CHALMERS

# Porting MeeGo to LEON

*Master of Science Thesis*

IVAN BERTONA

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, August 2011

Porting MeeGo to LEON

Ivan Bertona



Examiner: Arne Dahlberg

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, August 2011

POLITECNICO DI TORINO

III Faculty of Information Engineering
Master of Science in Computer Engineering

Master Thesis

# Porting MeeGo To LEON

**Supervisor:**
Bartolomeo Montrucchio

**Candidate:**
Ivan Bertona

August 2011

# Abstract

Portable multimedia devices are the flagship of a steadily growing market, from which the LEON/GRLIB hardware platform was excluded due to the lack of suitable software support.

This thesis work addressed such problem by initiating a port effort of MeeGo, a Linux-based mobility-oriented operating system, to the SPARC-compatible LEON processor and to the hardware components provided by the GRLIB IP core library.

The build infrastructure of MeeGo was modified and set up accordingly, and a partially working but usable MeeGo port was produced.

The LEON/GRLIB platform was evaluated from a multimedia application standpoint. Lacking areas such as OpenGL hardware acceleration were pinpointed and the necessary improvements outlined.

# Acknowledgements

First of all I would like to express my gratitude to all the team that makes Aeroflex Gaisler AB a great workplace. In particular to Jiri Gaisler, Kristoffer Glembo and Daniel Hellstrom who most closely helped me by providing support, infrastructure and precious advice. Additionally, I would like to thank my two academic supervisors, Arne Dahlberg at Chalmers and Bartolomeo Montrucchio at Politecnico di Torino, for their expert guidance and supportive attitude.

A special thanks goes to all my Erasmus friends. To Giampaolo Calestani and Isabella Mordeglia for remembering me that sometimes it is fine to take a break (and for making it remarkably good). To Fabio Castro and Tanja Schindler for staying a bit longer with me. To all the others that I cannot name here for teaching me that regardless where you go, you will always find great people to share your life with.

An equally special thanks goes to all my old friends from Italy. To Alessandro Bianchi for being a brother more than a friend. To Luca Cerri for stalking me with his persistent bad taste. To Francesca Di Meo for not calling me too often. To Alessandro Cannarozzo and Stefano Cannillo for all the adventures in Torino. To all of you who did not forget me, I did not forget you too.

Finally, and most importantly, I wish to thank my loving parents Maurizio Bertona and Lucia Simonotti, to whom I dedicate this thesis. Without you nothing would have been possible.

# List of Figures

# List of Tables

# Contents

# Chapter 1

# Introduction

This chapter provides an overview of the thesis work, the motivations behind it and the declared objectives. To clarify and support these elements, some background and context information is also included.

## 1.1 Project description and objectives

The main purpose of this project was to create a port of the MeeGo operating system for the LEON/GRLIB System-on-Chip (SoC) platform.

MeeGo is a Linux distribution designed to run on mobile and embedded devices such as netbooks, smartphones and in-vehicle computers. A custom desktop manager provides several user interfaces suitable for use with small screens, while a set of kernel patches guarantees enhanced compatibility with mobile hardware.

The hardware platform is developed by Aeroflex Gaisler AB and features a LEON synthesizable processor, which implements the SPARC v7 and v8 instruction sets[1]. The companion IP library GRLIB provides support components such as a system bus, several I/O interfaces and a framebuffer controller, all of which can be extensively configured and embedded on a single chip.

### 1.1.1 An informal introduction

In the last years we have observed a steady growth of market demand for portable multimedia devices. Mobile Internet traffic increased by a factor of 7 in Western Europe between May 2008 and May 2010, and by a factor of 13 in North America under the same period [1]. Hardware designers and manufacturers are consequently

[1]The v8 version differs from v7 only due to the introduction of hardware multiply and divide instructions, which can be optionally included in the synthesized LEON processor. A technical description of the instruction set can be found at [28].

exploring the possibility to enter such market, which of course requires know-how and investments. The first mandatory steps in this process include creating device prototypes, hardware and software, and estimating costs and marketability of products based on the developed prototypes.

The LEON/GRLIB platform is mainly used in aerospace applications, thanks to its optionally fault-tolerant and radiation-tolerant design. Use cases include Data Handling Systems (DHS) and Attitude and Orbit Control Subsystems (AOCS). On the other hand, nothing prevents its use in less critical contexts [14], such as the implementation of a set-top box or as a generic, multi-purpose embedded controller. As a matter of fact, though, multimedia applications are not a core feature of the platform and for this reason the available hardware support for multimedia is limited. For example, while a framebuffer device for video output is available, hardware video acceleration is not supported. Moreover, an AC97-compatible audio controller is under development but still not ready for use.

The objective of this thesis work was to provide the software required to evaluate LEON/GRLIB as a platform for multimedia and portable applications and pinpoint the areas where improvements are necessary. At the time of writing, the possible outcomes still range from the simple availability of useful test data to the prospective implementation of new products.

The classical areas in which the LEON/GRLIB platform is excellent, such as connectivity and reliability, combined with a refreshed multimedia outfit, might well become key points for platform adoption. For example, consider the availability of a Controller Area Network (CAN) bus interface. This interface is specifically designed for automotive applications to allow communication between the various vehicular subsystems. MeeGo provides and In-Vehicle Infotainment interface which might allow to develop a good car computer able to show telemetry and status information in real time, without the need for additional bridging hardware.

### 1.1.2 On the technical side

The LEON/GRLIB platform is designed as a collection of modular IP cores[2], implemented in VHDL, which describe the various available hardware components and the way in which they interact with each other. These modules can be configured extensively and synthesized in hardware using one of the two mainstream technologies: Field Programmable Gate Arrays (FPGA) or Application Specific Integrated Circuits (ASIC). Flexibility, performance, cost and time-to-market can be balanced every time according to the project requirements. The actual synthesis process and design of hardware, though, are outside the scope of this thesis work and will not

[2] The Intellectual Property (IP) core modular design pattern is commonly used in the industry to isolate hardware components in reusable blocks.

be covered in detail. Aeroflex Gaisler AB provided all the hardware required for testing, whose capabilities and specifications are described later in this report.

Without diving to much to details, for now, the available hardware modules range from the SPARC processor itself, with SMP support, to SDRAM and DDR memory controllers, a framebuffer interface and various I/O subsystems (e.g. USB, PCI, CAN, AMBA, SPI, I2C, Ethernet).

After the former official Linux kernel maintainer for the SPARC v7 and v8 architectures resigned in 2007 with no replacement, all the main Linux distributions dropped the support for them[3] and upgraded to the more recent v8+ version, which features 64-bit registers. This fact introduced some concern about the actual feasibility of the port, compared to the available time and workforce of one person for about six months. Thanks to the work of the Aeroflex Gaisler AB team, though, the Linux kernel can be patched to support the LEON/GRLIB platform. Moreover, all the core software components such as the GNU C Standard Library and the GNU Compiler Collection still retain support for the target architecture. Eventually most of the problems were addressed and solved. Known issues and uncompleted tasks are described in detail in section 3.2.

The core part of the porting process basically consisted in recompiling all the software packages and components to target the SPARC System V Application Binary Interface (ABI). Part of the packages had to be patched in order to work, while most did not yield any problem. Few packages, on the other hand, did not support the SPARC architecture at all and due to their size and complexity were left out from the port. A detailed list of the affected packages is available in appendix A.

The rebuild was performed as a two-step process. First, a bootstrap repository was created by manually cross-compiling a subset of the distribution. The bootstrap repository was later used to seed the official MeeGo build infrastructure and attempt to rebuild all the available packages. To succeed in these tasks a particular temporary build environment was created, blending a cross compiler and native SPARC hardware, while the official build infrastructure was modified to support full-system emulation of the SPARC build target. The porting process is covered in detail in chapter 2.

### 1.1.3 Road map and methodology

At planning time, a set of best practices to be applied throughout the project was defined. The idea was to keep a consistent behavior across the various phases of development and to help focusing on the final result.

- *General rule of thumb* — Rely as much as possible on the existing MeeGo

[3]As an example, the discussion that led Debian to drop SPARC v8 can be found at [3].

infrastructure and tools, modifying them if needed in order to support the SPARC architecture.

- *Regarding the compilation of packages* — In case of failures, first try to modify the build script in such a way that it does not affect the other architectures (e.g. using architecture if switches). Apply source code patches only in case no other fix can be developed.
- *Regarding the completeness of the port* — Give priority to the core distribution packages, which include the entire base system and the Xfce desktop environment[4]. Later concentrate on the netbook flavor of MeeGo, since it is the most appropriate for the available hardware.

Additionally, after a first period of study and documentation, a list of milestones was sketched in order to split the project in a number of phases, to be completed incrementally. The steps were:

1. Set up a temporary build environment and attempt to build some packages, make sure that they run on the LEON/GRLIB platform.
2. Using the temporary build environment, build enough packages to seed an initial bootstrap repository.
3. Add support for the SPARC architecture to the official MeeGo build system.
4. Seed the build system with the bootstrap repository and generate a SPARC repository.
5. Test the SPARC MeeGo images both in emulation and on real hardware, iterate the previous steps if necessary until a satisfactory result is obtained.
6. Package and document the port, keeping track of the modified components and providing a relevant set of patches.
7. Collect all the documentation produced in the previous steps and write a full report covering the various aspects of the thesis work.

## 1.2 Background

This section provides some brief background information about the current state of the mobile computing landscape and technical details about the structure of the MeeGo distribution and the LEON/GRLIB platform, useful to understand the remainder of the report.

[4]The Xfce desktop environment [30] is included in the MeeGo core repository but not installed by default in any official MeeGo image.

### 1.2.1 The mobile computing landscape

As said before, in the last years a strong increase in the availability and use of portable, connected, mobile devices has been observed. This development was backed by technological advancements in the areas of wireless connectivity, hardware miniaturization and energy storage and management.

At the same time the market fragmentation increased with the introduction of new devices such as netbooks, in-vehicle computers and smartphones capable of browsing the mainstream web.

Companies like ARM Holdings (which designs a low-power synthesizable hardware platform) and Texas Instruments (which produces, among others, ARM-based ASICs), were positively affected by this trend. ARM Holdings shares multiplied their value by almost three times over the last three years [4].

On the software side, the situation is completely different if compared to the desktop and laptop market. The embedded nature of portable devices, where the software is tightly coupled with the hardware, lowered the entrance barrier for new operating systems, resulting in the diffusion of several main actors:

- *Android* — Developed by Google, open source, Linux-based. Runs on smartphones and tablets produced by several manufacturers.
- *iOS* — Powers the Apple iPhone and iPad.
- *Windows Phone 7* — Developed by Microsoft, proprietary source. Runs on smarthpones produced by third-parties.
- *WebOS and Blackberry* — Run respectively on Palm and RIM smartphones and occupy a relatively niche market.
- *Several Linux distributions and Windows XP/7* — Can be adapted to run on netbooks and tablets with screens in the nine to twelve inches range.

In an attempt to make its Atom and Moorestown platforms more interesting to device manufacturers, Intel initiated the development of MeeGo in collaboration with Nokia [10], based on the previously separated efforts Moblin and Maemo. Unfortunately, after bad market performance and a change of CEO, Nokia revised its mobile strategy, moving from MeeGo to Windows Phone 7 [19].

Intel nevertheless confirmed its commitment to MeeGo and at the time of writing is working on the development of the 1.2 release.

### 1.2.2 The MeeGo OS

MeeGo is a fairly standard RPM-based Linux distribution, which differs from the others mostly in the user interface area. Additionally, MeeGo is a real open source project, backed by several companies, which employs best practices of open source development, providing convenient access to code base, forums, mailing lists and a transparent steering committee. All patches applied to MeeGo packages are also forwarded upstream in order to contribute to the open source ecosystem and increase maintainability.

To better adapt to the different kinds of available devices, various flavors of MeeGo are developed. These variants provide different user interfaces that fit diverse screen sizes and enhance usability for the intended usage patterns. Different kernels are also shipped to support the varying hardware capabilities. The MeeGo flavors are:

- *Netbook* — For laptops with a small form factor, usually equipped with a 10" screen.
- *Handset* — For smarthpones with Internet connectivity, usually equipped with a touch screen and able to make phone calls over mobile telephony networks.
- *In-Vehicle* — For vehicular computers, equipped with a touch screen, used to deliver information such as telemetry, weather and traffic reports and to entertain the passengers.
- *Smart TV* — For interactive TVs able to show Internet-based content and execute entertainment applications such as simple videogames.
- *Media phone* – For landline phones with extended capabilities such as video conferencing, message recording and contacts management.

Consequently, MeeGo is suitable for use with most of the portable devices available on market, provided that Linux drivers exist for the desired hardware.

#### Organization and components

As shown in Figure 1.4, which is provided by the development team at [16], MeeGo can be logically separated in three layers:

- *Core OS* — Contains the Linux kernel and related drivers and all the middleware and OS services used by higher layers, mostly resembling any other Linux distribution.

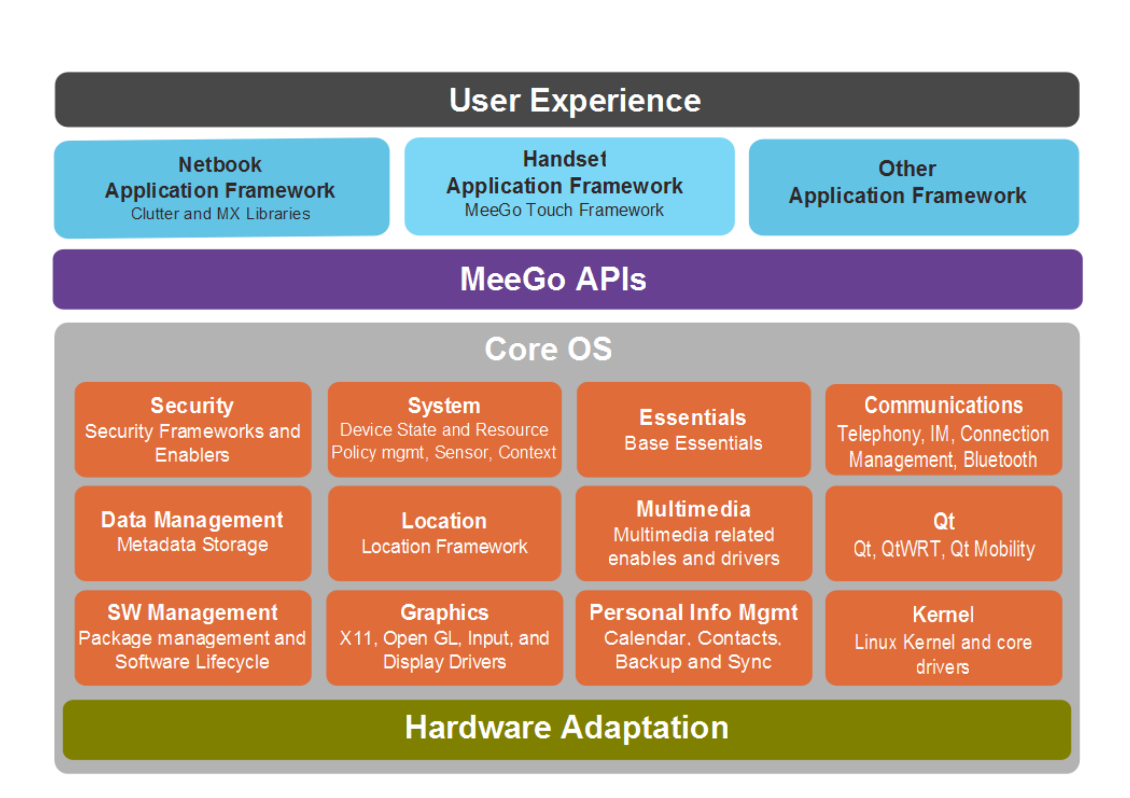


Figure 1.1. The MeeGo Architecture

- *MeeGo API* — Provides an interface for application development which simplifies the creation of portable applications with a standardized interface.
- *User Experience* — Contains the user interfaces for the various MeeGo flavors as well as additional flavor-specific API components.

### Distribution and build infrastructure

MeeGo is developed, maintained and compiled using a locally deployed instance of the OpenSUSE Build Service (OBS) [20]. The instance can be reached at [17].

The OBS allows to create a distributed build system able to keep track of the package sources (using a source control system) and of the package dependencies (by parsing the RPM specification files). The system consists of a main server where the packages are stored and of a variable number of build workers, which might be distributed on other machines over the network. Remote access is provided both via a command line client and a web interface. Security is implemented via access control lists where different accounts can be given different privileges regarding what can be done on the system, from read-only anonymous access to full administration capabilities.

The OBS is able to completely rebuild several Linux distributions, based on several package management systems. If a bootstrap repository and native workers are available, any architecture can be targeted. If native workers are not available the OBS can use x86 workers to target foreign architectures, either by cross-compiling or using software emulation.

#### SDK and application development

Developers interested in creating applications for MeeGo can use the MeeGo SDK, available both for Windows and Linux [23]. The SDK integrates development tools, the QEMU emulator and precompiled MeeGo images, allowing the developers to test their applications without the use of real hardware. Thanks to the use of MeeGo APIs, applications are easily portable. The included build tools can also cross-compile applications to the ARM target.

### 1.2.3 The LEON/GRLIB platform

As previously said, the entire LEON platform and GRLIB IP core library are implemented in VHDL and can be synthesized in hardware using different technological solutions. Currently two main options are available:

- *Field Programmable Gate Array (FPGA)* — This technology delivers low performance but has the property of being reprogrammable multiple times. Consequently, it is mainly used for small series production with a short time-to-market.

- *Application Specific Integrated Circuit (ASIC)* — This technology delivers better performance but produces integrated circuits that cannot be modified, whose cost-per-unit decreases with volume production. It is mainly used for final prototypes, evaluation boards and actual products produced in large series.

#### The LEON processor family

The LEON soft-processor family provides a certified implementation of the SPARC v7 and v8 architectures. The latest version, LEON4, is software compatible with the previous versions and has the following features [12]:

- 7-stage pipeline with branch prediction, hardware multiply and divide units, IEEE-754 compliant FPU.

- Separate instruction and data L1 cache (Harvard architecture) with snooping, configurable with 1 - 4 ways, 1 - 256 kbytes/way and random, LRR or LRU replacement policy, L2 cache up to 8 MB in size.
- Memory Management Unit (MMU) as of the SPARC specification, with configurable Translation Lookaside Buffer (TLB).
- Symmetric Multiprocessing (SMP) support up to sixteen cores.
- Advanced on-chip debug support with instruction/data trace buffer, hardware breakpoints, performance counters and more.

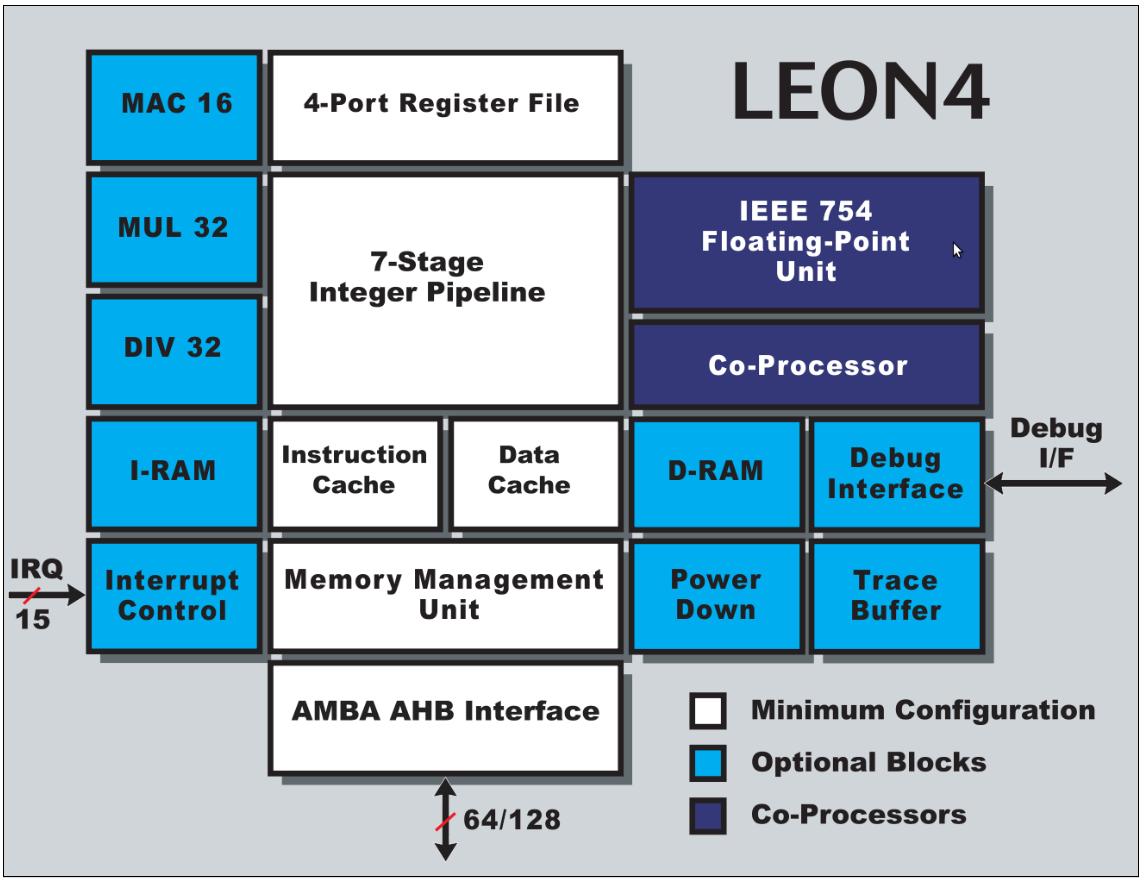


Figure 1.2. The LEON4 processor [13]

Performance wise, the LEON4 processor delivers 1.7 DMIPS/MHz, 2.1 CoreMark/MHz and 0.35 SPECint2000/MHz. The actual clock frequency is customizable, depending on the physical technology, and currently ranges from 125 MHz on a Virtex5 FPGA board to 1500 MHz using 32 nm ASIC.

### The GRLIB IP core library

The IP cores provided by the GRLIB library can be synthesized together with the processor, allowing the creation of a System-on-Chip (SoC) platform [25]. Interestingly, the library is released under the open source GNU GPL license. All the components are centered around a on-chip multi bus architecture and include, among others:

- PCI bridge with DMA.
- 32-bit PC133 SDRAM controller.
- 16/32/64-bit DDR/DDR2 controllers.
- PS/2 controller.
- 10/100/1000 Mbit Ethernet.
- USB 2.0 host and device controllers.
- Framebuffer video device with DVI interface.
- Basic system peripherals: timer, interrupt controller, UART, etc.
- Other interfaces: CAN, TAP, SPI, I2C, ATA, etc.

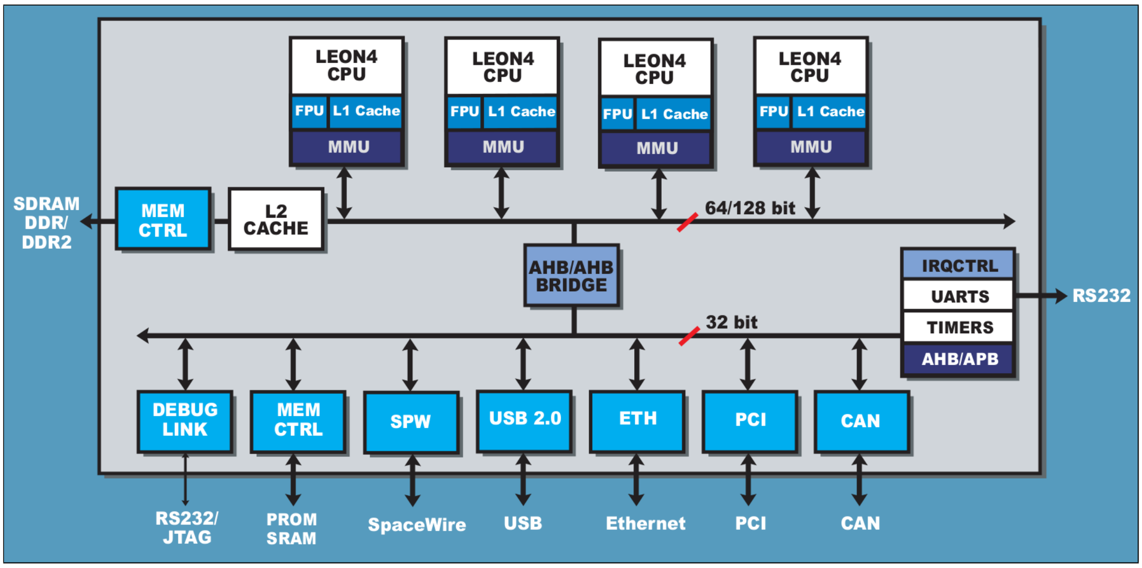


Figure 1.3. An example LEON4-based SoC schematic [13]

### Test and evaluation boards

The LEON platform is compatible with several FPGA test boards and synthesis tools developed by Actel, Altera, Lattice and Xilinx. A complete list is available at [25]. Additionally, some boards specifically designed for LEON/GRLIB are available as listed at [11].

These boards can be connected to a developer workstation using both Ethernet and serial interfaces. The GRMON utility leverages on the debug unit integrated in the LEON processors to provide a fully-featured step-by-step software debugger and processor inspector, with read and write capabilities (e.g. it is possible to write to memory addresses and registers when the processor is on hold).

A typical Linux development setup, which is relevant to this project, consists in loading a Linux kernel by issuing some commands in GRMON and mounting the root file system, hosted on the developer machine, using the Network File System (NFS) protocol. Thanks to this approach it is not necessary to regenerate and flash new images of the system every time a change occurs.

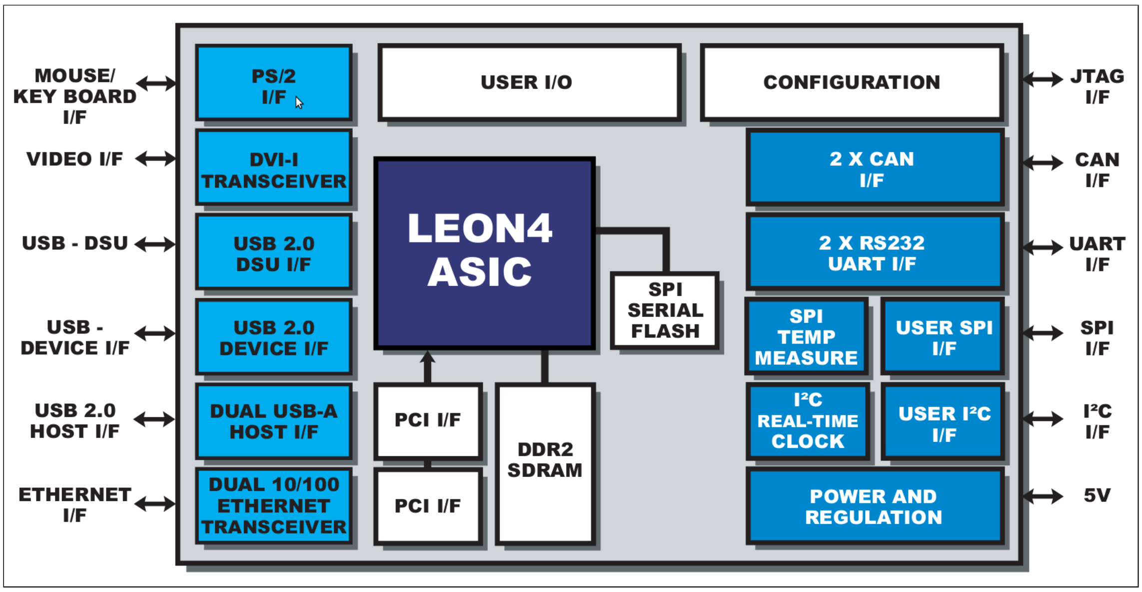


Figure 1.4. The GR-LEON4-ITX development board [6]

### The GR-LEON4-ITX development board

For the testing part of the thesis project a GR-LEON4-ITX board [7] was used. It is a general purpose, custom design, evaluation and testing board which provides the fastest piece of LEON/GRLIB hardware physically implemented to date. It has the following characteristics:

- Mini-ITX format with dual PCI slots.
- LEON4 SoC ASIC including dual LEON4 cores clocked at 200 MHz.
- 256 MB of 32-bit DDR2-400 SDRAM.
- 8 MB SPI serial flash PROM.
- Dual 32-bit 33 MHz PCI slots, dual 10/100 Ethernet interfaces, dual USB 2.0 host interfaces, dual CAN bus interfaces, SPI and I2C controller interfaces, dual PS/2 keyboard and mouse connectors, 44-bit generic user I/O.
- Debug support over Ethernet, USB or serial connection.

The testing of the MeeGo SPARC distribution is described in chapter 3, where also information about the board and its usage procedures is provided.

# Chapter 2

# The porting process

This chapter describes in detail the process that led to the creation of a working SPARC MeeGo repository. Some early failed attempts are also briefly discussed in order to clarify part of the design choices.

## 2.1 Tools, techniques and design choices

All the major Linux distributions consist in a collection of packages, or software archives, which can include source code, patches, build scripts and executable binary files. Usually, binary repositories for one or more architectures are generated starting from a set of source packages. For this purpose, the distribution maintainers develop and employ tools that allow to keep track of the package history and sources, distribute the maintenance tasks among a team of developers and automatically rebuild the packages when needed. For example, MeeGo relies on the OpenSUSE Build Service (OBS).

Thanks to these build systems, the transition towards new versions of the packages (and later to new releases of the distribution) is handled gracefully. In particular, once an initial binary repository is established for each of the supported architectures, every modification can be easily built and tested against the current code base, leading to a constant flow of updates which can be isolated and, if necessary, fixed from time to time.

What the tools cannot automatically handle, regardless the actual build technique, for various reasons that will be illustrated in the remainder of this chapter, is the generation of the initial bootstrap binary repository. Such repository, which might also come from a similar pre-existing Linux distribution, usually contains a minimal functional system, which is then used to initialize the build system.

### 2.1.1 Building techniques

Several techniques can be employed to build packages, being more or less convenient depending on a series of environmental factors such as the availability of fast native hardware, the support for emulation techniques and, last but not least, the way in which the available build scripts are written.

#### Cross or native builds

The first aspect to be considered concerns the relation between the architectures of the machine where the build is performed and of the generated binaries, and distinguishes the building techniques in two main groups:

- *Native build* — The target architecture of the compiler is the same of the compiler itself and, consequently, of the build machine.
- *Cross build* — The two architectures differ in such a way that the compiled binaries cannot be run on the building machine. This fact implies that the used compiler, which in this case is referred to as a cross compiler, is able to generate executable code for one or more foreign architectures.

It is worth to note that gray areas do exist. For example, consider the generation of 32-bit x86 binaries on a 64-bit x86_64 machine. Strictly speaking this is a cross build, but it is often regarded as a native build due to the fact that the generated binaries are still compatible with the target machine architecture.

#### Emulated native builds

Native builds can be executed on machines characterized by a different architecture by the means of emulation. A software interpreter reads the foreign binaries and, after a translation process, executes them on the local hardware. This technique is very flexible, but incurs in a high performance penalty due to the translation overhead. Still, it has the clear advantages of neglecting all the complexity related to cross compilation (which is often untested or entirely not supported by the package developers) and allowing for a native build even when real hardware is not available, affordable or for some other reason usable.

#### Emulation techniques

Few words must also be spent about emulation, in order to clarify then next paragraphs. It is possible to identify two main emulation techniques:

- *Full system emulation* — Implements the functionality of a complete system (which might even be fictional) from the ground up, simulating the interaction with hardware at low level. An entire operating system can be booted inside the emulator, which in turn runs as a process inside the host operating system. The emulated software might transparently access the hardware resources of the host machine (e.g. network cards) through an interface layer provided by the emulator.
- *User mode emulation* — Allows direct interaction between the running operating system and a foreign architecture binary by interpreting its instructions and translating all the executed system calls to the equivalents on the system. This translation mechanism usually requires the foreign binary to be compatible with a very similar environment and, due to the great amount of details involved, might result in partly faulty behaviors which are often hard to track down and fix.

Since it does not require an entire operating system to be installed or otherwise configured, user mode emulation is the most convenient technique to perform emulated native builds. Full system emulation, instead, is best suited for hardware and software testing or simulation.

### Chroot environments

Another aspect of the build is the interaction between the built software and the operating system, tools and libraries installed on the build machine. Application developers usually want to build and test their applications against a range of different user configurations without being forced to set up multiple test environments, reboot their workstations or employ additional hardware.

On the other hand, it is not easy to guarantee the reproducibility of the build on a heterogeneous range of configurations. For example some specific version of the build tools might be required for the build to succeed. The compiled binaries, moreover, should be isolated from the build system libraries, and linked against the libraries of the target system.

In other words, it would be convenient to be able to build applications in the same environment where they will be executed, with minimal effort by the application developers. A possible solution, which is not optimal for performance reasons, consists in booting such environment in a full system emulator. When working on Unix systems, though, a better alternative is available: the use of a chroot environment.

The chroot system call, introduced first in Version 7 Unix [29], allows to modify the execution environment of a process by temporarily changing the root of the file system to one of its subdirectories. The build technique consists in populating a directory on the host system with an image of the target system and chrooting into that directory. The two systems should be similar enough. In particular, the standard C library of the target system should be compatible with the system call layout of the running kernel.

The chroot environment might also include a blend of native and foreign binaries. The foreign binaries can be executed by an user mode emulator when necessary. This technique, known as mixed chroot, allows to target foreign architectures while mitigating the performance problems related to emulation by running part of the tools natively, when the results are not expected to be different.

### Building packages

Package managers usually provide the automation required to create chroot environments and build packages. For this reason, in most cases, two main kinds of packages are defined:

- *Source packages* — They contain the software sources, any relevant patch and the specifications of the content of the generated binary packages. Moreover, several scripts might be attached to implement tasks such as the actual compilation of the package.
- *Binary packages* — They contain the actual binaries, for one or more architectures. In the latter case, they are referred to as universal binary packages. Some package managers also support the inclusion of scripts to be executed before or after a particular event such as the installation of the package.

Several binary packages are usually generated starting from a single source package. For example, consider a software library. The development headers might be placed in a different subpackage than the dynamically loadable library, to allow their installation only on systems where they are actually needed.

The enclosed support scripts are usually interpreted (not compiled) in order to simplify portability. Nevertheless, package maintainers should pay particular attention to make sure that the scripts are indeed written in a portable way, for example by avoiding to hardcode specific paths or architecture names.

In its simplest form a package consists of a compressed archive containing the package files and shell scripts. Currently, more advanced systems are available, where packages are associated with metadata (e.g. build and installation dependencies, author names and e-mail addresses, etc.) and custom scripting languages are implemented to simplify the writing of maintenance scripts. Examples of package manager are the RedHat Package Manager [22], the Debian package management tools [27] and Slackware's pkgtool [24].

### Finding and retrieving packages

Most Linux distributions not only provide a package manager, but also additional tools used to search for packages and fetch them automatically from the repositories before build or installation. This functionality is usually separated from the package manager itself, so that different repository management tools can coexist using the package manager as an interface layer with its database of installed packages.

The list of available repositories is usually populated when the Linux distribution of choice is installed, while additional repositories can be configured later to provide third-party software not officially included in the distribution. The repositories are usually reached over the network and can provide both source and binary packages. The user might choose to customize a source package and rebuild it even though a prebuilt binary version might be already available for use.

Since the contents of the repositories may vary over time, as new packages are added or existing packages are updated, the repository management tools also usually provide a system update functionality.

## 2.1.2 The RedHat Package Manager

The RedHat Package Manager (RPM) system was born as a package manager for the RedHat Linux distribution, and was later adopted by several others including Fedora, OpenSUSE and MeeGo. It defines a custom file format for packages and provides a set of tools to manipulate them. More information about RPM and a complete guide for developers can be found at [15].

### The RPM package naming scheme

The packages follow a strict naming scheme, whose information is also included in the package file and which has the form *name-version-release.architecture.rpm*. The name components are defined as follows:

- *Name* — Identifies uniquely the package in the distribution namespace. It is often formed by a prefix, which identifies the included software, and a dash-separated suffix which specifies which part of the software is included. For example, a software library called *foo* might be split in the binary packages *foo*, *foo-devel* and *foo-static*.

- *Version* — It is the version of the included software, usually a sequence of numbers separated by dots. In some cases, patches that have not yet been included in an official release are attached to the package. Nevertheless the version field refers to the bundled software source package.

