QOM Vadis?
Taking Objects To The CPU And Beyond

Andreas Färber, B.Sc.
Expert Virtualization
SUSE LINUX Products GmbH
About Me
Introducing The Presenter

• QEMU contributor and maintainer for Cocoa, PReP, stable-0.15, QOM-CPU

• Working on KVM at SUSE
  - Level 3 support & feature development for SLE customers
  - Maintenance of SLE and openSUSE qemu package

• Studying Software Eng. for Embedded Systems
  - Aiming for a thesis on virtual development environments
Quo Vadis?
Outline Of This Presentation

• Introduction to QOM
• Remodeling the CPU with QOM
• Next steps
Introduction To QOM
QEMU
Conceptual Overview Of Device Emulation

• CPU-centric emulation!
• How: instructions (TCG only), PIO, MMIO
• Relevant: software-visible (black-box) behavior
• Irrelevant: simulation of inner workings / firmware
• QOM (formerly qdev) used for device encapsulation, reuse and parameterization
QEMU Object Model

QOM Terminology

- **Type**
  - Defines class
- **Class**
  - Stores static data & virtual method pointers
  - Lazily initialized from type and, optionally, static data
- **Object**
  - Stores dynamic data ("chunk of allocated memory")
- **Property**
  - Accessor to dynamic object data
  -Inspectable via monitor interface
A Simple Object Example
Declaration

static const TypeInfo example_type_info = {
    .name = "example",
    .parent = TYPE_OBJECT,
};

static void example_register_types(void) {
    type_register_static(&example_type_info);
}

type_init(example_register_types)
A Simple Object Example

Usage

• Add entry to Makefile.objs

• Instantiation
  - Object *obj = object_new("example");

• Finalization:
  - object_delete(obj);

• Note: More fine-grained control over lifetime and memory is available.
QOM Hooks
When What is Run

• `type_init()`: early during startup
• `TypeInfo::class_init`: when class is first created
  - `object_new()` / `object_initialize()`
  - `object_class_foreach()`
• `TypeInfo::instance_init`: for each instance created
  - `object_new()` / `object_initialize()`
• `TypeInfo::instance_finalize`: cleanup per instance
  - `object_delete()` / `object_finalize()`
qdev
Device Modeling Before And After QOM

• Forest of busses: PCI, ISA, … fallback: “SysBus”
• Two-stage construction via properties
• Hasn't changed … much
  - device_init() → type_init()
  - TypeInfo, .class_init
  - new type handling macros
• Possible to write a dummy device in < 10 minutes!
• Unit tests via qtest framework (some constraints!)
A Simple SysBusDevice Example

Declaration

#define TYPE_EXAMPLE = "example"

typedef struct ExampleState {
    SysBusDevice parent;
    MemoryRegion iomem;
} ExampleState;

static const TypeInfo example_type_info = {
    .name = TYPE_EXAMPLE,
    .parent = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(ExampleState),
    .class_init = example_class_init,
};

static void example_register_types(void) {
    type_register_static(&example_type_info);
}

type_init(example_register_types)
A Simple SysBusDevice Example
Implementation

```c
#define EXAMPLE_STATE(obj) \
  OBJECT_CHECK(ExampleState, (obj), TYPE_EXAMPLE)

static void example_device_init(SysBusDevice *dev) {
  ExampleState *s = EXAMPLE_STATE(dev);
  memory_region_init_io(&s->iomem, ...);
  sysbus_init_mmio(dev, &s->iomem);
}

static void example_class_init(ObjectClass *oc, void *data) {
  SysBusDeviceClass *sdc = SYS_BUSDEVICE_CLASS(oc);
  sdc->init = example_class_initfn,
}
```
A Simple SysBusDevice Example

Usage

• Add entry to Makefile.objs

• Instantiation
  - DeviceState *dev = qdev_create("example");

• Realization:
  - qdev_init_nofail(dev);

• Note: Today these are mostly wrapping QOM functions and can be inlined for QOM migration.
QOM Conventions
Which Examples To Follow (1/3)

- **DO** use `TYPE_FOO` constants defined in a header
- **DO** use verbose macro names
- **DO** use names-separated-by-dashes
- **DON'T** duplicate literal string type names

- `#define TYPE_EXAMPLE "example"
- `.name = TYPE_EXAMPLE,
- `object_new(TYPE_EXAMPLE)
- `qdev_create(TYPE_EXAMPLE)
QOM Conventions
Which Examples To Follow (2/3)

