Model-driven Distributed Software Deployment

Sander van der Burg

Delft University of Technology, EEMCS, Department of Software Technology

Philips Research, Healthcare Systems Architecture, Eindhoven

March 17, 2009
Software deployment

Software deployment is all of the activities that make a software system available for use:

- Building software components
- Transfering components from producer site to consumer site
- Installing components
- Activating components
- Upgrading components

Sander van der Burg

Model-driven Distributed Software Deployment
Software deployment is all of the activities that make a software system available for use.
Software deployment is all of the activities that make a software system available for use

- Building software components
- Transferring components from producer site to consumer site
- Installing components
- Activating components
- Upgrading components
A software deployment process is complex:

- A software system is modular and requires modules to make it work (dependencies).
- A software component requires the right version or variant of a dependency, e.g., DLL-hell.
- A software system should be able to find its dependencies, e.g., setting CLASSPATH in Java.
- Some software components require specific hardware, e.g., Intel/PowerPC processor.
- Uninstalling should be safe. We only want to remove unused/obsolete components.
- Keeping the system up to date should be safe and atomic. There could be a time window where the system is in an inconsistent state.
A software deployment process is complex:

- A software system is modular and requires modules to make it work (dependencies)
  - A software component requires the right version or variant of a dependency, e.g. DLL-hell
  - A software system should be able to find its dependencies, e.g. setting CLASSPATH in Java
- Some software components require specific hardware, e.g. Intel/PowerPC processor
- Uninstalling should be safe. We only want to remove unused/obsolete components.
- Keeping the system up to date should be safe and atomic. There could be a time window where the system is in an inconsistent state.
Distributed systems

A system in which:

- Components are distributed across different systems in a network and work together to reach a common goal
- Appears to a user as one logical system
- Example: Web application using a webservice and a database
The software deployment process of a distributed system is *more challenging*

- Dependencies on components on the same systems: *intra-dependencies*
- Dependencies on components running on different systems: *inter-dependencies*
- Upgrading cannot be done *atomically*
- Is expensive and tedious, if done manually:
  - Requires up-to-date documentation and people with skills
  - Time-consuming, error prone
Distributed systems

Machine: itchy

Machine: scratchy

 intra-dependency

 inter-dependency
Vision on software deployment

- 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.
Vision on software deployment

Configuration model → Distribution function → Distributed system
We already have:

- Nix deployment system
- Case study: Service Development Support System (SDS2)

In this thesis project:

- Disnix deployment system
- Make SDS2 deployable with Disnix
Nix deployment system

- Is a package manager, like RPM

Unique features:
- Stores components in isolation in a Nix store
- Builds packages from Nix expressions
- Upgrading is atomic
Conventional package managers e.g. RPM have nominal dependency specifications:

- Example: Firefox
- Requires: gtk+ >= 2.12.8
Dependency specifications

Conventional package managers e.g. RPM have nominal dependency specifications:

- Example: Firefox
- Requires: gtk+ >= 2.12.8
- What if gtk+ is compiled with GCC 3.3 and Firefox with GCC 4.2?
Dependency specifications

Conventional package managers e.g. RPM have nominal dependency specifications:

- **Example**: Firefox
- **Requires**: `gtk+ >= 2.12.8`
- **What if `gtk+` is compiled with GCC 3.3 and Firefox with GCC 4.2?**
- **Components cannot be linked together due to a different ABI.**
The Nix store

d8imsim3zs87dskyvmgdd24dn9ak4nbp-gtk+-2.12.8

Components can be stored next to each other because they do not share the same name if the inputs differ.
The Nix store

d8imsim3zs87dskyvmgdd24dn9ak4nbp-gtk+-2.12.8

- gtk+-2.12.8. Name of the component
- d8imsim3zs87dskyvmgdd24dn9ak4nbp: SHA 256 hash code based on all inputs such as the used to build the component such as the source code of gtk+, used compiler, linker.

All components are stored in the Nix store, which is usually /nix/store.

