Sculpt with Visual Composition (VC)

Norman Feske

September 21, 2018

Contents

1 Introduction

Sculpt is a component-based desktop operating system that puts the user in the position of full control. It is empowered by the Genode OS Framework, which provides a comprehensive set of building blocks, out of which custom system scenarios can be created. The name Sculpt hints at the underlying idea of crafting, molding, and tweaking the system interactively. Starting from a fairly minimalistic and generic base system, this tour through the Sculpt system will cover the following topics:

- A boot image that is a live system, rescue system, and bootstrap system all in one,
- Ways to tweak and introspect the system,
- Formatting a hard disk or USB stick and storing files on the file system,
- Connecting to a wired or wireless network,
- Installing and deploying software, and
- Running a guest operating system inside a virtual machine.

Feedback and contact Your feedback is appreciated!

Join the Genode mailing list for discussion

https://genode.org/community/mailing-lists

Get in touch with the developers at GitHub

https://github.com/genodelabs/genode

Contact Genode Labs for commercial inquiries

https://www.genode-labs.com

A printable PDF version of this document is available at https://genode.org/documentation/sculpt-vc.pdf.

2 Prerequisites

Sculpt with Visual Composition (VC) is the third of four revisions planned for 2018 with a successively increased ease of use. It features a graphical user interface for performing fundamental tasks like connecting to a wireless network, or installing and running software from packages. However, the full power of the system is accessible only via a textual interface.

2.1 Vim skills recommended

Sculpt VC leverages (a subset of) GNU coreutils, bash, and Vim as the user interface for sculpting the system. If you are not yet familiar with using Vim, you may take Sculpt VC as a welcome chance to get your toes wet. To enjoy the experience, you should be comfortable with the following operations:

- Opening and navigating within a text file (moving the cursor, using `/` to search),
- Using the insert mode to make modifications,
- Reverting accidental modifications (`u` undo),
- Saving a modified file (`:w`),
- Opening a file in a secondary buffer (`:e`),
- Switching between buffers (`:bn` for next, `:bp` for previous),
- Copy and paste (`v` start selection, `V` start line selection, `y` remember selection, `p` paste remembered selection),
- Exiting Vim (`:x` save and exit, `:q!` discard changes).

2.2 Hardware requirements and preparations

Sculpt VC should be compatible with recent Intel-based PC hardware featuring Intel graphics, E1000 networking, Intel wireless, and AHCI.

It is tested best on laptops of the Lenovo X and T series (X220, X250, X260, T430, T460, T470). For experimenting with Sculpt, we recommend getting a refurbished version of one of these. You may also find the unofficial hardware compatibility list https://usr.sysret.de/jws/genode/hcl.html helpful for finding Genode-compatible hardware.

Sculpt has been tested with screen resolutions up to 2560 x 1440. Displays with a higher resolution are not expected to work. The sweet spot is a full-HD display.

Please revisit the BIOS settings of your machine in the following respects:

VT-d enabled Even though Sculpt is able to run without an IOMMU, we advise to enable this option for the sandboxing of device drivers.

VT-x enabled Hardware-assisted virtualization is needed to run VirtualBox on top of Sculpt.

Boot from USB enabled Sculpt is usually booted from a USB stick.

UEFI boot enabled Sculpt boots via UEFI by default. The boot image is specially prepared such that it can be started via legacy boot on older machines. However, booting it via legacy boot on a modern machine is hit or miss.

UEFI secure boot disabled The Sculpt boot image is not cryptographically signed.

Optimize for performance when battery powered If the latter is not set, the hardware may behave erratically (e. g., non-working trackpoint when on battery).

3 Getting a first impression

Sculpt is best explored by first booting the prebuilt disk image downloadable from https://genode.org/download/sculpt. Right after booting the system, Sculpt's system-management user interface ("Leitzentrale") appears. The menu on the left provides convenient access to the connected storage devices and the network configuration. The center displays a live graph (runtime view) of the running components and their relationships. On the right, diagnostic messages are presented.

1. Select the in-memory file system as default storage location by clicking on the "ram" item of the "Storage" dialog and enabling the "Use" button. This instructs Sculpt that installed software is stored in memory without accessing any real storage device.

2. Enable networking in the "Network" dialog by selecting the "Wired" or "Wifi" option. In the latter case, select an access point and enter the corresponding passphrase (if needed). The successful network connection is indicated by the IP address displayed at the bottom of the network dialog.

3. With a storage location selected and network connectivity, it is time to install and start additional components by clicking on the + button of the runtime view and selecting a component from the context menu. When started for the first time, the ingredients of the selected subsystem are downloaded to the "used" storage location. Once the download is complete, the subsystem is started. As a first try, select the "backdrop" item. You can follow the progress of the installation procedure in the "Runtime" dialog. Once the installation is complete, you should notice a slight visual change.

4. Press F12 to toggle between the Leitzentrale and the actual runtime. Now, the backdrop should become visible in full glory.

5. Try adding additional components by selecting items in the "+" context menu of the runtime view. Most components expect the presence of a window manager. Hence, you should first select "wm". Please pay attention to diagnostic messages given in the runtime dialog on the left. Whenever a component depends on another one, a corresponding message appears.

6. You may click on any component in the runtime view to reveal additional information such as its memory usage. For components that you started manually, a remove button is displayed.

The following example subsystems are available from the "+" menu:

`fonts_fs` A file-system server that transforms TrueType fonts into glyph images, which become thereby accessible as virtual files. This provides a hook for customizing the font size of any component that uses the font server, and relieves components from depending on a specific font-rendering library. According to the <`route`> information, its configuration is taken from */config/managed/fonts*. The `fonts_fs` is used by the graphical terminal of the noux subsystem and the `top_view` application.

