Composing System Services in GuixSD

or how we came to wonder what a "system service" really is

FOSDEM, February 2017

GuixSD

```
(operating-system
  (host-name "schememachine")
  (timezone "Europe/Brussels")
  (locale "fr_BE.utf8")
  (bootloader (grub-configuration (device "/dev/sda")))
  (file-systems (cons (file-system
                        (device "my-root")
                        (title 'label)
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))
  (users (cons (user-account
                 (name "charlie")
                 (group "users")
                 (home-directory "/home/charlie"))
               %base-user-accounts))
  (services (cons* (dhcp-client-service)
                   (service openssh-service-type)
                   %base-services)))
```

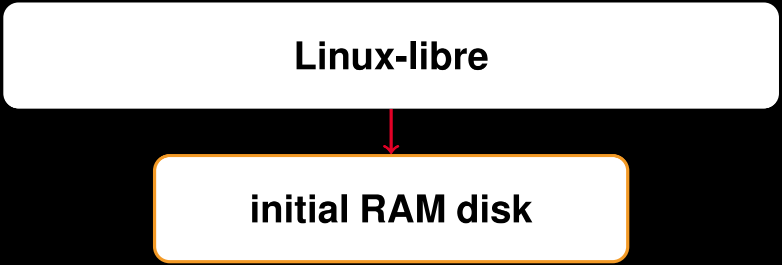
```
(service openssh-service-type
         (openssh-configuration
           (x11-forwarding? #t)
           (permit-root-login ’without-password)))
```

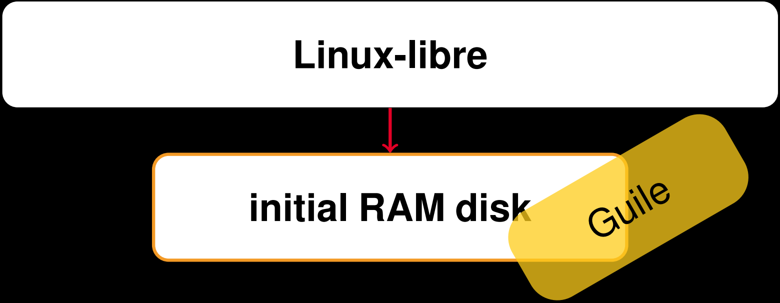
```
(operating-system
  ;; …
  (services (remove (lambda (service)
                      (eq? ntp-service-type
                           (service-kind service)))
                    %desktop-services)))
```

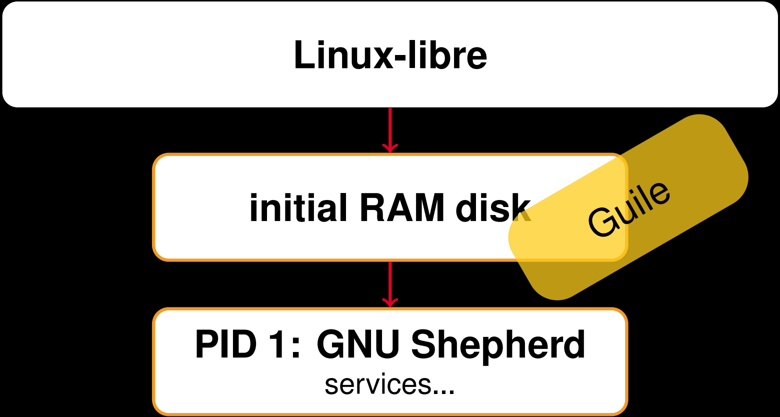
```scheme
(define %my-services
  ;; My very own list of services.
  (modify-services %desktop-services
    (mingetty-service-type config =>
                           (mingetty-configuration
                            (inherit config)
                            (motd (plain-file "motd"
                                    "Howdy FOSDEM!"))))
    (upower-service-type config =>
                         (upower-configuration
                          (inherit config)
                          (ignore-lid? #true)
                          (percentage-critical 5.)))))
```

