e2 factory
the emlix Embedded Build Framework
Agenda

- Motivation
- Basic Concepts
- Design and Implementation
- Working with e2 factory
e2 factory
Motivation
Motivation

Development Tools

- Source Code Management – about maintaining source code
- IDE or simply Editors – about development

- Build Framework – about reliable builds
Motivation

Requirements for embedded build systems:
- Automated builds
- Efficient development

Intended Audience
- Industrial Embedded Linux Developers
Motivation

Specific requirements for Industrial Embedded Linux Developers

- Reproducible builds
- Long term maintenance
- Development in distributed teams
- Support platform strategies
- Open Source specific: Care about licences
e2 factory
Basic Concepts
Basic Concepts

How to build an Embedded Linux Software System?

- Build a toolchain
- Build a kernel
- Build system software and libraries
- Build product specific software
- Compose things, usually into a kernel image and a root-filesystem image, ready to deploy

Component Based Software Engineering
Basic Concepts

The basic composition process
Basic Concepts

Cascading composition processes

- Toolchain
- Kernel Image
- Root Filesystem Image
e2 factory
Design and Implementation
Design and Implementation

Translating abstract terms into implementation terms

Composition translates into *build process*

Components are called

- *Sources* and *dependencies* when talking about *build process* input
- *Results* when talking about *build process* output
The build process

Source

Dependency

Build Configuration

Build Process

Build Environment

Result
Design and Implementation

The build process

- Setup the build environment
  - extract tarballs
The build process

- Copy things into the build environment
  - Sources,
  - Dependencies

- Install the build configuration
  - Build script
  - Shell environment
  - Build script library
Design and Implementation

The build process

- **Build**
  - Change the root directory to the build environment (`chroot()`)  
  - Run the build script
- The build script leaves the build output in a directory
Design and Implementation

The build process

- Store the result
  - Fetch the resulting files from the build environment
  - Create the result package
  - Store the result to the server
Design and Implementation

BuildId - Know what you build in advance

- Before building we calculate a cryptographic hash over any of the process inputs (sources, dependencies, build environment,...)
- We call that hash $\text{BuildId}$

- $e2$ factory stores results accessible through the $\text{BuildId}$
Design and Implementation

Build Cache

- Rebuilding is only done when any process input changed
- Results can be stored on a shared server
- They are available across multiple developers immediately
- Dependency tracking is fully automated and reliable

...unless the unlikely case of a hash collision happens. We use the sha1 hash algorithm which we think is strong enough to minimize risk here.
Reproducibility and Long Term Maintenance

- Industrial Embedded Systems need maintenance for many years
- Reproducibility is mandatory requirement to allow long term maintenance
Design and Implementation

Reproducibility and Long Term Maintenance

- e2 factory is split into
  - global tools, installed system wide
  - local tools, installed within each project environment
- Set of global tools
  - is small
    - maintains compatibility to former generations of local tools
- Local tools control the build process
- Version of local tools is locked to each single project
Reproducibility and Long Term Maintenance

- The project configuration is maintained within a Source Code Management System
- Sources are taken from
  - a SCM System
  - archive files and patches
Design and Implementation

Reproducibility and Long Term Maintenance

- The same, stable build environment is used
  - by all developers during development
  - in release builds
- Each build process runs in a fresh build environment
- Building is done with the root directory changed to the build environment
  - host system independence
  - build processes do not influence each other
Design and Implementation

Working in Teams – local or distributed

e2 factory is a distributed system and offers high flexibility

- Developers can share build results by automatically pushing them to a central server
- No more repeated builds across the team, results are looked up by their BuildId and reused
- A local cache can be used, for performance reasons
Design and Implementation

Working detached (or with limited network bandwidth)

e2 factory is flexible enough to support detached work

- The local cache can be filled in advance with relevant data
- Building and development does not require a network connection in this case

There are limitations: e2 factory relies on SCM System access. Detached work requires a distributed Source Code Management System (git)
e2 factory
Working with e2factory
What does a e2 factory project look like?

- Basic configuration entities are
  - Site configuration (system-wide, per user)
    - Servers
    - Policies
  - Project
    - Chroot
    - Licence
    - Environment
    - Sources
    - Results
What does a e2 factory project look like?