`wm` A window manager that displays clients in windows that can be arranged with the mouse.

`backdrop` A wallpaper that adjusts itself to any screen size.

`nano3d` A simple software-rendering demo, which can be adjusted at runtime by modifying its configuration via the textual interface described in Section 5. For example, by adding a custom config node directly inside the <`start`> node, the appearance can be changed on the fly:

```
<config painter="shaded" shape="cube"/>
```

`noux-system` A noux instance with a graphical terminal, similar to the inspect window of the Leitzentrale. Note the routing of the various file-system sessions when selecting the component in the runtime view.

`shared_fs` A file-system server that provides the */shared* sub directory of the Sculpt file system as a new file system. A client of this server will not see any other parts of the file system.

`usb_devices_rom` A hook for assigning USB devices to a virtual machine, explained in Section 7.4.

`vm_fs` A file-system server that provides the */vm/debian/* sub directory of the Sculpt file system as a new file system. It is explained in Section 6.

`top_view` An application that shows the CPU load, similar to `top`.

`2048` A *Threes!* inspired puzzle game running in a native Libretro runtime.

`vbox5-tc-browser` A throw-away virtual machine for running Firefox on TinyCore Linux. It uses VirtualBox as virtual-machine monitor.

`seoul-tc-browser` The same virtual machine as `vbox5-tc-browser` but executed inside the light-weight Seoul virtual-machine monitor.

`config_editor` Qt5-based text editor that is explicitly granted access to the config file system.

`arora` Qt5-based web browser, which does not touch any persistent file system.

`acpica` ACPI driver, which reports power-management state to */report/runtime/acpica/* and responds to changes of the */config/system* state.

`report_dump` A subsystem that periodically copies the content of the report file system to the default file system. Please refer to Section 7.5 for more information.

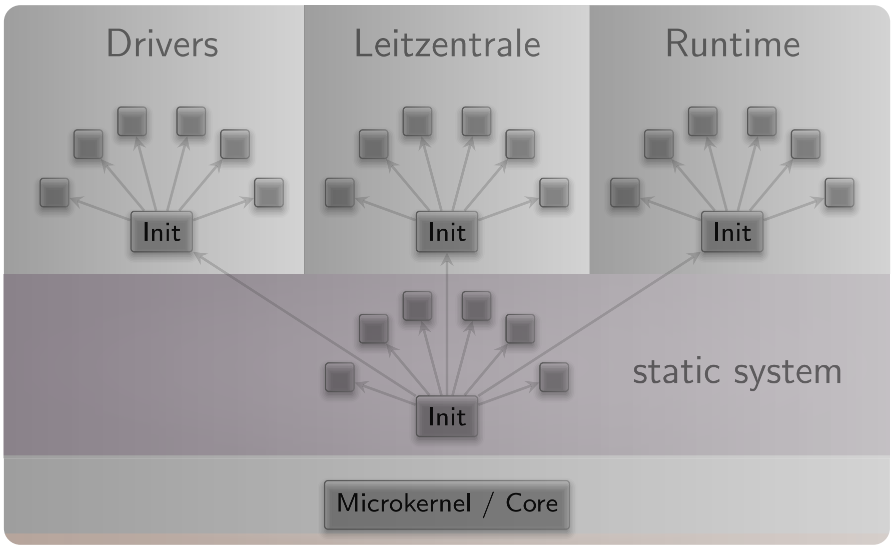

Figure 1: System overview

4 Base system

Unless customized, the Sculpt base system resides as a self-contained live operating system on a USB stick, not installed on disk. This has two advantages. First, it makes the update of the base system straight-forward and completely risk-free. Simply install the new version on a second USB stick. Should the new version cause any trouble, one can fall back to the original one by swapping the USB sticks. Second, it alleviates the need to install any boot-loader infrastructure on disk. In fact, one can use an entire disk as one file system without creating a partition table.

Note that Genode is not limited to booting from USB. It also supports the use of partitions. But for this guide, we keep things as simple as possible.

4.1 System overview

The Sculpt system consists of four parts living on top of the microkernel (Figure 1).

Static system The first - static - part of the system is baked into the boot image. It contains components that must be shared by the upper - dynamic - parts and defines the relationships between the upper parts via a static policy that is fixed by the creator of the boot image.

Besides a low-complexity GUI multiplexer called Nitpicker, the static system contains two in-memory file systems. The *config* file system stores configuration data whereas the *report* file system stores information reported by components. These file systems

