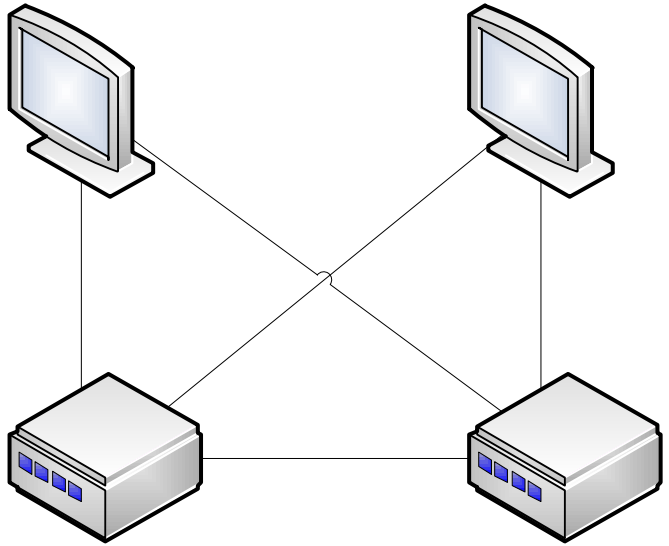
Weston Plugin Based IVI Layer Management

Ossama Othman

20 September 2012

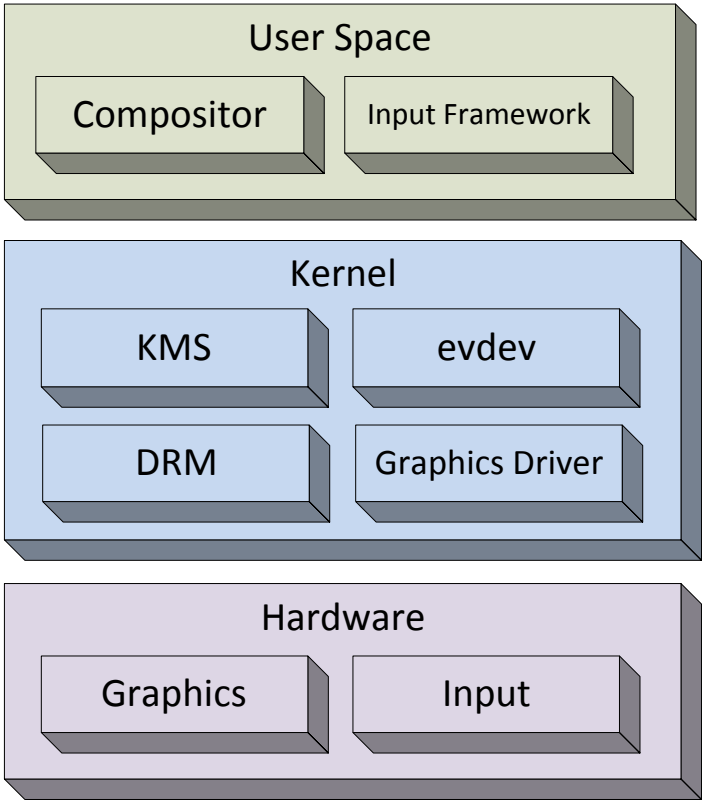
Context

- GNU/Linux based IVI platform
- Multiple screens
- Multiple ECUs
- Applications may be distributed across multiple screens and ECUs
- IVI-specific UI paradigms



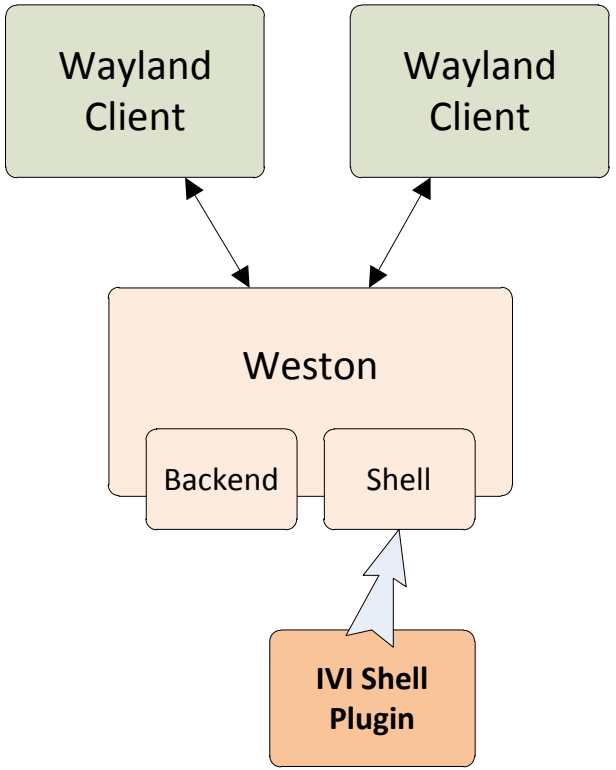
Problems

- Minimal resources for in-house development of suitable IVI compositor, etc.
- Costs
- Lack of expertise in interface between user space, kernel and graphics hardware
- Existing desktop solutions may not satisfy IVI UI requirements