- *Revision* — It is a version number related to the package creation and revisions. For example, the OpenSUSE Build System generates it automatically as two numbers separated by a dot: the first is incremented every time the source package is modified, the second every time the source package is built.
- *Architecture* — Describes the target of the package contents. For source packages, the literal *src* is used. For architecture-independent packages, instead, the literal *noarch* is used. For all the other packages it is the name of the target architecture. As a side note, RPM does not support the creation of universal binary packages.

### The RPM file format

The file structure of an RPM package consists of three metadata sections (lead, signature, headers) and of a data section (archive). The headers were not part of the first version of the format and were later added to support the new features developed for RPM. From a functionality standpoint, they duplicate and integrate the information present in the lead, which nevertheless is still included in current RPM packages for backward compatibility reasons. More in detail:

- *Lead* — It is a fixed-length data structure placed at the start of the file. The first four bytes consist of a constant magic value which can be used to identify the file as an RPM package. The following fields indicate the file format version, the type of package (source or binary), the package architecture, the original file name, the operating system on which the package was created and finally the package signature presence and type.
- *Signature* — It may or may not be present and consist of a cryptographic hash of the headers and data section of the file, which allows to check the integrity[1] of the package. The hash may also be signed using a public key cryptographic algorithm, which makes also possible to check the authenticity[2] of the package.
- *Headers* — They define a sequence of key-value pairs, holding metadata related to the package. A predefined set of keys is available, but not all of them must be actually present in the headers. Each entry consists of a key ID, a type marker (which specifies how to interpret the value), the length of the value

[1] Integrity is intended as the fact that the contents of the package actually correspond to the attached hash. Verifying this property is considered to be enough to protect against accidental data corruption, but not enough to protect against an attacker maliciously modifying the package.

[2] Authenticity is intended as the fact that the hash was indeed applied by the package maintainer and not modified afterwards. If verified together with integrity and against a trusted copy of the public key of the source it is enough to prevent package modification attacks. More information can be found at [21].

and the value itself. The headers duplicate the information available in the file name and in the lead and integrate that information with more details, such as package author and maintainer name, software license, build and install dependencies, installed size, archive size and the source code of the support scripts. Most of these values come from the specification files written by the package maintainer, whose structure is described later.

- *Archive* — All the files shipped together with the package are compressed using the GNU zip routines and appended to the package file, resulting in the archive section.

### How to create RPM packages

To create a RPM package, the maintainer first collects the source archives of the software that will be packaged. It is recommended to include a verbatim source archive from the software distribution and add separate patches which might be needed in the build environment. Then the maintainer writes a RPM specification file (usually referred to as spec file), which contains the package metadata and all the support scripts. Starting from these items, the rpmbuild tool is able to compile the sources and generate the RPM packages.

The spec files are written in a special purpose language, while the included support scripts are written in Bourne shell. The RPM spec file interpreter provides a set of predefined macros that help the maintainers to write the spec file and support scripts. These macros start with a percent sign and are expanded to their contents when the spec file is interpreted. Examples of macros are *%arch*, which gives the current target architecture, *%patch*, which expands to the shell command to apply a patch, *%configure* which expands to the classical GNU configure command with some of the most important switches set. A regular spec file consists of the following sections:

- *Preamble* — Defines all the metadata associated to the package, including but not limited to: name, version, release, license, description, build and install dependencies. Moreover, the same metadata can be specified for an arbitrary number of subpackages.

- *Prep section* — Includes the shell instructions required to unpack the source archives of the software and apply any patch included with the source package.

- *Build section* — Includes the shell instructions required to configure and build the source package. On Linux in most cases this boils down to the *configure* and *make* commands.

```
Summary: A hello world program
Name: helloworld
Version: 1.0
Release: 1
License: GPL
Group: Development/Tools
Source0: %{name}-%{version}.tar.gz
Patch0: fix-helloworld-spelling.patch
Patch1: fix-exit-value.patch

%description
A hello world program used to show an example of RPM spec file.

%prep
%setup -q
%patch0 -p1
%patch1 -p1

%build
%configure
make

%install
rm -rf %{buildroot}
mkdir -p %{buildroot}
make install DESTDIR=%{buildroot}

%postin
echo "Thank you for installing Hello World on %{arch}!"

%files
%defattr(-,root,root,-)
%{_bindir}/hello

%changelog
* Thu May 15 2011 Ivan Bertona <ivan.bertona@gmail.com> 1.0-1
- First Build
```

Figure 2.1. An example RPM spec file

- *Install section* — Includes the shell instructions required to install the compiled software in a specific directory, whose path is available through the macro *%{buildroot}*, where the files to be packaged are later found by the build tool. Again, on Linux this action is usually performed with the command *make install DESTDIR="..."*.

- *Install and uninstall scripts* — Four optional maintenance shell scripts can be included. Such scripts are copied in the header of the generated binary packages and executed when one of the following event occurs: the package is going to be installed, the package has just been installed, the package is going to be erased and the package has just been erased.

- *File lists* — The list of files that should be included in the package. In case more than one package is generated, more lists are provided, defining how the package contents are to be split among the generated subpackages.

### 2.1.3 The OpenSUSE Build System

The OpenSUSE Build System (OBS) [20] is a complete build system which supports most of the build techniques outlined before and integrates with the RedHat Package Manager and the Debian package management tools. Each instance of the OBS can host an arbitrary number of projects which basically are collections of packages to be built and linked against each other.

For each project, the build dependencies might be resolved either by providing a bootstrap repository and then recompiling all the packages in the project or by referring to an external package repository. The first mode of operation is well suited to maintain an entire Linux distribution, while the second can be used by developers to build and test their applications against one or more existing Linux distributions. Moreover, the OBS provides an accounting system that allows to limit the capabilities of users and restrict their access to selected projects or even single packages.

#### Components of the OBS

The OBS consists of several services written in Perl and Bash and a web interface whose server-side component is written in Ruby. The main components are:

- *Web interface* — It consists of a web application, written in Ruby, which might be run under Apache or lighttpd and uses MySQL as a database backend. Through the web interface it is possible to create, edit and delete projects and packages. The administrators of the OBS can also manage the local users and the granted permissions.

- *API* — Allows third-party applications to remotely access the facilities provided by the OBS by sending XML messages over HTTP. A command line client, available for several Linux flavors, allows to interact with the OBS through the API.

- *Scheduler* — This service continuously scans the local project and identifies the packages that need to be built or rebuilt, creating build jobs that become pending.

- *Workers* — Several instances of this service can be run, also on different machines on the network. It performs the actual build jobs, either natively or in emulation.

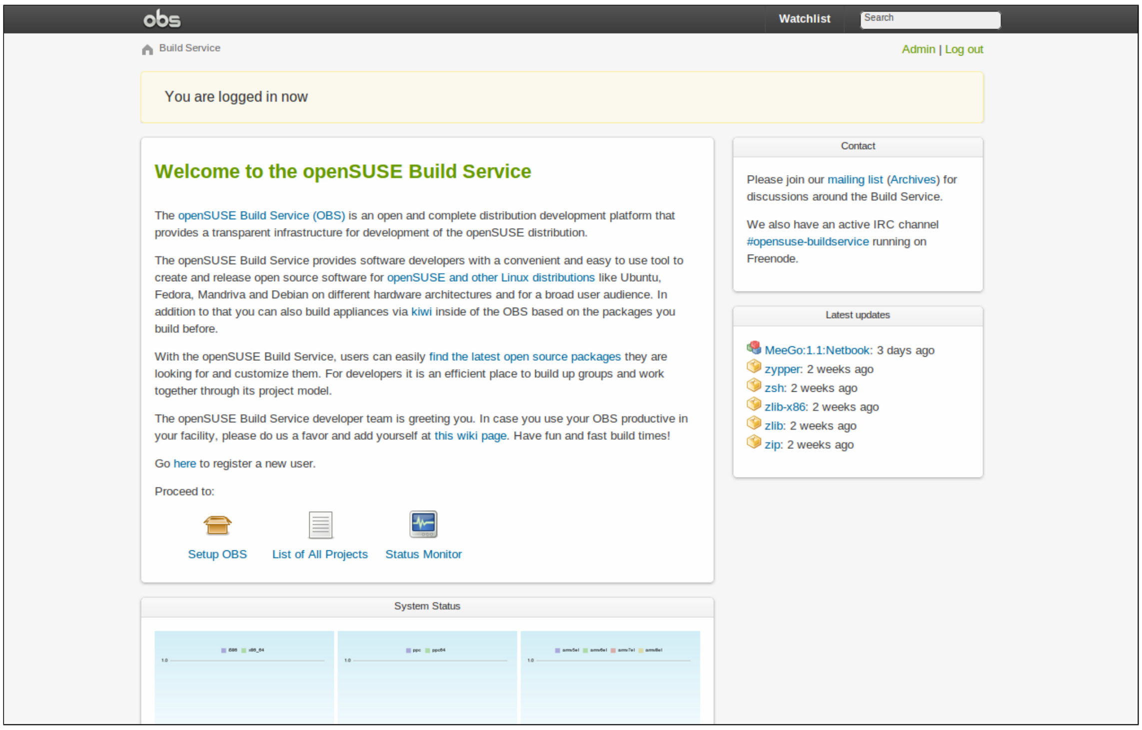


Figure 2.2. The OBS web interface

- *Dispatcher* — This service assigns the scheduled build jobs to the available workers according to a dispatching policy. Limited support for preferred build hosts and package priorities is also provided.

- *Warden* — This service continuously monitors the remote workers by pinging them. If a worker does not respond, the warden marks it as unavailable and forces the scheduler to reassign any lingering build job.

- *Publisher* — This service manages all the repositories generated by the OBS, collects and publishes the built packages in the right place. Moreover, it takes care of generating all the support metadata associated with the repositories and the repository specification files.

### Integration with QEMU

To accomplish all the tasks that require emulation capabilities, the OBS relies on the QEMU emulator. The QEMU emulator runs on x86 hardware and is able to emulate several architectures, as illustrated in table 2.1, even though not all of them with the same level of maturity. It supports both user mode and full system emulation, even though the OBS only uses it in user mode.

| | Full System | User Mode (Linux) |
|---|---|---|
| i386 | x | x |
| x86_64 | x | x |
| alpha | | x |
| arm | x | x |
| armeb | | x |
| cris | x | x |
| lm32 | x | |
| m68k | x | x |
| microblaze | x | x |
| microblazeel | x | x |
| mips | x | x |
| mipsel | x | x |
| mips64 | x | |
| mips64el | x | |
| ppc | x | x |
| ppcemb | x | |
| ppc64 | x | x |
| ppc64abi32 | | x |
| sh4 | x | x |
| sh4eb | x | x |
| sparc | x | x |
| sparc32plus | | x |
| sparc64 | x | x |

Table 2.1. Architectures emulated by QEMU

The user mode functionality of QEMU is enhanced by the binfmt_misc kernel module, which hooks itself at low level in the execution routines of the kernel. By default, the kernel will fail to execute any foreign architecture binary. The binfmt_misc module can be configured to intercept such event and force the kernel to execute a different native binary (in this case the QEMU emulator), passing as an argument the path to the foreign architecture binary so that it can be emulated in user mode.

**Supported building techniques**

The OBS supports most of the building techniques illustrated earlier and always executes them in a chroot environment. In particular, the OBS is able to execute native and cross vanilla builds, emulated native builds and emulated native builds with acceleration.

The emulated builds are executed in the same way as the native builds by implementing the transparent user mode emulation technique described above. The accelerated builds are implemented using a mixed chroot, where part of the software (e.g. package manager, compression utilities, compiler) is executed natively.

The following steps illustrate how the chroot environments are set up and the build jobs are executed, regardless which package manager is used:

1. An initial chroot environment is set up by installing a minimal set of packages under a specific directory. To do so, the package manager tools are invoked specifying a different root directory and disabling the execution of the support scripts. The requirement of the minimal chroot is to provide an environment capable of running the package manager. Each project in the OBS has a configuration file which includes the list of packages to be installed in the minimal chroot, while the packages are fetched from the repositories associated to the project.

2. The build dependencies of the package that is going to be built are resolved and a list of packages that need to be installed is stored in a file under the chroot. A script coming from the OBS is copied in the chroot as well.

3. The copied script is executed inside the chroot environment. It reads the list of needed packages and installs them through the package manager. All the foreign architecture binaries that might exist in the chroot are executed using the transparent user mode emulation technique outlined above.

4. Finally the script invokes the package manager to perform the build. The generated packages and logs are saved in a specific folder under the chroot, and can be later retrieved by the OBS.

### 2.1.4 The bootstrap problem

When targeting a novel, previously unsupported architecture, due to the lack of an existing binary repository, the build dependencies of almost all packages are not satisfied. Moreover, cycles usually exist in the build dependency graph of the distribution packages, making it difficult or even impossible to generate a satisfiable build list by the means of automated tools.

Once such list has been manually devised, there are two main methods to proceed with the build: natively compiling on compatible hardware or cross compiling. The viability and convenience of these two methods depend on the following environmental factors:

- *Availability of native hardware* — Some architectures, especially those oriented to embedded systems, might not be well suited to perform native builds. For example, they might not deliver enough performance or might not provide an adequate performance over cost ratio to make native builds practical.
- *Availability of a similar software target* — To perform native builds, a sufficiently similar software stack able to boot the target hardware is required. In case of the OBS, this software stack might be used as the bootstrap repository requiring the packages to have the same naming conventions of the target and similar versions of the software.
- *Cross-compile awareness of the build scripts* — The build scripts of some packages are written in such a way that the package cannot be cross compiled out of the box. For example, a build script might require a compiled binary to be executed on the build machine in order to generate headers, documentation or other kind of files. Additionally in some cases it might not be possible to correctly guess the capabilities of the target hardware at compile time.

| **Factors** | | | **Methods** | |
|---|---|---|---|---|
| Native Hardware | Similar Software | Cross Support | Native | Cross |
| Y | Y | N | Y | N |
| N | N | Y | N | Y |
| N | Y | Y | N | Y |
| Y | N | Y | N | Y |
| Y | Y | Y | Y | Y |
| Y | N | N | N | N |
| N | N | N | N | N |

Table 2.2. Bootstrap build methods

Table 2.2 illustrates which methods are viable depending on various notable real-world combinations of the above factors. The last two lines describe situations where none of the previously outlined build methods is directly employable.

These situations can be addressed either by modifying the package build scripts to add support for cross compilation or by indirectly solving the problems related to those scripts which do not cross compile. For this project indeed the latter solution was implemented, as described in section 2.2.2.

### 2.1.5 The ARM port and the SPARC design choices

The ARM port of MeeGo [2] was analyzed in order to gather useful information before initiating the SPARC porting effort. First, the bootstrap problem in the ARM case was heavily mitigated by the existence of very similar ARM port of Fedora. Several packages in the MeeGo distribution come indeed unchanged from their Fedora counterparts. The process is undocumented, but it is possible that the Fedora repositories could have been directly used to bootstrap the OBS. Unfortunately, the same is not true in the SPARC case. To create the bootstrap repository a temporary build environment was set up, as illustrated in 2.2.

As the first building attempts were performed, it became clear that a great number of MeeGo packages were not cross-compile friendly. At the same time, ARM hardware is subject to fast evolution and oriented to a low-power and low-performance market. For these reasons, to maintain the ARM port, the OBS is configured to work using the emulated native build mode with acceleration.

Given the nature of the available SPARC hardware, the same choice looked appropriated for the SPARC port. Some complications, though, emerged later in the implementation phase, due to the fact that the SPARC user mode support in QEMU is partial and not sufficient to run complex software. For example, the fork system call cannot be currently translated. Fixing QEMU was considered out of scope of the project work, also due to the time constraints, so an additional build method, based on full system emulation, was devised and implemented in the OBS.

## 2.2 The temporary build environment

This section describes the creation of the bootstrap repository used to initialize the OBS in order to build the SPARC MeeGo repositories. The repository had to be cross compiled from scratch due to the lack of an existing similar target.

The list of packages to build was first devised by observing the logs of the image creator for the official MeeGo 1.1 Core image [18]. The requirements related to build dependencies were relaxed by the fact that several tools could be executed natively in a chroot environment. For example, a package might have required the GNU make tool to be compiled. Such tool was not available in the SPARC bootstrap repository, but could be installed from the x86 repositories and directly used to perform the cross build.

An initial cross build environment was set up, but some problems emerged. Most of the build scripts turned out to be not friendly to cross compilation, requiring heavy modifications in order to build the bootstrap repository. The biggest complication was related to the required execution of the freshly compiled SPARC binaries, either for testing purposes or in order to generate header files and documentation. Instead of manually fixing every single issue, the execution of SPARC binaries was made possible with a workaround, resulting in a working general solution.

### 2.2.1 Initial cross build attempts

The hardware resources available for the project were few LEON test boards, some old SPARC workstations and several recent x86 machines. Given the higher performance of the x86 hardware, the first method of choice for building the bootstrap repository resulted to be cross compilation. The build machine was powered by a dual core AMD Phenom processor and 3GB of RAM. The installed operating system was Ubuntu 10.04 32-bit, since it was the most recent version supported by the MeeGo 1.1 SDK and tools. The following design choices were made:

1. *Chroot environment* — To minimize the differences between the build and the execution environment, the technique described in 2.1.1 was used. A chroot environment based on the official MeeGo 1.1 Netbook image was created and configured.

2. *Toolchain and sysroot* — To cross compile the packages, the official LEON toolchain was used. Such toolchain is based on the GNU Compiler Collection (GCC) version 4.4.2 and provides a minimal system root populated with the GNU Standard C Library (glibc) version 2.9.

3. *Automation scripts* — To reduce the amount of work required to build every single package, a set of scripts was implemented. The scripts allow to fetch the package sources, execute the builds, compare the contents of the built packages against the official MeeGo packages and manage a local repository.

#### Setup of the chroot environment

The first step consisted in adding the MeeGo repositories to Ubuntu and installing the required tools, in particular the MeeGo Image Creator (mic). Detailed instructions for all the supported operating systems can be found at [5].

To create the chroot environment, it was sufficient to unpack a MeeGo 1.1 Netbook image for the x86 architecture. The chroot environment was accessed using the *mic-chroot* command, which took care of bind mounting all the virtual filesystems necessary for the chroot environment to work properly (e.g. */proc* and */dev*).

```
root@host$ mkdir ~/meegochroot
root@host$ mic-chroot --unpack-only -s ~/meegochroot meego-netbook-ia32-1.1.img
root@host$ mic-chroot ~/meegochroot
root@meego$ zypper removerepo updates-core updates-netbook updates-non-oss
root@meego$ zypper refresh
root@meego$ zypper install man nano wget python perl rpm-build rpm-devel
    rpmdevtools
root@meego$ cd /opt
root@meego$ wget ftp://gaisler.com/gaisler.com/linux/linux-2.6/toolchains/sparc-
    linux-4.4.2/sparc-linux-ct-multilib-0.0.5.tar.bz2
root@meego$ tar -xvf sparc-linux-ct-multilib-0.0.5.tar.bz2
root@meego$ su - meego
meego@meego$ echo 'export PATH="/opt/sparc-linux-4.4.2-toolchains/multilib/bin:
    $PATH"' >> ~/.bashrc
meego@meego$ rpmdev-setuptree
```

Figure 2.3. Setup of a MeeGo chroot

The image came configured with two users: the *root* user, which was impersonated after invoking the *mic-chroot* command, and an unprivileged user called *meego*, which was used to perform the builds. In the chroot environment, it was possible to drop super user privileges using the *su - meego* command and regain them by typing *exit*.

The official LEON cross compile toolchain was installed by unpacking the distribution archive under the */opt* filesystem and permanently adding the its *bin* directory to the search path of the user *meego*.

The *rpmdev-setuptree* command created the directory structure used by the rpmbuild tool in the user home directory. In particular the following directories were created: */rpmbuild/BUILD*, */rpmbuild/BUILDROOT*, */rpmbuild/RPMS*, */rpmbuild/SOURCES*, */rpmbuild/SPECS* and */rpmbuild/SRPMS*. The input files for the command have to be placed in the *SPECS* and *SOURCES* directories, while the generated packages are saved under the *RPMS* and *SRPMS* directories.

### Configuration of RPM

MeeGo already provided custom settings for RPM through the package *meego-rpm-config*. Additionally, a platform-specific configuration file for the SPARC architecture had to be written. Most importantly, such file contained the platform-specific command line options to be passed to compiler, linker and configure scripts.

The first line in figure 2.4 specifies the compile flags to be passed to the compiler. The first three switches instruct GCC to generate code which includes the v8 multiply and divide instructions and targets a 32bit SPARC processor with hardware FPU. In LEON jargon this target is usually referred to as *hfleonv8*. The *–sysroot* switch forces GCC to refer all the inclusion paths, either for headers and libraries, to the path supplied as argument. The other lines in 2.4 specify the value of the *–host* option to be passed to any GNU configure script included in the source packages, to make it aware of the cross compilation scenario. The complete configuration file can be found in section C.1.1.

```
%optflags       -mcpu=v8 -m32 -mhard-float --sysroot=/home/meego/root
%_host_cpu      sparc
%_host_vendor   leon
%_host_os       linux
%_host          %{_host_cpu}-%{_host_vendor}-%{_host_os}-gnu
```

Figure 2.4. /usr/lib/rpm/platform/sparc-linux/macros (parts)

### The automation scripts

The build environment was enhanced with a set of scripts used to automate as much as possible the build tasks. The scripts were placed under the */home/meego/scripts* directory, which was added to the search path of the user *meego*. The first version of the script set is described here, while some additional functionality that was added later is described in 2.2.3.

| **Variable** | **Description** |
|---|---|
| *SM_VERSION* | MeeGo version to be built, used for example to select the correct source repository. |
| *SM_CACHE_MAXAGE* | Amount of seconds after which downloaded files such as source packages and repository indexes expire and have to be downloaded again. |
| *SM_EDITOR* | Text editor of choice. |
| *SM_PATH* | Location of the *rpmbuild* and *scripts* directories. |
| *SM_REPOSITORY* | Template of the URL to MeeGo source repositories. The SM_VERSION variable might be included in the value to select the correct repository. |
| *SM_RPM_TARGET* | Value of the *–target* switch to be passed to the rpmbuild tool. |

Table 2.3. Configuration variables for the build scripts

The scripts share a common set of utility functions defined in *include/functions.inc* and a certain amount of configuration variables, defined in *include/env.inc*, which are documented in table 2.3. The following commands are available:

- *buildclean* — Clears the rpmbuild directory tree and allows for a new build job to be prepared.

- *buildprepare "repository" "package" [local]* — Accepts the repository and package name (e.g. *core*, *tzdata*), downloads the source package from the MeeGo repository and unpacks it in the correct location so that it becomes ready to be built. If the *local* literal is specified, the source package is retrieved from the local repository of built packages, thus including any modification that was done in a previous build.

- *buildperform [prep/force]* — Builds the previously prepared package by invoking the rpmbuild tool with the correct switches. If the *prep* literal is specified, only the *%prep* stage in the spec file is executed. If the *force* literal is specified, the build is executed even if unresolved build dependencies are detected.

- *buildcheck "repository"* — Compares the built packages with the binary packages found in the specified repository (e.g. *core*) and notifies of any difference in the list of included files. Moreover, each binary file is checked to make sure it was indeed built for the SPARC target.

- *buildsave "repository"* — Saves the built packages in a local folder. The *repository* argument is usually the same value previously used to fetch the source package, but might be changed here for flexibility.

- *repoclean* — Deletes all the built packages saved in the local repository folder by the buildsave command.

- *rootpopulate* — Wipes the local system root directory (the one specified earlier in the *–sysroot* switch for GCC) and populates it using the initial system root shipped with the official LEON toolchain.

- *rootinstall "repository" "package"* — Installs the specified package from the local repository to the system root directory. This step is necessary to provide the include headers and libraries which might be needed by other packages in order to be built.

In figure 2.5 an example build session is shown. Eventually, an additional command *build "repository" "package"* was implemented to further reduce the verbosity by encapsulating all the shown commands in a single one. The sources of the referenced scripts can be found in section C.1.2.

```
meego@meego$ buildclean
meego@meego$ buildprepare core tzdata
meego@meego$ builperform
meego@meego$ buildcheck core
meego@meego$ buildsave core
meego@meego$ rootinstall core tzdata
```

Figure 2.5. Example usage of the automation scripts

### 2.2.2 A technique for remote execution

To overcome the limitations of the build scripts, a remote execution technique was devised and implemented. Two main design choices had to be made, concerning how and where the execution of the SPARC binaries had to take place. The implementation required to set up an available SPARC workstation and to modify the stock Ubuntu kernel and C library, as illustrated next.

#### How to modify and rebuild the Ubuntu kernel

Ubuntu provides a patched kernel which might differ from the mainline available on the official Linux kernel website. Moreover, Ubuntu provides a set of tools that simplify the retrieval of the correct sources and the installation of the new kernel. For this reason the recommended Ubuntu procedures were used, as described in [8].

Briefly, the required build tools were installed using the Ubuntu package manager. Then a copy of the sources of the kernel shipped with Ubuntu Lucid was fetched from its git repository and a new configuration file was created starting from the default one. The sources were duplicated, modified and a patch was generated with the *diff* command. The kernel was finally compiled and installed, and became automatically available in the system bootloader menu.

```
user@host$ sudo apt-get install fakeroot build-essential crash kexec-tools
    makedumpfile kernel-wedge build-dep linux git-core libncurses5 libncurses5-
    dev libelf-dev asciidoc binutils-dev
user@host$ git clone git://kernel.ubuntu.com/ubuntu/ubuntu-lucid.git
user@host$ cp -r ubuntu-lucid ubuntu-lucid.orig
user@host$ cp ubuntu-lucid/debian.master/config/i386/config.flavor.generic
    ubuntu-lucid/debian.master/config/i386/config.flavor.sparcmeego
...
user@host$ diff -Naur ubuntu-lucid.orig/fs/exec.c ubuntu-lucid/fs/exec.c >
    kernel-remote-execution.patch
user@host$ fakeroot ubuntu-lucid/debian/rules binary-sparcmeego
user@host$ sudo dpkg -i linux-headers-*-sparcmeego_*.deb linux-image-*-
    sparcmeego_*.deb
```

Figure 2.6. Rebuilding the Ubuntu Lucid kernel

### The execution hook mechanism

On Linux systems, binaries stored on the disk are executed via the *execve* system call, whose source code is located in the file *fs/exec.c* of the kernel source tree. When invoked on a foreign architecture binary, *execve* fails returning the *ENOEXEC* error code. This behavior was modified with a simple patch, as shown briefly in figure 2.7 and extensively in section C.1.3.

```
@@ -1359,6 +1401,7 @@
  struct files_struct *displaced;
  bool clear_in_exec;
  int retval;
+ char *remoteclient = "/home/meego/scripts/remoteclient";

  retval = unshare_files(&displaced);
  if (retval)
@@ -1379,6 +1422,11 @@
  clear_in_exec = retval;
  current->in_execve = 1;

+ if (bin_type_sparc(filename, (const char **)argv)) {
+   printk(KERN_DEBUG "remote execution activated\n");
+   filename = remoteclient;
+ }
+
  file = open_exec(filename);
  retval = PTR_ERR(file);
  if (IS_ERR(file))
```

Figure 2.7. Kernel patch to allow remote execution (part)

The *bin_type_sparc* function reads the ELF header of the binary and returns a non-zero value if the architecture is SPARC. In that case, the path on disk of the binary to be executed is replaced with the hardcoded path of a gateway program, which takes care of performing the remote execution.

### The remote execution gateway

The remote execution gateway script was designed to be as simple as possible. The main idea was to configure the execution target to replicate as closely as possible the build environment and to use the SSH protocol both to share files and execute commands remotely. The following details had to be taken care of:

- Configure SSH servers both on the host and the target. Generate RSA key pairs on both machines and add the public keys to the authorized hosts list on the other machine, so that no password will be asked upon connection[3].

[3]This setup refers to the OpenSSH public key authentication mode. More information can be found at [26].

- Create a user called *meego* on the target, with the same UID as the user *meego* on the host. On the target, mount under the same path the following directories of the host: */home/meego/rpmbuild*, */home/meego/root*.

The gateway was written in C, both for performance reasons and for convenience, and its source code can be found in section C.1.4. The gateway builds a string containing the command to be executed remotely, and passes it to the local SSH client. The command string is built as follows:

1. The SSH parameters required to connect to the target, such as user name, host name and port, are retrieved from the *SM_SSH_INTO_NATIVE* environment variable, which must be defined, and appended to the command string.
2. The current working directory is retrieved and the command to switch to it generated as *cd "..."*.
3. All the local environment variables except those specified in a predefined constant are dumped and each of them is exported to the remote target by appending the command *export VARIABLE_NAME="..."*.
4. The relative path to the SPARC binary (referred to the current working directory) and the provided parameters are quoted and appended to the command string. This is possible due to the fact that the kernel patch modifies the executable path but not the parameters, and that on Linux the first parameter is always the relative path to the invoked binary file.
5. Finally, the command string is completed with *exit "$?"* which forces the ssh client to return the exit code of the remotely invoked SPARC binary.

### Choice and configuration of the execution target

The remote execution gateway can work with any target, regardless its location and architecture, as long as it is reachable over the network and it accepts SSH connections. As a first attempt, the target was configured inside QEMU in full system emulation mode. Unfortunately the approach turned out to be impossible for two reasons:

1. The 32-bit SPARC QEMU was stable and complete enough to run a full Linux operating system, but the most recent compatible distribution, Debian Etch, was too old to be able to execute the SPARC binaries compiled against glibc version 2.9.

2. In theory, the 64-bit SPARC QEMU would have been able to execute a recent enough Linux distribution as well as the 32-bit SPARC binaries, but its development stage was still far from completion.

Consequently, the only choice left was to use real hardware. A spare SUN Ultra1 Creator workstation, featuring an UltraSPARC processor, was configured with Debian Lenny and successfully used as native remote target.

**Fixing the loader from glibc**

The GNU Standard C Library (glibc) provides the dynamic loader which loads at runtime the shared libraries needed by a program. The shared libraries are searched first in the paths specified by the environment variable *LD_LIBRARY_PATH*, then from the system library cache and finally from the standard paths */lib* and */usr/lib*.

Sometimes, in the cross compilation scenario, a dynamic library also used by a native build tool is built. Moreover the *LD_LIBRARY_PATH* variable might be set to include the build directory in order to run some tests. As a result, when the native tool is executed, the loader first finds the just built SPARC version of the library. The expected behavior would be to ignore it and proceed searching. Unfortunately, due to a glitch in the loader code, when the library and the build machine have different endianness (which is the case of x86 and SPARC) the execution is blocked and an error is returned.

The glitch was addressed by patching the 2.11.1 version of glibc shipped with Ubuntu, recompiling it and overwriting the stock dynamic loader, which is located at */lib/ld-2.11.1.so*. The complete patch can be found in section C.1.3.

### 2.2.3 Additional workarounds

While cross compiling some packages, additional workarounds had to be implemented to solve minor glitches. Since most of the packages rely on the GNU autotools and make for building, the workarounds proved to be reusable for packages showing the same problematic behavior. Consequently, the automation scripts described in 2.2.1 were integrated with a command used to enable or disable the available workarounds.

The command is *workaround "name" "action" [parameters]*. All workarounds support the actions *enable* and *disable*, with no parameters, while some of them also define other actions. Here is a description of the available workarounds:

- *compileflags* — Allows to specify additional flags that will be passed to the compiler. Once the workaround is enabled, the flags can be added using the action *add* and specifying them as parameters.

- *configurecache* — Allows to force the GNU configure scripts to load some variables from a cache file instead of attempting to evaluate them at runtime. This can be used in case some build parameters are not correctly detected (usually because they refer to the build machine instead of the target). The default cache file already contains the most common values, while more can be added with the action *add*.

- *configureflags* — Allows to specify additional flags that will be passed to the configure script. Once the workaround is enabled, the flags can be added using the action *add* and specifying them as parameters.

- *gccmasq* — The correct build tools (such as *gcc*, *g++*, *ar*, etc.) for the LEON target are prefixed with the literal *sparc-leon-linux-gnu-* to distinguish them from the local, native build tools[4]. Some build scripts ignore the RPM settings and do not prefix the tools, hence building x86 binaries. This workaround inserts some wrappers in the system path so that the correct tools are invoked in any case. The reason why this workaround is not always enabled is that some build scripts are indeed cross compilation aware and need both native and cross tools to work properly.

- *nocheck* — This workaround temporarily modifies the RPM configuration in order to disable the *%check* section of the build spec file.

- *nocross* — This workaround temporarily modifies the RPM configuration by removing the *–host* parameter for configure scritps, which consequently behave as the performed build was native. Often used in combination with the *gccmasq* workaround.

- *uname* — This workaround replaces the system *uname* command to return the information relative to the target. It is useful when the build scripts use *uname* to determine the characteristics of the target.

Finally, to complete the bootstrap repository, a custom package called *system-root* was created. Its contents were the system root shipped with the official LEON toolchain and a native SPARC version of GCC and Binutils. In order for the final build system to recognize its contents, the custom package was marked as provider (using the *Provides* clause in its spec file) of the following MeeGo packages: *glibc*, *glibc-common*, *glibc-devel*, *glibc-headers*, *glibc-static*, *glibc-utils*, *nscd*, *libstdc++*, *libstdc++-devel*, *kernel-headers*, *ldconfig*, *binutils*, *binutils-devel*, *gcc*, *gcc-c++*.

[4]This is the standard notation used for GNU-based cross build tools. The prefix is formed by four elements, which respectively represent target architecture, hardware vendor or subarchitecture, operating system and ABI.

## 2.3 The final build infrastructure

In order to build and maintain the full SPARC MeeGo repositories the OBS was set up and modified to target the SPARC architecture. The system was initialized using the bootstrap repository and the MeeGo source packages were imported and compiled automatically. Thanks to the source code history tracking features of the systems, all the patches applied to problematic packages were stored in the distribution history and are available for inspection through the web interface.

Due to a lack of functionality in the QEMU user mode emulation, the full system emulation mode had to be employed to perform the builds. As a consequence the mixed chroot acceleration approach could not be used and an alternative method based on the distributed building tool *distcc* was implemented. Finally, to further reduce the distribution rebuild time, a zero-configuration worker package was created and spread on several machines over the company network.

### 2.3.1 Setup and configuration of the OBS

An Intel machine equipped with a Core 2 Quad processor clocked at 3.00 GHz and 8 GB of RAM was dedicated to the build infrastructure. The machine was configured with OpenSUSE 11.4, the natural choice to set up an OBS instance, following the instructions available at [9].

#### Installing and configuring of the OBS

The OBS installation procedure is not fully automated. First the *Tools* repository referring to the installed OpenSUSE version has to be added to zypper. The complete OBS is available for installation through the packages *obs-server*, *obs-api* and *obs-worker*.

These packages also pull in MySQL, whose default settings are unsafe and can be fixed using the interactive tool *mysql_secure_installation*. Figure 2.8 shows the commands used to initialize the OBS databases.

```
root@host$ cd /srv/www/obs/api/
root@host$ RAILS_ENV="production" rake db:setup
root@host$ RAILS_ENV="production" rake db:migrate
root@host$ cd /srv/www/obs/webui/
root@host$ RAILS_ENV="production" rake db:setup
root@host$ RAILS_ENV="production" rake db:migrate
```

Figure 2.8. Initialization of the OBS MySQL databases

The OBS consists of several services, which all have to be started in order for the system to work. Figure 2.9 shows the startup procedure (the system can then be stopped by executing the same commands and replacing *start* with *stop*). Before starting the OBS services, though, several configuration files have to be modified as described next:

- */etc/lighttpd/lighttpd.conf*

  *include_shell "cat /etc/lighttpd/vhosts.d/*.conf"* — This line has to be uncommented to enable the virtual hosts functionality of lighttpd, used by the OBS web interface and API web applications.

- */etc/lighttpd/modules.conf*

  *include "conf.d/fastcgi.conf"* — This line has to be uncommented to enable the FastCGI interface used to invoke the Ruby interpreter and generate the dynamic web pages of the OBS.

- */etc/sysconfig/obs.server*

  *OBS_SCHEDULER_ARCHITECTURES="sparc"* — Makes sure that the scheduler only dispatches build jobs that target the SPARC architecture.

- */etc/sysconfig/obs.worker*

  *OBS_VM_TYPE="qemu"* — Forces the Virtual Machine mode of the OBS to use QEMU instead of Xen or KVM. This setting will also be used to enable the new full system emulation build method implemented for this project.

  *OBS_WORKER_INSTANCES="3* — Runs 3 workers at the same time. Can be changed depending on the available hardware resources.