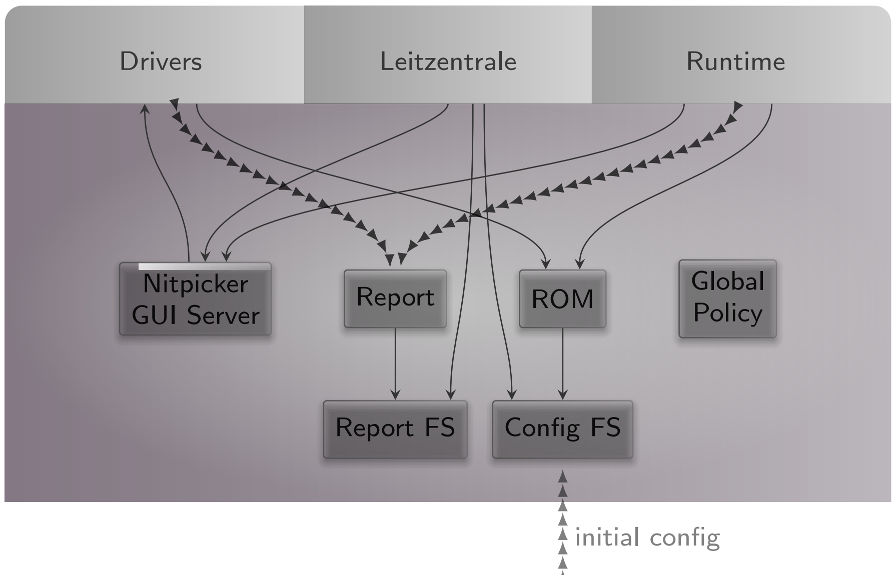

Figure 2: Detailed look at the static part of the system

are invisible to regular components. Components obtain their configuration data from a (read-only memory) ROM service, and report their state to a (write-only) report service.

At boot time, the config file system is pre-populated with information from the boot image. It stays in memory. Hence, after rebooting the system, any changes are gone.

Drivers subsystem The drivers subsystem provides all the basic services needed to realize an interactive system scenario: a framebuffer service for the graphical output, an input service to obtain user input, and a block service to access a storage device. All other drivers like networking or audio drivers are not covered by the drivers subsystem. They will enter the picture at a later stage and will use the platform service and USB service to access device resources.

As illustrated by Figure 3, some drivers like the framebuffer driver live in a dynamically managed subsystem that depends on runtime discovery of the hardware by the so-called driver-manager component. Whenever an Intel graphics device is present, the Intel framebuffer driver is spawned. Otherwise, a generic VESA driver or a driver for a boot-time-initialized framebuffer is used.

Several components of the drivers subsystem report their state. For example, when the Intel framebuffer is used, it reports the list of connectors present. Most importantly, the driver manager reports the available block devices.

As user input may enter the system in multiple ways - most prominently PS/2 and USB HID - the drivers subsystem contains a so-called input-filter component that

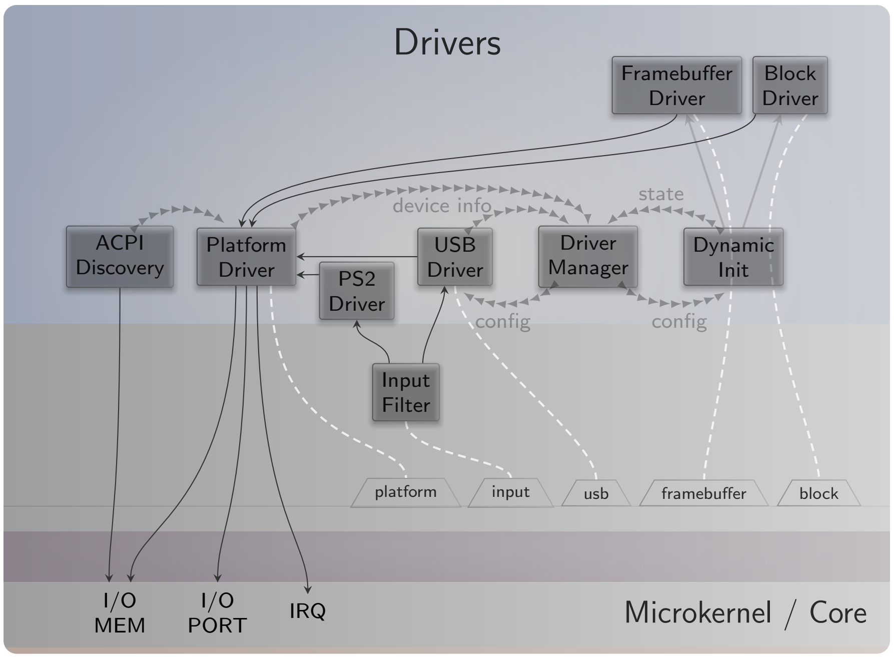

Figure 3: Services provided by the drivers subsystem

merges these event streams and applies transformations like key remappings or mouse acceleration.

Leitzentrale subsystem The Leitzentrale gives you - the user - full control over the config file system and the report file system. You are free to inspect and manipulate the system in any way you wish. The German term Leitzentrale refers to a control center that requires a certain degree of sophistication from the operator, which would be you. A typo at the wrong place may render your system temporarily inaccessible, eventually requiring a reboot. But don't be afraid. Since all manual changes performed in the Leitzentrale occur in memory only, you are not at risk of permanently bricking your machine.

The Leitzentrale can be toggled at any time by pressing F12 and will be enabled right after boot. It presents itself with a minimalistic GUI for accessing the storage devices attached to your machine and for configuring your network connectivity. Most importantly, however, it allows the user to spawn an interactive shell for manual *config* and *report* file systems access. To spawn this command-line interface, click on the "ram" item from the menu and select "Inspect". While inspecting file systems, the inspect window replaces the runtime view. However, both views can be toggled by clicking on the title of the storage dialog for the inspect window, or any other dialog for the runtime view.