Solution

- Wayland and Weston
  - Core compositing functionality
  - Handle kernel-space interaction
  - Efficient IPC
  - Open-source – FTW!
- Weston IVI plugin
  - Addresses IVI related UI feature gaps in Weston
- Simple and focused



Weston IVI Plugin Interface for GENIVI®-style Layer Management

Identifying Feature Gaps

GENIVI® Layer Management Client Interface Feature	Wayland Protocol
Surface Lifecycle Management	Yes
Surface Positioning	Yes, but through input device, not client supplied coordinate
Surface Visibility	No
Surface Opacity	No
Layer Lifecycle Management	No
Layer Positioning	No
Layer Visibility	No
Render order of surfaces in layer	No

Writing the Weston IVI Plugin Interface

- Group operations according to layer management object type
  - Surface
  - Layer
  - Display
- Weston plugin interface format matches Wayland protocol format
- Map Wayland protocol “interface” to each object type
  - One Wayland request per feature (operation)
    - Set position, opacity, visibility, etc

Wayland Protocol Interface XML Format

```
<protocol name="ivi_shell">
    <interface name="lm_surface" version="1">
        <description summary=application graphical content">
            Applications render content into a surface.
        </description>

        <request name="set_visibility">
            <arg name="visibility" type="uint" summary="Visibility…”/>
        </request>

        <event name="visibility">
            <arg name="visibility" type="uint" summary="Visibility…”/>
        </event>
    </interface>
    …
</protocol>
```


Subset of Wayland Protocol Elements

Protocol XML Element	Description
protocol	Top-level element that affects how files and `#include` guards generated by `wayland-scanner` are named
interface	Element that encapsulates a set of operations that may be performed on a corresponding instance
request	Server side operation called on object by client that is completed asynchronously
event	Client side operation asynchronously called by server side (e.g. Weston) when a change occurs

Wayland Protocol Argument Types

Wayland Argument Type	Native Type	Description
int	`int32_t`	32 bit signed integer
uint	`uint32_t`	32 bit unsigned integer
fixed	`wl_fixed_t`	double-precision floating point type, convert to native double using `wl_fixed_{to,from}_double()`
string	`char const *`	nil-terminated C string
array	`struct wl_array *`	Structure containing pointer to array and its length
fd	`int`	open file descriptor
new_id	`struct wl_object *` or `struct wl_resource *`	Object reference passed to factory-like requests
object	`struct wl_object *` or `struct wl_resource *`	Object reference (e.g. instance of `wl_surface`)

Caveats

- No concept of synchronous return values – e.g. “get” operations - in Wayland protocol
  - Clients should cache old value until `event` arrives asynchronously with new value

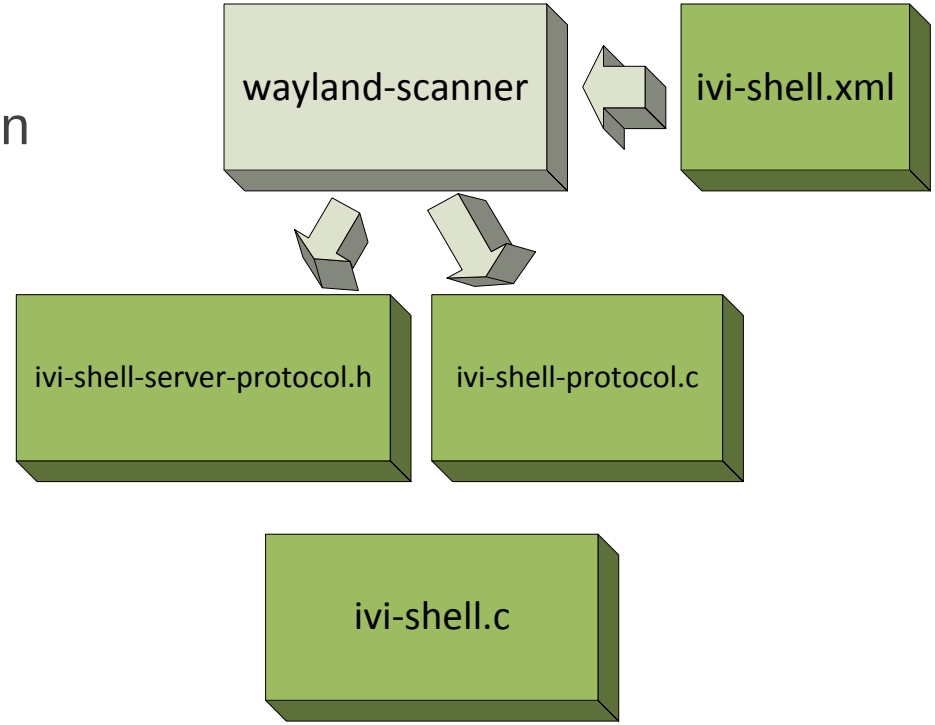
Weston IVI Plugin Implementation

Weston Plugin Requirements

- Plugin entry point
  - Initialization function called by Weston compositor upon successful load of plugin
  - `int module_init(struct weston_compositor *);`
    - Previously `shell_init()`
- Plugin must currently be built inside Weston source tree
  - Weston does not yet export plugin interface or library