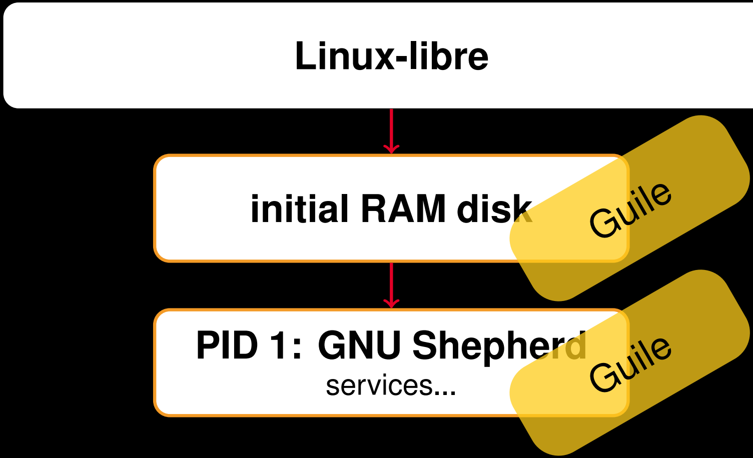
```
$ guix system build config.scm
…

$ guix system vm config.scm
…

$ guix system container config.scm
…

$ guix system reconfigure config.scm
…
```

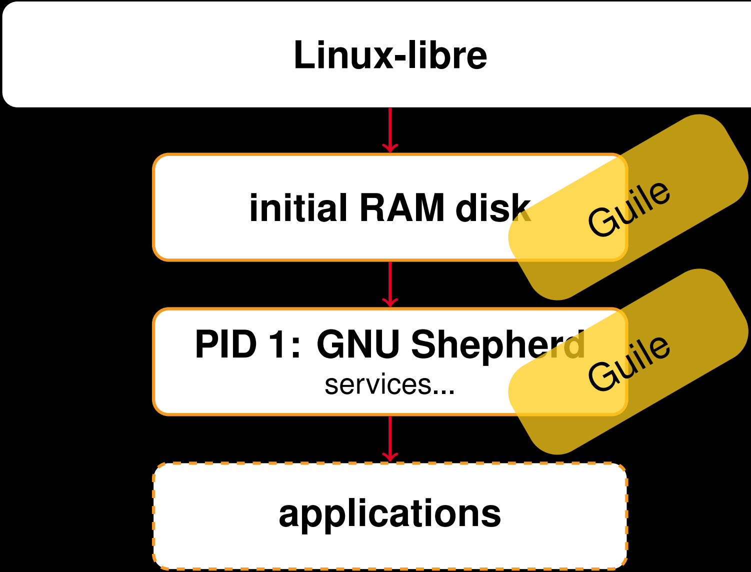
Linux-libre


Linux-libre
initial RAM disk


Linux-libre
initial RAM disk
Guile


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...
Guile


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...
Guile
applications

```scheme
;; Service definition for the GNU Shepherd (PID 1)
;; embedded in GuixSD.

(shepherd-service
  (provision '(mysql))
  (documentation "Run the MySQL server.")
  (start (let ((my.cnf (mysql-configuration-file config)))
           #~(make-forkexec-constructor
              (list (string-append #$mysql "/bin/mysqld")
                    (string-append "--defaults-file="
                                   #$my.cnf))
              #:user "mysql" #:group "mysql")))
  (stop #~(make-kill-destructor)))
```