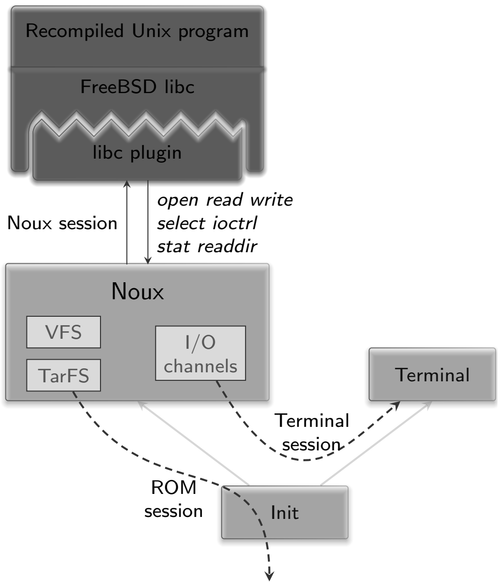

Figure 4: Noux runtime environment for executing Unix tools

The inspect window hosts a small Unix runtime called noux (Figure 4) as user interface. Don't let the presence of a Unix shell mislead you. Sculpt is not a Unix system. It merely uses Unix subsystems in the form of noux instances as convenient tools for managing and editing files. Within the inspect window, you can interact with both the report and config file systems using familiar commands such as the bash shell, a subset of coreutils, and Vim.

Besides the interactive shell, the Leitzentrale employs a noux instance that gives you a quick glance at the most recent log messages. The log is also available at *report/log* and can be browsed with Vim in the inspect window.

Noux is not bullet-proof. Should you get stuck, you may re-spawn it at any time by toggling the "Inspect" button.

4.2 Tweaking and inspecting the system

The Leitzentrale subsystem empowers you to interactively inspect and tweak the running system. Let's take a walk next.

Adjusting the user-input handling By default, Sculpt uses the US-English keyboard layout with a functioning capslock key. You may possibly want to adjust the former and - as a Vim user - most likely discharge the latter. As mentioned in Section [?],

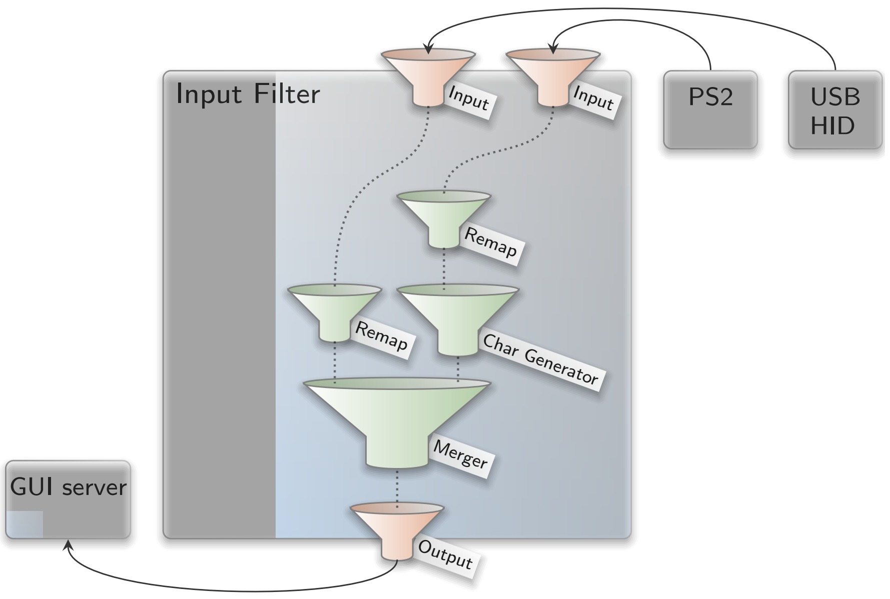

Figure 5: Filter chain for user-input events

user input is processed by the input-filter component. You can edit this component's configuration via

```
inspect:/> vim /config/input_filter
```

To change the keyboard layout to German, change "en_us.chargen" to "de.chargen" and save the file. The change becomes effective immediately at saving time.

To remap the capslock key to escape - a key often needed while using Vim - uncomment the corresponding <`remap`> rule

```
<key name="KEY_CAPSLOCK" to="KEY_ESC"/>
```

After saving the file, a Vim user's life suddenly becomes much more pleasant.

Take the time to review the remaining parts of the input-filter configuration. The nested configuration nodes define a hierarchy of filters that are applied in the order from the inside to outside (Figure 5). There are filters for merging events (<`merge`>), remapping buttons and keys (<`remap`>), supplementing symbolic character information (<`chargen`>), pointer acceleration (<`accelerate`>), and emulating a scroll wheel by moving the pointer while pressing the middle mouse button (<`button-scroll`>).

Display settings If you are running the Intel graphics driver, you can inspect the connected displays and their supported resolutions by taking a look at the report at */report/drivers/dynamic/intel_fb_drv/connectors*. This report is updated whenever a display is connected or disconnected. You can use this information to enable or disable a display in the driver's configuration, which you can find at */config/fb_drv*.

For a quick test, change the attribute `height="768"` to `force_height="768"` (you may modify `width` analogously). When saving the file, the screen real-estate will forcibly be limited to the specified size. This is helpful during presentations where the projector has a lower resolution than the laptop's internal display. By specifying the beamer's resolution, both the laptop and the beamer show the same content.

Exploring the drivers and Leitzentrale subsystems You can review the construction plan of the drivers subsystem by opening the file */config/drivers* in Vim. In particular, it is interesting to follow the <`route`> rules to see how the various components are connected. But there is more. The configuration is live. It enables you to reconfigure individual components on-the-fly. For example, search for the <`start`> node of the PS/2 driver and add the attribute `verbose_keyboard="yes"` to the embedded <`config`> node. By saving the file, the changed configuration becomes effective. Any key pressed or released on the PS/2 keyboard will result in a log message on the right. You may revert this change (vim: `u`) and save the original version of the file.

Note that not all components are dynamically reconfigurable but many modern ones - in particular the init component and most long-running server components - are.

It is possible to forcibly restart a component by adding a 'version' attribute to the <`start`> node. Whenever the `version` value is changed, the component is re-spawned.

