Dependable software deployment

Wouter Swierstra
13 October 2006
Software configuration mismanagement
Imperative software deployment

- Initial Installation
- Current Configuration
- Component
- Update

Diagram flow:
- Initial Installation leads to Current Configuration.
- Component leads to Update.
- Update leads back to Component.
Imperative software deployment

\[ h^{-1} \cdot g \cdot f \cdot h =? g \cdot f \]

- No reflection
Functional deployment
Functional deployment

**COMPONENTS**

- word = component
  - name = “MS Word”
- acroread = component
  - name = “Adobe Acrobat Reader”
- ... AcroRd32.exe

**RULES**

**BIND**

**CONFIGURATIONS**
Functional deployment

Install Word version > 10
&& only install trusted components
Functional deployment

RULES → BIND → CONFIGURATION

COMPONENTS

A → B → C → D
Functional deployment

Configurations are:

- Immutable
- Reproducible
- Analyzable
The deployment problem

- Given a set of available components and a set of rules and requirements, construct “the best” configuration.
What is a configuration?

C = configuration

notepad = component

name = “Notepad”

version = 5.0.3

notepad.exe = executable
What is a configuration?

C = configuration
    winc = component
        name = “Windows Kernel”
        version = 5.0.3
        msvcrtd.dll = library
        signal = proc
What is a configuration?

C = configuration

winc = component

  name = “Windows Kernel”
  version = 5.0.3
  msvcrtdll = library

  signal = proc
          ordinal = 759
Imports

C = configuration
   readline = component
      readline.dll

... 

ghc = component
   name = “Glasgow Haskell Compiler”
   version = 6.4.2
   requires = readline • readline.dll
When do configurations make sense?

- **Well-formed:**
  - A configuration should only have components as children:
    
    ```
    forall c in children C . sort c = component
    ```

  - Components should not be nested.
    
    ```
    sort v = component =>
    forall c in allChildren v . sort c != component
    ```
When do configurations make sense?

- **Resolved:**
  - Every name can be found:

\[
\text{freeVar}(C) = \emptyset
\]
What’s on disk?

- Store
  - Readline-3.2
    - environment
  - GHC-6.4.2
    - environment
  - Word-11
    - environment
What’s on disk?

- Store
  - Readline-3.2
    - environment
  - GHC-6.4.2
    - environment
  - Word-11
    - environment

- User environment
  - Word
  - ghc
What’s on disk?

- Multiple versions of components
- Minimize interference
- Hidden from user
- Unambiguous bindings
## Memory model

<table>
<thead>
<tr>
<th>Programming Languages</th>
<th>Example</th>
<th>Software Deployment</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td></td>
<td>Disk</td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td>5, “Hello”, …</td>
<td>Components</td>
<td>libc, ghc, …</td>
</tr>
<tr>
<td>Addresses</td>
<td>0x005aa772</td>
<td>Path names</td>
<td>“/usr/local/”</td>
</tr>
<tr>
<td>Pointer arithmetic</td>
<td>*(arr + 5)</td>
<td>String manipulation</td>
<td>“C:\wouter\” + configDir</td>
</tr>
</tbody>
</table>
What are components?

- COMPONENTS
- RULES
- BIND
- CONFIGURATION
Towards deployment

• How should a developer know how to refer to the component called “readline” on your system?

• We need to parameterize components:

ghc rl = component
  name = “Glasgow Haskell Compiler”
  version = 6.4.2
  requires = rl • name = “readline”
Deployment

- Finding a component with the right name might not be enough...

```
ghc rl = component
    name = "Glasgow Haskell Compiler"
    version = 6.4.2
    requires = rl • name = "readline" &&
               rl • version > 3.0
```
Predicates - I

- We don’t want to fix our predicate language.

- First-order predicate logic.

- Versions:
  
  - readline • version > 3.0
  
  - libc • version <= 5.0
Predicates – II

- Disjunctive dependencies:

  readline cc = component
  requires = cc • name == “GNU C Compiler”
  or cc • name “Visual Studio”
Predicates - III

- Defining recursive configurations allows global constraints:

  nvidia config = component
  requires =
    forall c in children config .
    c • name == “Monitor driver”
  => c == nvidia

- A good predicate language is really, really important.
Guiding the binding

COMPONENTS

RULES

BIND

CONFIGURATION
Policies - I

- What if you have more than one choice?
- A policy is a partial order on components.
- State of the art:
  - \( c\text{.name} == d\text{.name} \Rightarrow c\text{.version} > d\text{.version} \)
Many websites publish lists that rate software.

Security, given a rating function:
- \( \text{rate}(c) \geq \text{rate}(d) \)
Policies - III

- Parsimony, given a size measure and installed predicate:
  
  - if installed(c) then 0 else size(c)
  
  <= if installed (d) then 0 else size(d)
Windows Installer

- Analyzed lots of msi files
- Declares complete component contents…
- …but deploy files in shared directories
- …and allow custom actions to affect where files are deployed.
- No real predicate language.
Red Hat Package Manager

- Packages specify name, version, dependencies,
- Fixed, simple predicate language.
- No two versions of same component.
- Scripts to build and deploy can execute arbitrary actions.
Conclusions

- A good idea of what the problem is.

- Still open questions:
  - Plug-ins
  - User settings
  - Generating faithful component descriptions
  - ...

- Draft paper available.