Dependency Solving Is Still Hard

but We Are Getting Better at It

Stefano Zacchiroli
zack@irif.fr
@zacchiro
joint work with Pietro Abate, Roberto Di Cosmo, and Georgios Gousios

Université de Paris and Inria, France

21 February 2019
SANER 2020
27th Intl. Conf. on Software Analysis, Evolution and Reengineering
London, ON, Canada

Reflection on dependency solving (2012)

Pietro Abate, Roberto Di Cosmo, Ralf Treinen, and Stefano Zacchiroli
Dependency solving: a separate concern in component evolution management.
Journal of Systems and Software, 85(10):2228–2240, 2012.

Takeaways

- Dependency solving is harder than you think
- Dependency solving should be expressive and complete
- Dependency solvers should be reusable components shared across package managers

Has this vision been adopted since?

Dependency solving is hard

Definition (Dependency solving — simplified)

Input: a set of installed packages (*package status*), a *universe* of available packages, and an *user request* to alter current status.
Output: *upgrade plan* that produces a new package status which satisfies user request (or an error).

Dependency solving is NP complete

- Debian case—AND/OR + conflicts (Mancinelli et al. ASE 2006)
- all non-trivial cases—multiple package versions that cannot be co-installed (Abate et al. JSS 2012)

Dependency solving is an optimization problem

- not all solutions are equal, you want the “best” one
- AKA *minimal install problem* (Tucker et al. ICSE 2007)

Dependency solving is hard

Definition (Dependency solving — simplified)

Input: a set of installed packages (*package status*), a *universe* of available packages, and an *user request* to alter current status.
Output: *upgrade plan* that produces a new package status which satisfies user request (or an error).

Dependency solving is NP complete

- Debian case—AND/OR + conflicts (Mancinelli et al. ASE 2006)
- all non-trivial cases—multiple package versions that cannot be co-installed (Abate et al. JSS 2012)

Dependency solving is an optimization problem

- not all solutions are equal, you want the “best” one
- AKA *minimal install problem* (Tucker et al. ICSE 2007)

Dependency solving is hard

Definition (Dependency solving — simplified)

Input: a set of installed packages (*package status*), a *universe* of available packages, and an *user request* to alter current status.
Output: *upgrade plan* that produces a new package status which satisfies user request (or an error).

Dependency solving is NP complete

- Debian case—AND/OR + conflicts (Mancinelli et al. ASE 2006)
- all non-trivial cases—multiple package versions that cannot be co-installed (Abate et al. JSS 2012)

Dependency solving is an optimization problem

- not all solutions are equal, you want the "best" one
- AKA *minimal install problem* (Tucker et al. ICSE 2007)

Dependency solving desiderata

Correctness: the returned solution satisfies both the user request and all inter-package relationships (e.g., dependencies and conflicts)

Completeness: a solution is returned every time at least one (correct) solution exists

Expressivity: it should be possible:

a. for package maintainers to express fine-grained inter-package relationships; and
b. for users to express global optimization criteria (e.g., *"minimize the number of installed packages coming from the development branch"*)

Dependency solving desiderata

Correctness: the returned solution satisfies both the user request and all inter-package relationships (e.g., dependencies and conflicts)

Completeness: a solution is returned every time at least one (correct) solution exists

Expressivity: it should be possible:

- a for package maintainers to express fine-grained inter-package relationships; and
- b for users to express global optimization criteria (e.g., *“minimize the number of installed packages coming from the development branch”*)

Dependency solving desiderata

Correctness: the returned solution satisfies both the user request and all inter-package relationships (e.g., dependencies and conflicts)

Completeness: a solution is returned every time at least one (correct) solution exists

Expressivity: it should be possible:

a. for package maintainers to express fine-grained inter-package relationships; and
b. for users to express global optimization criteria (e.g., *“minimize the number of installed packages coming from the development branch”*)

Reusable dependency solvers

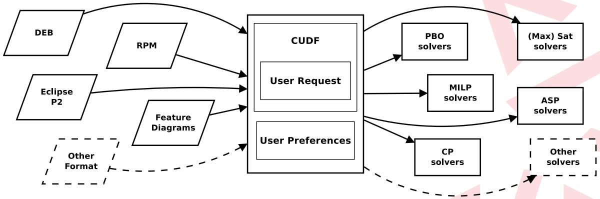


Figure: a formal model and exchange format for dep. solving scenarios

Reusable dependency solvers

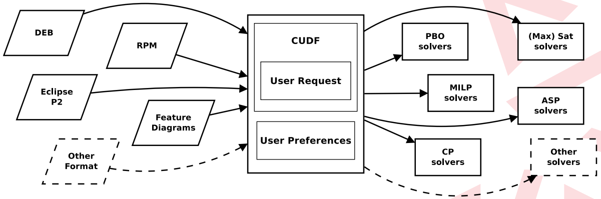
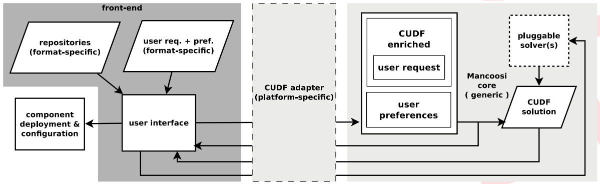