The component-specific configuration options are documented in the README files accompanying the respective components in the source tree.

Analogously to the drivers subsystem, you can find the construction plan for the Leitzentrale subsystem at */config/leitzentrale*. Try out the following tweaks:

- Change the transparency of the Leitzentrale by modifying the `alpha` attribute of the `fader` component.
- Change the font size of the `log_terminal` component from "10" to "18".

You may also enjoy tinkering with the configuration of the nitpicker GUI server, which is located at */config/nitpicker*. For example, you may change the background color or the labeling color of the "default" domain.

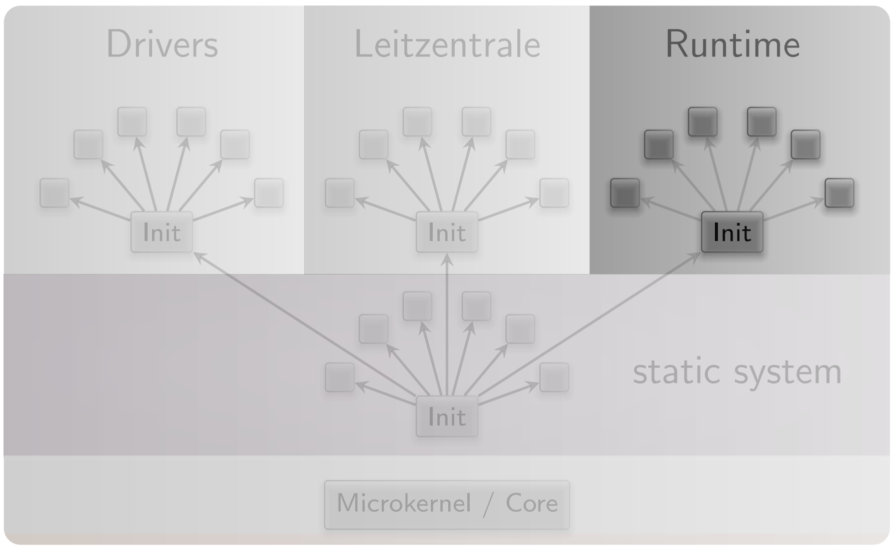

Figure 6

5 Runtime management

In contrast to the drivers subsystem and the Leitzentrale, which have a predefined purpose, the runtime subsystem is shaped by the user. The components present in the runtime subsystem are displayed by the runtime view. Some of them are managed by the Leitzentrale. For example, while inspecting a file system, the corresponding “inspect”, “inspect_terminal”, and “inspect_noux” components appear automatically. Other components correspond to subsystems deployed from installed packages, in particular the ones created via the runtime view’s "+" menu.

The current configuration of the runtime subsystem is available at */config/managed/runtime*. It is not recommended to modify this file manually. However, in some situations, it is useful to take manual control over the runtime configuration. This is possible by copying the file to *config/runtime*. Note that this will inhibit the management functionality of the Leitzentrale. You can yield back the control to the Leitzentrale by removing the */config/runtime* file.

As a prerequisite for deploying user-selected components, a default storage location must be defined by selecting the “Use” button of a file system in the menu. For the start, it is best to select the “ram” file system as storage location. Once you are comfortable with Sculpt, you may make the installation and customizations permanent by using a real storage device instead. The selection of a “used” file system has two immediate effects. First, any files present at *config/<version>/* at the selected file system are copied to the config file system. As the RAM file system is empty, no files are copied. Second, the so-called *depot/* is initialized at the selected file system. The depot is the designated place for the installation of software packages. By default, the depot is initialized such that the Sculpt system accepts software published by Genode's core developers. You may inspect the content of */ram/depot* using the inspect window.

With a file system and an Internet connection selected, additional software can be installed and run. The primary interface for software installation and deployment is the */config/deploy* file and the so-called launchers located at */config/launcher/*. The deploy file contains a number of commented-out template snippets for various subsystems. As a first test, uncomment the <`start`> entries for the *fonts_fs*, *wm*, and *backdrop*. When saving the file, the Sculpt manager will automatically kick off the download of the selected packages and its dependencies and thereby populate the depot. Once the download has completed, the packages are started.

Each <`start`> node refers a launcher according to the `name` attribute. It is possible to explicitly refer to a differently named launcher by specifying a `launcher` attribute. This way, one launcher can be instantiated multiple times. Pay special attention to the <`route`> definitions in the launchers. They define how the respective subsystem is connected to other parts of the system. For example, by default, the launcher for the backdrop connects the component directly to the nitpicker GUI server of the base system (parent). By changing the route from <`parent/`> to <`child name="wm"/`> the backdrop subsystem will be connected to the window manager instead. The files at */config/launcher/* are monitored by Sculpt and therefore can be edited on the fly. This is especially useful for editing <`config`> nodes.

A <`config`> node within a launcher - when present - overrides the one provided by the package. In turn, a <`config`> node within a node of the deploy config overrides any other <`config`> node. Both the launcher and a <`start`> node may contain a <`route`> node. The routing rules defined in the <`start`> node have precedence over the ones defined by the launcher. This way, the routing of a launcher can be parameterized at the deploy configuration.

Under the hood, the deployment is not directly controlled by */config/deploy*. Instead, Sculpt incorporates the user interaction with the runtime view and the information provided by */config/deploy* into the actually used deploy configuration at */config/managed/deploy*. Note that any modification of */config/deploy* resets */config/managed/deploy* to the state defined in */config/deploy*. To preserve interactive changes, you may copy */config/managed/deploy* to */config/deploy* before tweaking */config/deploy* manually.