  *OBS_WORKER_JOBS="2"* — Sets the number of parallel build jobs inside a worker to 2. Such value has been empirically selected by observing the behavior of the system.

  *OBS_VM_DISK_AUTOSETUP_ROOT_FILESIZE="6122"* — Sets the size of the system image file used to perform the builds to 6 GB.

  *OBS_INSTANCE_MEMORY="1024"* — Allocates 1 GB of RAM to every QEMU instance used to perform the builds.

- */srv/www/obs/api/config/database.yml* and *../webui/config/database.yml*

*username: root* — Sets the user name used to access the MySQL database.
*password: ****** — Sets the password used to access the MySQL database.

- */srv/www/obs/webui/config/environments/production.rb*

  *FRONTEND_HOST = "192.168.0.39"* — Should be changed to the IP address of the machine hosting the OBS server.

```
root@host$ rcmysql start
root@host$ rcobsrepserver start
root@host$ rcobssrcserver start
root@host$ rcobsscheduler start
root@host$ rcobsworker start
root@host$ rcobsdispatcher start
root@host$ rcobspublisher start
root@host$ rcobswarden start
root@host$ rclighttpd start
```

Figure 2.9. Startup of the OBS services

### Preparing QEMU

Several choices and arrangements had to be done in order to use the full system emulation capabilities of QEMU. The process can be summarized as follows:

- *Choice of the emulated hardware* — QEMU is able to emulate several workstations produced by Sun Microsystems. The SPARCstation 10 was chosen because quite reliable at the time of testing and able to support up to 1 GB of virtual RAM, which was considered to be necessary since compilation is a memory-consuming task. The switch *-M SS-10* was passed to QEMU in order to select the emulated machine.

- *Choice of the disk layout* — Since the OBS provided already some limited support for it, a disk layout with two single partition image files was used. The first image file was formatted in *ext3* and dedicated to the system image while the second was used as swap space. The path to the two image files was passed to QEMU using respectively the *-hda* and *-hdb* command line options.

- *Boot method and options* — QEMU is natively able to initialize the Linux kernel for improved flexibility, so no bootloader had to be installed in the system image file. The *-kernel* command line option was used to supply the path to a valid kernel *zImage* file. At boot, QEMU decompresses the image contents to a fixed location in RAM and sets the instruction pointer to that location. Moreover, using the *-append* command line option, it is possible to specify some options passed to the booted kernel. The most important employed options were:

  - *root=/dev/sda rw* — So that the kernel automatically mounts */dev/sda* as the file system root in read/write mode. Note that the device is *sda* and not *hda* since in recent kernels the IDE and SCSI subsystems were partially unified.
  - *ip=dhcp* — So that the kernel automatically attempts to configure all available network interfaces via the Dynamic Host Configuration Protocol (DHCP).
  - *elevator=noop* — So that the kernel does not reschedule I/O operations. The I/O scheduling policy is hence left only to the kernel running on the host machine, which is the one accessing the real hardware.
  - *console=ttyS0* — So that the kernel output is sent to a virtual serial interface and then redirected to the standard output stream of the QEMU process.

  Additionally, the *-m* option was passed to QEMU to set the amount of emulated RAM, the *-nographic* option was used to disable the optional emulated framebuffer and integrated VNC server and the *-no-reboot* option was used to make sure QEMU terminated in case of error, giving control back to the OBS.

- *Kernel version and build options* — Unfortunately, the QEMU team does not supply prebuilt Linux kernels suitable for the emulated hardware, nor a set of recommended options to be used. The latest available kernel version, 2.6.38, was then configured and cross compiled for the SPARC architecture using the LEON toolchain, whose output binaries are compatible with any SPARC v8 processor. The following notable settings were chosen:

  - *Regarding the cross toolchain* — The cross compiler tool prefix option was set to *sparc-leon-linux-gnu-*, in order for the LEON toolchain to be used.
  - *Regarding the loadable module support* — It was disabled to speed up the boot procedure. This setting makes sense only if the hardware where the kernel will be run is known at compile time and most of the subsystems are disabled, so the image does not grow exceedingly in size.
  - *Regarding the processor type and features* — The system timer frequency, which also regulates the interrupt handling, was set to the lowest possible value of 100 Hz. This setting was chosen due to the slowness of the

emulated machine and the non-interactive usage pattern. The symmetric multi-processing support was disabled because the emulated processor is single core.

– *Regarding the networking options* — Everything was disabled except Unix domain and PF_KEY sockets, TCP/IP networking and DHCP kernel-level autoconfiguration support.

– *Regarding the device drivers* — Everything was disabled except Loopback, RAM block, HID and GMII devices, the Real Time Clock and the specific drivers actually in use on the emulated system: 93CX6 EEPROM, Sun LANCE Ethernet interface and the */dev/openprom* virtual device.

– *Regarding the supported file systems* — The classic *ext2*, *ext3* and *ext4* file systems were selected as well as Dnotify, Inotify and kernel automounter support.

The kernel was compiled using the command *make ARCH=sparc* and the resulting *zImage* file was copied on the OBS machine under the path */opt/qemu-kernels/sparc-zImage*. A detailed list of the enabled kernel options can be found in section C.3.1.

### Preparing DISTCC

The DISTCC tool consists of a server and a client. The client runs as a wrapper to GCC and is able to offload part of the compilation processes to one or more servers, by transferring source and binary files over the network. In particular, the preprocessing phase is always executed on the build machine, while the already preprocessed files can be compiled somewhere else. This technique results in easier configuration, since the correct system headers do not have to be propagated on all the server machines, and good performance improvements, since most of the CPU time is anyway spent in the compilation phase.

Due to the use of full system emulation, the SPARC build jobs could not be accelerated using the same mixed chroot approach of the ARM port, so support for DISTCC was added to the OBS. OpenSUSE does not provide prebuilt DISTCC binaries, so DISTCC 2.18.3, the same version included in the *MeeGo 1.1 Core* repository, was compiled from source and installed on the OBS machine. The OBS chroot creation scripts were later modified to configure the DISTCC client to offload the build jobs to the host machine.

Figure 2.10 illustrates how to build DISTCC and run its server component. In this case the server is not executed as a daemon but on an interactive terminal, using the *–no-detach* option. The *–jobs* options specify how many concurrent compilation jobs to allow, while the *–allow* option specifies the IP addresses which are accepted as clients. Only the loopback address is specified since only the requests coming from the locally running instances of QEMU are supposed to be accepted.

```
user@host$ ./configure --prefix=/opt/distcc
user@host$ make
root@host$ make install
root@host$ /opt/distcc/bin/distccd --no-detach --jobs 4 --allow 127.0.0.1
```

Figure 2.10. Compilation and installation of DISTCC

### 2.3.2 Adding the necessary SPARC support

Most of the OBS code is written in Perl or Bash and supports the use of native build workers for a wide range of architectures out of the box, as long as a Linux distribution capable of running a recent Perl interpreter is available. In practice, the emulation functionality of the OBS is used only to target ARM systems.

If QEMU properly supported the SPARC architecture in user mode, adding it to the OBS would have been just a matter of changing some paths. Unfortunately, the only working emulator available for use was the full system QEMU, which required more complex changes to be performed, as described next.

#### Adding SPARC full system emulation to the OBS

The OBS was modified in several parts to accommodate the new SPARC build system. Patches against the OBS packages from the OpenSUSE 11.4 Tools repository can be found in section C.2.1. Briefly, the following changes were made:

- *Dispatching the SPARC build jobs* — The first change to be done regards the OBS dispatcher code. Inside the file */usr/lib/obs/server/bs_dispatch*, in fact, there is an hashtable which describes the target architectures that can be built by a worker running on a given architecture. The *sparc* literal has to be added on the lines related the *i586*, *i686* and *x86_64* workers, so that the dispatcher would assign SPARC jobs to them. The same change has to be performed in */usr/lib/obs/server/bs_worker*, so that the workers also become aware of their new SPARC build capabilities.

- *Fixes for the web interface* — The target repository selection page of the OBS web interface hides the SPARC target, making it impossible to select it for the builds. To fix this problem, the file */srv/www/obs/webui/config/options.yml* has to be changed by adding the literal *sparc* to the *visible_architectures* variable.

- *Adding full system emulation support* — The changes related to the emulation mode affect a tool called *build*, which is shipped with the OBS but can also be used independently. Such tool is able to create a chroot environment and build packages inside the chroot. The main idea behind the patches is to create the chroot environment inside a loop-mounted image file, boot it with QEMU and execute the build. Once the build is finished, the emulator shuts down and the built packages can be fetched from the image file. The files */usr/lib/build/build* and */usr/lib/build/init_buildsystem* were modified, most notably in the following ways:
  - The invocation of QEMU was changed to support full system emulation, by applying the settings described before.
  - The DISTCC client was configured by modifying some environment variables set at boot time. In particular, the path to the DISTCC wrappers (*/usr/lib/distcc/bin*) was added to the PATH variable, and the DISTCC_HOST variable was set to 10.0.2.2. Such IP address always maps to the host machine running QEMU.
  - Finally, a set of minor fixes was introduced to minimize the differences between the packages in the SPARC bootstrap repository and the packages in the existing MeeGo repositories. The changes mostly consist in harmless path fixes through symlinks and the introduction of the *–replacefiles* option in the RPM tool invocation.

**Initializing the SPARC MeeGo project**

Once the system was set up and ready to build, a new a project called *MeeGo* and a subproject called *MeeGo:1.1* were created through the web interface. Finally, a further subproject called *MeeGo:1.1:Core* was created and initialized following the steps:

1. The corresponding project configuration was fetched from the official MeeGo OBS instance, modified to include support for the SPARC target and added to the *MeeGo:1.1:Core* project from the *Project Config* tab of the web interface. The configuration lists the content of the minimal build chroot and also includes the project-related RPM settings such as compilation flags and support macros. It can be found in section C.2.2.

2. The output repository for the built SPARC packages was created from the *Repositories* tab of the web interface, by clicking on *Add*, then *pick one via advanced interface* and filling the displayed form with the following values: *Project: MeeGo:1.1:Core*, *Repository: standard*, *New name: standard*, *Architecture: sparc.*

3. All the packages from the bootstrap repository were copied to the OBS machine, under the path */srv/obs/build/MeeGo:1.1:Core/standard/sparc/:full*, making sure that they were owned by the user *obsrun*. The OBS was notified of this action by typing, as root, the command *obs_admin –rescan-repository Meego:1.1:Core standard sparc*.

4. The MeeGo 1.1 source packages from the *Core* repository were imported into the local OBS, using a small shell script written on purpose. The script downloaded the packages from the official repository and pushed them to the local OBS using the command line OBS client.

Other repositories, such as the *Netbook* repository, were imported following a slightly different procedure. The bootstrap repository was indeed not needed anymore, and the OBS had to be instructed to use the *Core* repository to satisfy the build dependencies and create the chroot environment for the builds. Steps 1 and 3 were skipped and step two was executed in the same way, hence selecting the *Core* repository instead of creating a new self-contained *Netbook* repository.

Finally, a repository called *Support* was created with the purpose of hosting additional packages and experimental builds specifically related to the SPARC ports. Currently, its contents are the *system-root* package described before, and the X.org video drivers for the machine emulated by QEMU.

### The porting workflow

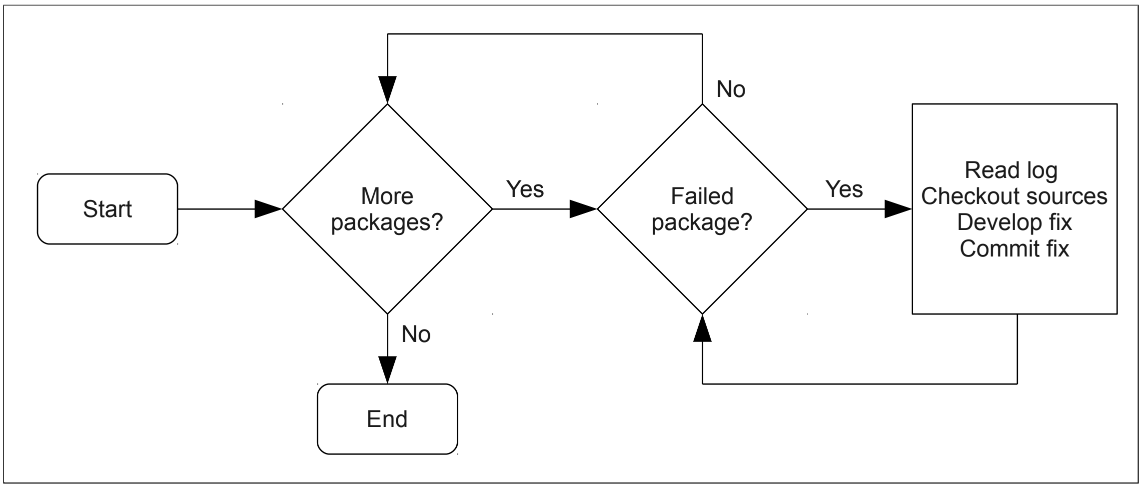


Figure 2.11. Porting workflow diagram

Once the *MeeGo:1.1:Core* project had been initialized and populated with the source packages, the OBS started to execute the build jobs. The build progress could be observed through the OBS web interface, which notified of failures. The OBS command line client was instead used to manipulate the package sources and apply the required fixes.

Figure 2.11 illustrates the porting workflow in the context of the OBS. Figure 2.12, instead, shows an example session where a package is fetched from the OBS, a patch is added and the changes are pushed back to the system.

```
user@host$ osc -Ahttp://192.168.0.39:81 co MeeGo:1.1:Core/package
user@host$ cd MeeGo:1.1:Core/package
...
user@host$ tar -xvf package-0.1.tar.gz
user@host$ cp -r package-0.1 package-0.1.orig
user@host$ nano package-0.1/src/file.c
user@host$ diff -Naur package-0.1.orig package-0.1 > sparc.patch
user@host$ nano package.spec
...
user@host$ osc -Ahttp://192.168.0.39:81 add sparc.patch
user@host$ osc -Ahttp://192.168.0.39:81 ci
```

Figure 2.12. Example porting session

### 2.3.3 A zero-configuration, flexible build cluster

Due to the emulation overhead and the lack of SMP support in QEMU, the build time of packages was too long for a complete system rebuild to be performed within the project deadline. Since the single package build time could not be reduced further, the system performance was improved by employing more hardware and exploiting package-level parallelism, hence attempting to build several packages concurrently.

The main idea was to create an easily deployable worker package and distribute it on different machines in the company network, possibly enabling it only for limited periods of time, for example at night. The main design requirements were ease of use and minimal external dependencies.

1. First, an OpenSUSE 11.4 chroot environment was created as shown in figure 2.13. The environment could then be accessed from any recent Linux distribution by bind mounting the required filesystems and typing the *chroot* command.

2. The environment was completed by installing the LEON toolchain, the DISTCC server, QEMU and the *obs-worker* package, following the same procedures described before. No patch had to be applied to the OBS code, since the modified *build* tool is downloaded automatically from the OBS server by the local worker service, every time a build job is started.

```
root@host$ mkdir /root/obs
root@host$ zypper --root /root/obs addrepo http://download.opensuse.org/
    distribution/11.4/repo/oss/suse/ repo-oss
root@host$ mkdir /root/obs/dev
root@host$ cp -a /dev/zero /root/obs/dev/
root@host$ zypper --root /root/obs install rpm zypper wget vim # this will pull
    in all the required dependencies
root@host$ mount -vt proc proc /root/obs/proc
root@host$ mount -vt sysfs sysfs /root/obs/sys
root@host$ mount -v -o bind /dev /root/obs/dev
root@host$ chroot /root/obs
...
```

Figure 2.13. Setup of an OpenSUSE chroot

3. An initialization script was written and saved under the path */root/obs*. It was designed to be run inside the chroot environment and start the DISTCC server and the OBS worker service.

4. Finally, an external startup script was added to be shipped together with the chroot directory. The script was able to adjust the chroot configuration files to the run environment, mount the required filesystems and run the internal initialization script.

The startup script and the chroot directory were compressed in a single archive for easy distribution. To run a new worker it was sufficient to unpack, edit the startup script by setting the correct OBS server IP address, and running it. When running the workers in a different environment, some configuration values might have to be changed in the startup script. Such values are documented in table 2.4, while the full sources are available in C.2.3.

| **Variable** | **Description** |
|---|---|
| *CFG_INSTANCES* | Number of worker instances to be run. |
| *CFG_INSTANCE_MEMORY* | Amount of memory to reserve for each instance. |
| *CFG_SERVER_IP* | Address of the main OBS server. |
| *CFG_SERVER_FQDN* | Fully qualified domain name of the main OBS server. Please note that this can be a simple hostname, not registered within a DNS, since the */etc/hosts* configuration file of the worker chroot is modified accordingly. |

Table 2.4. Configuration variables for the worker package

# Chapter 3

# System testing and images

This chapter describes the test hardware and setup, how the SPARC MeeGo images were assembled starting from the freshly compiled repositories and the results of the tests performed alongside.

## 3.1 The test environment

The tests were performed using the GR-LEON4-ITX board already introduced in section 1.2.3, which was connected to the local network and to a developer workstation as illustrated in figure 3.2. An initial bootable image of the SPARC MeeGo port was created manually, booted on the board and tested by incrementally adding new features of the system.

### 3.1.1 Test hardware, tools and configuration

A Linux-based developer workstation was used to host the system images and export them via NFS. The GRMON utility was used to remotely load a NFS-enabled kernel and to boot the board using the exported NFS share as root filesystem.

#### The developer workstation

Any recent Linux distribution could have been used as base for the developer workstation, as long as it was supported by GRMON. In particular, the same Ubuntu 10.04 machine previously employed to build the bootstrap repository was used. The following configuration steps were performed:

- *Connectivity* — The workstation was connected to the company network and the network parameters were configured automatically via DHCP. Moreover, the workstation was directly connected to the test board over a serial to USB interface. The tool *minicom* was used to display the output coming from the board, using the configuration illustrated in figure 3.1.

```
pu port           /dev/ttyUSB0
pu baudrate       38400
pu bits           8
pu parity         N
pu stopbits       1
pu rtscts         No
```

Figure 3.1. Connection configuration for minicom

- *NFS server* — It was configured to export an empty directory by adding a line in the configuration file */etc/exports*. The following NFS options were used:
  - *** — To allow access from any IP address on the company network.
  - *rw* — To export the directory in read/write mode.
  - *sync* — To make the server reply to requests only after all changes had been committed to stable storage[1].
  - *no_root_squash* — To disable the security feature which maps requests from the root UID/GID to the anonymous UID/GID. This was required to allow the right file and directory permissions to be set.
  - *insecure* — To allow requests whose source port number is higher than *IPPORT_RESERVED*[2].
- *SSH client* — It was installed in order to be used to login into the MeeGo images and execute commands.
- *grmon* — It was already configured and available for use from a NFS share on the company network. To execute it, two parameters had to be devised from the board documentation: the synchronization frequency and the static IP address of the hardware debug interface. For the GR-LEON4-ITX board in use the values were respectively *100* and *192.168.0.52*.

[1] The synchronous commit behavior is mandated by the NFS protocol, but the particular server implementation allowed a faster, albeit less safe, asynchronous commit mode.

[2] On UNIX-based systems only processes running with root privileges can make use port numbers lower or equal to *IPPORT_RESERVED*, whose value is usually 1024.

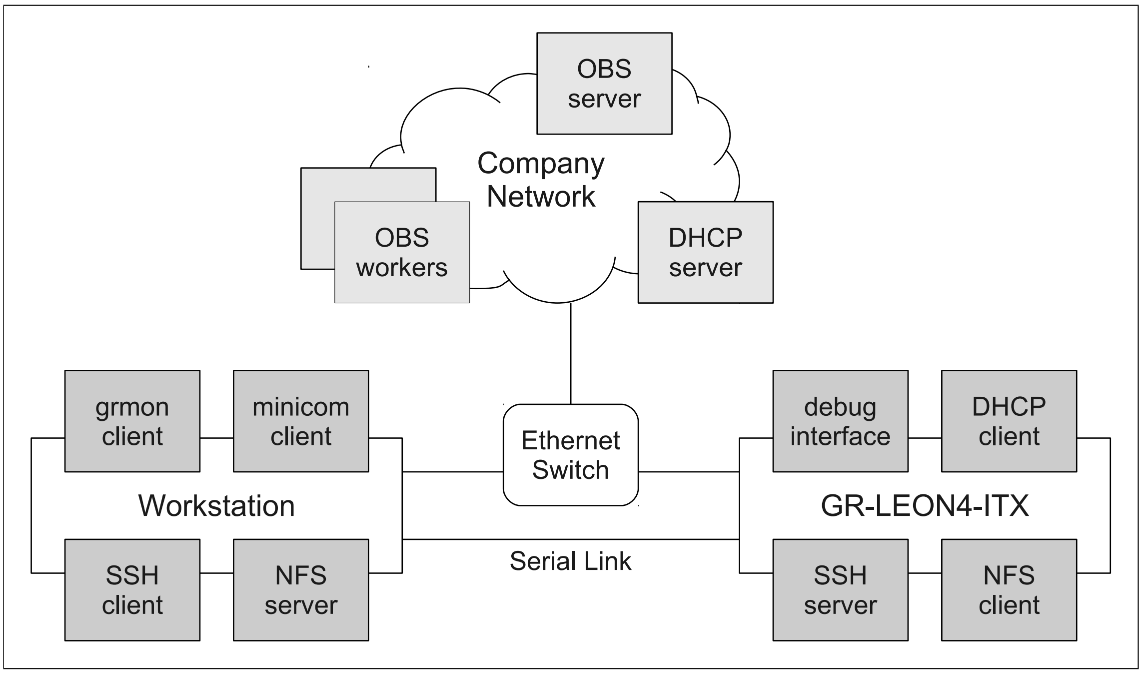


Figure 3.2. Test environment setup

### The test board kernel and boot method

Aeroflex Gaisler AB maintains a version of the Linux kernel compatible with LEON/-GRLIB hardware. Unfortunately, part of the required hardware drivers still had not been accepted into the kernel mainline, so a separate kernel repository was available for use inside the company network. The board kernel was obtained from the repository and compiled using the configuration shown in section C.3.2, and a binary image suitable for use with GRMON was created with the *mklinuximg* tool.

```
user@host$ git clone /usr/local/src/leon-linux-2.6 leon-linux-2.6
user@host$ cd leon-linux-2.6
user@host$ make ARCH=sparc
user@host$ mklinuximg arch/sparc/boot/image ../image.ram -base 0x40000000 -cmdline "video=grvga:800x600-16@60 console=ttyS0,38400 root=/dev/nfs nfsroot=192.168.0.103:/home/meego/SM/final-images/mp rw  ip=dhcp" -ethmac "00007ccc056d" -ipi 13
```

Figure 3.3. Creation of a Linux kernel image for the test board

Such tool accepts as input a kernel image and some configuration options which are included in the output, and generates a RAM image which also includes the code necessary to initialize the LEON processor and start the Linux kernel. The used image options were:

- *-base* — The starting address where the resulting image has to be loaded. This had to be specified since the image is not relocatable.
- *-cmdline* — The kernel command line arguments. Most notably, the *nfsroot* option was used to specity the location of the NFS export to be mounted as root filesystem.
- *-ethmac* — The MAC address of the board Ethernet interface through which the hardware debug functionality can be accessed.
- *-ipi* – The IRQ number used by the two available LEON cores to communicate.

The generated image is ready to be copied in RAM and executed from the first memory location. The GRMON tool was used to perform this operation, by issuing the *load* and *run* commands. At boot time, thanks to the integrated NFS client, the kernel mounted as root filesystem the system image available on the workstation.

### 3.1.2 Test workflow, methodology and criteria

The tests were performed by first booting an initial system image, incrementally adding packages and verifying that they worked. Several separated areas of functionality were identified and tested after installing the related packages, with the results shown in section 3.2. The tests were performed as follows:

1. A blank image file was created and its contents were formatted as an empty *ext3* partition, using the *dd* and *mkfs.ext3* commands.
2. The image file was loop mounted[3] on the workstation under the directory exported via NFS. This technique allowed to simplify the image management procedures and to allow to reuse the same kernel RAM image for different system images without having to move around consistent amounts of data[4].
3. A minimal bootable image able to run the package manager was created by invoking the Zypper and RPM tools with the *–root* option set to the NFS export path. The Zypper configuration had to be temporarily modified to force it to consider the SPARC packages, by adding the line *arch=sparc* in the configuration file */etc/zypp/zypp.conf*. The SPARC MeeGo repositories were added and the packages *rpm*, *zypper*, *wget* and *nano* were installed explicitly. Since Zypper resolves automatically the package dependencies, all the other needed packages were consequently installed. Figure 3.4 shows the issued commands in detail.

[3]The term loop mount refers to the fact that an image file on a filesystem is mounted under a directory of the same filesystem. The mount option *-o loop* has to be used.

[4]Due to the fact that the path to the NFS export is hardcoded in the RAM image.

```
user@host$ dd if=/dev/zero of="skel.img" bs=1MiB count="4092"
user@host$ mkfs.ext3 skel.img
user@host$ mkdir mountpoint
user@host$ su
root@host$ mount -oloop skel.img mountpoint
root@host$ rpm --root `realpath mountpoint` --initdb
root@host$ zypper --root `realpath mountpoint` addrepo http://192.168.0.39:82/
    MeeGo:/1.1:/Core/standard/MeeGo:1.1:Core.repo
root@host$ zypper --root `realpath mountpoint` addrepo http://192.168.0.39:82/
    MeeGo:/1.1:/Netbook/standard/MeeGo:1.1:Netbook.repo
root@host$ zypper --root `realpath mountpoint` addrepo http://192.168.0.39:82/
    MeeGo:/1.1:/Support/standard/MeeGo:1.1:Support.repo
root@host$ zypper --root `realpath mountpoint` install rpm zypper wget nano
root@host$ umount mountpoint
root@host$ exit
user@host$ cp skel.img minimal.img
```

Figure 3.4. Creation of the initial system image

4. The initial image contained some glitches caused by the fact that the pre and post installation scripts of the packages could not be executed, but it was enough to be booted over NFS, using GRMON as described above. To fix these glitches, the RPM database folder */var/lib/rpm* was deleted and the packages were reinstalled by invoking Zypper from the system shell prompt.

5. An iterative test procedure was started, as illustrated in figure 3.5. The packages related to an outlined area of functionality were installed, tested and fixed whenever it was necessary and possible. Once an area was tested, the procedure was repeated with the next one.

The MeeGo repositories contain many more packages than those actually used in the official stock MeeGo images. While allowing the users to empower their devices with ready-to-use software additions, this fact is probably related to the legacy of the traditional Linux distributions on which MeeGo is based on. To provide a figure, only the SPARC binary packages of the *Core* repository are more than 2800.

Testing everything was both impossible, due to the time constraints of the project, and pointless, since many packages were not interesting or even appropriate for use on a LEON/GRLIB system. For example, the sound server pulseaudio is superfluous because no audio output device is provided by GRLIB. The criteria used to choose whether to test or not a functionality area were two:

1. Is the functionality area required by or strictly related to MeeGo itself?

2. Is the functionality area interesting for LEON/GRLIB applications out of the scope of MeeGo?

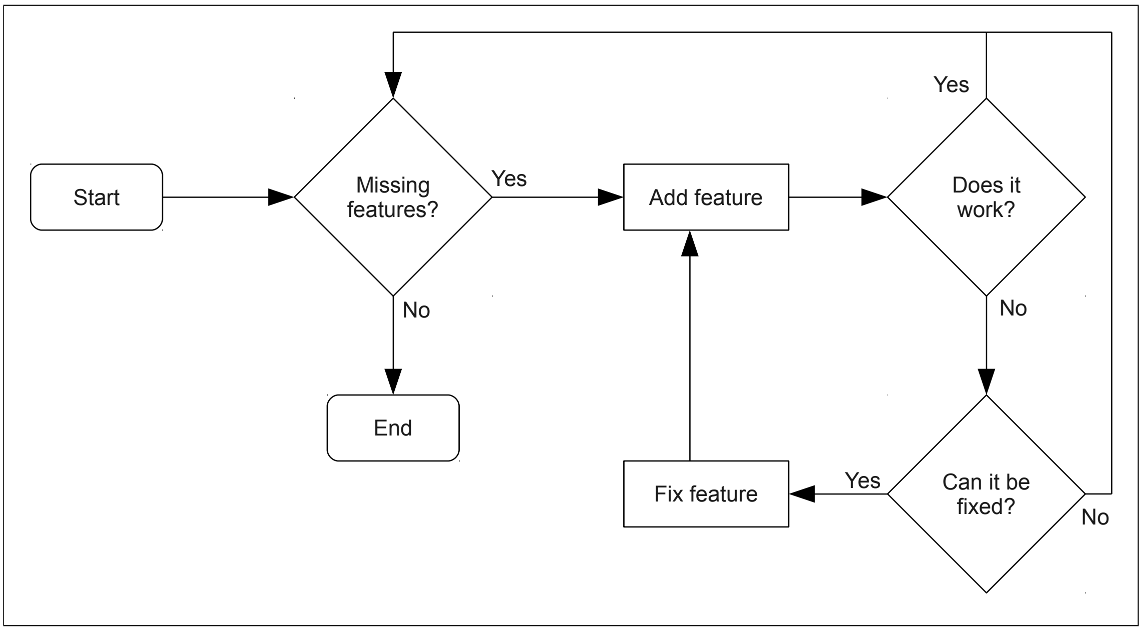


Figure 3.5. Test workflow

The second item is a consequence of the fact that, before the completion of this project, no recent Linux distribution was available for LEON/GRLIB. The SPARC MeeGo port contributed to fill this gap, hence becoming interesting also from a pure Linux functionality standpoint.

## 3.2 Functionality areas and test results

This section describes the functionality areas that were identified to group the testing of packages, and the results of the performed tests. Known issues and missing functionality are outlined as well. Several functionality areas are moreover grouped in macro areas, which correspond to the following subsections.

### 3.2.1 Basic system functionality

This macro area includes packages required to perform the boot sequence, implement the mandatory system services, allow the user to login, execute basic commands and remotely connect to the system.

### Boot sequence and startup scripts

It works. The file */etc/inittab* can be modified to set the default runlevel, where 3 is a non-graphical environment and 5 is a graphical multi-user environment. Since the root filesystem is mounted automatically by the kernel over NFS, it is not listed under */etc/fstab*. The startup script *rc.sysinit* prints a warning but the functionality is not affected. The available devices are correctly detected and exported to the */dev* filesystem by udev.

### Logging capabilities

The system logger daemon *rsyslog* is available. It works and it can be configured to start at boot using the *chkconfig* command, which works properly too. The daemon can be configured by modifying the file */etc/sysconfig/rsyslog*.

### Virtual terminals, text based login and session management

The boot sequence is correctly completed by spawning several virtual terminals, as defined in */etc/inittab*. The text-based login prompt works and can be accessed using the PS/2 keyboard and the screen connected to the board. The */etc/shadow* file had to be tweaked manually to set a known root password.

### Interactive shells

The repositories provide at least three different working shells: *bash*, *tcsh*, *zsh*. All of them are functional. The generic shell */bin/sh* is a symlink to *bash*. By default the system assumes an American English keyboard is connected.

### Basic UNIX system commands

The system commands provided by the GNU packages Coreutils and Findutils are available and working. These include filesystem management commands (such as *cp*, *mv*, *rm*, *chown*, *chmod*, etc.), text file manipulation commands (such as *base64*, *cat*, *expand*, *fmt*, *head*, *cut*, *sort*, *split*, *uniq*, etc.) and search commands such as *find*. It is moreover possible to change the user privileges through the *sudo* and *su* commands.

### Network configuration and utilities

The Ethernet interfaces available on the test board are working both in static and DHCP-based configurations. DNS resolution works. Configuration utilities such as *ifconfig* and *route* are available and working. Basic network utilities such as *ping*, *traceroute* and *nslookup* are available and working. The firewall subsystem *iptables* is available and functional.

#### Remote shell capabilities

The OpenSSH server and client are available and working. Both the server and the client can be used without any additional preparation besides installation and invocation. The first time the server is started, a new host keypair is generated. The procedure is CPU intensive and takes several minutes to be completed. As an alternative to OpenSSH, the lightweight SSH server dropbear is available and working.

#### Package management

The RPM and Zypper package management utilities are available and working. After installation Zypper does not have any repository configured. The SPARC MeeGo repositories can be added later using the commands *zypper addrepo* and *zypper ref*.

### 3.2.2 Text-based functionality and applications

This macro area includes packages which allow to perform advanced operations in a non-graphical environment.

#### Classical UNIX text manipulation tools

Most of the tools are available and working including *sed*, *grep*, *gawk*, *flex* and *bison*.

#### Data archiving and compression utilities

The following archiving and compression utilities were tested and resulted to be working properly: *zip*, *bzip2*, *gzip*, *xz* and *tar*.

#### Compilers and interpreters

Besides GCC, several other compilers and interpreters are available in the MeeGo repositories. The following were tested and resulted to be working properly: *perl*, *python*, *ruby*, *swi-prolog*, *lua*, *slang*, *tcl*, *tk* and *vala*. It is worth to note, though, that the test hardware did not deliver brilliant performance when interpreting code. Moreover, compilers usually require big amounts of memory which might force the system to swap, thus deteriorating performance.

### Documentation access utilities and data

The following utilities used to access system documentation are available and working: *man*, *info* and *help*. The documentation related to most system commands and configuration files can be installed through the package *man-pages*. If the latter is not installed, *man* fails with a misleading error message.

### Text editors

The following text-based text editors are available and working: *nano*, *emacs*, *vim* and *ed*.

## 3.2.3 Basic graphic functionality

This macro area includes packages required to provide minimal graphic functionality and hardware support for the GRLIB framebuffer and for common input devices.

### X.org graphic server

It works with the default configuration, using the generic X framebuffer driver *fbdev*. The supported screen refresh frequency is 60 Hz, while the supported screen resolutions and color depths are shown in table 3.1. The resolution of 1024x768 with 16-bit color depth works but is subject to screen flickering problems, as the system bus bandwidth is barely enough to transfer the graphic data. When the bus is heavily used also by other devices such as the Ethernet interface, the video data transfers might be delayed causing a loss of synchronization with the screen.

| | **8-bit** | **16-bit** | **24-bit** |
|---|---|---|---|
| **640x480** | x | x | x |
| **800x600** | x | x | |
| **1024x768** | x | x (partly) | |

Table 3.1. Supported screen resolutions and color depths

### Basic X-based graphic utilities

The basic window manager *twm* works allowing the setup of a bare-bones X-based desktop environment. The default utilities *xterm* and *xclock* work as well.

It is worth to note that the time displayed by *xclock* is not correct since the test board does not provide a system clock. At boot, the system assumes the conventional date of the 1st of January 1970 at 00:00[5].

### 3.2.4 Traditional Linux desktop functionality

This macro area includes packages required to setup a traditional Linux-based desktop system as well as graphical application for office automation, document viewing, image manipulation and similar.

#### Gnome desktop environment

Several components of Gnome infrastructure are included in the MeeGo repositories, but not enough to setup the complete desktop environment. Nevertheless both the system messaging bus *dbus* and the system configuration daemon *gconf* work properly. The traditional Gnome application libraries such as *glib* and *GTK* are available and working. These components are included since the MeeGo user interface is based also on parts of Gnome 3.

#### Xfce desktop environment

The full Xfce desktop environment is included in the MeeGo repositories, even though the reason is unclear. Anyway it was considered interesting and hence installed. In particular the following components were tested:

- *Window manager (xfwm4)* — OK.
- *Compositing features of the window manager* — OK.
- *Desktop manager (xfdestkop)* — OK.
- *Configuration daemon and access client* — OK.
- *Panel, application menu and common panel widgets* — OK.
- *File manager (thunar)* — OK.
- *Terminal emulator (terminal)* — OK.
- *Text editor (mousepad)* — OK.
- *System settings management dialogs* — OK.

[5] This date is usually referred to as the UNIX epoch, and is used to define UNIX timestamps as the number of seconds elapsed since then.

At this point, though, some concerns were raised about the actual capability of the hardware to run a fully fledged Linux system, especially in the area of graphics. The framebuffer device included in GRLIB, in fact, does not provide any hardware video acceleration, and all the rendering work has consequently to be done in software. This adds heavy load on the CPUs and results in a poor user experience, primarily characterized by a lack of responsiveness of the interface.