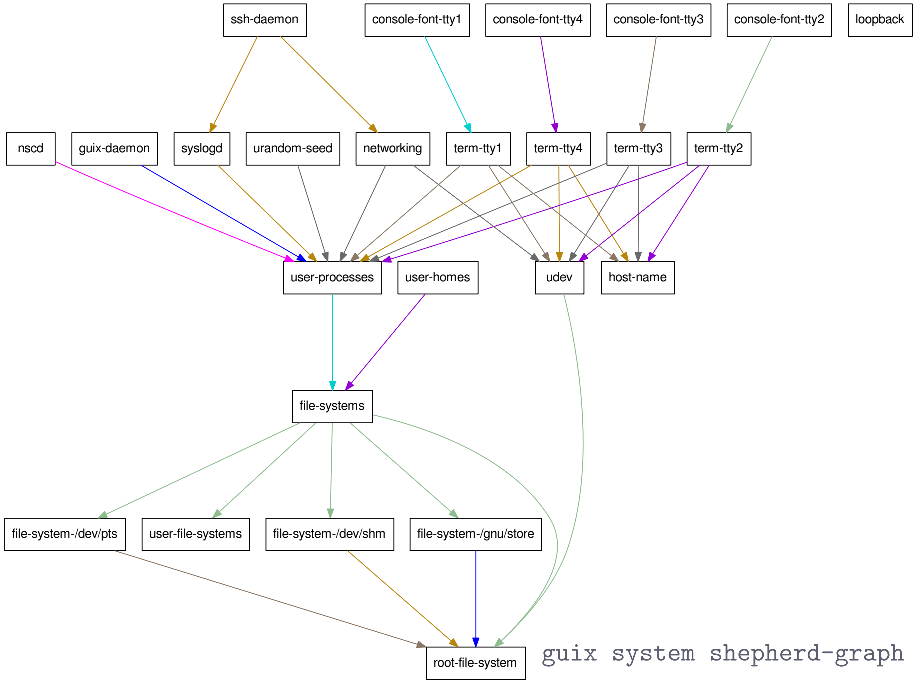
```
;; Shepherd service to mount/unmount a file system.

(with-imported-modules '((gnu build file-systems))
  (shepherd-service
    (provision '(file-system-/home))
    (start #~(lambda ()
               (mount "/dev/foo" "/home" "ext4")))
    (stop #~(lambda ()
              (umount "/home")))))
```

```
;; Shepherd service for the BitlBee IRC gateway daemon.


  (shepherd-service
    (provision '(bitlbee))
    (requirement '(loopback))
    (start #~(make-forkexec-constructor
              (list #$(file-append bitlbee "/sbin/bitlbee")
                    ...)))
    (stop  #~(make-kill-destructor)))
```

```
;; Shepherd service for the BitlBee IRC gateway daemon.
;; Running in a container!

(with-imported-modules '((gnu build linux-container))
  (shepherd-service
    (provision '(bitlbee))
    (requirement '(loopback))
    (start #~(make-forkexec-constructor/container
               (list #$(file-append bitlbee "/sbin/bitlbee"
                     ...)))
    (stop  #~(make-kill-destructor))))
```

world première!

Services, take #1.



`guix system shepherd-graph`

```
(service
  (provision '(postgres))
  (requirement '(user-processes loopback))
  (start #~(make-forkexec-constructor #$postgresql ...))
  (stop #~(make-kill-destructor)))
```

```
(service
  (provision '(postgres))
  (requirement '(user-processes loopback))
  (start #~(make-forkexec-constructor #$postgresql ...))
  (stop #~(make-kill-destructor))
  (activate #~(begin ...)))
```

```
(service
  (provision '(postgres))
  (requirement '(user-processes loopback))
  (start #~(make-forkexec-constructor #$postgresql ...))
  (stop #~(make-kill-destructor))
  (activate #~(begin ...))
  (user-groups (list (user-group
                      (name "postgres")
                      (system? #t))))
  (user-accounts (list (user-account
                        (name "postgres")
                        (group "postgres")
                        (system? #t)
                        (shell ...)))))
```

+ PAM

```
(service
  (provision '(postgres))
  (requirement '(user-processes loopback))
  (start #~(make-forkexec-constructor #$postgresql ...))
  (stop #~(make-kill-destructor))
  (activate #~(begin ...))
  (user-groups (list (user-group
                      (name "postgres")
                      (system? #t))))
  (user-accounts (list (user-account
                        (name "postgres")
                        (group "postgres")
                        (system? #t)
                        (shell ...)))))
```

```
(service
  (provision '(postgres))
  (requirement '(user-processes loopback))
  (start #~(make-forkexec-constructor #$postgresql ...))
  (stop #~(make-kill-destructor))
  (activate #~(begin ...))
  (user-groups (list (user-group
                       (name "postgres")
                       (system? #t))))
  (user-accounts (list (user-account
                         (name "postgres")
                         (group "postgres")
                         (system? #t)
                         (shell ...)))))
```