5.1 Storage device access and preparation

Whereas the RAM file system is practical for quick tests, it goes without saying that we want to persistently store data, programs, and configuration information on a storage device. Sculpt supports SATA disks, NVMe devices, and USB-storage devices. The storage dialog lists all devices detected by the drivers subsystem. A click on a device reveals possible operations or - if a partition table is present - more details about the device structure.

Depending on the operation selected by the user, the Sculpt manager will automatically reshape the runtime subsystem to perform the selected operation. For example, by selecting the "Format device" operation, the Sculpt manager will create a noux instance with the selected block device mounted at `/dev/block` and e2fsprogs mounted at `/`. The noux instance runs `mkfs.ext2` as init process. Likewise, an existing EXT2 file system can be checked by activating the "Check" button, which triggers the execution of `fsck.ext2` for the selected file system.

A particularly interesting option is presented at the last partition of the Sculpt USB stick. Initially - right after copying Sculpt's tiny disk image to the USB stick - the partition is only a few MiB in size. However, using the "Expand" operation, the partition can be extended to the full size of the USB stick, which makes enough room to use the USB stick as Sculpt file system. This clears the way for sculpting a custom live system stored entirely on the USB stick.

All file systems supported by Sculpt present an "Inspect" button. By toggling this button, the selected file system becomes accessible in the inspect window. Note that more than one file system can be inspected at a time. Each file system will appear as a directory at the root of the inspect directory tree, named after the corresponding device and partition number. This way, the inspect window becomes a convenient tool for copying files between file systems. Under the hood, the Sculpt manager spawns a file-system component for each inspected file system, which translates the notion of files and directories to block-device accesses.

5.2 Making customizations permanent

It is possible to make any customization of the config file system permanent by copying the modified files to a directory named `config/`<`version`> on a persistent file system where <`version`> corresponds to the Sculpt version number as found in the `/VERSION` file. Each time, this file system is selected for "Use", those files will be automatically copied to the in-memory config file system. Note that this mechanism works even for the `/config/deploy` file, which allows one to restore a once sculpted system composition directly at boot time.

To streamline the boot procedure into a customized Sculpt system even more, it is possible to mark one file system as default. At boot time - when the Sculpt manager discovers the attached storage devices - it automatically selects a file system for use according to the following order of preference:

1. Partition labeled as "GENODE*" on a USB device,
2. Partition labeled as "GENODE*" on a SATA or NVMe storage device,
3. An entire SATA or NVMe device used as a single EXT2 file system (as devised by Sculpt EA).

The storage dialog hosts a convenient "Default" button that allows one to toggle a partition label between "GENODE" and "GENODE*". For example, the last partition of the Sculpt USB stick can be marked as default or non-default using this button.

6 Hosting a guest operating system

The default deploy configuration found at */config/deploy* and the launcher at */config/launcher/vm* contain all the pieces needed to host a virtual machine on top of Sculpt. A virtual machine (VM) is a convenient stop-gap solution for running programs that are not yet available natively on Genode. It ultimately enables us Genode developers to use Sculpt as day-to-day OS.

By convention, we host the content of each VM in a dedicated directory */vm/<guest-os>/* at the file system. The VM directory contains a virtual disk image(s) as well as the VM configuration. To install the ingredients for running Debian in the VM, you may start the `download_debian` subsystem, which will automatically download the ISO image of the Debian installer and install a reasonable VM configuration. The subsystem requests a file-system session that points to the target directory. To pass the */vm/debian* directory to the subsystem, the file-system session is routed to the `vm_fs` component. Please make sure to uncomment this component along with the `download_debian` subsystem.

Please review and adjust the *machine.vbox* file as needed, in particular you may reconsider the amount of RAM by changing the `RAMSize` attribute. To start the VM, remove the comments around the following snippets within */config/deploy*:

1. "wm" - the window manager that will host a window of the VM,
2. "vm_fs" - the location of the virtual-disk image and VM configuration,
3. "shared_fs" - the location for sharing files between the guest OS and other parts of the Sculpt system,
4. "usb_devices_rom" - a configuration that contains a list of USB devices passed to the VM,
5. "vm" - the virtual machine.

After saving the file, VirtualBox should appear, starting the Debian installer.

After the installation is finished and the guest system was rebooted, it is time to install the guest additions of VirtualBox. To do that, the apt(1) configuration has to be adjusted. Edit the file

```
# vi /etc/apt/sources.list
```

and add the line

```
deb http://ftp.debian.org/debian stretch-backports main contrib non-free
```

Afterwards update the package cache

```
# apt update
```

and upgrade the packages

```
# apt upgrade
```

and install the Linux kernel headers

```
# apt install linux-headers-amd64
```

Just to be sure that the guest additions will use the newest kernel, reboot the guest system. Next, install all needed packages for the guest additions:

```
# apt install virtualbox-guest-dkms virtualbox-guest-x11
```

Having the Linux-header package is mandatory as the needed modules will not be built without it. After the packages are installed and the modules have been built, certain features like the dynamic mode-setting and shared folders can be used.

The example *machine.vbox* file already provides a configured shared folder called `shared`. By executing

```
# mount -t vboxsf shared /mnt/
```

it can be mounted and accessed via `/mnt`.

7 Advanced usage

7.1 Manual configuration

Thanks to the Sculpt-manager component of the Leitzentrale, many typical work flows and configuration tweaks are largely automated. For example,

- The management of storage devices,
- The creation of file-system components for used or inspected file systems,
- The selection and configuration of network access,
- Font size selection depending on the screen resolution,
- Triggering the download of missing depot content on demand.