#### GTK-based graphic applications

The MeeGo repositories include a full collection of GTK-based applications that can run without problems within the Xfce desktop environment. Several of them were installed, tested and proved to be working. Some notable applications in the areas of text editing, document and image visualization, office automation and network transfer are named here: *gedit*, *evince*, *eog*, *abiword*, *gnumeric* and *transmission*.

#### Web browsing

Unfortunately, no web browser could be built for the SPARC architecture (in the repositories are available both Chromium and Fennec, a Firefox derivative for mobile devices). All the major open source browsers, indeed, provide Just-in-Time javascript compilers which are required by the browser itself, but are not able to generate SPARC code.

### 3.2.5 MeeGo-specific functionality

This macro area includes packages that were explicitly designed and implemented for MeeGo systems.

#### MeeGo desktop manager and login manager

Due to its intended usage patterns MeeGo does no provide a prompt login manager, rather providing a desktop manager which initiates a session for a predefined user. The MeeGo desktop manager is spawned directly by *init* and is able to initialize both Xfce and the MeeGo interface launcher *uxlaunch*.

#### User interface initialization

The MeeGo desktop manager launches the interface launcher *uxlaunch*, which in turn is able to initialize all the different flavors of user interface available in MeeGo. Since the only MeeGo flavor encompassed by the port is the Netbook flavor, the tool was tested only with such user interface. It worked. The behavior of *uxlaunch* can be modified by editing the configuration file */etc/sysconfig/uxlaunch*.

### Mutter-based window manager

The MeeGo Netbook interfaces relies on a modified version of the Mutter window manager, which was developed originally for the Gnome 3 project. Mutter is a compositing-only window manager, which uses the Clutter library for graphic rendering. Clutter works on top of the OpenGL graphic libraries (optionally on the OpenGL ES subset for embedded applications).

The use of Clutter in combination with a software rasterizer had been reportedly problematic, for two main reasons:

1. Software rasterizers (included the one shipped with Mesa and used in the SPARC MeeGo setup) tend to be simple and to support a limited number of OpenGL extensions. In some cases, consequently, it might be impossible to render correctly Clutter-based applications.

2. Performance. Rendering complex scenes on a general purpose CPU is usually too slow to be practical for interactive use. This fact also explains why nor the software rasterizer developers nor the Clutter developer actually put some effort in enabling such combination to work in all cases.

The window manager relies only on supported extensions and hence works, albeit being extremely slow (in the order of 0.5 - 2 frames per second, depending on the scene). The hardware improvements required to deliver an acceptable user experience are discussed in detail in 4.2.1.

### Netbook user interface

The MeeGo Netbook user interface can be launched as well, and the top menu bar is shown on the screen. The icon animations work (e.g. the magnifying mouse-over effects), but they are to slow to be actually connected to the movements of the mouse.

### MeeGo-based application panels

The MeeGo Netbook application panels do not work. This is caused by a combination of factors, namely the use of the Clutter MX widget library and the software rasterizer. Two alternatives were considered to fix the problem:

- *Modify the software* — The software rasterizer could be modified to support all the required GL extensions. This solution, though, would have been pointless since it would not have addressed the performance problems related to software rendering.

- *Modify the hardware* — This was considered out of the project scope and, besides, it would have been enough work for a second thesis (or even more). Nevertheless, this solution was considered from a theoretical point of view and the necessary design parameters were outlined, as explained in 4.2.1.

## 3.3 Prebuilt system images

During the testing procedure, while the system was being assembled, it was considered interesting to create a series of preconfigured system images that could be used for platform evaluation and as starting point for Linux-based LEON/GRLIB projects. The images could be easily created by copying the testing image at a given point, somehow freezing its status and making it available for later use. This section describes the contents of the images and provides some documentation on how to use them.

### 3.3.1 Configuration and security considerations

The following list describes some important details about the predefined configuration of the images and, where relevant, describes how to modify it:

- *Users* — All the images are preconfigured with an administrative user *root* and an unprivileged user *meego*, which use by default the Bash shell. New users can be created using the classical UNIX commands.

- *Init* — The default runlevel is set to 3 on text-only images and to 5 on graphic images. This setting can be changed by editing the file */etc/inittab*.

- *Default user* — The default user for graphic sessions is *meego*. This setting can be changed by editing the file */etc/sysconfig/uxlaunch*.

- *SSH Server* — Some images provide a preinstalled SSH server, which can be started by typing the command */etc/init.d/sshd start*. To have it started automatically at boot, the command *chkconfig –add sshd* shall be executed once.

- *Repositories* — The SPARC MeeGo repositories are preconfigured in Zypper as reachable at the IP address 192.168.0.39, which was the address of the repository server when the images were created (no domain name had been allocated for that). If the repository server is moved, the preconfigured repositories have to be disabled and the correct ones have to be added using the command *zypper addrepo*. In future, in case new versions of the SPARC

MeeGo packages will be released, it will be possible to update the image with the command *zypper update*.

When using the images it is important to consider them as experimental software, which is not in any way production ready. MeeGo itself is still under heavy development and is not particularly focused on security issues. Moreover, when using the image, two security problems have to be addressed as soon as possible:

- *Default passwords* — The default user passwords have to be changed. Moreover, the *meego* user is enabled to escalate its privileges using the *sudo* command. This behavior can be disabled using the *visudo* command and commenting the line related to the user *meego*.
- *SSH keys* — The images that are shipped with the SSH server contain an already generated host RSA keypair. Such keypair should never be trusted, and a new one should be generated using the *ssh-keygen* command.

### 3.3.2 Description of the available images

The detailed contents of the prebuilt system images are available in appendix B.

#### minimal

Contains a really small set of packages that allow to boot the system, login on a Bash shell and use the package manager to install more software.

#### xorg

Contains a minimal X server which can be started by typing the command *startx* at prompt. The window manager *twm* is used by default.

#### xfce

Contains the Xfce desktop environment, which is automatically started at boot, and basic graphical utilities such as a file manager, a terminal emulator and a text editor.

#### netbook

Is the equivalent of the official MeeGo Netbook images, and allows to start the Netbook interface. Unfortunately, this image is only partially functional, as described in section 3.2.5.

# Chapter 4

# Conclusions

This chapter describes the outcome of the project and the possible future developments. The current situation is compared to the initial objectives.

## 4.1 The final result compared to initial objectives

This section provides an analysis of the project results from several points of view. Moreover, some personal considerations about the project and work experience are included.

### 4.1.1 Platform evaluation standpoint

Despite the project being named *Porting MeeGo to LEON*, one of its main objectives was indeed testing the limits of the LEON/GRLIB platform. From this point of view, the project can be considered successful, as it clearly outlined where the hardware improvements are most needed. Such improvements might be grouped in two main areas:

- *Hardware manufacturing* — The test board used to test the port, despite being one of the fastest available, was still too slow to provide a marketable user experience. This problem, though, does not regard the actual platform design, which can scale up to 16 cores clocked at 1.5 GHz, but rather the available hardware.

- *Hardware design* — Two main problems were outlined during the test phase. The first, and most important, is the lack of hardware video acceleration. Software rasterization is actually not supported by MeeGo and moreover cannot provide a smooth and satisfying user experience. The second problem is the lack of audio capabilities, which does not strictly impair the use of MeeGo but greatly limits its functionality.

On the other hand, the LEON/GRLIB platform has also proven to be very flexible and reliable. Since it is compatible with the preexisting SPARC ABI, which is already supported by several critical components of MeeGo, the effort required to port the software was greatly reduced. Additionally, the available hardware manufacturing options do not restrict the implementation to a specific form factor or performance level, which in theory allows the creation of any kind of computing device available currently, and probably also in future.

### 4.1.2 Software availability standpoint

As a result of the port effort, a vast amount of the available GNU/Linux userland as well as the MeeGo APIs and user interface elements were built for the SPARC v8 architecture. Parts of it still explicitly retained support for such architecture, while most of the software still worked because it used to support the architecture in previous versions or just because it was written properly, in a portable way. Finally, a small part of components did not support the architecture and needed substantial effort in order to be ported, hence it was excluded by the project.

As all the ported software can now be tested on the LEON/GRLIB platform and on other SPARC-based systems, it is possible to say that the project was successful. On the other hand, two critical issues still have to be addressed:

- *Thorough testing* — The project objectives and time deadline did not allow to thoroughly test every single software component. Moreover, some parts of MeeGo could not be tested due to a lack of hardware support, as already described in section 3.2. Surely, the manifestation of architecture-specific bugs is possible, especially due to the known endianness and storage size problems related with the porting of C code.

- *Maintenance* — As said, no recent Linux userland was available before the completion of this project. On the other hand, in the meantime, a new version of MeeGo was released. Without maintenance the SPARC MeeGo repositories are going to become outdated soon.

### 4.1.3 MeeGo standpoint

While most of the core system components were ported and working, it was possible to build all the MeeGo-specific components but not to produce an usable MeeGo

Netbook system, due to the hardware support problems outlined before. As a consequence, from this point of view, it is possible to say that the project was only partly successful.

To integrate what was done up to now, section 4.2 describes the actions that have to be performed to provide a working SPARC MeeGo, as well as possible future implementation scenarios.

## 4.2 Future developments

This section describes the next steps that might be taken to provide a fully fledged MeeGo experience on the LEON/GRLIB platform. While none of the following could be implemented within the time span of this project, it was interesting to anticipate some of the engineering decisions that might be taken in future.

### 4.2.1 Hardware support enhancements

The ported MeeGo was tested on a generic test board. In order to develop a MeeGo-based product a dedicated board would probably have to be designed, depending on the product requirements, and the following shortcomings should definitely be addressed:

- *System bus* — During the testing phase, when rendering frames at 1024x768 pixels resolution and 24-bit color depth, some annoying flickering effect affected the screen output. This problem was caused by the saturation of the bus which introduced some delay in the transmission of the frames and resulted in a loss of synchronization with the screen. The easiest solution would be to increase the memory bandwidth by using Double Data Rate (DDR) revision two controller and memory banks instead of the DDR revision one components used on the test board. Such controller is already available in GRLIB.
- *Video acceleration* — Given the embedded nature of the LEON/GRLIB platform and the MeeGo support for it, it is recommended to implement hardware OpenGL ES 2.0 acceleration and all the related Linux kernel and X server drivers. Given the size of the task, it might also be considered the possibility to license an IP core from a third-party.
- *Audio playback/recording* — The GRLIB core implementing an AC97-compatible controller should be completed and integrated in the GRLIB platform. Linux drivers should be written or modified as necessary. Even though not mandatory to run MeeGo, audio playback and recording are important features that any user would expect from the hardware.

- *Clock* — Any board designed to run MeeGo should include a real-world clock, in order to properly support the calendar and schedule functions of MeeGo.

- *Wireless Connectivity* — Bluetooth and WiFi connectivity features, which are also standard on current mobile devices, might be developed as IP cores or added as USB or PCI third-party expansions. The former case would require more internal work but lower final hardware costs, while the latter would result exactly in the opposite situation.

### 4.2.2 Software infrastructure enhancements

The modified infrastructure used for the project was enough to produce build a fixed version of SPARC MeeGo. In future, it might be necessary to modify it in order to better integrate the SPARC porting effort with the official MeeGo distribution. The following issues should be considered:

- *QEMU* — The QEMU project is a vital part of the MeeGo infrastructure, and it is used in several ways to maintain the ARM port: to build software in a mixed chroot, to create system images and to test applications by emulating a generic MeeGo device. As illustrated before, the SPARC support of QEMU is partial and should be improved in order to allow the SPARC port to be maintained more similarly to the ARM port.

- *Native workers* — The builds executed in full system emulation are quite slow, even when DISTCC is used to accelerate them. As an alternative to fixing the QEMU SPARC user mode emulation, some fast SPARC hardware should be acquired and used to perform the builds natively *This is now possible using the current SPARC MeeGo to run the build hosts.*.

- *Upstream synchronization* — This project focused on porting the stable MeeGo 1.1 version. In future it might be important to synchronize the SPARC MeeGo releases with the official MeeGo releases. A possible way to implement this would be contacting the MeeGo development team and obtaining developer access to the official OBS. Then it would be possible to connect a local OBS instance to the official OBS and forward automatically all the changes performed upstream to the local system.

### 4.2.3 MeeGo SDK and tools enhancements

To complete the SPARC MeeGo port, modified versions of the tools and SDK should be created, in order to support the SPARC target. In particularly, the following elements were inspected:

- *MeeGo Image Creator* — It is a tool written in Python, which allows to created custom MeeGo images starting from a repository and an image definition file, which describes the image contents as well as some configuration variants. The tool was inspected it was determined that it would easily generate SPARC images if a working user mode SPARC QEMU existed.

- *SDK* — The MeeGo SDK provides an environment to develop, build and test MeeGo applications. Modifying it to target SPARC would require to integrate a suitable cross compile toolchain and a full system SPARC QEMU. All the other components might be left unchanged.

# Appendix A

# Package details

This appendix describes in detail the port status, including information about the packages that were rebuilt with no modification, the packages that were modified and the packages that were excluded from the port.

## A.1 Unmodified packages

The following packages were successfully built for the SPARC architecture without any modification. This usually implies that the package supports SPARC and the spec file was written properly.

### A.1.1 Core repository

abrt
acct
acl
acpid
adns
alsa-lib
alsa-plugins
alsa-utils
anthy
apr
apr-util
aria2
asciidoc
asio
aspell
aspell-en
at
at-spi
atk
attr
audiofile
audiomanager
augeas
authconfig
autoconf
autoconf213
automake
automake14
automake17
automoc4
autotrace
avahi
baekmuk-ttf-fonts
basesystem
bash
bc
bind
bison
bitmap
bitstream-vera-fonts
bluez
bognor-regis
bootchart
booty
btrfs-progs
build
build-compare
busybox
buteo-mtp
buteo-sync-plugins
buteo-syncfw
buteo-syncml
byacc
bzip2
c-ares
ca-certificates
cabextract
cairomm
catdoc
ccache
ccss
cdrkit
check
chkconfig
chrpath
cjkuni-fonts

clucene
clutter
clutter-box2d
clutter-gesture
clutter-gst
clutter-gtk
clutter-imcontext
clutter-qt
cmake
cmake-gui
compface
comps-extras
connman
ConsoleKit
contactsd
contextkit
contextkit-maemo
coreutils
corewatcher
cpio
cppunit
createrepo
cronie
crontabs
cryptsetup-luks
cscope
ctags
cupscupsddk
curl
cvs
cyrus-sasl
d-feet
dblatex
dbus-c++
dbus-glib
dbus-python
dejagnu
dejavu-fonts
deltarpm
desktop-backgrounds
desktop-file-utils
devhelp
device-mapper
device-mapper-multipath
dhcp
dhcpv6
dialog
diffstat
diffutils
distcc
djvulibre
dmidecode
dnsmasq
docbook-dtds
docbook-style-dsssl
docbook-style-xsl
docbook-utils
dos2unix
dosfstools
doxygen
driconf
droid-fonts
dropbear
dsme
dvd+rw-tools
dvipdfm
dvipdfmx
dvipng
eat
ecryptfs-utils
ed
eggdbus
eject
elfutils
emacs
enchant
enscript
eom
epydoc
esound
etherboot
ethtool
evolution-data-server
exempi
exiv2
exo
expat
expect
fakeroot
farsight2
fastinit
fastjar
fdupes
fennec-qt-branding-meego
file
filesystem
findutils
firstboot
flac
flex
fontconfig
fontforge
fontpackages
foomatic
foomatic-db
fprintd
freeglut
freetype
fribidi
fslint
fuse
fuse-sshfs
fvkbd
gamin
gammu
garage-client-services
gawk
gc
GConf-dbus
gd
gdbm
generic-backgrounds
generic-logos
geoclue
gettext
ghostscript
ghostscript-fonts
giflib
gir-repository
git
glade3
glew
glib2
glibmm
gmime
gnet2
gnome-common
gnome-disk-utility
gnome-doc-utils
gnome-icon-theme
gnome-js-common
gnome-keyring
gnome-mime-data
gnome-python2
gnome-vfs2
gnupg
gnutls
gobject-introspection
gperf
gpgme
gpsbabel
gpsd
gpsdrive
graphviz
grep
groff
grubby
gsl
gsm
gssdp
gst-plugins-bad-free
gst-plugins-base
gst-plugins-farsight
gst-plugins-good
gstreamer
gstreamer-python
gtk-doc
gtk-nodoka-engine
gtk-xfce-engine
gtk2
gtk2-engines
gtkglext
gtkmm
gtkspell
guile
gupnp
gupnp-av
gupnp-igd
gupnp-ui
gvfs
gwenhywfar
gypsy
gzip
hardlink
help2man
hicolor-icon-theme

html2ps
hunspell
hunspell-en
hwdata
i2c-tools
icon-naming-utils
image-configs
image-manager
ImageMagick
imake
indent
inotify-tools
installer
installer-shell
intltool
iptables
iputils
iso-codes
isomd5sum
jadetex
jana
jasper
joe
json-glib
kasumi
kbd
kcalcore
keyutils
krb5
ladspa
latencytop
latex2html
lcms
less
libaccounts-glib
libaccounts-qt
libao
libarchive
libart_lgpl
libassuan
libasyncns
libatasmart
libatomic_ops
libbonobo
libbonoboui
libburn
libcanberra
libcap
libchamplain
libchewing
libcmtspeechdata
libcontentaction
libcreds2
libcroco
libdaemon
libdbus-c++
libdhcp
libdiscid
libdmx
libdres
libdsme
libedit
liberation-fonts
libevent
libexif
libfakekey
libffi
libfontenc
libfprint
libgcrypt
libgda
libgdbus
libgdl
libgee
libggz
libglade2
libglademm
libgnome
libgnome-keyring
libgnomecanvas
libgnomeui
libgpg-error
libgphoto2
libgsf
libgtop2
libgweather
libhangul
libical
libICE
libid3tag
libIDL
libidn
libiodata
libiptcdata
libisofs
libjingle
libjpeg
libksba
libmatchbox
libmeegochat
libmeegotouch
libmikmod
libmlocknice
libmng
libmp4v2
libmtp
libnice
libnl
libnotify
libofx
libogg
liboil
libopenraw
libpaper
libpcap
libpciaccess
libpng
libprolog
libpthread-stubs
libqmlog
libqttracker
libresource
librsvg2
libsamplerate
libsatsolver
libsexy
libsigc++
libsignon
libsilc
libSM
libsndfile
libsocialweb
libsocialweb-keys
libsocialweb-qt
libspectre
libspiro
libtalloc
libtar
libtasn1
libtdb
libtdb-compat
libtelepathy
libthai
libtheora
libthumbnailer
libtiff
libtool
libtrace
libuninameslist
libusb
libusb1
libuser
libutempter
libv4l
libva
libvisual
libvorbis
libwbxml2
libwmf
libwnck
libwsbm
libX11
libXau
libXaw
libxcb
libXcomposite
libXcursor
libXdamage
libXdmcp
libXevie
libXext
libxfce4menu
libxfce4util
libxfcegui4
libXfixes
libXfont
libXfontcache
libXft
libXi
libXinerama
libxkbfile
libxklavier
libxml2
libxml2-python
libXmu
libXpm

libXrandr
libXrender
libXres
libXScrnSaver
libxslt
libXt
libXTrap
libXtst
libXv
libXvMC
libXxf86dga
libXxf86misc
libXxf86vm
libzip
libzypp
linux-firmware
lklug-fonts
lockdev
logrotate
lohit-assamese-fonts
lohit-bengali-fonts
lohit-hindi-fonts
lohit-kannada-fonts
lohit-malayalam-fonts
lohit-oriya-fonts
lohit-punjabi-fonts
lohit-tamil-fonts
lohit-telugu-fonts
loudmouth
lpsolve
lrzsz
lsof
lua
lzo
m17n-contrib
m17n-db
m17n-lib
m4
maemo-video-thumbnailer
mailcap
mailx
make
makebootfat
MAKEDEV
man
man-pages
marmazon
matchbox-keyboard
meego-bookmarks
meego-osc-plugins
meego-packaging-tools
meego-release
meego-rpm-config
meegotouch-applauncherd
meegotouch-applifed
meegotouch-compositor
meegotouch-controlpanel
meegotouch-feedback
meegotouch-feedbackreactionmaps
meegotouch-home
meegotouch-inputmethodengine
meegotouch-inputmethodframework
meegotouch-inputmethodkeyboard
meegotouch-systemui
meegotouch-theme
memuse
mesa-demos
mic2
min
mingetty
minicom
mkcal
mkinitrd
mlocate
mm-common
mobile-broadband-provider-info
moblin-generic-backgrounds
moblin-live
moblin-menus
module-init-tools
mousepad
mozilla-filesystem
mpage
mpc
mpfr
mtd-utils
mtools
n900-camera-firmware
nano
nasm
nc
ncurses
neon
net-tools
netpbm
newt
newt-python
nodoka-theme-gnome
notification-daemon
notification-daemon-engine-nodoka
notify-python
nspr
nss
nss-mdns
ntp
o3read
obex-data-server
obexd
ofono
ohm
ohm-plugins-misc
opal
openconnect
OpenCV
openjade
openjpeg
openldap
openobex
opensp
openssh
openssl-certs
orage
ORBit2
org
osc
ots
PackageKit
packaging-tools
pakchois
pam
pam_pkcs11
pango
pangomm
paps
papyon
parted
passivetex
passwd
patch
patchutils
pavucontrol
pax
pciutils
pcre
perl
perl-Archive-Zip
perl-Array-Compare
perl-Config-IniFiles
perl-Convert-ASN1
perl-Convert-BinHex
perl-Crypt-SSLeay
perl-Crypt-SSLeay
perl-Devel-StackTrace
perl-Devel-Symdump
perl-Devel-Symdump
perl-ExtUtils-Depends
perl-ExtUtils-MakeMaker-Coverage
perl-ExtUtils-PkgConfig
perl-File-BaseDir
perl-File-DesktopEntry
perl-File-MimeInfo
perl-File-Which
perl-Finance-Quote
perl-Font-TTF
perl-gettext
perl-Glib
perl-HTML-Parser
perl-HTML-TableExtract
perl-HTML-Tagset
perl-HTML-Tree
perl-IO-Socket-INET6
perl-IO-Socket-SSL
perl-IO-stringy
perl-JSON
perl-libwww-perl
perl-libxml-perl
perl-MailTools
perl-MIME-Lite
perl-MIME-tools
perl-Net-LibIDN
perl-Net-SMTP-SSL
perl-Net-SSLeay
perl-Parse-Yapp
perl-Pod-Coverage
perl-SDL
perl-SGMLSpm

perl-SOAP-Lite
perl-Socket6
perl-Sub-Uplevel
perl-SVG
perl-SVG-Parser
perl-Test-Exception
perl-Test-MockObject
perl-Test-NoWarnings
perl-Test-Number-Delta
perl-Test-Pod
perl-Test-Pod-Coverage
perl-Test-Tester
perl-Test-Warn
perl-Text-Unidecode
perl-Tie-IxHash
perl-TimeDate
perl-Tk
perl-Tree-DAG_Node
perl-UNIVERSAL-can
perl-UNIVERSAL-isa
perl-URI
perl-XML-DOM
perl-XML-LibXML
perl-XML-NamespaceSupport
perl-XML-Parser
perl-XML-RegExp
perl-XML-RegExp
perl-XML-Simple
perl-XML-TreeBuilder
perl-XML-XQL
perl-YAML
phidgetlinux
phonesim
phonon
pidgin
pidgin-sipe
pixman
pkgconfig
plib
plymouth-lite
pm-utils
pmtools
poedit
polkit
polkit-gnome
poppler
popt
poster
powertop
ppl
ppp
prelink
presto-utils
procps
psb-headers
psmisc
psutils
pth
ptlib
pulseaudio
pulseaudio-settings-n900
pycairo
pyclutter
pyclutter-gtk
pygobject2
pygpgme
pygtk2
pygtkglext
pykickstart
PyOpenGL
pyorbit
pyparted
python
python-adns
python-chardet
python-cheetah
python-Coherence
python-configobj
python-crypto
python-dateutil
python-decorator
python-docutils
python-dtopt
python-enchant
python-formencode
python-fpconst
python-gdata
python-imaging
python-iniparse
python-louie
python-lxml
python-magic
python-markdown
python-mutagen
python-nose
python-numeric
python-paste
python-paste-deploy
python-pycurl
python-pygments
python-reportlab
python-setuptools
python-sexy
python-simplejson
python-sqlite2
python-telepathy
python-tempita
python-toscawidgets
python-tw-forms
python-twisted
python-twisted-conch
python-twisted-core
python-twisted-lore
python-twisted-mail
python-twisted-names
python-twisted-news
python-twisted-runner
python-twisted-web
python-twisted-web2
python-twisted-words
python-urlgrabber
python-webob
python-which
python-wsgiproxy
python-xklavier
python-zope-filesystem
python-zope-interface
python-ZSI
pytz
pyxdg
pyXML
PyYAML
qca2
qca2-ossl
qjson
qmf
qt-creator
qt-mobility
qt-obex-ftp-library
qt-web-runtime
qtcontacts-tracker
qtwebkit
quilt
rarian
readline
recode
rest
rhpl
rootfiles
rpm
rpmcheck
rpmdevtools
rpmlint
rpmlint-mini
rpmlint-Moblin
rpmorphan
rpmreaper
rsync
rsyslog
rtkit
ruby
samba
sample-media
sane-backends
scim
scim-anthy
scim-bridge
scim-chewing
scim-hangul
scim-panel-vkb-gtk
scim-pinyin
scim-skk
scons
screen
SDL
SDL_gfx
SDL_image
SDL_mixer
SDL_net
SDL_Pango
SDL_ttf
sed
seed
sensorfw
setup
setuptool

sg3_utils
sgml-common
shadow-utils
shared-mime-info
sharutils
skkdic
slang
slib
smartmontools
SOAPpy
sofia-sip
sos
sound-theme-freedesktop
soundtouch
spec-builder
spectacle
speex
speex
squashfs-tools
squeeze
ssmtp
startup-notification
strace
sudo
sw-updater
swi-prolog
swig
symlinks
syncevolution
syncevolution
sysfsutils
sysklogd
system-config-date
system-config-date-docs
system-config-display
system-config-keyboard
system-config-language
system-config-printer
system-config-rootpassword
system-config-users
sysvinit
t1lib
taglib
tar
tcl
tcp_wrappers
tcpdump
tcsh
teckit
telepathy-butterfly
telepathy-farsight
telepathy-filesystem
telepathy-gabble
telepathy-glib
telepathy-haze
telepathy-idle
telepathy-logger
telepathy-logger
telepathy-qt4
telepathy-ring
telepathy-salut
telepathy-sofiasip
telepathy-sofiasip
Terminal
test-definition
testrunner-lite
texi2html
texinfo
texlive-texmf
texlive-texmf-errata
Thunar
tig
time
timed
tinycdb
tix
tk
tmpwatch
tone-generator
totem-pl-parser
tpm-tools
traceroute
tracker
transifex-client
trousers
ttmkfdir
tumbler
twitter-glib
udev
udev-rules-handset-mrst
udev-rules-netbook
udisks
un-core-fonts
unique
unzip
upower
urw-fonts
usb-modeswitch
usb-modeswitch-data
usbutils
usermode
usleep
util-linux-ng
uuid
vala
vamp-plugin-sdk
vibrant-icon-theme
vim
vlgothic-fonts
vorbis-tools
vpnc
vte
WebKit
wget
WiMAX-Network-Service
wimax-tools
wireless-tools
wlanconfig
wpa_supplicant
wv
wxPython
Xaw3d
xbacklight
xbindkeys
xcb-proto
xcb-util
xdg-user-dirs
xdg-user-dirs-gtk
xdg-utils
xdvipdfmx
xerces-c
xfce-utils
xfce4-appfinder
xfce4-battery-plugin
xfce4-datetime-plugin
xfce4-desktop-branding-moblin
xfce4-dev-tools
xfce4-icon-theme
xfce4-mixer
xfce4-panel
xfce4-quicklauncher-plugin
xfce4-session
xfce4-settings
xfce4-taskmanager
xfconf
xfdesktop
xfwm4
xfwm4-theme-nodoka
xfwm4-themes
xhtml1-dtds
xhtml2fo-style-xsl
xinetd
xinput_calibrator
xkeyboard-config
xmlrpc-c
xmltex
xmlto
xorg-x11-apps
xorg-x11-drv-evdev
xorg-x11-drv-fbdev
xorg-x11-drv-intel
xorg-x11-drv-keyboard
xorg-x11-drv-kvm
xorg-x11-drv-mga
xorg-x11-drv-mouse
xorg-x11-drv-mtev
xorg-x11-drv-synaptics
xorg-x11-drv-vesa
xorg-x11-drv-vmmouse
xorg-x11-drv-vmware
xorg-x11-drv-void
xorg-x11-drv-wacom
xorg-x11-filesystem
xorg-x11-font-utils
xorg-x11-fonts
xorg-x11-proto-bigreqsproto
xorg-x11-proto-compositeproto
xorg-x11-proto-damageproto
xorg-x11-proto-dri2proto
xorg-x11-proto-evieext
xorg-x11-proto-fixesproto
xorg-x11-proto-fontcacheproto
xorg-x11-proto-fontsproto
xorg-x11-proto-glproto
xorg-x11-proto-inputproto
xorg-x11-proto-kbproto

xorg-x11-proto-randrproto
xorg-x11-proto-recordproto
xorg-x11-proto-renderproto
xorg-x11-proto-resourceproto
xorg-x11-proto-scrnsaverproto
xorg-x11-proto-trapproto
xorg-x11-proto-videoproto
xorg-x11-proto-xcmiscproto
xorg-x11-proto-xextproto
xorg-x11-proto-xf86bigfontproto
xorg-x11-proto-xf86dgaproto
xorg-x11-proto-xf86driproto
xorg-x11-proto-xf86miscproto
xorg-x11-proto-xf86rushproto
xorg-x11-proto-xf86vidmodeproto
xorg-x11-proto-xineramaproto
xorg-x11-proto-xproto
xorg-x11-proto-xproxymgmttproto
xorg-x11-server
xorg-x11-server-utils
xorg-x11-twm
xorg-x11-util-macros
xorg-x11-utils
xorg-x11-utils-iceauth
xorg-x11-utils-rgb
xorg-x11-utils-sessreg
xorg-x11-utils-xcmsdb
xorg-x11-utils-xdpyinfo
xorg-x11-utils-xdriinfo
xorg-x11-utils-xev
xorg-x11-utils-xfd
xorg-x11-utils-xfontsel
xorg-x11-utils-xgamma
xorg-x11-utils-xhost
xorg-x11-utils-xinput
xorg-x11-utils-xlsatoms
xorg-x11-utils-xlsclients
xorg-x11-utils-xlsfonts
xorg-x11-utils-xmodmap
xorg-x11-utils-xprop
xorg-x11-utils-xrandr
xorg-x11-utils-xrdb
xorg-x11-utils-xrefresh
xorg-x11-utils-xset
xorg-x11-utils-xsetroot
xorg-x11-utils-xvinfo
xorg-x11-utils-xwininfo
xorg-x11-xauth
xorg-x11-xbitmaps
xorg-x11-xinit
xorg-x11-xkb-utils
xorg-x11-xtrans-devel
xterm
xz
yasm
yelp
yum
yum-metadata-parser
yum-presto
yum-updatesd
yum-utils
zenity
zile
zip
zsh
zypper

### A.1.2 Netbook repository

abiword
aiksaurus
anerley
autogen
babl
banshee-1-branding-meego
bisho
brasero
cdrdao
celestia
cheese
chrome-meego-extension
contacts
dates
dcraw
deluge
dia
empathy
eog
eog-plugins
evince
evolution
file-roller
foobillard
fpm2
frozen-bubble
garage-netbook-ui
gcalctool
gcompris
gconf-editor
gdu-nautilus-extension
gedit
gimp
gnome-bluetooth
gnome-control-center-netbook
gnome-desktop
gnome-games
gnome-media
gnome-menus
gnome-packagekit
gnome-panel
gnome-python2-desktop
gnome-screensaver
gnome-session
gnome-settings-daemon
gnome-terminal
gnome-themes
gnome-user-docs
gnome-user-share
gnome-utils
gnuchess
gnumeric
goffice
google-gadgets
grisbi
gthumb
gtkhtml3
gtkmathview
gtksourceview2
gupnp-tools
homebank
ilmbase
impressive
libgail-gnome
libgdiplus0
libgnomecups
libgnomekbd
libgnomeprint22
libgnomeprintui22
libwpd
libwpg
libwps
link-grammar
marble
matchbox-panel
mathml-fonts
meego-cursor-theme
meego-help
meego-menus
meego-netbook-theme
meego-panel-applications
meego-panel-datetime
meego-panel-devices
meego-panel-myzone
meego-panel-networks
meego-panel-pasteboard
meego-panel-people
meego-panel-status
meego-panel-status
meego-panel-zones
meego-sound-theme
meego-ux-settings
meld
moblin-user-guide
moblin-user-skel
mutter-meego
mx
nautilus
nautilus-python

| | | |
|---|---|---|
| nbtk | planner | stellarium |
| netbook-backgrounds | pygtksourceview | syncevolution-gtk |
| netbook-icon-theme | quicksynergy | tasks |
| neverball | rawstudio | totem |
| OpenEXR | rhythmbox | transmission |
| opengl-games-utils | simple-scan | tuxpaint |

# A.2 Modified packages

The following packages were modified in order to successfully build for SPARC. In most cases the problems were related to poorly written spec files or minor code glitches.

## A.2.1 Core repository

### boost

Added the *–disable-long-double* build options in the spec file.

### cairo

Removed the build dependency to *binutils-devel* in the spec file.

### ctdb

Added a SPARC-specific patch to avoid the faulty generation of some headers at compile time.

### db4

Disabled DISTCC because it caused the build to fail.

### dbus

Fixed the environment PATH and added manually the correct build flags (ignored due to a glitch in the spec file).

### e2fsprogs

Disabled x86 specific tests when targeting SPARC.

### gdb

Excluded the gdb server because it did not support the SPARC architecture. The client was fine.

**gmp**

Disabled DISTCC because it caused the build to fail.

**icu**

Fixed some glitches in the make files which caused the build to fail.

**iproute**

Removed the build dependencies to *tetex-latex* and *tetex-dvips*.

**libcap-ng**

Changed the *–libdir* build option in the spec file.

**libdrm**

Removed the Intel and Radeon specific components.

**libproxy**

Removed the *libproxy-webkit* subpackage to resolve a dependency cycle.

**libsoup**

Removed the *–without-gnome* build option in the spec file.

**ltrace**

Backported a patch present in a more recent version of the software.

**m2crypto**

Fixed a constant definition in the spec file.

**meego-lsb**

Fixed some architecture conditionals in the spec file.

**mesa**

Removed the Intel graphic drivers.

**moblin-icon-theme**

Disabled the invokation of *gtk-update-icon-cache*.

**openssl**

Added the *sslarch=linux-sparcv8* and *sslflags=no-asm* build options in the spec file.

**orc**

Fixed an architecture conditional in the spec file.

**patchelf**

Disabled x86 specific tests when targeting SPARC.

**post-build-checks**

Fixed several architecture-related glitches in the spec file.

**pulseaudio-modules-meego**

Added the build option *-DAO_REQUIRE_CAS*.

**pyOpenSSL**

Removed the build dependency to *w3m* in the spec file.

**qt**

Disabled DISTCC because it caused the build to fail.

**rpm-python**

Fixed minor glitches in the spec file.

**texlive**

Fixed the path to cpp in the spec file.

**tzdata**

Included a working version of the ZIC utility.

**uxlaunch**

Added a patch that provides the I/O priority constants for SPARC.

**wxGTK**

Fixed some gcc invokation flags in the configure script.

### A.2.2 Netbook repository

**abrt-netbook**

Removed the build dependency to *pkgconfig(libgnome-control-center-extension)*.

**bickley**

Added a patch that provides the I/O priority constants for SPARC.

**bugle**

Disabled a patch that caused the build to fail.

**gegl**

Removed the build dependency to *w3m*.

**mutter**

Fixed the package file list in the spec file.

**sunbird**

Added the build option *–disable-jit* in the spec file.

**thunderbird**

Added the build option *–disable-jit* in the spec file.

## A.3 Excluded packages

The following packages were excluded from the port, usually because they explicitly did not support the SPARC architecture and the required changes were considered to consistent to fall withing the project scope. Most exclusions were caused by these five reasons:

1. The package explicitly did not support the SPARC architecture or was designed to work only on a different, specific architecture. For example some packages contained incompatible assembler code or Just-in-Time (JIT) compilers.
2. The package was meant to be compiled only for x86 systems and to be used to implement the mixed chroot build technique of the OBS. Since this technique could not be used to target SPARC, building these packages was pointless.
3. The package was designed to explicitly support or require hardware which was not available on the LEON/GRLIB platform. This included specific device drivers and packages containing binary blobs or firmware.
4. The package replicated the functionality already provided by the *system-root* package.
5. The package depended on another excluded package.