+ PAM

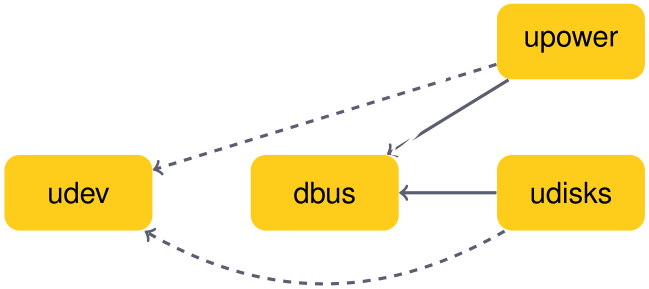
+ /etc

freedesktop.org

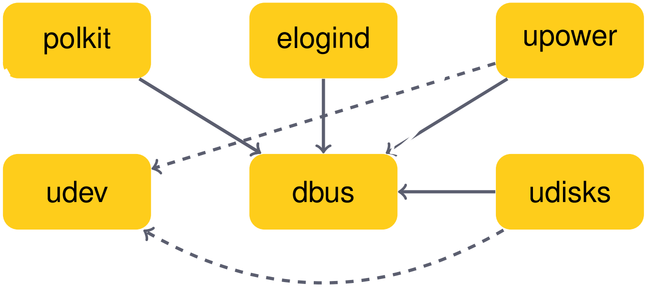
udev

dbus

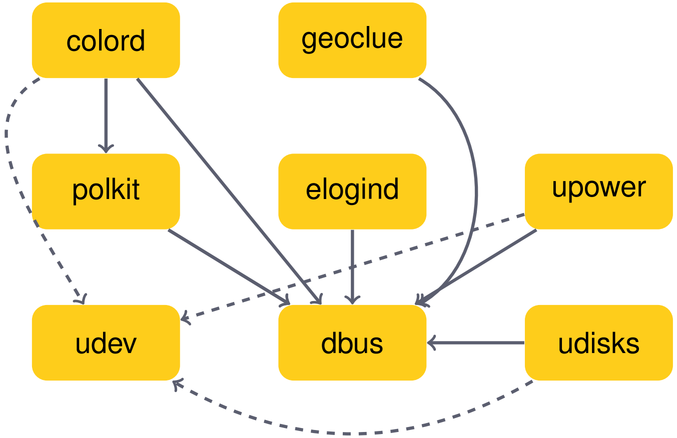
freedesktop.org



freedesktop.org



freedesktop.org



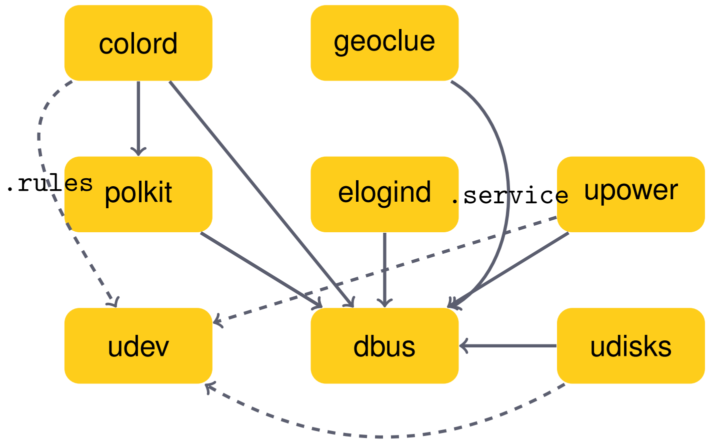
freedesktop.org

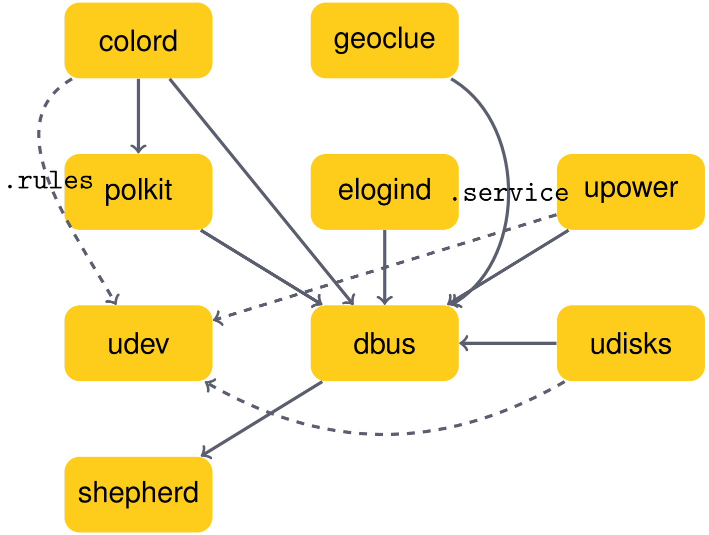
Composable services.

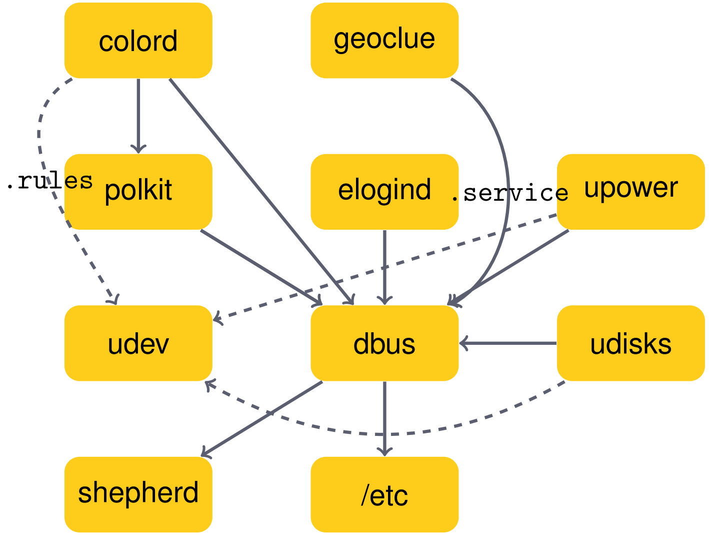
Key insight: services “extend” each other.

Digression:
NixOS configuration.

```
{ config, lib, pkgs, ... }:
let
  cfg = config.services.openssh;
in {
  options = …;

  config = mkIf cfg.enable {
    users.extraUsers.sshd = { isSystemUser = true; };
    environment.etc = authKeysFiles //
      { "ssh/moduli".source = cfg.moduliFile; };
    systemd.services.sshd-service =
      {  wantedBy = "multi-user.target";
         # …
      };
    security.pam.services.sshd =
      { startSession = true;
        unixAuth = cfg.passwordAuthentication;
      };
}
```