Figure: a formal model and exchange format for dep. solving scenarios



front-end
repositories (format-specific)
user req. + pref. (format-specific)
component deployment & configuration
user interface
CUDF adapter (platform-specific)
CUDF enriched
user request
user preferences
Mancoosi core (generic)
pluggable solver(s)
CUDF solution

Figure: a modular package manager architecture

Package manager	Version scheme	Solver	Distribution granularity		Version locking	Qualif.	Dependency range operators				Range modifiers		Resolution process			Approximate solutions	
							gt/lt	and	or	not	flex patch	flex minor	correctness	completeness	user prefs	missing deps	conflict
Go (dep)	git tags	ad hoc	github	branch	yes	no	no	no	no	no	no	no	yes	yes	no	error	error
npm	semver	ad hoc	archive	package	yes	no	yes	yes	yes	no	yes	yes	?		no	error	keep both
Packagist	git tags	ad hoc	github	branch	yes	no	yes	yes	yes	no	yes	no	yes	?	?	?	error
opam	debian	CUDF (any)	git	package	work-around	yes	yes	yes	yes	yes	no	no	yes	yes	yes	error	error
PyPI / pip	pep-440	ad hoc	archive	package	yes	conda	yes	yes	no	yes	yes	yes	yes	yes	no	error	error
Nuget	semver	ad hoc	archive	package	yes	no	yes	yes	no	no	no	no	yes	yes	no	error	nearest wins
Paket	semver	ad hoc	archive, github	package, branch	yes	no	yes	yes	no	no	yes	no	yes	yes	no	error	error
Maven	semver	ad hoc	archive	package	no	yes	yes	yes	yes	yes	no	no	yes	yes	with plug-ins	latest	nearest wins
RubyGems	semver	ad hoc	archive	package	yes	bundler	yes	yes	no	no	yes	no	?	?	?	error	error
Cargo	semver	ad hoc	archive, git	package, branch	no	yes	yes	yes	no	no	yes	yes	yes	yes	no	latest	name man-gling
CPAN	strings	ad hoc	archive	package	no	yes	yes	yes	yes	yes	no	no	no	no	no	error	error
Bower	semver	ad hoc	git	package	?	?	yes	yes	yes	no	yes	yes	yes	yes	no	error	use res-olu-tions
Clojars	semver	ad hoc	archive	package	?	?	yes	yes	yes	yes	no	no	yes	yes	error	error	error
CRAN	debian	ad hoc	archive, git	package	?	yes	yes	yes	yes	yes	no	no	no	no	no	error	error
Hackage / cabal	semver	?	archive	package	?	no	yes	yes	yes	yes	yes	no	?	no	no	error	error
Debian (apt)	debian	CUDF (any)	package	package	pinning	yes	yes	yes	yes	yes	no	no	yes	yes	yes	error	error
RedHat (dnf)	dnf	libzypp	archive	package	?	yes	yes	yes	yes	yes	yes	yes	yes	yes	?	error	error
Eclipse P2	semver	sat4j	archive	package	?	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	error	error

A dependency solving census — discussion

+ increased availability of expressive inter-package dependencies
- almost no support for flexible user preferences
+ several package managers now rely on complete solvers based on state-of-the-art constraint resolution techniques
  - e.g., Eclipse, SUSE, RedHat, Opam, Debian (opt-in)
- very scarce adoption of the separation of concern approach
  - Opam, Debian (opt-in)

What were the blockers?

- CUDF limitations
  - e.g., supporting P2 qualifiers or NPM intervals takes some work
- reluctance to have external dependencies in a core component of your ecosystem
- "if you build it, they will come" is not enough: adoption strongly correlated with direct involvement from researchers

A dependency solving census — discussion

+ increased availability of expressive inter-package dependencies
– almost no support for flexible user preferences
+ several package managers now rely on complete solvers based on state-of-the-art constraint resolution techniques
    - e.g., Eclipse, SUSE, RedHat, Opam, Debian (opt-in)
– very scarce adoption of the separation of concern approach
    - Opam, Debian (opt-in)

What were the blockers?

- CUDF limitations
    - e.g., supporting P2 qualifiers or NPM intervals takes some work
- reluctance to have external dependencies in a core component of your ecosystem
- “if you build it, they will come” is not enough: adoption strongly correlated with direct involvement from researchers

- we are now well into the 2nd wave of dependency solving
  - first distro-, now language-specific package managers
- old mistakes were remade
  - e.g., *who needs conflicts?*, *who needs a dependency solver?*, *how hard could it be to implement a dependency solver?*, ...
- things are getting better, with increased adoption of solid constraint solving technology (mostly SAT)
- reusable dependency solvers will likely remain a dream

Pietro Abate, Roberto Di Cosmo, Georgios Gousios, Stefano Zacchiroli
Dependency Solving Is Still Hard, but We Are Getting Better at It
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering. IEEE

preprint: `http://bit.ly/saner20-deps`

Stefano Zacchiroli
zack@irif.fr
@zacchiro