Weston Plugin Components

- ivi-shell.xml
  - Wayland protocol based plugin interface
- ivi-shell.c
  - Plugin implementation
- Generated files
  - ivi-shell-server-protocol.h
  - ivi-shell-protocol.c



Weston Plugin Implementation

- Request implementation prototypes found in generated `ivi-shell-server-protocol.h` header
- ```
  static void lm_surface_set_opacity(struct wl_client * client,
                                     struct wl_resource * resource,
                                     struct wl_resource * opacity_resource);

  static const struct lm_surface_interface lm_surface_implementation = {
     lm_surface_set_opacity
  };
  ```
- Register implementation
  - ```
    struct wl_resource * resource =
      wl_client_add_object(client, &lm_surface_interface,
                           &lm_surface_implementation, id, surface);
    ```


Using the Weston IVI Plugin

Configure Weston to Use IVI Plugin

- `weston.ini` search path
  - `$XDG_CONFIG_HOME/.config/`
  - `$HOME/.config/`
- Automake-style plugin install directory
  - `$(libdir)/weston/`
- Set name of plugin in `weston.ini`

```
[shell]
type=ivi-shell.so
…
```

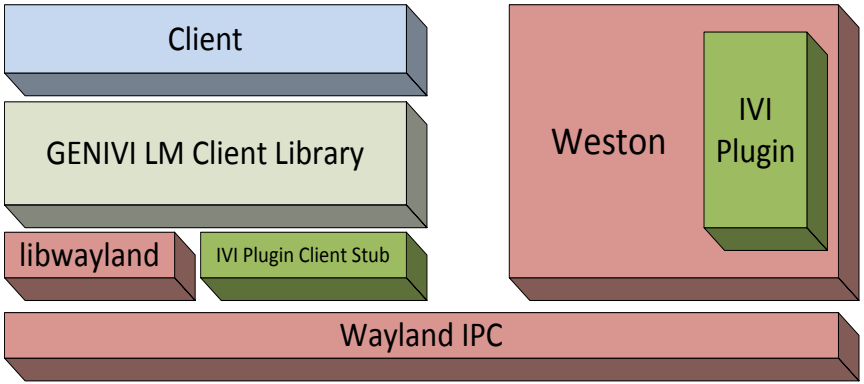

Invocation of IVI Plugin Operations

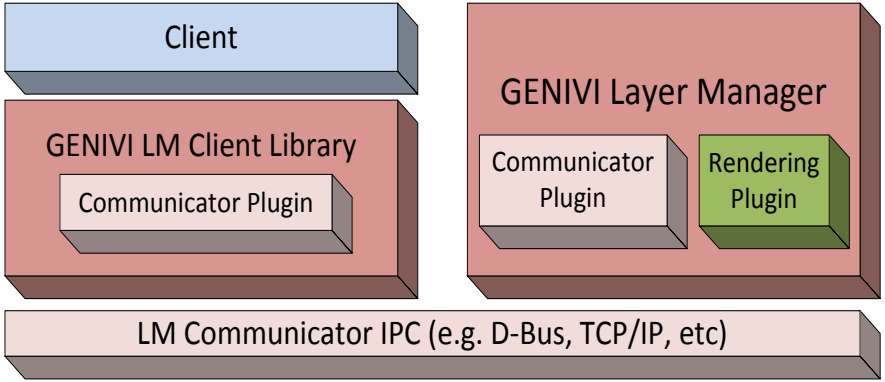
- Client side build
  - Add `ivi-shell-protocol.c` to client source build
- Client side code
  - …

```
#include “ivi-shell-client-protocol.h”
…
void foo()
{
   lm_surface_set_visibility(…);
}
```


Weston IVI Plugin vs. GENIVI Layer Manager

Architectures





Architectural Comparison

- Weston IVI Plugin
  - Weston-specific
  - Small codebase
  - Efficient IPC
  - Simple architecture
  - Network transparency support in Wayland/Weston is still pending
- GENIVI Layer Manager
  - Highly extensible
    - Rendering plugins
      - Wayland
      - X
      - Direct FB
    - Communicator plugins
      - D-Bus
      - TCP/IP

Summary

- Existing desktop style window management is insufficient for IVI in some cases
  - Applications are not necessarily run in windows
- Weston’s extensibility is key to creating IVI UI with minimal effort
  - No need for IVI UI developers to create compositor from scratch
  - Allows IVI UI developers to focus on their own IVI-specific needs

References

- Wayland
  - http://wayland.freedesktop.org/
  - http://cgit.freedesktop.org/wayland

- GENIVI®
  - http://www.genivi.org/
  - Media & Graphics Expert Group

Legal


TURE INTEL LINUX WIRELESS
OP
CS YOCTO
CONNMAN XEN
SYNCEVOLUTION
GUPNP
KVM POKY
OFONO
LINUX KERNEL
SIMPLE FIRMWARE INTERFACE (SFI) ENTERPRISE SECURITY INFRASTRUCTURE
INTEL OPEN SOURCE
TECHNOLOGY CENTER