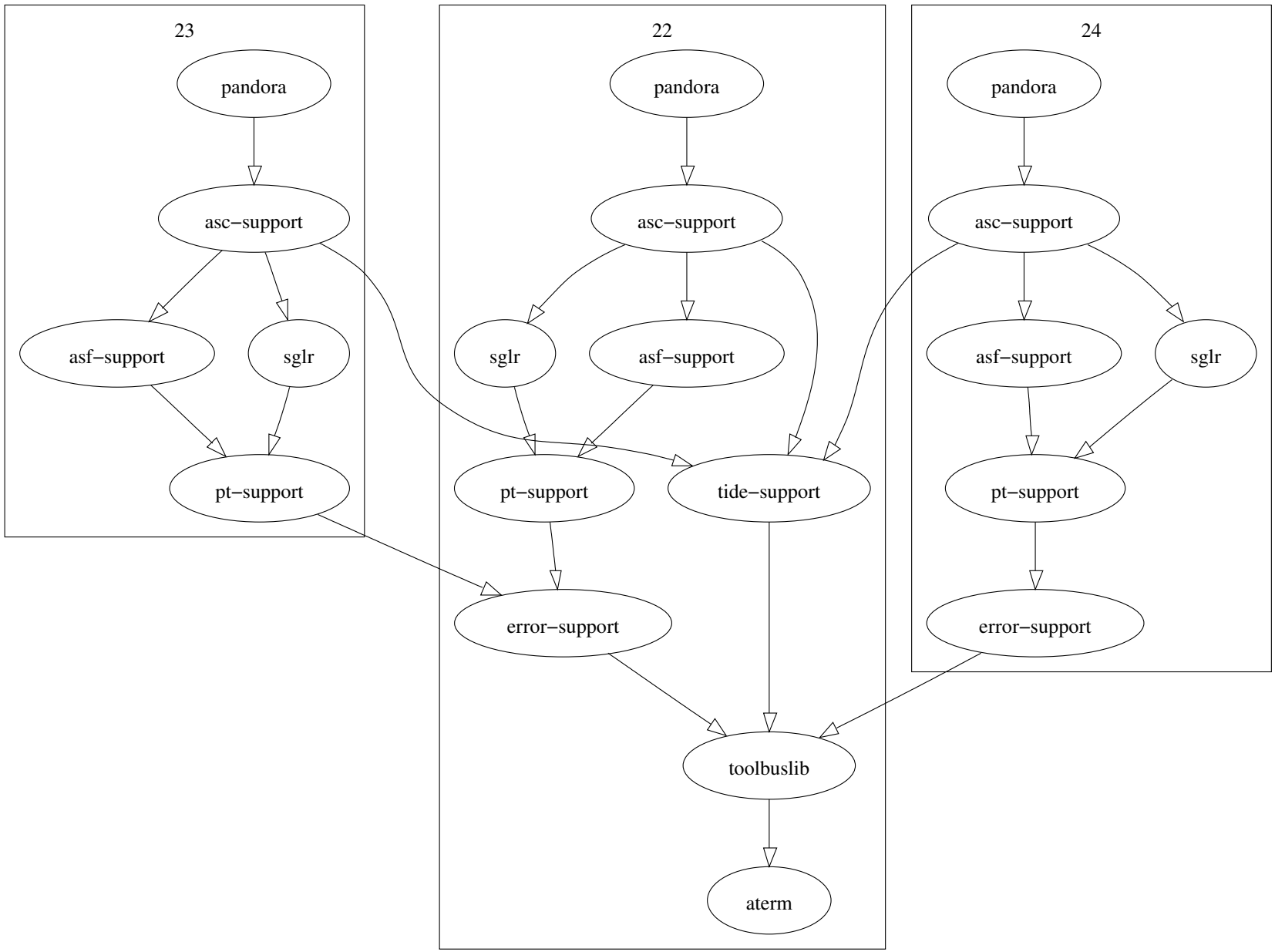
What goes into the SKB?

- Host (uname -a)
- Component revision (name + version)
- Versions of configuration files
- Full logs + success/failure of build actions
- Used component dependencies

Rationale

- Why not branch every successfully built component?
  - no support for dependencies
  - coupled to VCS (hard to do generically)
  - need write access on repositories
- Database achieves the same generically

Incremental BOMs



Some technical details

- Language: Ruby
- Object Relational Mapping: ActiveRecord
- Database: PostgreSQL
- Web: Ruby On Rails
- Builder size: 1600 LOC

Some Observations

- Relational formalization turned out not to be a good starting point for implementation
- Portability remains a problem (MSWindows)
- Compiler dependencies are not handled satisfactory (“alles of Nix?”)

Feature Summary

- Component-based
- Distributed builds
- Incremental builds
- Reproducible builds
- Portable implementation
- Accurate Bills of Materials

Conlusion/Future Work

- It works, finally...
- Future work:
  - support for building branches
  - backtracking to improve continuity
  - automatic releasing
  - patch/update generation

Thank you