This convenience comes at the price of built-in policy, which may stand in the way of sophisticated scenarios. For this reason, almost all policies of the Sculpt manager can be manually overriden by manually managing configuration files.

The Sculpt manager interacts with the rest of the system solely by monitoring reports aggregated in the report file system, and writing configuration files into the config file systems. All files generated by the Sculpt manager are located at */config/managed/*. By manually creating a same-named file at */config/*, one can supply a custom managed configuration to the Sculpt manager. A useful practice is taking a snapshot of the generated configuration as a starting point for the custom version. For example, by copying the NIC router configuration while it is connected to a network:

```
cp /config/managed/nic_router /config
```

Now, the copy at */config/nic_router* can be edited. Note that changes usually take immediate effect.

Examples of manual customization are:

- Adding custom NIC router policies such as port-forwarding rules,
- Installing depot content manually by managing */config/installation* by hand. This includes the ability to download the source code for any package.
- Disarming the automated update mechanism by using a */config/installation* file with no <`archive`> entries.
- Fixing the current state of the runtime subsystem by copying */config/managed/runtime* to */config/runtime*. This allows one to manually tweak and inspect the runtime in any way, e. g., to enable additional reporting when troubleshooting.

- Manually defining the default font sizes by creating a custom *config/fonts* configuration.
- Managing Wifi credentials manually by supplying a custom *config/wifi* file.

To revert any manual customization, delete the corresponding file. In this case, the Sculpt manager will take over again. Note that all manual customizations can be made permanent by following the steps explained in Section 5.2.

7.2 Building the boot image

The following steps assume that you have the Genode tool chain installed on a GNU/Linux system. For reference, Ubuntu 16.04 is known to work well. If you don't know your way around Genode's source tree yet, please consider the "Getting started" section of the Genode Foundations book that is available as a free download at https://genode.org.

1. Clone Genode's Git repository:

   ```
   git clone https://github.com/genodelabs/genode.git
   cd genode
   git checkout -b sculpt_vc sculpt_vc
   ```

2. Download the support for the NOVA microkernel

   ```
   ./tool/depot/download genodelabs/bin/x86_64/base-nova/2018-09-19
   ```

   The content is downloaded to the *public/* directory and extracted to the *depot/* directory.

3. Download all ingredients for the Sculpt boot image

   ```
   ./tool/depot/download genodelabs/pkg/x86_64/sculpt/2018-09-21
   ```

4. Create a build directory

   ```
   ./tool/create_builddir x86_64
   ```

5. Configure the build directory by editing *build/x86_64/etc/build.conf*. Most importantly, enable the `gems` source-code repository where the Sculpt scenario resides. In addition, the `ports`, `dde_linux` and `dde_ipxe` repository are needed as well. Second, change the default configuration of the `QEMU_RUN_OPT` variable to `image/disk` instead of `image/iso`. This way, the build process will produce a valid disk image with a GPT partition table instead of a legacy ISO image.

6. Create the Sculpt boot image (defined by the run script at *repos/gems/run/sculpt.run*)

   ```
   make -C build/x86_64 run/sculpt KERNEL=nova
   ```

   The boot image is created at *build/x86_64/var/run/sculpt.img*.

7. Write the boot image to a USB stick:

   ```
   sudo dd if=build/x86_64/var/run/sculpt.img of=/dev/sdx bs=1M conv=fsync
   ```

   Here, `/dev/sdx` refers to the device node of your USB stick. To determine it, you may inspect the output of `dmesg` after plugging it in.

7.3 Reproducing the system from source

Section 7.2 presents the creation of the boot image from pre-built packages. You may want to build those packages from source, in particular for customizing the system.

Before building the packages, various ports of 3rd-party software need to be prepared. The following command prepares all of them at once:

```
<genode-dir>/tool/ports/prepare_port \
     bash coreutils curl dde_ipxe dde_linux \
     dde_rump e2fsprogs gnupg grub2 jitterentropy \
     libarchive libc libgcrypt libiconv libssh \
     lwip_legacy ncurses nova openssl qemu-usb \
     stdcxx vim virtualbox5 x86emu xz zlib libpng \
     ttf-bitstream-vera stb
```

The ingredients of the boot image are subsumed by the `pkg/sculpt` package. The default set of software installed by the update runtime is defined by the `pkg/sculpt-installation` package. You can find the depot recipes for these packages at *repos/gems/recipes/pkg/*.

The *repos/gems/run/sculpt.run* script can be executed to build a boot image. By default, the boot image refers to `genodelabs/pkg/sculpt` and to `genodelabs/pkg/sculpt-installation` for the runtime-installed software. You may want to install your version of these packages instead by changing the package provider from `genodelabs` to <`you`> by adding the line

```
RUN_OPT += --depot-user <you>
```

to your *<build-dir>/etc/build.conf*.
To build the packages for the boot image:

```
<genode-dir>/tool/depot/create \
     UPDATE_VERSIONS=1 FORCE=1 REBUILD= \
     <you>/pkg/x86_64/sculpt \
     <you>/bin/x86_64/base-nova
```

The `FORCE=1` argument ensures that source archives are re-created and checked for the consistency with their versions. If the source code of any of the archives changed, the `UPDATE_VERSIONS=1` argument automatically updates its version. Please don't forget to commit the updated `hash` files. The empty `REBUILD=` argument limits the creation of binary packages to those that do not yet exist in binary form. If not specified, the command would recompile all packages each time. You may further add `-j`<`N`> to parallelize the build process where <`N`> is the level of parallelism.
Building the `sculpt-installation` package works analogously to the `sculpt` package.

```
<genode-dir>/tool/depot/create \
     UPDATE_VERSIONS=1 FORCE=1 REBUILD= \
     <you>/pkg/x86_64/sculpt-installation
```

To make the `sculpt-installation` available for download from within the boot image, you must publish it. This involves the archiving, signing, and uploading of the content. The former two steps are covered by the *tool/depot/publish* tool, which expects one to specify a concrete version. The current version of the `sculpt-installation` can be obtained via

```
cat <genode-dir>/repos/gems/recipes/pkg/sculpt-installation/hash
```

The first part is the version. The second part is the content hash of the version. For more information about working with the depot tool, refer to https://genode.org/documentation/developer-resources/package_management.