- **Basic configuration entities are**
  - Project
  - Chroot
  - Licence
  - Environment
  - **Sources**
  - Results

```
e2source {
  name = "busybox",
  licences = {
    "gpl2",
  },
  file = {
    {
      server = "upstream",
      location = "busybox-1.15.0.tar.bz2",
      unpack = "busybox-1.15.0",
    },
  },
}
```
What does a e2 factory project look like?

- Basic configuration entities are
  - Project
  - Chroot
  - Licence
  - Environment
  - Sources
  - Results

```bash
e2source {
    name = "busybox-config",
    file = {
        { server = ".",
          location = "src/busybox/busybox.config",
          copy = "busybox.config",
        },
    },
}
```
Working with e2 factory

What does a e2 factory project look like?

- Basic configuration entities are
  - Project
    - Chroot
    - Licence
  - Environment
  - Sources (git)
  - Results

```yaml
e2source {
  licences = {
    "gpl2",
  },
  type = "git",
  server = "git",
  location = "linux-2.6.git",
  branch = "master",
  tag = "v2.6.31",
}
```
What does a e2 factory project look like?

- Basic configuration entities are
  - Project
    - Chroot
    - Licence
    - Environment
  - Sources
  - Results
    - Configuration
    - Build script

```plaintext
e2result {
    name = "busybox",
    chroot = {
        "base",
    },
    depends = {
        "toolchain",
    },
    sources = {
        "busybox",
        "busybox-config",
    },
}
```
What does a e2 factory project look like?

- Basic configuration entities are
  - Project
  - Chroot
  - Licence
  - Environment
  - Sources
  - Results
    - Configuration
    - Build script

```
  cd busybox
  cp ../busybox-config/busybox.config .config
  make ARCH=${cross_arch} CROSS_COMPILE=${target_platform}-
  make ARCH=${cross_arch} CROSS_COMPILE=${target_platform}-
      CONFIG_PREFIX=${ROOT} install
  tar -czf ${OUT}/busybox.tar.gz -C ${ROOT} .
```
What does a e2 factory project look like?

- Basic configuration entities are
  - Project
    - Chroot
    - Licence
    - Environment
  - Sources
  - Results
    - Configuration
    - Build script

```cmake
e2result {
  name = "rootfs",
  chroot = {
    "base",
  },
  depends = {
    "libc",
    "busybox",
    "zlib",
  },
  sources = {
  }
}
```
Working with e2 factory

What does a e2 factory project look like?

- Basic configuration entities are
  - Project
  - Chroot
  - Licence
  - Environment
  - Sources
  - Results
    - Configuration
    - Build script

```
tar -xzf ${DEP}/busybox/busybox.tar.gz
   -C ${ROOT}

tar -xzf ${DEP}/zlib/zlib.tar.gz
   -C ${ROOT}

tar -czf ${OUT}/rootfs.tar
   -C ${ROOT} .
```
Working with e2 factory

Basic use cases

- Reproducible Builds
- Development

```
$ e2-build busybox
 skipping binutils  [abcdef...]
 skipping gcc       [5176ab...]
 skipping libc      [123abc...]
...
 skipping toolchain [443456...]
 building busybox   [456123...]
$
```
Working with e2 factory

Basic use cases

- Reproducible Builds
- Development
  - The playground, a shell inside the build environment

```bash
$ e2-build --playground busybox
building busybox   [456123...] [playground]

$ e2-playground busybox
entering playground...
#
```
Working with e2 factory

An approach to platform based development

- Maintain a common platform for multiple products
- Keep development close together
  - share as much as possible
- Keep the products independent enough
  - different product life-cycles
Working with e2 factory

An approach to platform based development

- The generic part has well-defined interfaces for product development
Working with e2 factory

An approach to platform based development

- Products depend on the generic platform
- Products represented by results
Working with e2 factory

An approach to platform based development

- Project is self-contained
  - Toolchain included
  - Fully automated dependency handling

- Rebasing products onto different hardware is easy
  - Required due to discontinued hardware or
  - Growing hardware requirements
Thank you for your attention!

www.e2factory.org

e2factory@emlix.com
www.emlix.com