colord
geoclue
polkit
elogind
upower
udev
dbus
udisks
.rules
.service

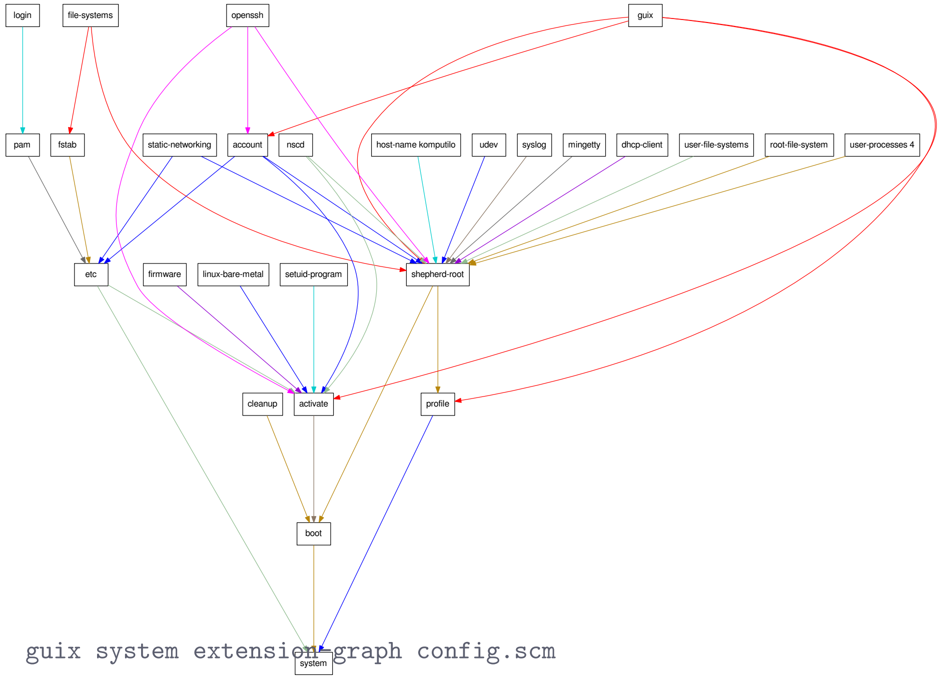

colord
geoclue
.rules
polkit
elogind
.service
upower
udev
dbus
udisks
shepherd


colord
geoclue
polkit
elogind
upower
.rules
.service
udev
dbus
udisks
shepherd
/etc

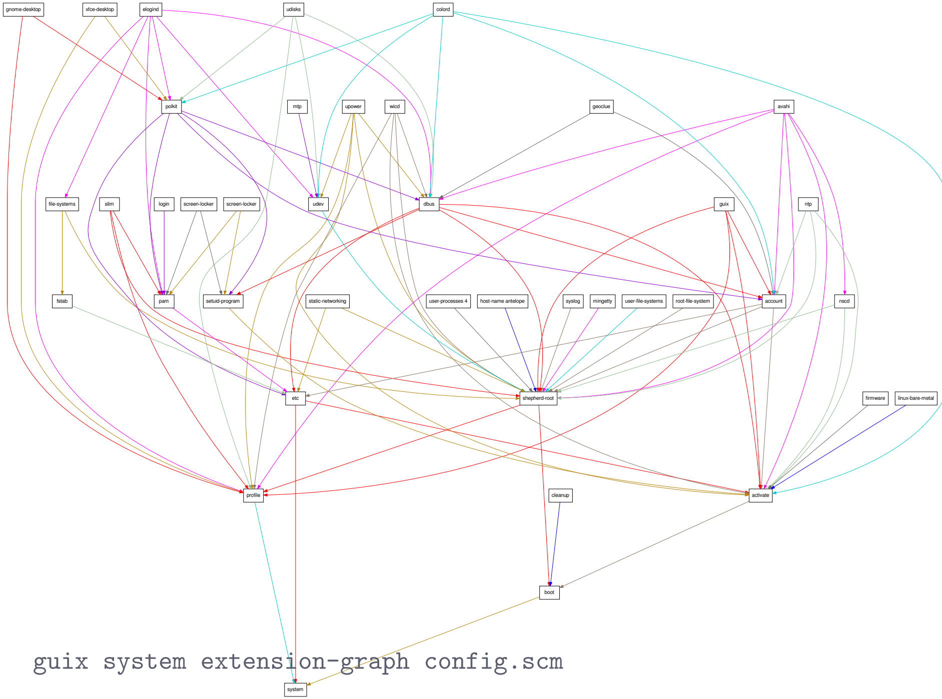
what users type

```
(operating-system
  (host-name "schememachine")
  ;; ...
  (services (cons* (dhcp-client-service)
                   (service openssh-service-type
                            (openssh-configuration
                              (x11-forwarding? #t)
                              (permit-root-login
                                'without-password)))
                   (service nginx-service-type ...)
                   %base-services)))
```

Services,
service types.



```
guix system extension-graph config.scm
```



```
guix system extension-graph config.scm
```

fold-services.

Dear Haskeller,
this is a monoid!

Wrap-up.

GuixSD leverages
a holistic approach
to system services.

- services can **use and extend** PID 1
- “service extensions” capture
  ***all* the service aspects**
- makes complex configurations **tractable**

- services can **use and extend** PID 1
- “service extensions” capture
  ***all* the service aspects**
- makes complex configurations **tractable**
- **come up with your own services!**

GuixSD

ludo@gnu.org

https://gnu.org/software/guix/