### A.3.1 Core repository

```
bash-x86 (2)
binutils (4)
bootstub (1)
chromium (5)
cloog (1)
compat-libstdc++-33 (1)
cross-armv5tel-binutils (1)
cross-armv5tel-binutils-accel (2)
cross-armv5tel-gcc (1)
cross-armv5tel-gcc-accel (2)
cross-armv7l-binutils (1)
cross-armv7l-binutils-accel (2)
cross-armv7l-gcc (1)
cross-armv7l-gcc-accel (2)
dev86 (1)
doxymacs (5)
fakechroot (1)
fennec-qt (1)
fw-update (3)
gcc (4)
glibc (4)
glibc-x86 (2)
gmp-x86 (2)
gnu-efi (3)
grub (1)
gupnp-vala (1)
intel-gpu-tools (3)
kernel (3)
kernel-headers (4)
kernel-ivi (3)
kernel-mrst (3)
kernel-netbook (3)
kexec-tools (1)
libgcc-x86 (2)
libsmbios (3)
meego-cross-armv5tel-sysroot (2)
meego-cross-armv7l-sysroot (2)
mpc-x86 (2)
mpfr-x86 (2)
ncurses-libs-x86 (2)
sreadahead (1)
subversion (1)
syslinux (1)
sysprof (1)
system-config-boot (5)
v8 (1)
valgrind (1)
w3m (1)
zlib-x86 (2)
```

### A.3.2 Netbook repository

```
anjuta (5)
banshee-1 (5)
gnome-sharp2 (5)
gtk-sharp2 (5)
inkscape (5)
mono-addins (5)
mono-core (1)
mono-zeroconf (5)
ndesk-dbus (5)
ndesk-dbus-glib (5)
notify-sharp (5)
taglib-sharp (5)
vym (1)
```

# Appendix B

# Images content

This appendix lists the packages installed in the system images that were prepared alongside the testing phase.

## B.1 minimal

augeas-libs
basesystem
bash
bzip2
bzip2-libs
chkconfig
ConsoleKit
ConsoleKit-libs
coreutils
cpio
curl
db4
db4-utils
dbus
dbus-glib
dbus-libs
e2fsprogs
e2fsprogs-libs
eggdbus
elfutils-libelf
expat
fastinit
file-libs
filesystem
findutils
gamin
gawk
gdbm
glib2
gnupg2
grep
gzip
hwdata
info
iputils
less
libacl
libattr
libblkid
libcap
libcom_err
libcurl
libgcrypt
libgpg-error
libidn
libksba
liblua
libss
libudev
libusb
libuser
libuuid
libxml2
libzypp
logrotate
MAKEDEV
meego-release
mingetty
moblin-user-skel
module-init-tools
nano
ncurses
ncurses-base
ncurses-libs
net-tools
nspr
nss
nss-softokn-freebl
nss-sysinit
openssh
openssh-server
openssl
pam
passwd
pcre
perl
perl-Compress-Raw-Zlib
perl-CPAN
perl-devel
perl-ExtUtils-MakeMaker
perl-ExtUtils-ParseXS
perl-IO-Compress-Base
perl-IO-Compress-Zlib
perl-libs
perl-Module-Pluggable
perl-Pod-Escapes
perl-Pod-Simple
perl-Test-Harness
polkit
popt
procps
psmisc
pth
readline
rootfiles
rpm
rpm-libs
rsyslog
satsolver-tools
sed
setup
shadow-utils

sqlite
sudo
system-root
sysvinit
sysvinit-tools
tzdata
udev
usleep
util-linux-ng
vim-minimal
wget
xz-libs
zlib
zypper

## B.2 xorg

augeas-libs
basesystem
bash
bzip2
bzip2-libs
chkconfig
ConsoleKit
ConsoleKit-libs
ConsoleKit-x11
coreutils
cpio
curl
db4
db4-utils
dbus
dbus-glib
dbus-libs
e2fsprogs
e2fsprogs-libs
eggdbus
elfutils-libelf
expat
fastinit
file-libs
filesystem
findutils
fontconfig
freetype
gamin
gawk
gdbm
glib2
gnupg2
grep
gzip
hwdata
info
iputils
less
libacl
libattr
libblkid
libcap
libcom_err
libcurl
libdrm
libfontenc
libgcrypt
libgpg-error
libICE
libidn
libksba
liblua
libpciaccess
libpng
libSM
libss
libtalloc
libudev
libusb
libuser
libutempter
libuuid
libX11
libXau
libXaw
libxcb
libXcursor
libXdmcp
libXext
libXfixes
libXfont
libXft
libxkbfile
libxml2
libXmu
libXpm
libXrender
libXt
libzypp
logrotate
MAKEDEV
meego-release
mesa-dri-swrast-driver
mingetty
moblin-user-skel
module-init-tools
nano
ncurses
ncurses-base
ncurses-libs
net-tools
nspr
nss
nss-softokn-freebl
nss-sysinit
openssl
pam
passwd
pcre
perl
perl-Compress-Raw-Zlib
perl-CPAN
perl-devel
perl-ExtUtils-MakeMaker
perl-ExtUtils-ParseXS
perl-IO-Compress-Base
perl-IO-Compress-Zlib
perl-libs
perl-Module-Pluggable
perl-Pod-Escapes
perl-Pod-Simple
perl-Test-Harness
pixman
pkgconfig
polkit
popt
procps
psmisc
pth
readline
rootfiles
rpm
rpm-libs
rsyslog
satsolver-tools
sed
setup
shadow-utils
sqlite
sudo
system-root
sysvinit
sysvinit-tools
tzdata
udev
usleep
util-linux-ng
vim-minimal
wget
xkeyboard-config
xorg-x11-apps
xorg-x11-drv-evdev
xorg-x11-drv-fbdev
xorg-x11-drv-keyboard
xorg-x11-drv-mouse
xorg-x11-filesystem
xorg-x11-fonts-100dpi
xorg-x11-fonts-75dpi
xorg-x11-font-utils
xorg-x11-server
xorg-x11-server-common
xorg-x11-server-Xorg-setuid

xorg-x11-twm
xorg-x11-utils-xhost
xorg-x11-utils-xmodmap
xorg-x11-utils-xrdb
xorg-x11-utils-xsetroot
xorg-x11-xauth
xorg-x11-xbitmaps
xorg-x11-xinit
xorg-x11-xkb-utils
xterm
xz-libs
zlib
zypper

## B.3 xfce

alsa-lib
atk
augeas-libs
avahi
basesystem
bash
bzip2
bzip2-libs
cairo
chkconfig
ConsoleKit
ConsoleKit-libs
ConsoleKit-x11
coreutils
cpio
cups-libs
curl
db4
db4-utils
dbus
dbus-glib
dbus-libs
dbus-x11
dejavu-fonts-common
dejavu-lgc-sans-fonts
dejavu-lgc-sans-mono-fonts
dejavu-lgc-serif-fonts
dejavu-sans-fonts
dejavu-sans-mono-fonts
dejavu-serif-fonts
desktop-backgrounds-basic
desktop-file-utils
devhelp
e2fsprogs
e2fsprogs-libs
eggdbus
elfutils-libelf
enchant
exo
expat
fastinit
file-libs
filesystem
findutils
fontconfig
fontpackages-filesystem
freetype
gamin
gawk
GConf-dbus
gdbm
generic-backgrounds
glade3
glade3-libgladeui
glib2
gnome-icon-theme
gnupg2
gnutls
grep
gst-plugins-base
gstreamer
gtk2
gtk-nodoka-engine
gtk-nodoka-engine-extras
gtk-xfce-engine
gzip
hicolor-icon-theme
hunspell
hwdata
info
iputils
jasper
jasper-libs
less
libacl
libattr
libblkid
libcap
libcom_err
libcurl
libdaemon
libdrm
libexif
libfontenc
libgcrypt
libglade2
libgnome-keyring
libgpg-error
libICE
libicu
libidn
libjpeg
libksba
liblua
libnotify
libogg
libpciaccess
libpng
libproxy
libSM
libsoup
libss
libtalloc
libtasn1
libthai
libtheora
libtiff
libudev
libusb
libuser
libutempter
libuuid
libvisual
libvorbis
libwnck
libX11
libXau
libXaw
libxcb
libXcomposite
libXcursor
libXdamage
libXdmcp
libXext
libxfce4menu
libxfce4util
libxfcegui4
libXfixes
libXfont
libXft
libXi
libXinerama
libxkbfile
libxklavier
libxml2
libXmu
libXpm
libXrandr
libXrender
libXres
libxslt
libXt
libXv
libXxf86vm
libzypp
logrotate
MAKEDEV
meego-release
mesa-dri-swrast-driver
mingetty
moblin-user-skel
module-init-tools
mousepad

nano
ncurses
ncurses-base
ncurses-libs
net-tools
nodoka-filesystem
nspr
nss
nss-softokn-freebl
nss-sysinit
openssl
orc
pam
pango
passwd
pcre
perl
perl-Compress-Raw-Zlib
perl-CPAN
perl-devel
perl-ExtUtils-MakeMaker
perl-ExtUtils-ParseXS
perl-Glib
perl-IO-Compress-Base
perl-IO-Compress-Zlib
perl-libs
perl-Module-Pluggable
perl-Pod-Escapes
perl-Pod-Simple
perl-Test-Harness
pixman
pkgconfig
polkit
popt
procps
psmisc
pth
rarian
rarian-compat
readline
rootfiles
rpm
rpm-libs
rsyslog
satsolver-tools
sed
setup
shadow-utils
shared-mime-info
sqlite
startup-notification
sudo
system-root
sysvinit
sysvinit-tools
Terminal
Thunar
tzdata
udev
un-core-fonts-batangbold
un-core-fonts-dinaru
un-core-fonts-dinarubold
un-core-fonts-dinarulight
un-core-fonts-dotum
un-core-fonts-dotumbold
un-core-fonts-graphic
un-core-fonts-graphicbold
un-core-fonts-gungseo
un-core-fonts-pilgi
un-core-fonts-pilgibold
unique
urw-fonts
usleep
util-linux-ng
uxlaunch
vim-minimal
vte
WebKit-gtk
wget
xdg-user-dirs
xdg-user-dirs-gtk
xfce4-appfinder
xfce4-battery-plugin
xfce4-datetime-plugin
xfce4-desktop-branding-moblin
xfce4-icon-theme
xfce4-mixer
xfce4-panel
xfce4-quicklauncher-plugin
xfce4-session
xfce4-session-engines
xfce4-settings
xfce4-taskmanager
xfce-utils
xfconf
xfconf-perl
xfdesktop
xfwm4
xfwm4-theme-nodoka
xfwm4-themes
xkeyboard-config
xorg-x11-apps
xorg-x11-drv-evdev
xorg-x11-drv-fbdev
xorg-x11-drv-keyboard
xorg-x11-drv-mouse
xorg-x11-filesystem
xorg-x11-fonts-100dpi
xorg-x11-fonts-75dpi
xorg-x11-font-utils
xorg-x11-server
xorg-x11-server-common
xorg-x11-server-utils
xorg-x11-server-Xorg-setuid
xorg-x11-twm
xorg-x11-utils-iceauth
xorg-x11-utils-rgb
xorg-x11-utils-sessreg
xorg-x11-utils-xcmsdb
xorg-x11-utils-xgamma
xorg-x11-utils-xhost
xorg-x11-utils-xmodmap
xorg-x11-utils-xrandr
xorg-x11-utils-xrdb
xorg-x11-utils-xrefresh
xorg-x11-utils-xset
xorg-x11-utils-xsetroot
xorg-x11-xauth
xorg-x11-xbitmaps
xorg-x11-xinit
xorg-x11-xkb-utils
xterm
xz-libs
zlib
zypper

## B.4 netbook

acl
alsa-lib
alsa-utils
anerley
aspell
aspell-en
atk
augeas-libs
autoconf
automake
avahi
avahi-glib
avahi-gobject
avahi-ui
basesystem
bash
bisho
bluez
bluez-libs
btrfs-progs
buteo-mtp
buteo-syncfw
buteo-syncml
buteo-sync-plugins
bzip2
bzip2-libs
ca-certificates
cairo
cheese
chkconfig
chrome-meego-extension
cjkuni-fonts
clutter

clutter-gesture
clutter-gtk
clutter-imcontext
connman
ConsoleKit
ConsoleKit-libs
ConsoleKit-x11
contextkit
coreutils
coreutils-libs
cpio
cryptsetup-luks
cups
cups-libs
curl
cyrus-sasl-lib
cyrus-sasl-md5
cyrus-sasl-plain
db4
db4-utils
dbus
dbus-glib
dbus-libs
dbus-python
dbus-x11
dejavu-fonts-common
dejavu-sans-fonts
deltarpm
desktop-file-utils
device-mapper
device-mapper-libs
dhclient
dialog
docbook-dtds
dosfstools
droid-sans-fonts
droid-sans-mono-fonts
droid-serif-fonts
dsme
e2fsprogs
e2fsprogs-libs
ecryptfs-utils
eggdbus
eject
elfutils-libelf
empathy
enchant
enchant-aspell
eog
evince
evince-libs
evolution-data-server
exempi
expat
farsight2
fastinit
file
file-libs
file-roller
filesystem
findutils
firstboot
flac
fontconfig
fontpackages-filesystem
foomatic
foomatic-db
foomatic-db-filesystem
foomatic-db-ppds
freetype
frozen-bubble
fuse
fuse-libs
gamin
garage-client-services
garage-netbook-ui
gawk
gcalctool
gcalctool-doc
GConf-dbus
gdbm
gdu-nautilus-extension
gedit
generic-logos
genisoimage
geoclue
gettext-libs
ghostscript
ghostscript-fonts
giflib
glew
glib2
glx-utils
gmime
gmp
gnome-bluetooth
gnome-bluetooth-libs
gnome-bluetooth-meego
gnome-control-center-netbook
gnome-desktop
gnome-disk-utility
gnome-disk-utility-libs
gnome-disk-utility-ui-libs
gnome-doc-utils
gnome-doc-utils-stylesheets
gnome-games
gnome-games-help
gnome-icon-theme
gnome-keyring
gnome-keyring-pam
gnome-media
gnome-media-libs
gnome-menus
gnome-mime-data
gnome-python2
gnome-python2-canvas
gnome-python2-desktop
gnome-python2-gnomekeyring
gnome-screensaver
gnome-settings-daemon
gnome-terminal
gnome-utils
gnome-vfs2
gnupg2
gnutls
google-gadgets
google-gadgets-meego
gpgme
grep
grubby
gssdp
gst-plugins-bad-free
gst-plugins-base
gst-plugins-good
gstreamer
gthumb
gtk2
gtkhtml3
gtksourceview2
guile
gupnp
gupnp-igd
gvfs
gvfs-gphoto2
gvfs-obexftp
gvfs-smb
gvfs-trash
gypsy
gzip
hicolor-icon-theme
hunspell
hwdata
info
installer-launch
iproute
iputils
iso-codes
isomd5sum
jana
jasper
jasper-libs
json-glib
kbd
kcalcore
keyutils
keyutils-libs
kpartx
krb5-libs
lcms
lcms-libs
less
libaccounts-glib
libaccounts-qt
libacl
libarchive
libart_lgpl
libasyncns
libatasmart
libattr
libblkid
libbonobo
libbonoboui
libcanberra
libcanberra-gtk2
libcap
libchamplain

libchewing
libcom_err
libcreds2
libcroco
libcurl
libdaemon
libdeclarative-contacts
libdeclarative-multimedia
libdeclarative-publishsubscribe
libdeclarative-sensors
libdeclarative-serviceframework
libdres
libdrm
libdsme
libedit
libevent
libexif
libffi
libfontenc
libgcrypt
libgdbus
libgdiplus0
libgee
libglade2
libgnome
libgnomecanvas
libgnomekbd
libgnome-keyring
libgnomeui
libgpg-error
libgphoto2
libgsf
libgtop2
libgudev1
libgweather
libhangul
libical
libICE
libicu
libIDL
libidn
libiodata
libiphb
libiptcdata
libjpeg
libksba
liblua
libmeegotouch
libmng
libmtp
libnice
libnl
libnotify
libogg
libopenraw
libopenraw-gnome
libpciaccess
libphonon4
libpng
libprolog
libproxy
libproxy-gnome
libproxy-python
libpurple
libqmlog
libqtcontacts1
libqtcore4
libqtdbus4
libqtdeclarative4
libqtdeclarative4-folderlistmodel
libqtdeclarative4-gestures
libqtdeclarative4-particles
libqtdesigner4
libqtgui4
libqtlocation1
libqtmessaging1
libqtmultimediakit1
libqtnetwork4
libqtopengl4
libqtopiamail1
libqtpublishsubscribe1
libqtscript4
libqtsensors1
libqtserviceframework1
libqtsql4
libqtsql4-sqlite
libqtsvg4
libqtsysteminfo1
libqttest4
libqttracker
libqtversit1
libqtwebkit4
libqtwebkit-qmlwebkitplugin
libqtxml4
libqtxmlpatterns4
libresource
libresource-client
librsvg2
libsignon
libsignon-passwordplugin
libsignon-saslplugin
libsilc
libSM
libsmbclient
libsndfile
libsocialweb
libsocialweb-keys
libsoup
libspectre
libss
libtalloc
libtasn1
libtdb
libtelepathy
libthai
libtheora
libtiff
libtool-ltdl
libtrace
libudev
libusb
libusb1
libuser
libuser-python
libutempter
libuuid
libvisual
libvorbis
libwbxml2
libwnck
libX11
libXau
libXaw
libxcb
libXcomposite
libXcursor
libXdamage
libXdmcp
libXext
libXfixes
libXfont
libXft
libXi
libXinerama
libxkbfile
libxklavier
libxml2
libxml2-python
libXmu
libXpm
libXrandr
libXrender
libXres
libXScrnSaver
libxslt
libXt
libXtst
libXv
libXxf86misc
libXxf86vm
libzypp
linux-firmware
lockdev
logrotate
m2crypto
m4
mailcap
MAKEDEV
matchbox-panel
meego-cursor-theme
meego-help
meego-menus
meego-netbook-theme
meego-panel-applications
meego-panel-datetime
meego-panel-devices
meego-panel-myzone
meego-panel-networks
meego-panel-pasteboard
meego-panel-people
meego-panel-status
meego-panel-web
meego-panel-zones
meego-release
meego-sound-theme
meegotouch-applauncherd

meegotouch-applifed
meegotouch-controlpanel
meegotouch-feedback
meegotouch-feedbackreactionmaps
meegotouch-inputmethodengine
meegotouch-inputmethodframework
meegotouch-theme
meego-ux-settings
mesa-dri-swrast-driver
mesa-libGL
mesa-libGLU
mesa-libGLUT
mesa-libOSMesa
mic2
mingetty
minizip
mkinitrd
mobile-broadband-provider-info
moblin-live
moblin-user-skel
module-init-tools
mozilla-filesystem
mtools
mutter
mutter-meego
mutter-meego-branding-upstream
mx
nano
nautilus
nautilus-extensions
ncurses
ncurses-base
ncurses-libs
netbook-backgrounds
netbook-icon-theme
net-tools
neverball
notify-python
nspr
nss
nss-mdns
nss-softokn-freebl
nss-sysinit
ntp
ntpdate
o3read
obexd
obex-data-server
ofono
ohm
ohm-config
ohm-plugin-core
ohm-plugin-resolver
ohm-plugins-misc
opengl-games-utils
openjpeg-libs
openldap
openobex
openssh
openssh-clients
openssh-server
openssl
ORBit2
orc
PackageKit
PackageKit-browser-plugin
PackageKit-device-rebind
PackageKit-glib
PackageKit-gtk-module
PackageKit-qt
PackageKit-zypp
pam
pango
papyon
parted
passwd
pciutils
pcre
perl
perl-Compress-Raw-Zlib
perl-CPAN
perl-devel
perl-ExtUtils-MakeMaker
perl-ExtUtils-ParseXS
perl-File-BaseDir
perl-File-DesktopEntry
perl-File-MimeInfo
perl-gettext
perl-IO-Compress-Base
perl-IO-Compress-Zlib
perl-libs
perl-Locale-Maketext-Simple
perl-Module-Pluggable
perl-Pod-Escapes
perl-Pod-Simple
perl-SDL
perl-Test-Harness
pidgin-sipe
pixman
pkgconfig
plymouth-lite
pm-utils
polkit
polkit-gnome
poppler
poppler-glib
poppler-utils
popt
prelink
procps
psmisc
pth
pulseaudio
pulseaudio-module-x11
pycairo
pygobject2
pygpgme
pygtk2
pygtk2-libglade
pykickstart
pyOpenSSL
pyparted
python
python-decorator
python-iniparse
python-libs
python-numeric
python-pycurl
python-simplejson
python-telepathy
python-urlgrabber
qjson
qtcontacts-tracker
qt-mobility
qt-web-runtime
rarian
rarian-compat
readline
rest
rhpl
rootfiles
rpm
rpm-libs
rpm-python
rsync
rtkit
samba-winbind-clients
sample-media
satsolver-tools
scim
scim-bridge
scim-bridge-clutter
scim-bridge-gtk
scim-chewing
scim-hangul
scim-pinyin
scim-skk
SDL
SDL_gfx
SDL_image
SDL_mixer
SDL_net
SDL_Pango
SDL_ttf
sed
sensorfw
servicefw
setup
sg3_utils-libs
sgml-common
shadow-utils
shared-mime-info
skkdic
sofia-sip
sofia-sip-glib
sound-theme-freedesktop
speex
sqlite
squashfs-tools
startup-notification
strace
sudo
swi-prolog
swi-prolog-lib
swi-prolog-lib-core
syncevolution

syncevolution-evolution
syncevolution-gtk
sysklogd
system-config-date
system-config-date-docs
system-config-language
system-config-printer
system-config-printer-libs
system-root
sysvinit
sysvinit-tools
taglib
tar
tasks
telepathy-butterfly
telepathy-farsight
telepathy-filesystem
telepathy-gabble
telepathy-glib
telepathy-haze
telepathy-idle
telepathy-mission-control
telepathy-qt4
telepathy-qt4-farsight
telepathy-ring
telepathy-salut
telepathy-sofiasip
telepathy-stream-engine
time
timed
tinycdb
tmpwatch
totem-pl-parser
tracker
trousers
ttmkfdir
tzdata
udev
udisks
unique
unzip
upower
urw-fonts
usbutils
usermode
usermode-gtk
usleep
util-linux-ng
uxlaunch
vim-minimal
vlgothic-fonts
vlgothic-fonts-common
vte
WebKit-gtk
wget
wireless-tools
wpa_supplicant
xcb-util
xdg-user-dirs
xdg-user-dirs-gtk
xdg-utils
xinetd
xkeyboard-config
xml-common
xorg-x11-apps
xorg-x11-drv-evdev
xorg-x11-drv-fbdev
xorg-x11-drv-keyboard
xorg-x11-drv-mouse
xorg-x11-drv-synaptics
xorg-x11-drv-vesa
xorg-x11-fonts-100dpi
xorg-x11-fonts-ISO8859-1-100dpi
xorg-x11-fonts-misc
xorg-x11-fonts-Type1
xorg-x11-font-utils
xorg-x11-server
xorg-x11-server-common
xorg-x11-server-Xorg
xorg-x11-twm
xorg-x11-utils
xorg-x11-utils-xdpyinfo
xorg-x11-utils-xdriinfo
xorg-x11-utils-xev
xorg-x11-utils-xfd
xorg-x11-utils-xfontsel
xorg-x11-utils-xhost
xorg-x11-utils-xlsatoms
xorg-x11-utils-xlsclients
xorg-x11-utils-xlsfonts
xorg-x11-utils-xmodmap
xorg-x11-utils-xprop
xorg-x11-utils-xrandr
xorg-x11-utils-xrdb
xorg-x11-utils-xsetroot
xorg-x11-utils-xvinfo
xorg-x11-utils-xwininfo
xorg-x11-xauth
xorg-x11-xinit
xorg-x11-xkb-utils
xterm
xz-libs
yelp
yum
yum-metadata-parser
zenity
zlib
zypper

# Appendix C

# Referenced source code

This appendix contains the relevant part of the source code and the patches that were produced while working on the project, which are referenced elsewhere in the text.

## C.1 Temporary environment

This section contains code and patches related to the temporary build environment.

### C.1.1 RPM configuration

This section provides the configuration files added to the RPM installation in the MeeGo 1.1 chroot in order to target the SPARC architecture.

#### platform/sparc-linux/macros

```
# Per-platform rpm configuration file.

#==========================
# ---- per-platform macros.
#
%optflags   -mcpu=v8 -m32 -mhard-float --sysroot=/home/meego/root

%_arch      sparc
%_vendor    leon
%_os      linux
%_gnu     -gnu
%_target_platform %{_target_cpu}-%{_vendor}-%{_target_os}

%_host_cpu    sparc
%_host_vendor   leon
%_host_os   linux
%_host      %{_host_cpu}-%{_host_vendor}-%{_host_os}-gnu

%_build     i686-build_pc-linux-gnu
```

```
%_build_cpu   i686
%_build_vendor    build_pc
%_build_os    linux

#=========================
# ---- configure macros.
#
%_prefix    /usr
%_exec_prefix   %{_prefix}
%_bindir    %{_exec_prefix}/bin
%_sbindir   %{_exec_prefix}/sbin
%_libexecdir    %{_exec_prefix}/libexec
%_datarootdir   %{_prefix}/share
%_datadir   %{_datarootdir}
%_sysconfdir    /etc
%_sharedstatedir  /var/lib
%_localstatedir   /var
%_lib     lib
%_libdir    %{_prefix}/lib
%_includedir    %{_prefix}/include
%_oldincludedir   /usr/include
%_infodir   %{_datarootdir}/info
%_mandir    %{_datarootdir}/man
%_initddir    %{_sysconfdir}/rc.d/init.d
# Deprecated misspelling, present for backwards compatibility.
%_initrddir   %{_initddir}

%_defaultdocdir   %{_datadir}/doc

%_smp_mflags %([ -z "$RPM_BUILD_NCPUS" ] \\\
  && RPM_BUILD_NCPUS="`/usr/bin/getconf _NPROCESSORS_ONLN`"; \\\
  [ "$RPM_BUILD_NCPUS" -gt 1 ] && echo "-j$RPM_BUILD_NCPUS")

#=========================
# ---- Build system path macros.
#
%__ar     %{_host}-ar
%__as     %{_host}-as
%__cc     %{_host}-gcc
%__cpp      %{_host}-gcc -E
%__cxx      %{_host}-g++
%__ld     %{_host}-ld
%__nm     %{_host}-nm
%__objcopy    %{_host}-objcopy
%__objdump    %{_host}-objdump
%__ranlib   %{_host}-ranlib
%__remsh    %{__rsh}
%__strip    %{_host}-strip

#=========================
# ---- Build policy macros.
#
#-------------------------
# Expanded at end of %install scriptlet.
#

%__arch_install_post %{nil}

%__os_install_post \
    %{_rpmconfigdir}/brp-compress \
    brp-strip-sparc \
    brp-strip-static-archive-sparc \
```

```
    brp-strip-comment-note-sparc \
%{nil}

%__spec_install_post\
    %{?__debug_package:%{__debug_install_post}}\
    %{__arch_install_post}\
    %{__os_install_post}\
%{nil}

#-------------------------
# Expanded at end of %prep
#
%__id_u   %{__id} -u
%__chown_Rhf  %{__chown} -Rhf
%__chgrp_Rhf  %{__chgrp} -Rhf
%_fixperms  %{__chmod} -Rf a+rX,u+w,g-w,o-w
#-------------------------
# Always use %defattr(-,root,root) in %files (added in rpm-4.0.4)
#
#%files(n:f:) %%files%{?-f: -f %{-f*}}%{?-n: -n %{-n*}} %{?1}\
#%defattr(-,root,root,-)\
#%{nil}
```

### C.1.2 Automation scripts

This section provides the source code of the automation scripts used to build the bootstrap repository.

#### include/env.inc

```
#!/bin/bash

# BASH SETTINGS
set +h
umask 022

# USER CONFIGURATION
export SM_VERSION="1.1" # meego version to be built
export SM_CACHE_MAXAGE="3600" # amount of seconds after cached files expire
export SM_EDITOR="nano" # preferred text editor
export SM_REMOTEPORT="5011"

# GENERATED CONFIGURATION
export SM_REPOSITORY="http://repo.meego.com/MeeGo/releases/$SM_VERSION"
export SM_PATH="/home/meego"

# GENERATED PATHS
export SM_PATH_RPMBUILD="$SM_PATH/rpmbuild" # rpmbuild root directory
export SM_PATH_SCRIPTS="$SM_PATH/scripts" # sm scripts
export SM_PATH_REPOSITORY="$SM_PATH/repository" # generate packages
export SM_PATH_ROOT="$SM_PATH/root" # populated sysroot
export SM_PATH_CACHE="$SM_PATH/cache" # downloaded packages
export SM_PATH_LOGS="$SM_PATH/logs" # downloaded packages
export SM_PATH_CONFIGURE_FLAGS="$SM_PATH_SCRIPTS/workarounds/data/configure.
    flags"
export SM_PATH_COMPILE_FLAGS="$SM_PATH_SCRIPTS/workarounds/data/compile.flags"
export SM_PATH_CONFIGURE_CACHE="$SM_PATH_SCRIPTS/workarounds/data/configure.
    cache"
export SM_PATH_CONFIGURE_HOST="$SM_PATH_SCRIPTS/workarounds/data/configure.host"
```

```
export SM_PATH_CONFIGURE_TARGET="$SM_PATH_SCRIPTS/workarounds/data/configure.target"
export SM_PATH_CONFIGURE_BUILD="$SM_PATH_SCRIPTS/workarounds/data/configure.build"

# SSH COMMANDS
export SM_SSH_INTO_CHROOT="ssh meego@ivan"
export SM_SSH_INTO_NATIVE="ssh meego@ultra1"

# COMPILER TARGETS AND FLAGS
export SM_COMPILE_BUILD="i686-build_pc-linux-gnu" # where the cross compiler is executed
export SM_COMPILE_HOST="sparc-leon-linux-gnu" # where the compiled code is executed
export SM_COMPILE_TARGET="sparc-leon-linux-gnu" # where the compiled code is executed
export SM_COMPILE_CFLAGS="-mcpu=v8 -m32 -mhard-float --sysroot=$SM_PATH_ROOT -Xlinker --build-id -L$SM_PATH_ROOT/lib/ -L$SM_PATH_ROOT/usr/lib/"
export SM_COMPILE_CXXFLAGS="-mcpu=v8 -m32 -mhard-float --sysroot=$SM_PATH_ROOT -Xlinker --build-id -L$SM_PATH_ROOT/lib/ -L$SM_PATH_ROOT/usr/lib/"
export SM_COMPILE_LDFLAGS="--sysroot=$SM_PATH_ROOT -L$SM_PATH_ROOT/lib/ -L$SM_PATH_ROOT/usr/lib/"

# REMOTECLIENT CONFIGURATION
export SM_REMOTECLIENT_OVERRIDE=""

# RPM COMMAND SWITCHES
export SM_RPM_TARGET="--target sparc-linux"

# USEFUL CONFIGURE ENVIRONMENT VARIABLES
export CFLAGS="$SM_COMPILE_CFLAGS"
export CXXFLAGS="$SM_COMPILE_CXXFLAGS"
export CC="${SM_COMPILE_TARGET}-gcc"
export CXX="${SM_COMPILE_TARGET}-g++"
export AR="${SM_COMPILE_TARGET}-ar"
export AS="${SM_COMPILE_TARGET}-as"
export RANLIB="${SM_COMPILE_TARGET}-ranlib"
export LD="${SM_COMPILE_TARGET}-ld"
export STRIP="${SM_COMPILE_TARGET}-strip"
export PKG_CONFIG="$SM_PATH_SCRIPTS/wrappers/pkg-config"
export PKG_CONFIG_PATH="$SM_PATH_ROOT/usr/lib/pkgconfig"
export PKG_CONFIG_SYSROOT_DIR="$SM_PATH_ROOT"
```

## include/functions.inc

```
#!/bin/bash

# ASK FOR CONFIRMATION
# Usage: confirm <question>
# Returns: 0 -> Yes, 1 -> No
function confirm()
{
  # prompt question and read answer
  echo -n "$@ "
  read -e ANSWER

  # check the answer
  for RESPONSE in y Y yes YES Yes
  do
    if [ "_$ANSWER" == "_$RESPONSE" ]
    then
```

```
        return 0
    fi
  done

  # any answer other than the list above is considerred a "no" answer
  return 1
}

# CHECK RETURN CODE
# Usage: check <program> <code>
function check()
{
  # check error code
  if [[ $2 -ne 0 ]]; then
    echo "!! Error: $1 exited with error code $2."
    exit 1
  fi

  # all good
  return 0
}

# PRINT LINE
# Usage: line <character>
function line()
{
  # get terminal width
  WIDTH=`tput cols`

  # print line
  for I in `seq $WIDTH`; do
    echo -n $1
  done
  echo
}

# CHECK IF CACHED FILE IS MISSING/STALE
# Usage: cache <path>
function cache()
{
  # build file path
  FILE="$SM_PATH_CACHE/$1"

  # first check if file exists
  if [[ ! -f $FILE ]]; then
    echo "missing"
    return 0
  fi

  # get relevant timestamps and age
  MODIFIED=`stat -c %Y $FILE`
  NOW=`date ++%s`
  (( AGE = $NOW - $MODIFIED ))

  # CHECK AGE
  if [[ $AGE -gt $SM_CACHE_MAXAGE ]]; then
    echo "stale"
    return 0
  fi

  # still good
  echo "cached"
```

```
  return 0
}

# PRINT MEEGO REPOSITORY URL
# Usage: repourl <repo> <arch>
function repourl()
{
  # start building url
  REPOSITORY="$SM_REPOSITORY/$1/repos/$2"

  # add "/packages" for binary architectures
  if [[ $2 != "source" ]]; then
    REPOSITORY="$REPOSITORY/packages"
  fi

  # print result
  echo $REPOSITORY
}

# PRINT LOCAL REPOSITORY PATH
# Usage: repourl <repo> <arch>
function repopath()
{
  # start building path
  REPOSITORY="$SM_PATH_REPOSITORY/MeeGo/releases/$SM_VERSION/$1/repos/$2"

  # add "/packages" for binary architectures
  if [[ $2 != "source" ]]; then
    REPOSITORY="$REPOSITORY/packages"
  fi

  # print result
  echo $REPOSITORY
}
```

## buildclean

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [[ $# -ne 0 ]]; then
        echo "Usage: buildclean"
        exit 1
fi