The launchers integrated in the boot image are defined at *gems/run/sculpt/launcher/*. Each file contains a node with a mandatory pkg attribute. If the attribute value contains one or more `/` characters, it is assumed to be a complete pkg path of the form <`user`>`/pkg/`<`name`>`/`<`version`>. Otherwise it is assumed to be just the pkg name and is replaced by the current version of the current depot user's pkg at system-integration time.

7.4 Updating the USB boot device from within VirtualBox

The */config/deploy* example is prepared to assign USB storage devices directly to a running virtual machine. You may inspect the report */report/drivers/usb_active_config* to get a list of attached USB devices. Use Vim to copy the <`policy`> node of the selected device into the <`inline`> section within the */config/launcher/usb_devices_rom* file, and adjust the line as follows:

- Replace the node type <`policy`> by <`device`>, and
- Rename the attribute `label_suffix` to `label`.

The updated `usb_devices` ROM prompts VirtualBox to open a USB session at the drivers subsystem. Hence, when saving the modified */config/launcher/usb_devices_rom* file, the guest OS should detect a new USB device (check the output of `dmesg`). You may now write a new version of the Sculpt ISO image to the device by following the steps described in Section 7.2.

7.5 Sculpt as a hardware-probing instrument

Sculpt can be used as a convenient tool for probing Genode's compatibility with new hardware via the so-called `report_dump` subsystem, which periodically copies the content of Sculpt's report file system to the default file system.

First, a USB stick with a fresh Sculpt image is booted on a fully supported machine. The user then customizes the USB stick within the running system by expanding the USB stick's Genode partition, setting it as the default storage location, and deploying the `report_dump` subsystem. The last step triggers the installation of the `report_dump` package onto the USB stick. Finally, the user copies the deploy configuration from the in-memory config file system (*/config/deploy*) to the USB stick (*/usb-<N>/config/<VERSION>/deploy*). When booting this prepared USB stick, this deployment configuration becomes active automatically. At this point, the Sculpt system will copy a snapshot of the report file system to the Genode partition of the USB stick every 10 seconds. The snapshots captured on the USB stick can later be analyzed on another machine.

The snapshots not only contain all log messages (*/report/log*) but also the reports generated by various components of the drivers subsystem and any other deployed components. For example, with `acpica` present in the deploy configuration, the battery state is captured as well.

8 Credits

Sculpt is an example system scenario of the Genode project, which is an operating-system technology designed and developed from scratch.

Genode OS Framework https://genode.org

That said, Genode is not developed in a vacuum. It rather stands on the shoulders of giants and greatly benefits from the free-software/open-source community. The following projects play a particularly important role for the Sculpt scenario.

NOVA microhypervisor

Sculpt's kernel is a derivate of NOVA, maintained by Genode Labs. NOVA was originally created by Udo Steinberg http://hypervisor.org.

Linux kernel https://kernel.org

Sculpt reuses several Linux subsystems as individual components, in particular the USB stack, the Intel wireless stack, the Intel graphics driver, and the TCP/IP stack.

NetBSD's rump kernel https://wiki.netbsd.org/rumpkernel/

The file-system support is based on NetBSD kernel code, which became reusable on Genode thanks to the rump kernel project.

FreeBSD https://www.freebsd.org/

The C runtime that is used by most 3rd-part software is based on FreeBSD's libc.

Device drivers

WPA supplicant https://w1.fi/wpa_supplicant/ *(used by the wireless driver)*

iPXE http://ipxe.org *(basis of the wired network driver)*

xf86emu http://xorg.freedesktop.org/ *(used by the VESA driver)*

Programs and libraries used within the noux runtime

Vim http://www.vim.org

ncurses https://www.gnu.org/software/ncurses/ncurses.html

GNU coreutils https://www.gnu.org/software/coreutils/coreutils.html

GNU bash https://www.gnu.org/software/bash/

E2fsprogs http://e2fsprogs.sourceforge.net/

Libraries used for the package-management infrastructure

curl https://curl.haxx.se *(basis of the fetchurl tool)*

libssh https://www.libssh.org

OpenSSL https://www.openssl.org

XZ Utils https://tukaani.org/xz/ *(support for tar.xz archives)*

libarchive https://www.libarchive.org *(basis of the extract tool)*

zlib https://www.zlib.net

GnuPG https://www.gnupg.org *(basis of the verify tool)*

Applications

VirtualBox https://www.virtualbox.org *(used for hosting virtual machines)*

Crucial tools used during development

GNU/Linux (various distributions)

Git https://git-scm.com

GNU compiler collection https://gcc.gnu.org

GNU binutils https://www.gnu.org/software/binutils/

GNU make https://www.gnu.org/software/make/

Tcl https://www.tcl.tk

Expect http://expect.sourceforge.net

Qemu https://www.qemu.org

GitHub issues https://github.com