Model-driven Distributed Software Deployment

ing. S. van der Burg
Healthcare Systems Architecture/Delft University of Technology

September 16, 2008
Outline

• Introduction
• SDS2
• Nix
• Disnix
• Conclusions
• Future work
• Demonstration
Introduction

• Software deployment is all of the activities that make a software system available for use
  – Install/Uninstall
  – Update
  – Release
  – Activate/Deactivate
Introduction

• The software deployment process should be a simple problem, but it turns out to be much more complex
  – A software component is almost never self-contained
  – A software system should be able to find its dependencies
  – Some software components require specific hardware
  – Uninstalling should be safe
  – Keeping the system up to date should be safe and atomic
Introduction

• Distributed systems are systems that appear to a user as one logical system
  – Components are distributed across different systems in a network and work together to reach a common goal
• The software deployment process of distributed systems is even more difficult:
  – Dependencies of components on the same systems (intra-dependencies)
  – Dependencies of components running on different systems (inter-dependencies)
Introduction

• Information technology infrastructure of hospitals is heterogeneous
  – Workstations
  – Medical equipment (e.g. MRI scanners)
  – Connected through wired and wireless network
Introduction

• The software deployment process in hospital environments is a semi-automatic process
• Software deployment in such an environment requires
  – Up-to-date documentation
  – People with necessary skills
• A semi-automatic deployment process is
  – Error prone
  – Time consuming
  – Costly
Vision

• The software deployment process should be a simple process, not a complex one
• The software deployment process should be fully automatic, not semi-automatic
• The configuration of a system should be captured in a model
Distributed Environment
Service Development Support System (SDS2)

• Developed here at HSA
• Asset management and utilization services for mobile medical equipment
• Service Oriented Architecture
  – Information as a service
  – Services could be distributed across different machines
Components, Protocols & Technologies

- XMPP
  - XMPP Generator
  - XMPP Logger
  - XMPP Server
- SQL
  - Equipment logs
  - Equipment log summaries
- SOAP
  - Floor plan
  - Utilization service

Technologies:
- GWT
- smack
- mysql
- Java
- Axis2
- tomcat
- ejabberd
Deployment View

**Hospital**

- **XMPP Generator**
- **XMPP server**
- **XMPP Logger**
- **floor plan**

**Philips Enterprise**

- **Mobile Equipment**
- **Mobile Equipment**
- **TomCat server**
- **equipment logs**
- **equipment log summaries**

**Component types**

- client
- transformer
- repository

**Communication protocols**

- XMPP
- SOAP
- SQL
Nix

- Nix is a purely functional package manager
- Installing/upgrading/removing package automatically (like others RPM, dpkg etc.)
- Nix expression language, a domain specific language to describe how components are built
- All dependencies are explicit, this yields:
  - Multiple versions of a package to be installed side-by-side
  - Ensures that dependency specifications are complete
  - Supports atomic upgrades and rollbacks
  - “Niks” + UNIX = Nix
- Nix deploys packages on local systems but not on distributed systems though
Example Nix expression

```
{stdenv, fetchsvnssh, apacheAnt, jdk, mysql_jdbc, smack, config, username, password }:

stdenv.mkDerivation {
    name = "XMPPLogger";

    src = fetchsvnssh {
        url = "gforge.natlab.research.philips.com/svnroot/public/sds2/branches/NixDeploy/SDS2Applications/XMPPLogger";
        md5 = "62401d706079cb61d3b7e0badc818a1d";
        inherit username;
        inherit password;
    };

    builder = ./builder.sh;

    inherit jdk mysql_jdbc smack config;
    buildInputs = [apacheAnt jdk smack config];
}
```
Packages model

```scala
rec {
  username = "user";
  password = "secret";

  config = import ..../config {
    ...
  };

  smack = ...;
  stdenv = ...;
  apacheAnt = ...;

  XMPPLogger = import ..../SDS2Applications/XMPPLogger {
    inherit username password;
    inherit stdenv fetchsvnssh apacheAnt jdk;
    inherit smack mysql_jdbc;
    inherit config;
  };
  ...
}
```
Package Dependency Graph AssetTracker Service

328 Packages
1,374 Dependencies
Nix deployment operations

- Installing XMPP Logger:
  - `nix-env -f pkgs.nix -i XMPPLogger`

- Uninstalling XMPP Logger:
  - `nix-env -f pkgs.nix -e XMPPLogger`

- Upgrading XMPP Logger:
  - `nix-env -f pkgs.nix -u XMPPLogger`
Disnix

- An extension to Nix to allow distributed software deployment tasks
- A webservice interface which allows remote access to the Nix store and Nix profiles
- Introduces three model types to model a distributed system:
  - Services model
  - Infrastructure model
  - Distribution model
- Uses a variant of the two-phase commit algorithm to allow distributed atomic commits
Disnix overview
Services model

```javascript
ME2MSService = {
    name = "ME2MSService";
    pkg = pkgs.SDS2.webservices.ME2MSService;
    dependsOn = [ MELogService FloorPlanService ];
};

MULogService = {
    name = "MULogService";
    pkg = pkgs.SDS2.webservices.MULogService;
    dependsOn = [ ME2MSService ];
};

MELogService = {
    name = "MELogService";
    pkg = pkgs.SDS2.webservices.MELogService;
    dependsOn = [ ... ];
};
```
Infrastructure model

```

{  
    dtk15 = {  
        hostname = "dtk15";
        targetEPR = http://dtk15:8080/axis2/services/DisnixService;
        tomcatPort = 8080;
    };

    dt2d1 = {  
        hostname = "dt2d1";
        targetEPR = http://dt2d1:8080/axis2/services/DisnixService;
        tomcatPort = 8080;
        mysqlPort = 3306;
    };

}  

```
Distribution model

{services, infrastructure}:

[
  { service = services.FloorPlanService; target = infrastructure.dtk15; }  
  { service = services.ME2MSService; target = infrastructure.dtk15; }  
  { service = services.MELogService; target = infrastructure.dt2d1; }  
  { service = services.MULogService; target = infrastructure.dtk15; }  
...
]
Distributed atomic commits

• Commit-request phase:
  – Build all components on the coordinator machine
  – Transfer the closures to the cohort machines through the webservice interface

• Request phase, on each cohort:
  – Install component in profile
  – Activate the component
Conclusion

• We have adapted and modeled the SDS2 system in Nix expressions to make it automatically deployable
• We extended the Nix approach of software deployment of single system to distributed systems
• We have demonstrated that we can deploy SDS2 in a distributed environment
Future work

• Dynamic distribution based on Quality of Service models
  – Reliability by reducing the inter-dependencies
• Symmetric build and deliver parts
• Support for heterogeneous environments, example:
  – Linux coordinator
  – Windows/FreeBSD/Linux based network
• Support for other types of distributed systems with other protocols
• Investigate development constraints
SDS2 Deployment Demo

- Initial deployment state:
  - nbu5
  - disnix
  - VirtualBox/NixOS

- Deployment over two machines:
  - dtk15
  - VirtualBox/NixOS

- Garbage collect to initial state:
  - dt2d1
  - VirtualBox/NixOS

- Services and Software:
  - SDS2Utilisation
  - FloorplanService
  - ME2MSService
  - MULogService
  - EventMerger
  - DemoPortal
  - MySQL database
  - SDS2AssetTracker
  - EMLLogService
  - HIBService
  - MELogService
  - XMPPGenerator
  - XMPPLLogger
  - VirtualBox/NixOS