Polynomial Dependencies

Eliminating NP-Complete Problems in Design of Modular Systems

Jaroslav Tulach
NetBeans Platform Architect

NetBeans

Agenda

- NP-Completeness
- Semantic Versioning
- Range Dependencies
- Jigsaw like Services
  > Little language change
- Future plan

NP-Completeness

- Polynomial time with non-deterministic computer
  - > Hard to find solution with current hardware
  - > Polynomial to verify solution
- Conversion of problems
  - > Clique, Hamiltonian, Traveler, Knapsack
- 3SAT

$$E = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4)$$

- Randomized algorithms

http://en.wikipedia.org/wiki/NP-complete

Semantic Versioning NP-Complete

- Dotted versions
  - 1.2, 1.2.3, 1.4
- Need for incompatibility versioning
  - 2.0, 2.3
- NP-Complete
  - For each variable x generate $M_{1.0}$ and $M_{2.0}$
  - For each formula (x_1 || x_2 || x_3) generate $F_{1.1}$ $F_{1.2}$ $F_{1.3}$
  - Generate module T that depends on each F
- The more incompatibilities - the more NP

Polynomial Semantic Versioning

- Transitive Repository
  - When compiling T one knows
  - Can't compile against both $M_{1.0}$ and $M_{2.0}$ at once
- Each module carries all dependencies
  - Easy to turn on
- Assembling libraries together
  - During compilation
  - Verify two sets of dependencies compatible
- Simple polynomial checks

Range Dependencies NP-Complete

- Makes the “semantic” example more realistic
  - Narrow range simulates incompatibility
  - Defensive “QA” style
- NP-Complete 3SAT conversion
  - For each variable x generate $M_{1.0}$ and $M_{1.1}$
  - For each formula generate three version of F
  - Positive variable $F_{1.1} \rightarrow M_{[1.1,1.1]}$
  - Negative variable $F_{1.2} \rightarrow M_{[1.0,1.0]}$
- No incompatibilities, just ranges

Polynomial Range Dependencies

- Transitive Dependencies
  - > Same trick as before just on ranges
  - > During compilation record all dependencies
- Composition of two dependency sets
  - > Selects most narrow range for each dependency
  - > Reject when empty
- Gives power to end users
  - > Actual compatibility, not declared one
- No longer “source of all evil”

The Services Problem

- Reversed dependencies with “requires service”
  - Unknown at compile time
  - Can't pre-compute transitive dependencies
- Easy to show its NP-Complete again
  - Looked unsolvable
- Injectable singletons
  - Inherently initialized (have default implementation)
  - Injectable (testing & runtime)
  - “requires service optional” style

http://wiki.apidesign.org/wiki/Injectable_Singleton

Default Service Implementation

- “requires service optional”
  - Default provided in the same module
- “requires service”
  - Hint in which module to find default
  - jaxp@1.9 knows the default is com.oracle.xerces@1.9
  - Default included in transitive dependencies
- Providing a hint
  - Default by the module M itself
  - Can be changed by other modules depending on M
  - Re-configurable when launching the application

Language Change

- “requires service optional”
  - > Remains unchanged
- “requires service” -
  - > Always needs default:

```
module M1 {
  requires service S with default M2@3.8;
}
```

NetBeans

Conclusion

- Transitive dependencies with service defaults
  - > Powerful and easy to resolve
  - > Satisfies real world use-cases
- Compiler records transitive closures
  - > Incremental
  - > Validation of merged dependency sets
- Scientific proof
  - > Can “return back” to academic sphere for a while
- To NP or not to NP?

Q&A

NetBeans