Components can be stored next to each other because they do not share the same name if the inputs differ.
The Nix store

```
/nix/store
  19w6773m1msy...-openssh-4.6p1
    bin
    sbin
      sshd
  smkabrbibq7...-openssl-0.9.8e
    lib
      libssl.so.0.9.8
  c6jbqm2mc0a7...-zlib-1.2.3
    lib
      libz.so.1.2.3
  im276akmsrhv...-glibc-2.5
    lib
      libc.so.6
```
{stdenv, fetchsvn, apacheAnt, jdk, mysql_jdbc, smack, config}:

stdenv.mkDerivation {
    name = "XMPPLogger";

    src = fetchsvn {
        url = svn://gforge/.../SDS2Applications/XMPPLogger;
        md5 = "62401d706079cb61d3b7e0badc818a1d";
    };

    builder = ./builder.sh;

    inherit jdk mysql_jdbc smack config;
    buildInputs = [apacheAnt jdk smack config];
}
rec {
    config = import ../config {
        ...
    };

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

    XMPPLLogger = import ../SDS2Applications/XMPPLLogger {
        inherit stdenv fetchsvn apacheAnt jdk;
        inherit smack mysql_jdbc;
        inherit config;
    };
    ...
}
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`
Service Development Support System (SDS2)

- Developed at the Healthcare Systems Architecture department of Philips Research
- Asset management and utilization services for mobile medical equipment
- Service Oriented Architecture
  - Information as a service
  - Services could be distributed across different machines in the network
Asset Tracker Service

Sander van der Burg

Model-driven Distributed Software Deployment
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
let pkgs = ..../top-level/pkgs.nix;
in
{
    XMPPLogger = {
        name = "XMPPLogger";
        pkg = pkgs.SDS2.webservices.XMPPLogger;
        dependsOn = [ EjabberdService MySQLService ];
    };

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

    ...
}

{  
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;
  };
}
{services, infrastructure}:

[
    { service = services.FloorPlanService;
      target = infrastructure.dtk15; }
    { service = services.ME2MSService;
      target = infrastructure.dtk15; }
    { service = services.MELogService;
      target = infrastructure.dt2d1; }
    { service = services.XMPPLogger;
      target = infrastructure.dtk15; }
    ...
]

Sander van der Burg

Model-driven Distributed Software Deployment
Distributed upgrading

Commit-request phase:
- Build all components on the coordinator machine
- Transfer the closures to the cohort machines through the webservice interface
- If one of the cohorts fail: no undo operations are needed

Request phase, on each cohort:
- Install component in profile
- Activate the component
- If one of the cohorts fail: rollback on every cohort
- A proxy can be used to drain connections and block/queue access during the transition
- (Almost) atomic
Adapting SDS2

Nix is known as the purely functional deployment model. The deployment model of Nix/Disnix is normative. We have to adapt systems to fit in the deployment model (normalize) in order to make dynamic reconfiguration possible.
Nix is known as the purely functional deployment model
The deployment model of Nix/Disnix is normative
We have to adapt systems to fit in the deployment model (normalize) in order to make dynamic reconfiguration possible.
Adapting SDS2

- **Local deployment.** Creating build scripts and Nix expressions for SDS2 platform and infrastructure components.
- **Less strict inter-dependencies.** Modifying SDS2 services to reconnect in case of a network failure.
- **Dynamic binding.** Implementing a lookup service which provides locations of services. Modified SDS2 services to use the lookup service for each request to another service.
Conclusions

- 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 systems to distributed systems.
- We identified design constraints to make a distributed system deployable with Nix/Disnix.
- We have demonstrated that we can deploy SDS2 in a distributed environment.
Future work

- Dynamic distribution based on Quality of Service models
- Investigate more development constraints
  - Implementing atomic-commit without blocking
- Support for heterogeneous environments, example:
  - Linux coordinator
  - Windows/FreeBSD/Linux based network
- Support for other types of distributed systems with other protocols
- Security