# CLEAN RPMBUILD
echo "-> Cleaning rpmbuild..."

for DIRECTORY in SOURCES SPECS BUILD BUILDROOT TEMP RPMS SRPMS; do
  fakeroot rm -rf $SM_PATH_RPMBUILD/$DIRECTORY/*
  check rm $?
done

echo "-> Done."
```

## buildprepare

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [[ -z $1 || -z $2 || ( ! -z $3 && $3 != "local"  ) ]]; then
  echo "Usage: buildprepare <repo> <package> [local]"
  exit 1
fi

if [[ -z $3 ]]; then
  # GET REPOSITORY URL
  echo "-> Building repository URL..."
  REPOSITORY_URL=`repourl $1 source`
  echo "$REPOSITORY_URL"

  # GET REPOSITORY INDEX IF NEEDED
  REPOSITORY_INDEX_NAME="repository_index_$1_source"
  REPOSITORY_INDEX_PATH="$SM_PATH_CACHE/$REPOSITORY_INDEX_NAME"
  REPOSITORY_INDEX_CACHE=`cache $REPOSITORY_INDEX_NAME`

  if [[ $REPOSITORY_INDEX_CACHE != "cached" ]]; then
    echo "-> No cached index found, downloading..."
    wget -nv -O $REPOSITORY_INDEX_PATH $REPOSITORY_URL
    check wget $?
    touch $REPOSITORY_INDEX_PATH
  else
    echo "-> Cached index found, no need to download..."
  fi

  # GET PACKAGE LIST
  echo "-> Extracting package list..."
  PACKAGE_LIST=`cat $REPOSITORY_INDEX_PATH | grep -E -o "href=\".*\.rpm\"" | grep -E -o "\".*\"" | sed 's/^.//' | sed 's/.$//'`
else
  # GET REPOSITORY PATH
  echo "-> Building repository path..."
  REPOSITORY_PATH=`repopath $1 source`
  echo "$REPOSITORY_PATH"

  # GET PACKAGE LIST
  echo "-> Extracting package list..."
  PACKAGE_LIST=`ls $REPOSITORY_PATH | grep '\.rpm$'`
fi

echo "-> Found `echo $PACKAGE_LIST | wc -w` packages."

# IDENTIFY PACKAGE
echo "-> Identifying package..."
PACKAGE_NAME=`echo $PACKAGE_LIST | tr ' ' '\n' | grep -E "^$2\-[0-9a-zA-Z\~\.\_]+\-[0-9a-zA-Z\~\.\_]+src\.rpm"`

if [[ -z $PACKAGE_NAME ]]; then
  echo "!! Error: package does not exist."
  exit 1
fi

# GET PACKAGE IF NEEDED
if [[ -z $3 ]]; then
  echo "-> Found \"$PACKAGE_NAME\", probing cache..."
```

```
  PACKAGE_PATH="$SM_PATH_CACHE/$PACKAGE_NAME"
  PACKAGE_URL="$REPOSITORY_URL/$PACKAGE_NAME"
  PACKAGE_CACHE=`cache $PACKAGE_NAME`

  if [[ $PACKAGE_CACHE != "cached" ]]; then
    echo "-> No cached package found, downloading..."
    wget -nv -O $PACKAGE_PATH $PACKAGE_URL
    check wget $?
    touch $PACKAGE_PATH
  else
    echo "-> Cached package found, no need to download..."
  fi
else
  echo "-> Found \"$PACKAGE_NAME\", copying..."
  PACKAGE_PATH="$REPOSITORY_PATH/$PACKAGE_NAME"
fi

# CLEAN RPMBUILD
buildclean
check buildclean $?

# EXTRACT PACKAGE
echo "-> Extracting package contents..."
rpmdev-extract -f -C $SM_PATH_RPMBUILD/TEMP $PACKAGE_PATH
check rpmdev-extract $?

# MOVE FILES IN THE CORRECT FOLDERS
echo "-> Moving files.."
for FILE in `find $SM_PATH_RPMBUILD/TEMP/ -name *.spec`; do
  mv -v $FILE $SM_PATH_RPMBUILD/SPECS;
  check mv $?
done

rm -rfv $SM_PATH_RPMBUILD/SOURCES
check rm $?

cp -rv `find $SM_PATH_RPMBUILD/TEMP -mindepth 1 -maxdepth 1 -type d`
    $SM_PATH_RPMBUILD/SOURCES
check cp $?

# PERFORM COMMON TESTS AND DISPLAY WARNINGS
for FILE in `find $SM_PATH_RPMBUILD/SPECS/ -name *.spec`; do
  echo "-> Analyzing \"$FILE\"..."

  RESULT=`cat $FILE | grep '%check'`
  if [[ ! -z $RESULT ]]; then
    echo -e "\tWarning: check section present."
  fi

  RESULT=`cat $FILE | grep '%reconfigure'`
  if [[ ! -z $RESULT ]]; then
    echo -e "\tWarning: %reconfigure macro present."
  fi
done

echo "-> Done."
```

## buildperform

```
#!/bin/bash

```

```
# INCLUDE COMMON HEADERS

source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS

if [[ ! ( $# -eq 0 || ( $# -eq 1 && ( $1 == "prep" || $1 == "force" ) ) ) ]];
    then
  echo "Usage: buildperform [prep|force]"
  exit 1
fi

# GET SPEC FILE AND CHECK FOR MULTIPLES

SPEC_FILE=`find $SM_PATH_RPMBUILD/SPECS -name "*.spec"`
SPEC_COUNT=`find $SM_PATH_RPMBUILD/SPECS -name "*.spec" | wc -l`

case $SPEC_COUNT in
  0)
    echo "-> No spec file found."
    echo "-> Nothing to do."
    exit 0 ;;
  1)
    echo "-> Found 1 spec file." ;;
  *)
    echo "-> Found $SPEC_COUNT spec files."
    echo "-> Sorry, behaviour undefined." ;;
esac

# FIX RPMBUILD CONFIGURATION

#echo "-> Fixing rpmbuild configuration... (Overwriting \"$SM_PATH/.rpmrc\")"
#echo "optflags: sparc $SM_COMPILE_CFLAGS" > $SM_PATH/.rpmrc

# MAKE SURE THE BUILD DIRECTORIES ARE NOT POLLUTED

for DIRECTORY in BUILD BUILDROOT; do
  rm -rfv $SM_PATH_RPMBUILD/$DIRECTORY/*
  check rm $?
done

# SETUP THE RPMBUILD FLAGS

if [[ $1 == "force" ]]; then
  FLAGS="--nodeps"
else
  FLAGS=""
fi

# PERFORM BUILD

if [[ ! -z $1 && $1 == "prep" ]]; then
  echo "-> Executing %prep stage..."
  rpmbuild -bp $SM_RPM_TARGET $SPEC_FILE
  check "rpmbuild" $?
  echo "-> Done."
else
  echo "-> Executing build..."
  rpmbuild -ba $FLAGS $SM_RPM_TARGET $SPEC_FILE
  check "rpmbuild" $?
  echo "-> Done."
```

```
fi
```

## buildcheck

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [ -z "$1" ]; then
  echo "Usage: buildcheck <repo>"
  exit 1
fi

# FIND BINARY PACKAGES
echo "-> Looking for binary packages..."
BINARY_PACKAGES=`find $SM_PATH_RPMBUILD/RPMS -name '*.rpm'`
BINARY_COUNT=`echo -n $BINARY_PACKAGES | wc -w`
echo "-> Found $BINARY_COUNT."

if [[ $BINARY_COUNT -eq 0 ]]; then
  echo "-> Nothing to do."
  exit 0
fi

# GET REPOSITORY URLs
echo "-> Building repository base URL..."
REPOSITORY_URL_BASE=`repourl $1 ia32`
echo "$REPOSITORY_URL_BASE"

# GET REPOSITORY INDEXES AND PACKAGE LISTS IF NEEDED
declare -A REPOSITORY_INDEX_NAME
declare -A REPOSITORY_INDEX_PATH
declare -A REPOSITORY_INDEX_CACHE
declare -A PACKAGE_LIST

for ARCH in i586 i686 noarch; do
  REPOSITORY_INDEX_NAME[$ARCH]="repository_index_${1}_ia32_${ARCH}"
  REPOSITORY_INDEX_PATH[$ARCH]="$SM_PATH_CACHE/${REPOSITORY_INDEX_NAME[$ARCH]}"
  REPOSITORY_INDEX_CACHE[$ARCH]=`cache ${REPOSITORY_INDEX_NAME[$ARCH]}`

  if [[ ${REPOSITORY_INDEX_CACHE[$ARCH]} != "cached" ]]; then
    echo "-> No cached index found for arch \"$ARCH\", downloading..."
    wget -nv -O ${REPOSITORY_INDEX_PATH[$ARCH]} $REPOSITORY_URL_BASE/$ARCH
    check wget $?
    touch ${REPOSITORY_INDEX_PATH[$ARCH]}
  else
    echo "-> Cached index found for arch \"$ARCH\", no need to download..."
  fi

  echo "-> Extracting package list for arch \"$ARCH\"..."
  PACKAGE_LIST[$ARCH]=`cat ${REPOSITORY_INDEX_PATH[$ARCH]} | grep -E -o "href=\".*\.rpm\"" | grep -E -o "\".*\"" | sed 's/^.//' | sed 's/.$//'`
  echo "-> Found `echo ${PACKAGE_LIST[$ARCH]} | wc -w` packages."
done

# FOR EACH LOCAL PACKAGE
for PACKAGE_PATH in `echo $BINARY_PACKAGES`; do
  # GET REAL PACKAGE NAME
```

```
PACKAGE_NAME=`basename $PACKAGE_PATH | sed -r 's/\-[0-9a-zA-Z\~\.\_]+\-[0-9a-zA-Z\~\.\_]+(sparc|noarch).rpm$//'`
echo "-> Investigating \"$PACKAGE_NAME\"..."

# CHECK BINARIES ARCHITECTURE
echo -e "\t-> Verifying that all ELF binaries are Sparc..."
if [[ -d $SM_PATH_RPMBUILD/TEMP/buildcheck ]]; then
  sudo rm -rf $SM_PATH_RPMBUILD/TEMP/buildcheck
  check rm $?
fi
mkdir $SM_PATH_RPMBUILD/TEMP/buildcheck
check mkdir $?
rpmdev-extract -f -C $SM_PATH_RPMBUILD/TEMP/buildcheck $PACKAGE_PATH > /dev/null
check rpmdev-extract $?
PACKAGE_TEMP_DIR=`find $SM_PATH_RPMBUILD/TEMP/buildcheck -mindepth 1 -maxdepth 1 -type d`

for FILE in `find $PACKAGE_TEMP_DIR`; do
  FILE_RELATIVE=${FILE#$PACKAGE_TEMP_DIR}
  if [[ -L $FILE ]]; then
    echo -e "\t\t-> Found link \"$FILE_RELATIVE\" to \"`readlink $FILE`\" "
  else
    OUTPUT=`sparc-leon-linux-gnu-readelf -h $FILE 2>&1`
    if [[ $? -eq 0 ]]; then
      echo -ne "\t\t-> Found binary \"$FILE_RELATIVE\"... "
      CHECK=`echo $OUTPUT | grep "Machine: Sparc"`
      if [[ $? -eq 0 ]]; then
        echo "OK"
      else
        echo "BAD"
        line "-"
        echo $OUTPUT
        line "-"
      fi
    fi
  fi
done

sudo rm -rf $SM_PATH_RPMBUILD/TEMP/buildcheck
check rm $?

# IDENTIFY PACKAGE
echo -e "\t-> Trying to locate package in the official repository..."

PACKAGE_FOUND=0
for ARCH in i586 i686 noarch; do
  PACKAGE_NAME_OFFICIAL=`echo ${PACKAGE_LIST[$ARCH]} | tr ' ' '\n' | grep -E "^$PACKAGE_NAME\-[0-9a-zA-Z\~\.\_]+\-[0-9a-zA-Z\~\.\_]+$ARCH\.rpm"`
  if [[ -z $PACKAGE_NAME_OFFICIAL ]]; then
    echo -e "\t-> Not found in arch \"$ARCH\"."
  else
    echo -e "\t-> Found in arch \"$ARCH\" with name \"$PACKAGE_NAME_OFFICIAL\"."
    PACKAGE_ARCH_OFFICIAL=$ARCH
    PACKAGE_FOUND=1
    break
  fi
done

if [[ $PACKAGE_FOUND -eq 0 ]]; then
  echo "!! Error: package does not exist in official repository."
```

```
    continue # there may be more packages to investigate, so we do not exit
  fi

  # GET PACKAGE FROM OFFICIAL REPOSITORY IF NEEDED
  PACKAGE_PATH_OFFICIAL="$SM_PATH_CACHE/$PACKAGE_NAME_OFFICIAL"
  PACKAGE_URL_OFFICIAL="$REPOSITORY_URL_BASE/$ARCH/$PACKAGE_NAME_OFFICIAL"
  PACKAGE_CACHE_OFFICIAL=`cache $PACKAGE_NAME_OFFICIAL`

  if [[ $PACKAGE_CACHE_OFFICIAL != "cached" ]]; then
    echo -e "\t-> No cached package found, downloading..."
    wget -nv -O $PACKAGE_PATH_OFFICIAL $PACKAGE_URL_OFFICIAL
    check wget $?
    touch $PACKAGE_PATH_OFFICIAL
  else
    echo -e "\t-> Cached package found, no need to download..."
  fi

  # GET PACKAGE CONTENT LISTS
  echo -e "\t\t-> Comparing package contents..."

  rpm -qlp $PACKAGE_PATH > $SM_PATH_RPMBUILD/TEMP/${$}_list_local
  check rpm $?

  rpm -qlp $PACKAGE_PATH_OFFICIAL > $SM_PATH_RPMBUILD/TEMP/${$}_list_official
  check rpm $?

  # COMPUTE DIFFERENCES AND DISPLAY RESULTS
  diff -y $SM_PATH_RPMBUILD/TEMP/${$}_list_local $SM_PATH_RPMBUILD/TEMP/${$}
      _list_official > $SM_PATH_RPMBUILD/TEMP/${$}_diff_all
  RESULT=$?
  cat $SM_PATH_RPMBUILD/TEMP/${$}_diff_all | grep -E '>|\||<' >
      $SM_PATH_RPMBUILD/TEMP/${$}_diff

  if [[ $RESULT -eq 0 ]]; then
    echo -e "\t\t-> GOOD: Both packages have the same file list."
  else
    echo -e "\t\t-> BAD: There are some differences:"
    line "-"
    echo -e " Local\t\t\t\t\t\t\t\tOfficial"
    line "-"
    cat $SM_PATH_RPMBUILD/TEMP/${$}_diff | sed 's/^/ /g'
    line "-"
  fi

  rm $SM_PATH_RPMBUILD/TEMP/${$}_list_local
  check rm $?
  rm $SM_PATH_RPMBUILD/TEMP/${$}_list_official
  check rm $?
  rm $SM_PATH_RPMBUILD/TEMP/${$}_diff_all
  check rm $?
  rm $SM_PATH_RPMBUILD/TEMP/${$}_diff
  check rm $?
done
```

## buildsave

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc
```

```

# CHECK ARGUMENTS
if [ -z "$1" ]; then
  echo "Usage: buildsave <repo>"
  exit 1
fi

# MOVE BINARY PACKAGES
echo "-> Looking for binary packages..."
BINARY_COUNT=`find $SM_PATH_RPMBUILD/RPMS -name '*.rpm' | wc -l`
BINARY_PATH=`repopath $1 sparc`
echo "-> Found $BINARY_COUNT."

if [[ $BINARY_COUNT -gt 0 ]]; then
  cp -rv $SM_PATH_RPMBUILD/RPMS/* $BINARY_PATH
  check mv $?
  echo "-> Updating repository metadata..."
  #createrepo -c $SM_PATH_CACHE/repository -pd $BINARY_PATH
  #check createrepo $?
fi

# MOVE SOURCE PACKAGES
echo "-> Looking for source packages..."
SOURCE_COUNT=`find $SM_PATH_RPMBUILD/SRPMS -name '*.rpm' | wc -l`
SOURCE_PATH=`repopath $1 source`
echo "-> Found $SOURCE_COUNT."

if [[ $SOURCE_COUNT -gt 0 ]]; then
        cp -rv $SM_PATH_RPMBUILD/SRPMS/* $SOURCE_PATH
        check mv $?
  echo "-> Updating repository metadata..."
  #createrepo -c $SM_PATH_CACHE/repository -pd $SOURCE_PATH
  #check createrepo $?
fi

echo "-> Done."
```

## repoclean

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [[ $# -ne 0 ]]; then
        echo "Usage: repoclean"
        exit 1
fi

# ASK FOR CONFIRMATION
confirm "-> This will completely wipe your local (generated) repository. Would
    you like to continue? [y/N]"
if [[ $? -ne 0 ]]; then
        echo "-> Ok, aborting."
        exit 0
fi

# DELETE ROOT DIRECTORY CONTENTS
echo "-> Deleting local repository..."
```

```
rm -rfv $SM_PATH_REPOSITORY
check rm $?
echo "-> Done."

echo "-> Recreating repository structure..."
for REPOSITORY_NAME in core netbook; do
  mkdir -pv `repopath $REPOSITORY_NAME source`
  check mkdir $?
  mkdir -pv `repopath $REPOSITORY_NAME sparc`
  check mkdir $?
done

echo "-> Done."
```

## rootpopulate

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [[ $# -ne 0 ]]; then
        echo "Usage: rootpopulate"
        exit 1
fi

PACKAGE_CURRENT=1
PACKAGE_COUNT=`find $SM_PATH/repository -wholename */releases/$SM_VERSION/*/repos/sparc/packages/*.rpm -not -name *.src.rpm | wc -l`

for PACKAGE_FILE in `find $SM_PATH/repository -wholename */releases/$SM_VERSION/*/repos/sparc/packages/*.rpm -not -name *.src.rpm`; do
  echo -e "\t($PACKAGE_CURRENT/$PACKAGE_COUNT) $PACKAGE_FILE"
  (( PACKAGE_CURRENT++ ))
  sudo rpm -i --noscripts --nodeps --ignorearch --ignoreos --force --root=$SM_PATH_ROOT $PACKAGE_FILE
  check rpm $?
done
```

## rootinstall

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [[ $# -ne 2 ]]; then
        echo "Usage: rootinstall <repo> <package>"
        exit 1
fi

# IDENTIFY PACKAGE
REPOSITORY_PATH=`repopath $1 sparc`
PACKAGE_NAME=`echo "$2" | sed 's/\+/\\\+/g'`
PACKAGE_PATH=`find $REPOSITORY_PATH -regextype posix-extended -type f -regex "^$REPOSITORY_PATH/(sparc|noarch)/$PACKAGE_NAME\-[0-9a-zA-Z\~\.\_\+]+\-[0-9a-zA-Z\~\.\_\+]+(sparc|noarch)\.rpm$"`
```

```
if [[ -z $PACKAGE_PATH ]]; then
        echo "-> Error: package does not exist."
        exit 1
else
      echo "-> Identified package \"$PACKAGE_PATH\"."
fi

# INSTALL PACKAGE
echo "-> Installing package..."
sudo rpm -i -vv --noscripts --nodeps --ignorearch --ignoreos --force --root=
    $SM_PATH_ROOT $PACKAGE_PATH
check rpm $?
echo "-> Done."
```

## workaround

```
#!/bin/bash

# INCLUDE COMMON HEADERS
source `dirname $0`/include/env.inc
source `dirname $0`/include/functions.inc

# CHECK ARGUMENTS
if [[ $# -lt 2 ]]; then
  echo "Usage: workaround <name> <action> [parameter]"
  exit 1
fi

# CHECK WORKAROUND and ACTION
WORKAROUND_ACTIONS=`find $SM_PATH_SCRIPTS/workarounds -name $1-* -exec basename
    {} \; | sed "s/$1-//g"`

if [[ -z $WORKAROUND_ACTIONS ]]; then
  echo "!! Error: workaround not found."
  exit 1
fi

if [[ ! -f $SM_PATH_SCRIPTS/workarounds/$1-$2 ]]; then
  echo "!! Error: workaround \"$1\" has no action \"$2\"."
  exit 1
fi

# CHECK STATUS
if [[ -f `dirname $0`/workarounds/status/$1 ]]; then
  STATUS="enabled"
else
  STATUS="disabled"
fi

# EXECUTE REQUEST
case $2 in
  status)
    echo "-> Workaround $STATUS." ;;
  enable)
    if [[ $1 == "all" ]]; then
      `dirname $0`/workarounds/$1-$2
      exit 0
    fi
    if [[ $STATUS == "enabled" ]]; then
      echo "!! Error: workaround already enabled."
```

```
      exit 1
    fi
    touch `dirname $0`/workarounds/status/$1
    `dirname $0`/workarounds/$1-$2
    echo "-> Workaround enabled." ;;
  disable)
    if [[ $1 == "all" ]]; then
      `dirname $0`/workarounds/$1-$2
      exit 0
    fi
    if [[ $STATUS == "disabled" ]]; then
      echo "!! Error: workaround already disabled."
      exit 1
    fi
    rm `dirname $0`/workarounds/status/$1
    `dirname $0`/workarounds/$1-$2
    echo "-> Workaround disabled." ;;
  *)
    if [[ $1 == "all" ]]; then
      `dirname $0`/workarounds/$1-$2
      exit 0
    fi
    if [[ $STATUS == "disabled" ]]; then
      echo "!! Error: workaround is disabled."
      exit 1
    fi
    $SM_PATH_SCRIPTS/workarounds/$1-$2 $3
    if [[ $? -eq 0 ]]; then
      echo "-> Workaround action executed."
    fi
    ;;
esac
```

### C.1.3 Kernel and Loader patches

This section contains the patches that were applied to the Linux kernel and the glibc loader in order to implement the transparent remote execution capabilities.

#### kernel-remote-execution.patch

```
--- ubuntu-lucid.orig/fs/exec.c 2011-06-17 14:29:19.000000000 +0200
+++ ubuntu-lucid.new/fs/exec.c  2011-06-17 14:32:47.000000000 +0200
@@ -56,9 +56,11 @@
 #include <linux/fsnotify.h>
 #include <linux/fs_struct.h>
 #include <linux/pipe_fs_i.h>
+#include <linux/elf.h>

 #include <trace/events/fs.h>

+#include <asm/byteorder.h>
 #include <asm/uaccess.h>
 #include <asm/mmu_context.h>
 #include <asm/tlb.h>
@@ -1346,6 +1348,46 @@

 EXPORT_SYMBOL(search_binary_handler);

+int bin_type_sparc(const char *filename, const char **argv)
```

```
+{
+ int fd, len;
+ Elf32_Ehdr hdr;
+ mm_segment_t old_fs = get_fs();
+ set_fs(KERNEL_DS);
+
+ fd = sys_open(filename, O_RDONLY, 0);
+ if (fd < 0)
+   return 0;
+
+       /* Read ELF header */
+ len = sys_read(fd, (char __user *)&hdr, sizeof(hdr));
+ if (len != sizeof(hdr)) {
+   sys_close(fd);
+   return 0;
+ }
+
+       /* Check for 32-bit, Executable, SPARC and Big-endian binary */
+ if ((hdr.e_ident[EI_MAG0] == ELFMAG0) &&
+     (hdr.e_ident[EI_MAG1] == ELFMAG1) &&
+     (hdr.e_ident[EI_MAG2] == ELFMAG2) &&
+     (hdr.e_ident[EI_MAG3] == ELFMAG3) &&
+     (hdr.e_ident[EI_CLASS] == ELFCLASS32) &&
+     (hdr.e_ident[EI_DATA] == ELFDATA2MSB)) {
+   if ((be16_to_cpu(hdr.e_type) == ET_EXEC) &&   /* EXEC */
+       (be16_to_cpu(hdr.e_machine) == 0x0002)) {   /* SPARC */
+       if (argv && argv[0])
+       printk(KERN_DEBUG "check_elf: SPARC binary %s (argv[0]=%s)\n", filename,
     argv[0]);
+     else
+       printk(KERN_DEBUG "check_elf: SPARC binary %s\n", filename);
+     return 1;
+   }
+ }
+ sys_close(fd);
+ set_fs(old_fs);
+
+ return 0;
+}
+
 /*
  * sys_execve() executes a new program.
  */
@@ -1359,6 +1401,7 @@
  struct files_struct *displaced;
  bool clear_in_exec;
  int retval;
+ char *remoteclient = "/home/meego/scripts/remoteclient";

  retval = unshare_files(&displaced);
  if (retval)
@@ -1379,6 +1422,11 @@
  clear_in_exec = retval;
  current->in_execve = 1;

+ if (bin_type_sparc(filename, (const char **)argv)) {
+   printk(KERN_DEBUG "remote execution activated\n");
+   filename = remoteclient;
+ }
+
  file = open_exec(filename);
  retval = PTR_ERR(file);
```

```
  if (IS_ERR(file))
```

### loader-keep-searching.patch

```
diff -Naur glibc-2.11-12-g24c0bf7.orig/elf/dl-load.c glibc-2.11-12-g24c0bf7.new/
    elf/dl-load.c
--- glibc-2.11-12-g24c0bf7.orig/elf/dl-load.c 2011-03-09 11:09:20.000000000
    +0100
+++ glibc-2.11-12-g24c0bf7.new/elf/dl-load.c  2011-03-09 10:12:16.000000000
    +0100
@@ -1683,10 +1683,15 @@
      }
    else if (ehdr->e_ident[EI_DATA] != byteorder)
      {
-       if (BYTE_ORDER == BIG_ENDIAN)
+       /* SM: we want to ignore this file and keep searching
+    even though the byte order is wrong because the cause might
+    just be a different architecture */
+           goto close_and_out;
+
+       /*if (BYTE_ORDER == BIG_ENDIAN)
    errstring = N_("ELF file data encoding not big-endian");
       else
-   errstring = N_("ELF file data encoding not little-endian");
+   errstring = N_("ELF file data encoding not little-endian");*/
      }
    else if (ehdr->e_ident[EI_VERSION] != EV_CURRENT)
      errstring
```

## C.1.4 Remote execution gateway

This section provides the source code for the remote execution gateway.

### remoteclient.c

```
#define IGNORE_VARIABLES ":LD_LIBRARY_PATH:RPM_SOURCE_DIR:RPM_OPT_FLAGS:
    RPM_BUILD_ROOT:RPM_PACKAGE_NAME:RPM_OS:RPM_ARCH:RPM_BUILD_DIR:RPM_DOC_DIR:
    RPM_PACKAGE_RELEASE:RPM_PACKAGE_VERSION:HOSTNAME:SM_COMPILE_TARGET:SM_PATH:
    SHELL:TERM:HISTSIZE:SM_PATH_CACHE:OLDPWD:SM_COMPILE_CXXFLAGS:
    AG_SERVICE_TYPES:SM_PATH_CONFIGURE_FLAGS:SM_PATH_CONFIGURE_TARGET:USER:
    LS_COLORS:SM_REMOTEPORT:AG_SERVICES:MAIL:PATH:SM_PATH_CONFIGURE_CACHE:
    SM_VERSION:PWD:SM_COMPILE_HOST:SM_PATH_SCRIPTS:SM_PATH_CONFIGURE_HOST:
    SM_PATH_CONFIGURE_BUILD:SM_PATH_LOGS:SM_RPM_TARGET:SM_COMPILE_BUILD:
    SM_COMPILE_LDFLAGS:HISTCONTROL:SM_COMPILE_CFLAGS:SHLVL:HOME:
    SM_SSH_INTO_NATIVE:GNOME_DESKTOP_SESSION_ID:SM_REPOSITORY:AG_PROVIDERS:
    LOGNAME:SM_EDITOR:CVS_RSH:LESSOPEN:SM_PATH_REPOSITORY:DISPLAY:M_DECORATED:
    SM_CACHE_MAXAGE:G_BROKEN_FILENAMES:_:"
#define T_BUFFER 1024
#define Q_SINGLE 1
#define Q_DOUBLE 2

#define _GNU_SOURCE

#ifdef _DEBUG_ON
#define DEBUG_MACRO(x) x
#else
#define DEBUG_MACRO(x)
#endif

```

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <syslog.h>

struct t_buffer {
  int size, length;
  char *data;
};

int search_string(char *needle, char *haystack)
{
  int result;
  char *search;

  if( (search = calloc(strlen(needle) + 3, sizeof(char))) == NULL) {
    perror("calloc failed");
    exit(EXIT_FAILURE);
  }

  search[0] = '\0';

  strcat(search, ":");
  strcat(search, needle);
  strcat(search, ":");

  result = (strstr(haystack, search) == NULL) ? 0 : 1;
  free(search);
  return result;
}

int
should_export(char *name)
{
  return search_string(name, IGNORE_VARIABLES) == 1 ? 0 : 1;
}

void
buffer_init(struct t_buffer *buffer)
{
  if( (buffer->data = calloc(T_BUFFER, sizeof(char))) == NULL) {
    perror("calloc failed");
    exit(EXIT_FAILURE);
  }

  buffer->size = 1;
  buffer->length = 1;
  buffer->data[0] = '\0';
}

void
buffer_append(struct t_buffer *buffer, char *append)
{
  int length;

  length = strlen(append);

  if(buffer->length + length > buffer->size * T_BUFFER) {
    do buffer->size = buffer->size + 1;
    while(buffer->length + length > buffer->size * T_BUFFER);
```

```
    if( (buffer->data = realloc(buffer->data, buffer->size * T_BUFFER * sizeof(char))) == NULL) {
      perror("calloc failed");
      exit(EXIT_FAILURE);
    }
  }

  buffer->length += length;
  strcat(buffer->data, append);
}

int
how_to_quote(char *string)
{
  return (strstr(string, "'") == NULL) ? Q_SINGLE : Q_DOUBLE;
}

int
is_prefix(char *path, char *prefix)
{
  int i;

  for(i = 0; path[i] != '\0' && prefix[i] != '\0' && path[i] == prefix[i]; i++);
  return prefix[i] == '\0' ? 1 : 0;
}

int
main(int argc, char **argv, char **envp)
{
  int i, q;
  char *arguments[4], *ssh, *directory, *variable, *token, *cursor, *path, *rpmbuild, *root, *library, *override;
  struct t_buffer command;

  DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient starting..."));

  for(i = 0; i < argc; i++)
    DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient : argv[%d] = \"%s\"", i, argv[i]));

  // check if we want to force local execution anyway
  if( (override = getenv("SM_REMOTECLIENT_OVERRIDE")) != NULL) {
    if(search_string(basename(argv[0]), override)) {
      DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient override condition detected..."));
      argv[0] = basename(argv[0]);
      execvpe(argv[0], argv, envp);
      // if we arrive here, something wrong happened
      perror("execvpe failed");
      return EXIT_FAILURE;
    }
  }

  // get current working directory
  if( (directory = get_current_dir_name()) == NULL) {
    perror("get_current_dir_name failed");
    return EXIT_FAILURE;
  }

  DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient : directory = \"%s\"", directory));

```

```
// get SSH string
if( (ssh = getenv("SM_SSH_INTO_NATIVE")) == NULL) {
  fprintf(stderr, "error: $SM_SSH_INTO_NATIVE must be set\n");
  return EXIT_FAILURE;
}

// get rpmbuild path
if( (rpmbuild = getenv("SM_PATH_RPMBUILD")) == NULL) {
  fprintf(stderr, "error: $SM_PATH_RPMBUILD must be set\n");
  return EXIT_FAILURE;
}

// get root path
if( (root = getenv("SM_PATH_ROOT")) == NULL) {
  fprintf(stderr, "error: $SM_PATH_ROOT must be set\n");
  return EXIT_FAILURE;
}

DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient : ssh = \"%s\"", ssh));
DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient creating command..."));

// create command
buffer_init(&command);
buffer_append(&command, "cd ");
buffer_append(&command, directory);
buffer_append(&command, "; export PATH=\"");

if( (token = getenv("PATH")) != NULL) {
  if( (path = strdup(token)) == NULL) {
    perror("strdup failed");
    return EXIT_FAILURE;
  }
  for(token = strtok_r(path, ":", &cursor); token != NULL; token = strtok_r(NULL, ":", &cursor)) {
    if(is_prefix(token, rpmbuild)) {
      buffer_append(&command, token);
      buffer_append(&command, ":");
    }
  }
  free(path);
}

buffer_append(&command, "/home/meego/scripts/wrappers:");
buffer_append(&command, root);
buffer_append(&command, "/bin:");
buffer_append(&command, root);
buffer_append(&command, "/usr/bin:");
buffer_append(&command, "$PATH\";");

buffer_append(&command, " export LD_LIBRARY_PATH='");

if( (token = getenv("LD_LIBRARY_PATH")) != NULL) {
  if( (library = strdup(token)) == NULL) {
    perror("strdup failed");
    return EXIT_FAILURE;
  }
  for(token = strtok_r(library, ":", &cursor); token != NULL; token = strtok_r(NULL, ":", &cursor)) {
      buffer_append(&command, token);
      buffer_append(&command, ":");
  }
  free(library);
```

```
  }

  buffer_append(&command, root);
  buffer_append(&command, "/lib:");
  buffer_append(&command, root);
  buffer_append(&command, "/usr/lib");
  buffer_append(&command, "';");

  for(i = 0; envp[i] != NULL; i++) {
    DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient envp[%d] = \"%s\"", i, envp[i])
        );
    variable = strdup(envp[i]);
    token = strtok_r(variable, "=", &cursor);
    if(should_export(token)) {
      buffer_append(&command, " export ");
      buffer_append(&command, token);
      buffer_append(&command, "='");
      token = strtok_r(NULL, "=", &cursor);
      DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient token = \"%s\"", token));
      if(token != NULL) // strok_r would have returned an empty string
        buffer_append(&command, token);
      buffer_append(&command, "';");
    }
    free(variable);
  }

  for(i = 0; i < argc; i++) {
    q = how_to_quote(argv[i]);
    buffer_append(&command, q == Q_SINGLE ? " '" : " \"");
    buffer_append(&command, argv[i]);
    buffer_append(&command, q == Q_SINGLE ? "'" : "\"");
  }

  buffer_append(&command, "; exit $?;");

  DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient creating arguments..."));

  // build arguments
  arguments[0] = "ssh";
  arguments[1] = strdup(ssh + 4); // remove the "ssh " prefix
  arguments[2] = command.data;
  arguments[3] = NULL;

  DEBUG_MACRO(syslog(LOG_NOTICE, "remoteclient invoking SSH..."));

  // run remote command
  syslog(LOG_NOTICE, "remoteclient <<< %s >>>\n", command.data);
  execvp("ssh", arguments);

  // if we arrive here, something wrong happened
  perror("execvp failed");
  return EXIT_FAILURE;
}
```

# C.2 Final build environment and OBS

## C.2.1 OBS patches

### bs_dispatch.patch

```
--- orig/usr/lib/obs/server/bs_dispatch 2011-06-21 15:32:53.000000000 +0200
+++ new/usr/lib/obs/server/bs_dispatch  2011-06-21 15:47:54.000000000 +0200
@@ -75,16 +75,18 @@
 my $port = 5252;        #'RR'
 $port = $1 if $BSConfig::reposerver =~ /:(\d+)$/;

+# SM: the table has been modified to build sparc on intel machines
+
 my %cando = (
 # this code sucks and is on the list to be rewritten
 # switch on next 3 lines if you want arm, mips, ppc and sh4 qemu emulated builds on a x86 worker
-#  'i586'    => [          'i586',          'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4'],
-#  'i686'    => [          'i586', 'i686', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4'],
-#  'x86_64'  => ['x86_64', 'i586', 'i686', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4'],
+  'i586'    => [          'i586',                            'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4', 'sparcv8', 'sparc' ],
+  'i686'    => [          'i586',          'i686',           'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4', 'sparcv8', 'sparc' ],
+  'x86_64'  => ['x86_64', 'i586:linux32', 'i686:linux32', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4', 'sparcv8', 'sparc' ],
 # switch on next 3 lines if you want only arm qemu emulated builds on a x86 worker
-  'i586'    => [          'i586',          'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel',       'sh4'],
-  'i686'    => [          'i586', 'i686', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel',       'sh4'],
-  'x86_64'  => ['x86_64', 'i586', 'i686', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel',       'sh4'],
+#  'i586'    => [          'i586',          'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel',       'sh4'],
+#  'i686'    => [          'i586', 'i686', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel',       'sh4'],
+#  'x86_64'  => ['x86_64', 'i586', 'i686', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel',       'sh4'],
 #
   'ppc'     => [
       'ppc'                ],
   'ppc64'   => [
       'ppc', 'ppc64',      ],
@@ -99,7 +101,8 @@
   'ia64'    => ['ia64'],
   's390'    => ['s390'],
```

```
   's390x'   => ['s390x', 's390'],
-  'sparc'   => ['sparcv8', 'sparc'],
+  'sparc'   => ['sparc'],
+  'sparcv8' => ['sparcv8', 'sparc'],
   'sparc64' => ['sparc64v', 'sparc64', 'sparcv9v', 'sparcv9', 'sparcv8:linux32', 'sparc:linux32'],
   'mips'    => ['mips'],
   'mips64'  => ['mips64', 'mips'],
```

## bs_publish.patch

```
--- orig/usr/lib/obs/server/bs_publish  2011-06-21 15:32:53.000000000 +0200
+++ new/usr/lib/obs/server/bs_publish 2011-06-21 15:47:53.000000000 +0200
@@ -397,6 +397,7 @@
 ARCH.armv8el arm armel armv4l armv5el armv5tel armv6el armv6l armv6vl armv7el armv7l armv7vl armv8el armv8l armv8vl noarch
 ARCH.i686 i686 i586 i486 i386 noarch
 ARCH.i586 i586 i486 i386 noarch
+ARCH.sparc sparc noarch
 DEFAULTBASE i586
 DESCRDIR descr
 DATADIR .
@@ -497,7 +498,7 @@
   my ($extrep, $projid, $repoid, $signargs, $pubkey, $repoinfo, $patterns) = @_;

   deletepatterns_rpmmd($extrep);
-  return unless @{$patterns || []};
+  #return unless @{$patterns || []};

   # create patterns data structure
   my @pats;
@@ -508,6 +509,8 @@
   my $pats = {'pattern' => \@pats, 'count' => scalar(@pats)};
   writexml("$extrep/repodata/patterns.xml", undef, $pats, $BSXML::patterns);
   qsystem('modifyrepo', "$extrep/repodata/patterns.xml", "$extrep/repodata") && print("    modifyrepo failed: $?\n");
+
+  # SM: we want to retain the patterns.xml file, otherwise the image creator will not support package groups
   unlink("$extrep/repodata/patterns.xml");

 #  for my $pattern (@{$patterns || []}) {
@@ -1171,8 +1174,10 @@
     deletepatterns_ymp($extrep, $projid, $repoid, $signargs, $pubkey);
   }
   if ($patterntype{'rpm-md'}) {
+    print "creating patterns for rpm-md: $extrep, $projid, $repoid, $signargs, $pubkey, $repoinfo, $patterns\n";
     createpatterns_rpmmd($extrep, $projid, $repoid, $signargs, $pubkey, $repoinfo, $patterns);
   } else {
+    print "deleting patterns for rpm-md";
     deletepatterns_rpmmd($extrep, $projid, $repoid, $signargs, $pubkey);
   }
   if ($patterntype{'comps'}) {
```