• DO place parent field first
• DON'T use “busdev” or similar qdev conventions

• typedef struct MyState {
  Object parent; /* or PCIDevice parent etc. */
  uint32_t some_register_value;
} MyState;
QOM Conventions
Which Examples To Follow (3/3)

• DO use cast macros (based on struct layout)
• DON'T rely on DO_UPCAST() (field names)
• DO use per-type variable declarations
• Avoid using cast macros other than OBJECT() inline

• void do_something_with(MyDeviceState *s) {
  PCIDevice *pci = PCI DEVICE(s);
  pci->field = foo;
  /* not s->pci.field or PCI(s)->field */
QOM ABI
Stability Rules

• Properties are externally visible (like command line)!
• A property MAY be
  - dropped
  - renamed
• But: A property MAY NOT change its type.
Remodeling The CPU With QOM
vCPU Use Cases
Data Center Meets System-on-a-Chip

• Homogeneous environment (largely)
• Standardized machine
• Long-running guests
  - Live migration
  - Hot-plug of resources
• Users: Sysadmins

• Highly fragmented hardware/software landscape
• “Weird” hardware is out there in the wild!
  - Heterogeneous cores
  - Kernel bring-up, drivers…
• Users: Developers
CPU State
Data Layout Before QOM

CPUX86State a.k.a. CPUState

CPU_COMMON

cpu-defs.h

target-i386/cpu.h

Common fields at varying offset make it impossible to use different CPUxxxStates at once!
CPU State
New Data Layout With QOM

CPUSState

include/qemu/cpu.h

X86CPU

CPUX86State (CPUArchState)

CPU_COMMON

cpu-defs.h
target-i386/cpu.h
target-i386/cpu-qom.h

Ongoing migration
CPU State
Understanding The Migration

- CPUState can now be used in every file!
- CPUArchState is still dependent on cpu.h
- CPUArchState → CPUState: ENV_GET_CPU()
- CPUState → CPUArchState requires knowledge of CPU type (no symmetry guarantees across targets)
- First set of fields was moved to CPUState for v1.3
CPU State

API Guidelines

• New target-independent code should use CPUState
  - Series adapting kvm_arch_*( ) under way
• Target-specific code should use FooCPU
  - Provides easy access to both CPUFooState and new fields
• CPUArchState is usually reset by zeroing the front part
  - Pointers or persistent data needs to be in CPUFooState behind CPU_COMMON,
  - or in FooCPU – before CPUArchState if accessed by TCG
CPU subclasses
How We Create vCPUs

• Subclasses prepared per -cpu name (~90% there)
• Currently flat hierarchy of, e.g., TYPE_OBJECT → TYPE_CPU → TYPE_ARM_CPU → “cortex-a9”
• More advanced hierarchies of CPU families possible (requested for sparc)
• Goal: Get rid of cpu_init() in favor of QOM/qdev
Next Steps
Reference Counting
Solving Object Finalization Issues

• Device is unplugged – but object still referenced
  - Current solution: object_unparent()
• Drop object_delete() in favor of object_unref()?  
  - Less predictable whether or when memory is freed
Realization
Two-stage Initialization For All Objects

• Idea: Generalize DeviceClass::init()
  - void realizefn(Object *obj, Error **err)
  - void unrealizefn(Object *obj) ?

• Provide “realized” property to inspect / set
Monitor Improvements
Extending QMP Plumbing For Handling Objects

• Today: qom-list, qom-get, qom-set
• qom-create?
  - Currently device_add requires DeviceState
Static Properties For QOM?
Generalizing qdev Properties

- Facilitate read-only-when-realized
- Facilitate getters / setters for simple value types?
- How to handle global default values?
- Suggestions and patches welcome!
Preparing For CPU Hotplug
Standardizing CPU Creation and Initialization

• Make the CPU a device
  - Add qdev support to linux-user / bsd-user for v1.3

• Properties for x86 CPU manipulation / inspection
  - Versioning of CPU models via global qdev properties?
QOM'ifying SoCs
Using QOM For Grouping Of Devices

• Early prototype: SuperH 7750
• Drafts: Tegra2
• Common pattern: one SoC on multiple boards
  - Helper function for CPU and device creation
  - Idea: Group in inspectable way using QOM Container

• Open issues:
  - How to deal with -cpu?
  - Static properties for parameterization?
Spreading QOM
Standardizing Object Creation and Initialization Elsewhere

• blockdev?
• chardev?
• netdev?
Submit your models upstream to not get left behind!

www.qemu.org

Thank you.