## bs_worker.patch

```
--- orig/usr/lib/obs/server/bs_worker 2011-06-21 15:32:53.000000000 +0200
+++ new/usr/lib/obs/server/bs_worker  2011-06-21 15:47:54.000000000 +0200
```

```
@@ -84,6 +84,8 @@
 my $xenstore_maxsize = 20 * 1000000;
 my $gettimeout = 3600; # 1 hour timeout to avoid forever hanging workers

+# SM: the table has been modified to build sparc and sparcv8 on intel machines
+
 my %cando = (
   'armv4l'  => [                                        'armv4l'                                        ],
   'armv5el' => [                                        'armv4l', 'armv5el'                             ],
@@ -93,13 +95,13 @@
   'sh4'     => [                                        'sh4' ],
 # this code sucks and is on the list to be rewritten
 # switch on next 3 lines if you want arm, mips, ppc and sh4 qemu emulated builds on a x86 worker
-#  'i586'    => [           'i586',                     'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4' ],
-#  'i686'    => [           'i586',        'i686',      'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4' ],
-#  'x86_64'  => ['x86_64', 'i586:linux32', 'i686:linux32', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4' ],
+  'i586'    => [           'i586',                     'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4', 'sparcv8', 'sparc' ],
+  'i686'    => [           'i586',        'i686',      'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4', 'sparcv8', 'sparc' ],
+  'x86_64'  => ['x86_64', 'i586:linux32', 'i686:linux32', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'mips64', 'mips64el', 'ppc', 'ppc64', 'sh4', 'sparcv8', 'sparc' ],
 # switch on next 3 lines if you want only arm qemu emulated builds on a x86 worker
- 'i586'    => [           'i586',                     'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'sh4' ],
- 'i686'    => [           'i586',        'i686',      'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'sh4' ],
- 'x86_64'  => ['x86_64', 'i586:linux32', 'i686:linux32', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'sh4' ],
+# 'i586'    => [           'i586',                     'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'sh4' ],
+# 'i686'    => [           'i586',        'i686',      'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'sh4' ],
+# 'x86_64'  => ['x86_64', 'i586:linux32', 'i686:linux32', 'armv4l', 'armv5el', 'armv6el', 'armv7el', 'armv8el', 'mips', 'mipsel', 'sh4' ],
 #
   'parisc'  => ['hppa', 'hppa64:linux64'],
   'parisc64'=> ['hppa64', 'hppa:linux32'],
@@ -108,8 +110,9 @@
   'ia64'    => ['ia64'],
   's390'    => ['s390'],
   's390x'   => ['s390x', 's390:s390'],
-  'sparc'   => ['sparcv8', 'sparc'],
-  'sparc64' => ['sparc64v', 'sparc64', 'sparcv9v', 'sparcv9', 'sparcv8:linux32', 'sparc:linux32'],
+  'sparc'   => ['sparc'],
```

```
+  'sparcv8'   => ['sparcv8', 'sparc'],
+  'sparc64' => ['sparc64v', 'sparc64', 'sparcv9v', 'sparcv9', 'sparcv8', 'sparc'],
   'mips'    => ['mips'],
   'mips64'  => ['mips64', 'mips:mips32'],
 );
@@ -179,6 +182,8 @@

        --xen       : enable xen

+     --qemu      : enable qemu
+
        --device    : set kvm or xen root device (default is <root>/root file)

        --swap      : set kvm or xen swap device (default is <root>/swap file)
@@ -290,6 +295,11 @@
      shift @ARGV;
      next;
    }
+   if ($ARGV[0] eq '--qemu') {
+     $vm = ' --qemu';
+     shift @ARGV;
+     next;
+   }
    if ($ARGV[0] eq '--xendevice' || $ARGV[0] eq '--device') {
      shift @ARGV;
      $vm_root = shift @ARGV;
@@ -1200,7 +1210,9 @@
        push @meta, sort {substr($a, 34) cmp substr($b, 34)} @m;
      }
    }
-   die("getbinaries: missing packages: @todo\n") if @todo;
+   # SM: temporarily disabled
+   # die("getbinaries: missing packages: @todo\n") if @todo;
+   print("SM: normally we would die here with: getbinaries: missing packages: @todo\n") if @todo;

    if (!$kiwimode) {
      # generate meta data
@@ -1395,7 +1407,9 @@
      } elsif (-e "$pkgdir/$bin.deb") {
        push @rpmlist, "$bin $pkgdir/$bin.deb";
      } else {
-       die("missing package: $bin\n");
+    # SM: temporarily disabled
+    # die("missing package: $bin\n");
+    print("SM: normally we would die here with: missing package: $bin\n");
      }
    }
    push @rpmlist, "localkiwi $localkiwi/localkiwi.rpm" if $localkiwi && -e "$localkiwi/localkiwi.rpm";
```

## build.patch

```
--- orig/usr/lib/build/build  2011-06-21 15:32:06.000000000 +0200
+++ new/usr/lib/build/build 2011-06-21 16:52:34.000000000 +0200
@@ -331,6 +331,8 @@
 toshellscript()
 {
  echo "#!/bin/sh -x"
+ echo "export PATH=\"/usr/lib/distcc/bin:/sbin:/usr/sbin:$PATH\";"
```

```
+ echo "export DISTCC_HOSTS=\"10.0.2.2\";"
  echo -n exec
  shellquote "$@"
  echo
@@ -592,7 +594,7 @@
  if ! test -b "$VM_SWAP" ; then
      rm -f "$VM_SWAP"
      umask 027
-     mknod "$VM_SWAP" b 3 2
+     mknod "$VM_SWAP" b 8 16
      umask 022
  fi
  swapon -v "$VM_SWAP" || exit 1
@@ -795,7 +797,7 @@
  BUILD_DIST="$ARG"
  shift
       ;;
-      *-xen|*-kvm|--uml|--qemu)
+      *-xen|*-kvm|--uml|*--qemu)
  VM_TYPE=${PARAM##*-}
  if [ -n "$ARG" ]; then
      VM_IMAGE="$ARG"
@@ -1514,12 +1516,20 @@
                    KVM64_WORKAROUND="-cpu kvm64"
    fi

-   set -- $qemu_bin -no-reboot -nographic -net none $KVM64_WORKAROUND \
-       -kernel $vm_kernel \
-       -initrd $vm_initrd \
-       -append "root=$qemu_rootdev panic=1 quiet no-kvmclock rw elevator=noop console=ttyS0 init=$vm_init_script" \
-       ${MEMSIZE:+-m $MEMSIZE} \
-       "${qemu_args[@]}"
+   if [[ "$BUILD_ARCH" == 'sparc' ]]; then
+     set -- qemu-system-sparc -M SS-10 -no-reboot -nographic \
+       -kernel /opt/qemu-kernels/sparc-zImage \
+       -append "root=$qemu_rootdev ip=dhcp panic=1 rw elevator=noop console=ttyS0 init=$vm_init_script" \
+       ${MEMSIZE:+-m $MEMSIZE} \
+       "${qemu_args[@]}"
+   else
+     set -- $qemu_bin -no-reboot -nographic -net none $KVM64_WORKAROUND \
+         -kernel $vm_kernel \
+         -initrd $vm_initrd \
+         -append "root=$qemu_rootdev panic=1 quiet no-kvmclock rw elevator=noop console=ttyS0 init=$vm_init_script" \
+         ${MEMSIZE:+-m $MEMSIZE} \
+         "${qemu_args[@]}"
+   fi

    if test "$PERSONALITY" != 0 ; then
        # have to switch back to PER_LINUX to make qemu work
@@ -1634,6 +1644,23 @@
     fi

     #
+    # fix SPARC MeeGo environment glitches
+    #
+    if [[ -n $RUNNING_IN_VM -a "$BUILD_ARCH" == 'sparc' ]]; then
+        ln -s /usr/bin/ar /usr/bin/sparc-leon-linux-gnu-ar
+        ln -s /usr/bin/as /usr/bin/sparc-leon-linux-gnu-as
+        ln -s /usr/bin/ranlib /usr/bin/sparc-leon-linux-gnu-ranlib
```

```
+         ln -s /usr/bin/g++ /usr/bin/sparc-leon-linux-gnu-g++
+         ln -s /usr/bin/cpp /lib/cpp
+         ln -s /usr/bin/gcc /usr/bin/sparc-leon-linux-gnu-gcc
+         ln -s /usr/bin/gcc /usr/bin/cc
+         ln -s /usr/bin/gcc /usr/bin/sparc-leon-linux-gnu-cc
+         cat /usr/lib/rpm/find-debuginfo.sh | sed 's/strict=true/strict=false/g' > /usr/lib/rpm/find-debuginfo.sh.fixed
+         cp -f /usr/lib/rpm/find-debuginfo.sh.fixed /usr/lib/rpm/find-debuginfo.sh
+         cp -rf /usr/lib/gcc/sparc-leon-linux-gnu/4.4.2/* /usr/lib/gcc/sparc-leon-linux-gnu/4.4.5/
+     fi
+
+     #
      # install dummy sign program if needed
      #
      test -f $BUILD_ROOT/usr/bin/sign_installed && mv $BUILD_ROOT/usr/bin/sign_installed $BUILD_ROOT/usr/bin/sign
```

## init_buildsystem.patch

```
--- orig/usr/lib/build/init_buildsystem 2011-06-21 15:32:06.000000000 +0200
+++ new/usr/lib/build/init_buildsystem  2011-06-21 15:47:52.000000000 +0200
@@ -513,7 +513,6 @@
 # store that we start to build system
 #
 mkdir -p $BUILD_ROOT
-mkdir -p $BUILD_ROOT/.build
 touch $BUILD_IS_RUNNING

 if test -n "$PREPARE_VM" ; then
@@ -711,7 +710,14 @@
      test -c $BUILD_ROOT/dev/null || create_devs
     fi
     test -e $BUILD_ROOT/etc/fstab || touch $BUILD_ROOT/etc/fstab
-    test -e $BUILD_ROOT/etc/ld.so.conf || cp $BUILD_ROOT/etc/ld.so.conf.in $BUILD_ROOT/etc/ld.so.conf
+ # SM: create empty ld.so.conf if ld.so.conf.in is missing
+ if [ ! -e $BUILD_ROOT/etc/ld.so.conf ]; then
+   if [ -e $BUILD_ROOT/etc/ld.so.conf.in ]; then
+     cp $BUILD_ROOT/etc/ld.so.conf.in $BUILD_ROOT/etc/ld.so.conf
+   else
+     touch $BUILD_ROOT/etc/ld.so.conf
+   fi
+ fi
     if test -z "$PREPARE_VM" ; then
  run_pkg_scripts
  init_db
@@ -720,6 +726,7 @@
 fi

 if test -n "$PREPARE_VM" ; then
+    mkdir -p $BUILD_ROOT/.build
     echo "copying packages..."
     for PKG in $PACKAGES_TO_INSTALL ; do
  rm -f $BUILD_ROOT/.init_b_cache/$PKG.$PSUF
@@ -938,7 +945,8 @@
  rm -f $BUILD_ROOT/.init_b_cache/$PKG.rpm
  cp $BUILD_ROOT/.init_b_cache/rpms/$PKG.rpm $BUILD_ROOT/.init_b_cache/$PKG.rpm || cleanup_and_exit 1
     fi
```

```
-    ( chroot $BUILD_ROOT rpm --ignorearch --nodeps -U --oldpackage --ignoresize $RPMCHECKOPTS \
+ # SM: added --replacefiles to fix some initial package glitch
+    ( chroot $BUILD_ROOT rpm --ignorearch --nodeps -U --oldpackage --ignoresize --replacefiles $RPMCHECKOPTS \
    $ADDITIONAL_PARAMS .init_b_cache/$PKG.rpm 2>&1 || \
    touch $BUILD_ROOT/exit ) | \
        grep -v "^warning:.*saved as.*rpmorig$"
@@ -1080,9 +1088,13 @@
  chroot $BUILD_ROOT bash -c ". /etc/profile ; $PROG"
 done

-if test -e $BUILD_ROOT/usr/share/zoneinfo/UTC ; then
-    chroot $BUILD_ROOT zic -l UTC
-fi
+# SM: temporarily disable zic due to a package glitch
+#if test -e $BUILD_ROOT/usr/share/zoneinfo/UTC ; then
+#    chroot $BUILD_ROOT zic -l UTC
+#fi
+
+# SM: fix perl permissions and some executable paths due to a package glitch
+chmod +x /usr/bin/perl

 test -e $BUILD_ROOT/.build/init_buildsystem.data || HOST=`hostname`
 test -e $BUILD_ROOT/etc/hosts || echo "127.0.0.1 localhost" > $BUILD_ROOT/etc/hosts
```

## C.2.2 Project config

This section provide the configuration file for the SPARC MeeGo Core repository.

### project-config

```
###########################
# Header
###########################

Patterntype: rpm-md comps
Support: build build-compare
Release: <CI_CNT>.<B_CNT>

###########################
# Export Filters
###########################

ExportFilter: \.x86_64\.rpm$ x86_64
ExportFilter: \.i586\.rpm$ i586

ExportFilter: \.sparc\.rpm$ sparc

ExportFilter: \.armv5el\.rpm$ armv5el
ExportFilter: \.armv5tel\.rpm$ armv5el
ExportFilter: \.armv6el\.rpm$ armv6el
ExportFilter: \.armv6l\.rpm$ armv6el
ExportFilter: \.armv6vl\.rpm$ armv6el
ExportFilter: \.armv7el\.rpm$ armv7el
ExportFilter: \.armv7l\.rpm$ armv7el
ExportFilter: \.armv7vl\.rpm$ armv7el

```

```
ExportFilter: .*vanish\.rpm
PublishFilter: .*vanish\.rpm
ExportFilter: .*dontuse\.rpm
PublishFilter: .*dontuse\.rpm

###########################
# ARM Section
###########################

%ifarch %arm

%define cross_5 1
%define cross_7 1
%define native 1

%ifarch armv5el
Changetarget: armv5tel-meego-linux
%define _gnu gnueabi
%if %{cross_5}
%define speedcommon 1
%define speedbash 1
%define speedbinutils 1
%define speedgcc 1
%define native 0
%endif
%endif

%ifarch armv6el
Changetarget: armv6l-meego-linux
%define _gnu gnueabi
%endif

%ifarch armv7el
Changetarget: armv7l-meego-linux
%define _gnu gnueabi
%if %{cross_7}
%define speedcommon 1
%define speedbash 1
%define speedbinutils 1
%define speedgcc 1
%define native 0
%endif
%endif

%if %speedcommon
Preinstall: aaa-meego-accelerator glibc-x86-arm
Runscripts: aaa-meego-accelerator
Required: aaa-meego-accelerator
%endif

%if %speedbash
Preinstall: bash-x86-arm ncurses-libs-x86-arm
Runscripts: bash-x86-arm
%endif

%if %speedbinutils
Required: cross-arm-binutils-accel
%endif

%if %speedgcc
Required: cross-arm-gcc-accel
%endif
```

```

Preinstall: rpm
Preinstall: rpm-libs
Required:   rpm
Prefer:     rpm-libs
Prefer:     rpm

%endif

###########################
# Intel Section
###########################

%ifarch %{ix86}
Ignore: ncurses-libs-x86
Preinstall: rpm rpm-libs
Required:   rpm
%endif

###########################
# SPARC Section
###########################

%ifarch %{sparc}
Ignore: ncurses-libs-x86
Preinstall: rpm rpm-libs sysroot remcall readline corefixes util-linux-ng distcc
     sysvinit
Required: rpm
Order: sysroot:bash
%endif

###########################
# Chroot Definitions
###########################

Preinstall: liblua
Preinstall: bash bzip2 coreutils diffutils  db4
Preinstall: filesystem grep glibc glibc-common libacl libattr
Preinstall: libgcc pam pcre nss nspr libcap
Preinstall: popt readline sed tar zlib
Preinstall: sqlite  ncurses-libs
Preinstall: elfutils-libelf  perl-libs
Preinstall: bzip2-libs  libstdc++ setup
Preinstall: file-libs
Preinstall: nss-softokn-freebl xz-libs

VMinstall: util-linux-ng perl perl-libs libblkid e2fsprogs-libs libuuid grep
    pcre

Required: binutils gcc glibc rpm-build libtool

Support: cpio gcc-c++  perl-libs perl  net-tools findutils
Support: file findutils  zlib bzip2 info
Support: gzip  xz-lzma-compat ncurses-libs
Support: make  patch sed  gawk tar grep coreutils pkgconfig  autoconf automake
Support: unzip  groff shadow-utils
Support: m4 file-libs tzdata meego-rpm-config meego-release
Support: kernel-headers glibc-headers

Keep: binutils cpp  cracklib file findutils gawk gcc  gcc-ada gcc-c++
Keep: gdbm  gzip libada libunwind  glibc-devel pcre xz-lzma-compat
```

```
Keep: make pam-modules shadow-utils gmp libcap groff cpio kernel-headers  glibc-
    headers
Keep: patch rcs rpm-build  nss nspr elfutils python grep libgcc gcc-c++
Keep: mpc mpfr

%ifarch %ix86
Keep: cloog cloog-ppl ppl
%endif

Prefer: libgnome-keyring
Prefer: xorg-x11-server-Xorg
Prefer: libtool-ltdl
Prefer: db4-cxx
Prefer: libtdb
Prefer: db4
Prefer: xulrunner
Prefer: readline
Prefer: xz-lzma-compat
Prefer: mutter-devel
Prefer: perl-Archive-Tar
Prefer: util-linux-ng
Prefer: kernel-netbook
Prefer: mesa-dri-i965-driver
Prefer: GConf2
Prefer: w3m
Prefer: nspr nspr-devel nss nss-devel
Prefer: generic-logos
Prefer: text-www-browser:lynx
Prefer: docbook-utils:lynx
Prefer: kdepim:pinentry-qt
Prefer: syslogd sysklogd
Prefer: -libgcc-mainline -libstdc++-mainline -gcc-mainline-c++
Prefer: -libgcj-mainline -viewperf -compat -compat-openssl097g
Prefer: -zmd -OpenOffice_org -pam-laus -libgcc-tree-ssa -busybox-links
Prefer: -crossover-office

Conflict: ghostscript-library:ghostscript-mini

Ignore: udev:udev-rules
Ignore: cups:xinetd
Ignore: cups:xinitd
Ignore: alsa-lib:alsa-plugins-pulseaudio
Ignore: meego-cross-armv5tel-sysroot
Ignore: nautilus:gvfs
Ignore: polkit:ConsoleKit
Ignore: iso-codes:xml-common
Ignore: libzypp:gnupg
Ignore: WebKit:libproxy
Ignore: gvfs:gnome-disk-utility
Ignore: installer:system-config-date
Ignore: libproxy:xulrunner
Ignore: system-config-date:authconfig
Ignore: authconfig:pam,usermode,python
Ignore: firstboot:system-config-date
Ignore: SDL:mkinitrd
Ignore: SDL:kernel,kernel-netbook,kern-ivi
Ignore: pulseaudio:kernel
Ignore: alsa-lib:kernel,kernel-netbook,kern-ivi
Ignore: alsa-plugins:kernel,kernel-netbook,kern-ivi
Ignore: gst-plugins-good:kernel,kernel-netbook,kernel-ivi
Ignore: libzypp:expect
Ignore: gtk2:moblin-icon-theme
```

```
Ignore: brasero:moblin-menus
Ignore: udev:meego-udev-rules
Ignore: pulseaudio:rtkit
Ignore: rpm:libcap
Ignore: rpm-libs:libcap
Ignore: mutter-meego:meego-panel-applications,meego-panel-myzone,meego-panel-pasteboard,meego-panel-people,meego-panel-status,meego-web-browser-panel,meego-panel-media,zenity
Ignore: db4:util-linux-ng
Ignore: fuse-sshfs:fastinit
Ignore: dhcp:fastinit
Ignore: libgnomeprint22:fastinit
Ignore: gvfs:fastinit
Ignore: meego-ux-settings:mutter,mutter-meego,mojito,gnome-vfs2,nautilus,meego-gtk-engine
Ignore: mutter-moblin:clutter-gtk,zenity
Ignore: gnome-desktop:gnome-user-docs
Ignore: gnome-settings-daemon:gnome-control-center
Ignore: avahi:fastinit
Ignore: fastinit:udev
Ignore: udev:fastinit
Ignore: PackageKit:udev
Ignore: cvs:vim-minimal
Ignore: bluez:fastinit
Ignore: aspell:aspell-en
Ignore: installer:squashfs-tools
Ignore: fuse:kernel
Ignore: fuse:fastinit
Ignore: fastinit:module-init-tools
Ignore: hwdata:module-init-tools
Ignore: gzip:less
Ignore: xmlto:text-www-browser
Ignore: docbook-utils:text-www-browser
Ignore: gtk2:hicolor-icon-theme
Ignore: docbook-dtds:openjade
Ignore: xmlto:passivetex
Ignore: GConf-dbus:openldap
Ignore: perl:rsyslog,tcsh,logrotate
Ignore: rpm:curl,crontabs,logrotate
Ignore: texinfo-tex:tetex
Ignore: xorg-x11-server:hal-info
Ignore: gcc:libgomp
Ignore: autoconf:imake
Ignore: ConsoleKit:dbus,dbus-devel
Ignore: fastinit:kernel,udev,ethtool,mingetty
Ignore: tetex:tetex-fonts,desktop-file-utils
Ignore: pam:glib2
Ignore: util-linux-ng:ConsoleKit-libs
Ignore: gettext-devel:libgcj,libstdc++-devel
Ignore: pam-modules:resmgr
Ignore: bind-utils:bind-libs
Ignore: alsa:dialog,pciutils
Ignore: portmap:syslogd
Ignore: fontconfig:freetype2
Ignore: fontconfig-devel:freetype2-devel
Ignore: xorg-x11-libs:freetype2
Ignore: xorg-x11:x11-tools,resmgr,xkeyboard-config,xorg-x11-Mesa,libusb,freetype2,libjpeg,libpng
Ignore: arts:alsa,audiofile,resmgr,libogg,libvorbis
Ignore: libxml2-devel:readline-devel
Ignore: gnome-vfs2:gnome-mime-data,desktop-file-utils,cdparanoia,dbus-1,dbus-1-glib,krb5,hal,libsmbclient,fam,file_alteration
```

```
Ignore: libgda:file_alteration
Ignore: gnutls:lzo,libopencdk
Ignore: libgnomecanvas-devel:glib-devel
Ignore: libgnomeui:gnome-icon-theme,shared-mime-info
Ignore: gnome-pilot:gnome-panel
Ignore: postfix:pcre
Ignore: docbook_4:iso_ent,sgml-skel,xmlcharent
Ignore: docbook-xsl-stylesheets:xmlcharent
Ignore: tetex:xorg-x11-libs,expat,fontconfig,freetype2,libjpeg,libpng,ghostscript-x11,xaw3d,gd,dialog,ed
Ignore: mailx:smtp_daemon
Ignore: cron:smtp_daemon

###########################
# Compile Flags
###########################

%define  __global_cflags        -O2 -g -pipe -Wall  -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector --param=ssp-buffer-size=4 -Wformat -Wformat-security

Optflags: i386 %{__global_cflags} -m32 -march=i386 -mtune=generic -fasynchronous-unwind-tables
Optflags: i486 %{__global_cflags} -m32 -march=i486 -fasynchronous-unwind-tables
Optflags: i586 %{__global_cflags} -m32 -march=core2 -mssse3 -mtune=atom -mfpmath=sse -fasynchronous-unwind-tables -fno-omit-frame-pointer
Optflags: i686 %{__global_cflags} -m32 -march=core2 -mssse3 -mtune=atom -mfpmath=sse -fasynchronous-unwind-tables -fno-omit-frame-pointer

Optflags: armv5tel %{__global_cflags} -fmessage-length=0 -march=armv5te -mlittle-endian
Optflags: armv6l %{__global_cflags} -fmessage-length=0 -march=armv6 -mlittle-endian -mfpu=vfp -mfloat-abi=softfp -D__SOFTFP__
Optflags: armv7l %{__global_cflags} -fmessage-length=0 -march=armv7-a -mtune=cortex-a8 -mlittle-endian -mfpu=vfpv3 -mfloat-abi=softfp -D__SOFTFP__

Optflags: sparc %{__global_cflags} -mcpu=v8 -m32 -mhard-float -Xlinker --build-id

###########################
# RPM Macros
###########################

Macros:

%moblin_version 2
%meego_version 1
%meego 1.1
%opensuse_bs 1
%vendor MeeGo
%_vendor meego
%_default_patch_fuzz   2

%py_ver         %(echo `python -c "import sys; print sys.version[:3]"`)
%py_prefix      %(echo `python -c "import sys; print sys.prefix"`)
%py_libdir      %{py_prefix}/lib/python%{py_ver}
%py_incdir      /usr/include/python%{py_ver}
%py_sitedir     %{py_libdir}/site-packages
%py_dyndir      %{py_libdir}/lib-dynload
%py_comp        python -c "import compileall; import sys; compileall.compile_dir(sys.argv[1], ddir=sys.argv[1][len('$RPM_BUILD_ROOT'):])"
```

```
%py_ocomp       python -O -c "import compileall; import sys; compileall.
    compile_dir(sys.argv[1], ddir=sys.argv[1][len('$RPM_BUILD_ROOT'):])"

%ext_info .gz
%ext_man .gz

%info_add(:-:) test -x /sbin/install-info -a -f %{?2}%{?!2:%{_infodir}}/%{1}%
    ext_info && /sbin/install-info --info-dir=%{?2}%{?!2:%{_infodir}}
    %{?2}%{?!2:%{_infodir}}/%{1}%ext_info \
%{nil}

%info_del(:-:) test -x /sbin/install-info -a ! -f %{?2}%{?!2:%{_infodir}}/%{1}%
    ext_info && /sbin/install-info --quiet --delete --info-dir=%{?2}%{?!2:%{
    _infodir}} %{?2}%{?!2:%{_infodir}}/%{1}%ext_info \
%{nil}

%_smp_mflags -j1
```

### C.2.3 Deployable worker

This section provides the startup scripts written for the deployable worker package.

#### obs.sh

```
#!/bin/bash

CFG_INSTANCES='1' # number of parallel build jobs, 0 means all the available
    cores
CFG_INSTANCE_MEMORY='1024' # amount of RAM allocated for each instance
CFG_SERVER_IP='192.168.0.39' # IP address of the obs server
CFG_SERVER_FQDN='ivan.site' # fully qualified domain name of the obs server

OBS_PATH="`dirname $0`/chroot"

LINE=`tput cols`
LINE="-`seq -s "-" $LINE | sed 's/[0-9]//g'`"

if [[ "`whoami`" != 'root' ]]; then
  tput setaf 1
  echo "!! ERROR: this script needs root privileges"
  tput sgr0
  exit 1
fi

if [[ "$CFG_INSTANCES" == '0' ]]; then
  tput setaf 1
  echo "!! ERROR: please edit the script and set CFG_INSTANCES"
  tput sgr0
  exit 1
fi

if [[ ! -d $OBS_PATH ]]; then
  tput setaf 1
  echo "!! ERROR: the OBS directory does not exist"
  tput sgr0
  exit 1
fi

echo $LINE
```

```
echo "-> PINGING SERVER..."
ping -c 1 $CFG_SERVER_IP > /dev/null
if [[ $? != 0 ]]; then
  tput setaf 1
  echo "!! ERROR: server at $CFG_SERVER_IP did not answer ping"
  tput sgr0
  exit 1
fi

echo "-> CHECKING LOCAL NETWORK PORTS..."
BUSY_PORTS=`netstat -l --numeric --numeric-ports --protocol=inet | grep -E '^tcp.*$' | awk '{ print $4 }' | cut -d':' -f'2' | grep -E '^(5252)|(5352)$' | tr '\n' ' ' | sed 's/\ $//g'`
if [[ ! -z $BUSY_PORTS ]]; then
  tput setaf 1
  echo "!! ERROR: some required ports are already bound ($BUSY_PORTS)"
  tput sgr0
  exit 1
else
  echo "required ports are free"
fi

echo "-> FIXING CONFIGURATION..."

echo "dns configuration"
cp /etc/resolv.conf $OBS_PATH/etc

echo "worker configuration"
echo "CFG_SERVER_IP=\"$CFG_SERVER_IP\"" > $OBS_PATH/etc/buildhost.config
echo "OBS_REPO_SERVERS=\"$CFG_SERVER_IP:5252\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_SRC_SERVER=\"$CFG_SERVER_IP:5352\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_WORKER_INSTANCES=\"$CFG_INSTANCES\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_WORKER_JOBS=\"1\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_VM_TYPE=\"qemu\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_VM_KERNEL=\"none\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_VM_INITRD=\"none\"" >> $OBS_PATH/etc/buildhost.config
echo "OBS_INSTANCE_MEMORY=\"$CFG_INSTANCE_MEMORTY\"" >> $OBS_PATH/etc/buildhost.config

echo "hosts configuration"
cp /etc/hosts $OBS_PATH/etc
echo "$CFG_SERVER_IP $CFG_SERVER_FQDN" >> $OBS_PATH/etc/hosts

echo "-> CLEANING OLD LOGS..."
rm -vf $OBS_PATH/root/distccd.log
rm -vf $OBS_PATH/root/worker_logs/*

echo "-> MOUNTING REQUIRED FILESYSTEMS..."
mount -vt proc proc $OBS_PATH/proc
mount -vt sysfs sysfs $OBS_PATH/sys
mount -v -o bind /dev $OBS_PATH/dev

echo "-> CHROOTING INTO OBS..."

echo $LINE
tput setaf 4
chroot $OBS_PATH i386 /root/obs.sh $CFG_INSTANCES $CFG_SERVER_IP $CFG_SERVER_FQDN $CFG_INSTANCE_MEMORY
tput sgr0
echo $LINE
```

```

echo "-> SAVING LOGS..."
DIR_LOGS="‘dirname $0‘/logs/‘date +’%s’‘"
mkdir -p $DIR_LOGS/worker_logs
cp -v $OBS_PATH/root/distccd.log $DIR_LOGS
cp -v $OBS_PATH/root/worker_logs/* $DIR_LOGS/worker_logs

echo "-> UNMOUNTING FILESYSTEMS..."
umount -v $OBS_PATH/dev
umount -v $OBS_PATH/sys
umount -v $OBS_PATH/proc

OBS_MOUNTS="‘cat /etc/mtab | grep $OBS_PATH‘"

if [[ ! -z $OBS_MOUNTS ]]; then
  tput setaf 1
  echo "!! ERROR: some mountpoints were not correctly unmounted, please do that manually"
  echo $OBS_MOUNTS
  tput sgr0
fi

echo $LINE
tput setaf 5
echo "-> THANKS FOR YOUR CYCLES!!!"
tput sgr0
echo $LINE
```

## obs-in.sh

```
#!/bin/bash

echo "‘whoami‘ @ ‘uname -a‘"

echo -n "-> STARTING DISTCCD... "
export DISTCCD_PATH="/opt/sparc-linux-4.4.2-toolchains/multilib/bin"
/opt/distcc/bin/distccd --no-detach --jobs $1 --allow 127.0.0.1 --log-stderr --verbose &> /root/distccd.log &
echo "DONE"

echo -n "-> STARTING WORKERS... "
rcobsworker start &> /dev/null
echo "DONE"

tput bold
echo
echo -e "   :-----------------------------------------------------------------:"
echo -e "   |                          UP & RUNNING...                        |"
echo -e "   | (DO NOT PRESS CTRL+C, KILL OR OTHERWISE TERMINATE THE PROCESS) |"
echo -e "   |                       PRESS RETURN TO QUIT!                     |"
echo -e "   :-----------------------------------------------------------------:"
tput sgr0
tput setaf 4

read

echo -n "-> TERMINATING DISTCCD... "
killall distccd &> /dev/null
echo "DONE"

echo -n "-> TERMINATING WORKERS... "
```

```
rcobsworker stop &> /dev/null
killall qemu-system-sparc &> /dev/null
echo "DONE"
```

# C.3 Kernel configurations

This section provides the kernel configuration files generated for the project. The configuration files have been shortened here by removing all comments.

## C.3.1 QEMU

This section provides the configuration file for the kernel used to boot QEMU. The configuration applies to the 2.6.38 stock Linux kernel.

### kernel-config-qemu

```
CONFIG_SPARC=y
CONFIG_SPARC32=y
CONFIG_BITS=32
CONFIG_ARCH_USES_GETTIMEOFFSET=y
CONFIG_GENERIC_CMOS_UPDATE=y
CONFIG_AUDIT_ARCH=y
CONFIG_MMU=y
CONFIG_HIGHMEM=y
CONFIG_ZONE_DMA=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_NEED_SG_DMA_LENGTH=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_ARCH_NO_VIRT_TO_BUS=y
CONFIG_CONSTRUCTORS=y
CONFIG_HAVE_IRQ_WORK=y
CONFIG_EXPERIMENTAL=y
CONFIG_BROKEN_ON_SMP=y
CONFIG_INIT_ENV_ARG_LIMIT=32
CONFIG_CROSS_COMPILE="sparc-leon-linux-gnu-"
CONFIG_LOCALVERSION=""
CONFIG_LOCALVERSION_AUTO=y
CONFIG_SWAP=y
CONFIG_SYSVIPC=y
CONFIG_SYSVIPC_SYSCTL=y
CONFIG_POSIX_MQUEUE=y
CONFIG_POSIX_MQUEUE_SYSCTL=y
CONFIG_TINY_RCU=y
CONFIG_LOG_BUF_SHIFT=14
CONFIG_NAMESPACES=y
CONFIG_UTS_NS=y
CONFIG_IPC_NS=y
CONFIG_USER_NS=y
CONFIG_PID_NS=y
CONFIG_NET_NS=y
CONFIG_SYSCTL=y
CONFIG_ANON_INODES=y
CONFIG_UID16=y
CONFIG_SYSCTL_SYSCALL=y
CONFIG_KALLSYMS=y
CONFIG_HOTPLUG=y
CONFIG_PRINTK=y
CONFIG_BUG=y
CONFIG_ELF_CORE=y
CONFIG_BASE_FULL=y
CONFIG_FUTEX=y
CONFIG_EPOLL=y
CONFIG_SIGNALFD=y
CONFIG_TIMERFD=y
CONFIG_EVENTFD=y
CONFIG_SHMEM=y
CONFIG_AIO=y
CONFIG_VM_EVENT_COUNTERS=y
CONFIG_PCI_QUIRKS=y
CONFIG_COMPAT_BRK=y
CONFIG_SLAB=y
CONFIG_HAVE_OPROFILE=y
CONFIG_HAVE_ARCH_TRACEHOOK=y
CONFIG_HAVE_DMA_ATTRS=y
CONFIG_HAVE_DMA_API_DEBUG=y
CONFIG_HAVE_ARCH_JUMP_LABEL=y
CONFIG_SLABINFO=y
CONFIG_RT_MUTEXES=y
CONFIG_BASE_SMALL=0
CONFIG_BLOCK=y
CONFIG_LBDAF=y
CONFIG_IOSCHED_NOOP=y
CONFIG_IOSCHED_DEADLINE=y
CONFIG_IOSCHED_CFQ=y
CONFIG_DEFAULT_CFQ=y
CONFIG_DEFAULT_IOSCHED="cfq"
CONFIG_INLINE_SPIN_UNLOCK=y
CONFIG_INLINE_SPIN_UNLOCK_IRQ=y
CONFIG_INLINE_READ_UNLOCK=y
CONFIG_INLINE_READ_UNLOCK_IRQ=y
CONFIG_INLINE_WRITE_UNLOCK=y
CONFIG_INLINE_WRITE_UNLOCK_IRQ=y
```

```
CONFIG_HZ_100=y
CONFIG_HZ=100
CONFIG_RWSEM_GENERIC_SPINLOCK=y
CONFIG_GENERIC_FIND_NEXT_BIT=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_ARCH_MAY_HAVE_PC_FDC=y
CONFIG_EMULATED_CMPXCHG=y
CONFIG_SELECT_MEMORY_MODEL=y
CONFIG_FLATMEM_MANUAL=y
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
CONFIG_PAGEFLAGS_EXTENDED=y
CONFIG_SPLIT_PTLOCK_CPUS=4
CONFIG_ZONE_DMA_FLAG=1
CONFIG_BOUNCE=y
CONFIG_DEFAULT_MMAP_MIN_ADDR=4096
CONFIG_NEED_PER_CPU_KM=y
CONFIG_SUN_PM=y
CONFIG_SERIAL_CONSOLE=y
CONFIG_SBUS=y
CONFIG_SBUSCHAR=y
CONFIG_PCI=y
CONFIG_PCI_SYSCALL=y
CONFIG_SUN_OPENPROMFS=y
CONFIG_SPARC32_PCI=y
CONFIG_BINFMT_ELF=y
CONFIG_CORE_DUMP_DEFAULT_ELF_HEADERS=y
CONFIG_BINFMT_MISC=y
CONFIG_NET=y
CONFIG_PACKET=y
CONFIG_UNIX=y
CONFIG_XFRM=y
CONFIG_XFRM_USER=y
CONFIG_NET_KEY=y
CONFIG_INET=y
CONFIG_IP_FIB_HASH=y
CONFIG_IP_PNP=y
CONFIG_IP_PNP_DHCP=y
CONFIG_TCP_CONG_CUBIC=y
CONFIG_DEFAULT_TCP_CONG="cubic"
CONFIG_UEVENT_HELPER_PATH="/sbin/hotplug"
CONFIG_STANDALONE=y
CONFIG_PREVENT_FIRMWARE_BUILD=y
CONFIG_FW_LOADER=y
CONFIG_FIRMWARE_IN_KERNEL=y
CONFIG_EXTRA_FIRMWARE=""
CONFIG_OF=y
CONFIG_OF_PROMTREE=y
CONFIG_OF_DEVICE=y
CONFIG_OF_NET=y
CONFIG_BLK_DEV=y
CONFIG_BLK_DEV_LOOP=y
CONFIG_BLK_DEV_CRYPTOLOOP=y
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=16
CONFIG_BLK_DEV_RAM_SIZE=4096
CONFIG_MISC_DEVICES=y
CONFIG_EEPROM_93CX6=y
CONFIG_HAVE_IDE=y
CONFIG_SCSI_MOD=y
CONFIG_SCSI=y
CONFIG_SCSI_DMA=y
CONFIG_SCSI_NETLINK=y
CONFIG_SCSI_PROC_FS=y
CONFIG_BLK_DEV_SD=y
CONFIG_CHR_DEV_SG=y
CONFIG_SCSI_SPI_ATTRS=y
CONFIG_SCSI_FC_ATTRS=y
CONFIG_SCSI_ISCSI_ATTRS=y
CONFIG_SCSI_LOWLEVEL=y
CONFIG_SCSI_QLOGICPTI=y
CONFIG_SCSI_SUNESP=y
CONFIG_NETDEVICES=y
CONFIG_DUMMY=y
CONFIG_MII=y
CONFIG_NET_ETHERNET=y
CONFIG_SUNLANCE=y
CONFIG_INPUT=y
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_PSAUX=y
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1024
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=768
CONFIG_INPUT_JOYDEV=y
CONFIG_INPUT_EVDEV=y
CONFIG_INPUT_EVBUG=y
CONFIG_INPUT_KEYBOARD=y
CONFIG_KEYBOARD_ATKBD=y
CONFIG_KEYBOARD_SUNKBD=y
CONFIG_INPUT_MOUSE=y
CONFIG_MOUSE_PS2=y
CONFIG_MOUSE_PS2_ALPS=y
CONFIG_MOUSE_PS2_LOGIPS2PP=y
CONFIG_MOUSE_PS2_SYNAPTICS=y
CONFIG_MOUSE_PS2_TRACKPOINT=y
CONFIG_MOUSE_SERIAL=y
CONFIG_SERIO=y
CONFIG_SERIO_SERPORT=y
CONFIG_SERIO_LIBPS2=y
CONFIG_VT=y
CONFIG_CONSOLE_TRANSLATIONS=y
CONFIG_VT_CONSOLE=y
CONFIG_HW_CONSOLE=y
CONFIG_VT_HW_CONSOLE_BINDING=y
CONFIG_DEVKMEM=y
CONFIG_SERIAL_8250=y
CONFIG_SERIAL_8250_PCI=y
CONFIG_SERIAL_8250_NR_UARTS=4
CONFIG_SERIAL_8250_RUNTIME_UARTS=4
CONFIG_SERIAL_SUNCORE=y
CONFIG_SERIAL_SUNZILOG=y
CONFIG_SERIAL_SUNZILOG_CONSOLE=y
CONFIG_SERIAL_SUNSU=y
CONFIG_SERIAL_SUNSU_CONSOLE=y
CONFIG_SERIAL_SUNSAB=y
CONFIG_SERIAL_SUNSAB_CONSOLE=y
CONFIG_SERIAL_CORE=y
CONFIG_SERIAL_CORE_CONSOLE=y
CONFIG_CONSOLE_POLL=y
CONFIG_UNIX98_PTYS=y
CONFIG_DEVPTS_MULTIPLE_INSTANCES=y
CONFIG_LEGACY_PTYS=y
CONFIG_LEGACY_PTY_COUNT=256
CONFIG_HW_RANDOM=y
```

```
CONFIG_DEVPORT=y
CONFIG_ARCH_WANT_OPTIONAL_GPIOLIB=y
CONFIG_SSB_POSSIBLE=y
CONFIG_VGA_ARB=y
CONFIG_VGA_ARB_MAX_GPUS=16
CONFIG_VIDEO_OUTPUT_CONTROL=y
CONFIG_FB=y
CONFIG_FIRMWARE_EDID=y
CONFIG_FB_CFB_FILLRECT=y
CONFIG_FB_CFB_COPYAREA=y
CONFIG_FB_CFB_IMAGEBLIT=y
CONFIG_FB_MODE_HELPERS=y
CONFIG_FB_TILEBLITTING=y
CONFIG_FB_SBUS=y
CONFIG_FB_TCX=y
CONFIG_BACKLIGHT_LCD_SUPPORT=y
CONFIG_LCD_CLASS_DEVICE=y
CONFIG_LCD_PLATFORM=y
CONFIG_BACKLIGHT_CLASS_DEVICE=y
CONFIG_BACKLIGHT_GENERIC=y
CONFIG_DISPLAY_SUPPORT=y
CONFIG_DUMMY_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE_ROTATION=y
CONFIG_FONT_SUN8x16=y
CONFIG_FONT_SUN12x22=y
CONFIG_HID_SUPPORT=y
CONFIG_HID=y
CONFIG_RTC_LIB=y
CONFIG_RTC_CLASS=y
CONFIG_RTC_HCTOSYS=y
CONFIG_RTC_HCTOSYS_DEVICE="rtc0"
CONFIG_RTC_INTF_SYSFS=y
CONFIG_RTC_INTF_PROC=y
CONFIG_RTC_INTF_DEV=y
CONFIG_RTC_DRV_M48T59=y
CONFIG_SUN_OPENPROMIO=y
CONFIG_EXT2_FS=y
CONFIG_EXT2_FS_XATTR=y
CONFIG_EXT2_FS_POSIX_ACL=y
CONFIG_EXT2_FS_SECURITY=y
CONFIG_EXT2_FS_XIP=y
CONFIG_EXT3_FS=y
CONFIG_EXT3_DEFAULTS_TO_ORDERED=y
CONFIG_EXT3_FS_XATTR=y
CONFIG_EXT3_FS_POSIX_ACL=y
CONFIG_EXT3_FS_SECURITY=y
CONFIG_EXT4_FS=y
CONFIG_EXT4_FS_XATTR=y
CONFIG_EXT4_FS_POSIX_ACL=y
CONFIG_EXT4_FS_SECURITY=y
CONFIG_EXT4_DEBUG=y
CONFIG_FS_XIP=y
CONFIG_JBD=y
CONFIG_JBD2=y
CONFIG_FS_MBCACHE=y
CONFIG_FS_POSIX_ACL=y
CONFIG_FILE_LOCKING=y
CONFIG_FSNOTIFY=y
CONFIG_DNOTIFY=y
CONFIG_INOTIFY_USER=y
CONFIG_AUTOFS4_FS=y
CONFIG_FUSE_FS=y
CONFIG_CUSE=y
CONFIG_GENERIC_ACL=y
CONFIG_PROC_FS=y
CONFIG_PROC_KCORE=y
CONFIG_PROC_SYSCTL=y
CONFIG_PROC_PAGE_MONITOR=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
CONFIG_TMPFS_POSIX_ACL=y
CONFIG_CONFIGFS_FS=y
CONFIG_MISC_FILESYSTEMS=y
CONFIG_ROMFS_FS=y
CONFIG_ROMFS_BACKED_BY_BLOCK=y
CONFIG_ROMFS_ON_BLOCK=y
CONFIG_NETWORK_FILESYSTEMS=y
CONFIG_NFS_FS=y
CONFIG_LOCKD=y
CONFIG_NFS_COMMON=y
CONFIG_SUNRPC=y
CONFIG_SUNRPC_GSS=y
CONFIG_RPCSEC_GSS_KRB5=y
CONFIG_MSDOS_PARTITION=y
CONFIG_SUN_PARTITION=y
CONFIG_NLS=y
CONFIG_NLS_DEFAULT="iso8859-1"
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_ENABLE_MUST_CHECK=y
CONFIG_FRAME_WARN=1024
CONFIG_MAGIC_SYSRQ=y
CONFIG_DEBUG_KERNEL=y
CONFIG_DETECT_HUNG_TASK=y
CONFIG_BOOTPARAM_HUNG_TASK_PANIC_VALUE=0
CONFIG_BKL=y
CONFIG_DEBUG_BUGVERBOSE=y
CONFIG_DEBUG_MEMORY_INIT=y
CONFIG_HAVE_ARCH_KGDB=y
CONFIG_KGDB=y
CONFIG_KGDB_SERIAL_CONSOLE=y
CONFIG_KGDB_TESTS=y
CONFIG_DEFAULT_SECURITY_DAC=y
CONFIG_DEFAULT_SECURITY=""
CONFIG_CRYPTO=y
CONFIG_CRYPTO_ALGAPI=y
CONFIG_CRYPTO_ALGAPI2=y
CONFIG_CRYPTO_AEAD2=y
CONFIG_CRYPTO_BLKCIPHER=y
CONFIG_CRYPTO_BLKCIPHER2=y
CONFIG_CRYPTO_HASH=y
CONFIG_CRYPTO_HASH2=y
CONFIG_CRYPTO_RNG2=y
CONFIG_CRYPTO_PCOMP2=y
CONFIG_CRYPTO_MANAGER=y
CONFIG_CRYPTO_MANAGER2=y
CONFIG_CRYPTO_MANAGER_DISABLE_TESTS=y
CONFIG_CRYPTO_NULL=y
CONFIG_CRYPTO_WORKQUEUE=y
CONFIG_CRYPTO_CBC=y
CONFIG_CRYPTO_ECB=y
CONFIG_CRYPTO_PCBC=y
CONFIG_CRYPTO_CRC32C=y
CONFIG_CRYPTO_MD4=y
```

```
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_MICHAEL_MIC=y
CONFIG_CRYPTO_SHA256=y
CONFIG_CRYPTO_SHA512=y
CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_ARC4=y
CONFIG_CRYPTO_BLOWFISH=y
CONFIG_CRYPTO_CAST5=y
CONFIG_CRYPTO_CAST6=y
CONFIG_CRYPTO_DES=y
CONFIG_CRYPTO_SERPENT=y
CONFIG_CRYPTO_TWOFISH=y
CONFIG_CRYPTO_TWOFISH_COMMON=y
CONFIG_BITREVERSE=y
CONFIG_GENERIC_FIND_LAST_BIT=y
CONFIG_CRC16=y
CONFIG_CRC32=y
CONFIG_LIBCRC32C=y
CONFIG_HAS_IOMEM=y
CONFIG_HAS_IOPORT=y
CONFIG_HAS_DMA=y
CONFIG_NLATTR=y
```

### C.3.2 GR-LEON4-ITX

This section provides the configuration file for the kernel used to boot the test board. The configuration applies to the LEON branch of the 2.6.38 Linux kernel.

#### kernel-config-leon

```
CONFIG_SPARC=y
CONFIG_SPARC32=y
CONFIG_BITS=32
CONFIG_ARCH_USES_GETTIMEOFFSET=y
CONFIG_GENERIC_CMOS_UPDATE=y
CONFIG_AUDIT_ARCH=y
CONFIG_MMU=y
CONFIG_HIGHMEM=y
CONFIG_ZONE_DMA=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_NEED_SG_DMA_LENGTH=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_ARCH_NO_VIRT_TO_BUS=y
CONFIG_CONSTRUCTORS=y
CONFIG_EXPERIMENTAL=y
CONFIG_LOCK_KERNEL=y
CONFIG_INIT_ENV_ARG_LIMIT=32
CONFIG_CROSS_COMPILE=""
CONFIG_LOCALVERSION=""
CONFIG_LOCALVERSION_AUTO=y
CONFIG_SWAP=y
CONFIG_SYSVIPC=y
CONFIG_SYSVIPC_SYSCTL=y
CONFIG_TREE_RCU=y
CONFIG_RCU_FANOUT=32
CONFIG_LOG_BUF_SHIFT=14
CONFIG_SYSFS_DEPRECATED=y
CONFIG_SYSFS_DEPRECATED_V2=y
CONFIG_BLK_DEV_INITRD=y
CONFIG_INITRAMFS_ROOT_UID=0
CONFIG_INITRAMFS_ROOT_GID=0
CONFIG_RD_GZIP=y
CONFIG_INITRAMFS_COMPRESSION_NONE=y
CONFIG_SYSCTL=y
CONFIG_ANON_INODES=y
CONFIG_EMBEDDED=y
CONFIG_UID16=y
CONFIG_SYSCTL_SYSCALL=y
CONFIG_KALLSYMS=y
CONFIG_PRINTK=y
CONFIG_BUG=y
CONFIG_ELF_CORE=y
CONFIG_BASE_FULL=y
CONFIG_FUTEX=y
CONFIG_EPOLL=y
CONFIG_SIGNALFD=y
CONFIG_TIMERFD=y
CONFIG_EVENTFD=y
CONFIG_SHMEM=y
CONFIG_AIO=y
CONFIG_HAVE_PERF_EVENTS=y
CONFIG_PERF_USE_VMALLOC=y
CONFIG_VM_EVENT_COUNTERS=y
CONFIG_PCI_QUIRKS=y
CONFIG_COMPAT_BRK=y
CONFIG_SLAB=y
CONFIG_HAVE_OPROFILE=y
CONFIG_HAVE_ARCH_TRACEHOOK=y
CONFIG_HAVE_DMA_ATTRS=y
CONFIG_USE_GENERIC_SMP_HELPERS=y
CONFIG_HAVE_DMA_API_DEBUG=y
CONFIG_SLABINFO=y
CONFIG_RT_MUTEXES=y
CONFIG_BASE_SMALL=0
CONFIG_MODULES=y
CONFIG_MODULE_UNLOAD=y
CONFIG_STOP_MACHINE=y
CONFIG_BLOCK=y
CONFIG_LBDAF=y
CONFIG_IOSCHED_NOOP=y
CONFIG_IOSCHED_DEADLINE=y
CONFIG_IOSCHED_CFQ=y
CONFIG_DEFAULT_CFQ=y
CONFIG_DEFAULT_IOSCHED="cfq"
CONFIG_INLINE_SPIN_UNLOCK=y
CONFIG_INLINE_SPIN_UNLOCK_IRQ=y
```

```
CONFIG_INLINE_READ_UNLOCK=y
CONFIG_INLINE_READ_UNLOCK_IRQ=y
CONFIG_INLINE_WRITE_UNLOCK=y
CONFIG_INLINE_WRITE_UNLOCK_IRQ=y
CONFIG_MUTEX_SPIN_ON_OWNER=y
CONFIG_SMP=y
CONFIG_NR_CPUS=32
CONFIG_HZ_100=y
CONFIG_HZ=100
CONFIG_RWSEM_GENERIC_SPINLOCK=y
CONFIG_GENERIC_FIND_NEXT_BIT=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_ARCH_MAY_HAVE_PC_FDC=y
CONFIG_EMULATED_CMPXCHG=y
CONFIG_SPARC32_SMP=y
CONFIG_SELECT_MEMORY_MODEL=y
CONFIG_FLATMEM_MANUAL=y
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
CONFIG_PAGEFLAGS_EXTENDED=y
CONFIG_SPLIT_PTLOCK_CPUS=4
CONFIG_ZONE_DMA_FLAG=1
CONFIG_BOUNCE=y
CONFIG_DEFAULT_MMAP_MIN_ADDR=4096
CONFIG_SUN_PM=y
CONFIG_SERIAL_CONSOLE=y
CONFIG_SPARC_LEON=y
CONFIG_UBOOT_LOAD_ADDR=0x40004000
CONFIG_UBOOT_FLASH_ADDR=0x00080000
CONFIG_UBOOT_ENTRY_ADDR=0xf0004000
CONFIG_SBUS=y
CONFIG_SBUSCHAR=y
CONFIG_PCI=y
CONFIG_PCI_SYSCALL=y
CONFIG_PCI_DEBUG=y
CONFIG_SUN_OPENPROMFS=y
CONFIG_SPARC32_PCI=y
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_MISC=y
CONFIG_NET=y
CONFIG_PACKET=y
CONFIG_UNIX=y
CONFIG_XFRM=y
CONFIG_INET=y
CONFIG_IP_FIB_HASH=y
CONFIG_IP_PNP=y
CONFIG_IP_PNP_DHCP=y
CONFIG_INET_TUNNEL=y
CONFIG_INET_XFRM_MODE_TRANSPORT=y
CONFIG_INET_XFRM_MODE_TUNNEL=y
CONFIG_INET_XFRM_MODE_BEET=y
CONFIG_INET_DIAG=y
CONFIG_INET_TCP_DIAG=y
CONFIG_TCP_CONG_CUBIC=y
CONFIG_DEFAULT_TCP_CONG="cubic"
CONFIG_IPV6=y
CONFIG_INET6_XFRM_MODE_TRANSPORT=y
CONFIG_INET6_XFRM_MODE_TUNNEL=y
CONFIG_INET6_XFRM_MODE_BEET=y
CONFIG_IPV6_SIT=y
CONFIG_IPV6_NDISC_NODETYPE=y
CONFIG_DNS_RESOLVER=y
CONFIG_RPS=y
CONFIG_WIRELESS=y
CONFIG_STANDALONE=y
CONFIG_PREVENT_FIRMWARE_BUILD=y
CONFIG_OF=y
CONFIG_OF_DEVICE=y
CONFIG_OF_MDIO=y
CONFIG_BLK_DEV=y
CONFIG_BLK_DEV_LOOP=y
CONFIG_BLK_DEV_CRYPTOLOOP=y
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=16
CONFIG_BLK_DEV_RAM_SIZE=4096
CONFIG_XILINX_SYSACE=y
CONFIG_HAVE_IDE=y
CONFIG_SCSI_MOD=y
CONFIG_SCSI=y
CONFIG_SCSI_DMA=y
CONFIG_SCSI_PROC_FS=y
CONFIG_BLK_DEV_SD=y
CONFIG_SCSI_WAIT_SCAN=m
CONFIG_SCSI_LOWLEVEL=y
CONFIG_NETDEVICES=y
CONFIG_PHYLIB=y
CONFIG_NET_ETHERNET=y
CONFIG_MII=y
CONFIG_GRETH=y
CONFIG_NET_PCI=y
CONFIG_E100=y
CONFIG_8139TOO=y
CONFIG_8139TOO_PIO=y
CONFIG_NETDEV_1000=y
CONFIG_DL2K=y
CONFIG_INPUT=y
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_PSAUX=y
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1024
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=768
CONFIG_INPUT_JOYDEV=y
CONFIG_INPUT_EVDEV=y
CONFIG_INPUT_EVBUG=y
CONFIG_INPUT_KEYBOARD=y
CONFIG_KEYBOARD_ATKBD=y
CONFIG_KEYBOARD_SUNKBD=y
CONFIG_INPUT_MOUSE=y
CONFIG_MOUSE_PS2=y
CONFIG_MOUSE_PS2_ALPS=y
CONFIG_MOUSE_PS2_LOGIPS2PP=y
CONFIG_MOUSE_PS2_SYNAPTICS=y
CONFIG_MOUSE_PS2_TRACKPOINT=y
CONFIG_MOUSE_SERIAL=y
CONFIG_SERIO=y
CONFIG_SERIO_SERPORT=y
CONFIG_SERIO_LIBPS2=y
CONFIG_VT=y
CONFIG_CONSOLE_TRANSLATIONS=y
CONFIG_VT_CONSOLE=y
CONFIG_HW_CONSOLE=y
CONFIG_DEVKMEM=y
CONFIG_SERIAL_SUNCORE=y
CONFIG_SERIAL_SUNZILOG=y
```

```
CONFIG_SERIAL_SUNZILOG_CONSOLE=y
CONFIG_SERIAL_CORE=y
CONFIG_SERIAL_CORE_CONSOLE=y
CONFIG_SERIAL_GRLIB_GAISLER_APBUART=y
CONFIG_SERIAL_GRLIB_GAISLER_APBUART_CONSOLE=y
CONFIG_UNIX98_PTYS=y
CONFIG_LEGACY_PTYS=y
CONFIG_LEGACY_PTY_COUNT=256
CONFIG_HW_RANDOM=y
CONFIG_DEVPORT=y
CONFIG_ARCH_WANT_OPTIONAL_GPIOLIB=y
CONFIG_SSB_POSSIBLE=y
CONFIG_MFD_SUPPORT=y
CONFIG_VGA_ARB=y
CONFIG_VGA_ARB_MAX_GPUS=16
CONFIG_FB=y
CONFIG_FB_CFB_FILLRECT=y
CONFIG_FB_CFB_COPYAREA=y
CONFIG_FB_CFB_IMAGEBLIT=y
CONFIG_FB_TILEBLITTING=y
CONFIG_FB_GRVGA=y
CONFIG_BACKLIGHT_LCD_SUPPORT=y
CONFIG_LCD_CLASS_DEVICE=m
CONFIG_BACKLIGHT_CLASS_DEVICE=y
CONFIG_BACKLIGHT_GENERIC=y
CONFIG_DUMMY_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE=y
CONFIG_FONT_SUN8x16=y
CONFIG_LOGO=y
CONFIG_LOGO_LINUX_MONO=y
CONFIG_LOGO_LINUX_VGA16=y
CONFIG_LOGO_LINUX_CLUT224=y
CONFIG_LOGO_SUN_CLUT224=y
CONFIG_SOUND=y
CONFIG_SOUND_OSS_CORE=y
CONFIG_SOUND_OSS_CORE_PRECLAIM=y
CONFIG_SND=y
CONFIG_SND_TIMER=y
CONFIG_SND_PCM=y
CONFIG_SND_RAWMIDI=y
CONFIG_SND_SEQUENCER=y
CONFIG_SND_OSSEMUL=y
CONFIG_SND_MIXER_OSS=y
CONFIG_SND_PCM_OSS=y
CONFIG_SND_PCM_OSS_PLUGINS=y
CONFIG_SND_SUPPORT_OLD_API=y
CONFIG_SND_VERBOSE_PROCFS=y
CONFIG_SND_VMASTER=y
CONFIG_SND_RAWMIDI_SEQ=y
CONFIG_SND_MPU401_UART=y
CONFIG_SND_AC97_CODEC=y
CONFIG_SND_DRIVERS=y
CONFIG_SND_PCI=y
CONFIG_SND_ALI5451=y
CONFIG_SND_ATIIXP=y
CONFIG_SND_ATIIXP_MODEM=y
CONFIG_SND_INTEL8X0=y
CONFIG_SND_USB=y
CONFIG_SND_SPARC=y
CONFIG_AC97_BUS=y
CONFIG_HID_SUPPORT=y
CONFIG_HID=y
CONFIG_USB_HID=y
CONFIG_USB_SUPPORT=y
CONFIG_USB_ARCH_HAS_HCD=y
CONFIG_USB_ARCH_HAS_OHCI=y
CONFIG_USB_ARCH_HAS_EHCI=y
CONFIG_USB=y
CONFIG_USB_DEVICEFS=y
CONFIG_USB_DEVICE_CLASS=y
CONFIG_USB_EHCI_HCD=y
CONFIG_USB_EHCI_TT_NEWSCHED=y
CONFIG_USB_OHCI_HCD=y
CONFIG_USB_OHCI_LITTLE_ENDIAN=y
CONFIG_USB_STORAGE=y
CONFIG_RTC_LIB=y
CONFIG_RTC_CLASS=y
CONFIG_RTC_HCTOSYS=y
CONFIG_RTC_HCTOSYS_DEVICE="rtc0"
CONFIG_RTC_INTF_SYSFS=y
CONFIG_RTC_INTF_PROC=y
CONFIG_RTC_INTF_DEV=y
CONFIG_RTC_DRV_M48T59=y
CONFIG_SUN_OPENPROMIO=y
CONFIG_EXT2_FS=y
CONFIG_EXT2_FS_XATTR=y
CONFIG_EXT2_FS_POSIX_ACL=y
CONFIG_EXT2_FS_SECURITY=y
CONFIG_EXT3_FS=y
CONFIG_EXT3_DEFAULTS_TO_ORDERED=y
CONFIG_EXT3_FS_XATTR=y
CONFIG_EXT4_FS=y
CONFIG_EXT4_FS_XATTR=y
CONFIG_JBD=y
CONFIG_JBD2=y
CONFIG_FS_MBCACHE=y
CONFIG_FS_POSIX_ACL=y
CONFIG_FILE_LOCKING=y
CONFIG_FSNOTIFY=y
CONFIG_DNOTIFY=y
CONFIG_INOTIFY_USER=y
CONFIG_AUTOFS4_FS=y
CONFIG_ISO9660_FS=y
CONFIG_UDF_FS=y
CONFIG_UDF_NLS=y
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_FAT_DEFAULT_CODEPAGE=437
CONFIG_FAT_DEFAULT_IOCHARSET="iso8859-1"
CONFIG_PROC_FS=y
CONFIG_PROC_KCORE=y
CONFIG_PROC_SYSCTL=y
CONFIG_PROC_PAGE_MONITOR=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
CONFIG_MISC_FILESYSTEMS=y
CONFIG_ROMFS_FS=y
CONFIG_ROMFS_BACKED_BY_BLOCK=y
CONFIG_ROMFS_ON_BLOCK=y
CONFIG_NETWORK_FILESYSTEMS=y
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
CONFIG_ROOT_NFS=y
```

```
CONFIG_LOCKD=y
CONFIG_LOCKD_V4=y
CONFIG_NFS_COMMON=y
CONFIG_SUNRPC=y
CONFIG_SUNRPC_GSS=y
CONFIG_RPCSEC_GSS_KRB5=y
CONFIG_PARTITION_ADVANCED=y
CONFIG_MSDOS_PARTITION=y
CONFIG_LDM_PARTITION=y
CONFIG_SUN_PARTITION=y
CONFIG_NLS=y
CONFIG_NLS_DEFAULT="iso8859-1"
CONFIG_NLS_CODEPAGE_437=y
CONFIG_NLS_CODEPAGE_850=y
CONFIG_NLS_CODEPAGE_852=y
CONFIG_NLS_ISO8859_1=y
CONFIG_NLS_ISO8859_2=y
CONFIG_NLS_UTF8=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_ENABLE_MUST_CHECK=y
CONFIG_FRAME_WARN=1024
CONFIG_MAGIC_SYSRQ=y
CONFIG_DEBUG_KERNEL=y
CONFIG_DETECT_HUNG_TASK=y
CONFIG_BOOTPARAM_HUNG_TASK_PANIC_VALUE=0
CONFIG_DEBUG_BUGVERBOSE=y
CONFIG_DEBUG_INFO=y
CONFIG_RCU_CPU_STALL_DETECTOR=y
CONFIG_KEYS=y
CONFIG_DEFAULT_SECURITY_DAC=y
CONFIG_DEFAULT_SECURITY=""
CONFIG_CRYPTO=y
CONFIG_CRYPTO_ALGAPI=y
CONFIG_CRYPTO_ALGAPI2=y
CONFIG_CRYPTO_AEAD=y
CONFIG_CRYPTO_AEAD2=y
CONFIG_CRYPTO_BLKCIPHER=y
CONFIG_CRYPTO_BLKCIPHER2=y
CONFIG_CRYPTO_HASH=y
CONFIG_CRYPTO_HASH2=y
CONFIG_CRYPTO_RNG2=y
CONFIG_CRYPTO_PCOMP2=y
CONFIG_CRYPTO_MANAGER=y
CONFIG_CRYPTO_MANAGER2=y
CONFIG_CRYPTO_MANAGER_DISABLE_TESTS=y
CONFIG_CRYPTO_NULL=y
CONFIG_CRYPTO_WORKQUEUE=y
CONFIG_CRYPTO_AUTHENC=y
CONFIG_CRYPTO_CBC=y
CONFIG_CRYPTO_ECB=y
CONFIG_CRYPTO_PCBC=y
CONFIG_CRYPTO_HMAC=y
CONFIG_CRYPTO_CRC32C=y
CONFIG_CRYPTO_MD4=y
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_MICHAEL_MIC=y
CONFIG_CRYPTO_SHA1=y
CONFIG_CRYPTO_SHA256=y
CONFIG_CRYPTO_SHA512=y
CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_ARC4=y
CONFIG_CRYPTO_BLOWFISH=y
CONFIG_CRYPTO_CAST5=y
CONFIG_CRYPTO_CAST6=y
CONFIG_CRYPTO_DES=y
CONFIG_CRYPTO_SERPENT=y
CONFIG_CRYPTO_TWOFISH=y
CONFIG_CRYPTO_TWOFISH_COMMON=y
CONFIG_CRYPTO_DEFLATE=y
CONFIG_BITREVERSE=y
CONFIG_GENERIC_FIND_LAST_BIT=y
CONFIG_CRC16=y
CONFIG_CRC_ITU_T=y
CONFIG_CRC32=y
CONFIG_LIBCRC32C=y
CONFIG_ZLIB_INFLATE=y
CONFIG_ZLIB_DEFLATE=y
CONFIG_DECOMPRESS_GZIP=y
CONFIG_HAS_IOMEM=y
CONFIG_HAS_IOPORT=y
CONFIG_HAS_DMA=y
CONFIG_NLATTR=y
```

# References


[1] *Admob Mobile Metrics Metrics Highlights May 2010.*
http://metrics.admob.com/wp-content/uploads/2010/06/May-2010-AdMob-Mobile-Metrics-Highlights.pdf
http://goo.gl/7Cu2h
Accessed 18 Jul 2011.

[2] *ARM MeeGo Wiki.*
http://wiki.meego.com/ARM
Accessed 18 Jul 2011.

[3] *Dropping sparc32 for lenny.*
http://lists.debian.org/debian-sparc/2007/04/msg00044.html
http://goo.gl/JedtE
Accessed 18 Jul 2011.

[4] *FT.com Markets Data ARM Holdings PLC.*
http://markets.ft.com/ft/tearsheets/performance.asp?s=ARMH:NSQ
http://goo.gl/SLTH2
Accessed 18 Jul 2011.

[5] *Getting Started with the MeeGo SDK 1.1 for Linux MeeGo Wiki.*
http://wiki.meego.com/SDK/Docs/1.1/Getting_started_with_the_MeeGo_SDK_for_Linux
http://goo.gl/fmFMo
Accessed 18 Jul 2011.

[6] *GRLEON4ITX Brochure.*
http://www.gaisler.com/doc/LEON4_Mini-ITX_Mainboard.pdf
http://goo.gl/FRDm0
Accessed 18 Jul 2011.

[7] *GRLEON4ITX LEON4 Development Board.*
http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=339&Itemid=

http://goo.gl/V6mCV
Accessed 18 Jul 2011.

[8] *How to compile a Ubuntu Lucid kernel.*
http://blog.avirtualhome.com/2010/05/05/how-to-compile-a-ubuntu-lucid-kernel
http://goo.gl/tyMhz
Accessed 18 Jul 2011.

[9] *Installation Quick Start OpenSUSE 11.4.*
http://doc.opensuse.org/products/opensuse/openSUSE/opensuse-startup/art.osuse.installquick.html
http://goo.gl/GD3zh
Accessed 18 Jul 2011.

[10] *Intel and Nokia Merge Software Platforms for Future Computing Devices.*
http://www.intel.com/pressroom/archive/releases/2010/20100215corp.htm
http://goo.gl/AnIU7
Accessed 18 Jul 2011.

[11] *LEON Development Boards.*
http://www.gaisler.com/cms/index.php?option=com_content&task=section&id=9&Itemid=29
http://goo.gl/G6CQM
Accessed 18 Jul 2011.

[12] *LEON4 Processor.*
http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=338&Itemid=231
http://goo.gl/9mYgu
Accessed 18 Jul 2011.

[13] *LEON4 Product Sheet.*
http://www.gaisler.com/doc/LEON4_32-bit_processor_core.pdf
http://goo.gl/XvKav
Accessed 18 Jul 2011.

[14] *Markets & Applications.*
http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=119&Itemid=40
http://goo.gl/BVpI2
Accessed 18 Jul 2011.

[15] *Maximum RPM.*
http://www.rpm.org/max-rpm
Accessed 18 Jul 2011.

[16] *MeeGo Architecture Layer View.*
https://meego.com/developers/meego-architecture/
meego-architecture-layer-view
http://goo.gl/YLx0F
Accessed 18 Jul 2011.

[17] *MeeGo Build Service.*
http://build.meego.com
Accessed 18 Jul 2011.

[18] *meegocoreia32maddesysroot1.1.log.*
http://mirrors4.kernel.org/meego/releases/1.1/core/images/
meego-core-ia32-madde-sysroot/meego-core-ia32-madde-sysroot-1.1.log
http://goo.gl/uPlly
Accessed 18 Jul 2011.

[19] *Nokia and Microsoft Announce Plans for a Broad Strategic Partnership to Build a New Global Mobile Ecosystem.*
http://www.microsoft.com/presspass/press/2011/feb11/02-11partnership.mspx
http://goo.gl/APPAF
Accessed 18 Jul 2011.

[20] *OpenSUSE Build Service.*
http://en.opensuse.org/Portal:Build_Service
http://goo.gl/hr0qg
Accessed 18 Jul 2011.

[21] *Public Key Criptography Wikipedia, the free encyclopedia.*
http://en.wikipedia.org/wiki/Public-key_cryptography
http://goo.gl/h4gAv
Accessed 18 Jul 2011.

[22] *RPM Package Manager.*
http://rpm5.org
Accessed 18 Jul 2011.

[23] *SDK MeeGo Wiki.*
http://wiki.meego.com/SDK
Accessed 18 Jul 2011.

[24] *Slackware Package Management.*
http://www.slackbook.org/html/package-management.html%
urlhttp://goo.gl/QGy3e
Accessed 18 Jul 2011.

[25] *SOC Library.*
http://www.gaisler.com/cms/index.php?option=com_content&task=section&id=13&Itemid=125
http://goo.gl/BrfJG
Accessed 18 Jul 2011.

[26] *SSH Authentication Protocol.*
http://www.ietf.org/rfc/rfc4252.txt
Accessed 18 Jul 2011.

[27] *The Debian package management tools.*
http://www.debian.org/doc/FAQ/ch{-}pkgtools.en.html
% urlhttp://goo.gl/FQFko
Accessed 18 Jul 2011.

[28] *The SPARC Architecture Manual Version 8.*
http://www.sparc.org/standards/V8.pdf
http://goo.gl/tfG9f
Accessed 18 Jul 2011.

[29] *Version 7 Unix Wikipedia, the free encyclopedia.*
http://en.wikipedia.org/wiki/Version_7_Unix
http://goo.gl/jQVjO
Accessed 18 Jul 2011.

[30] *Xfce Desktop Environment.*
http://www.xfce.org
Accessed 18 Jul 2011.