Phonons in Solids

Internals and Usage Explained

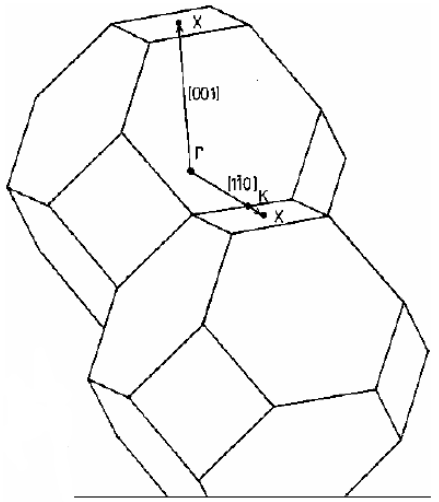
Matthias Kretz & Kevin Ottens

September 23rd 2006

Phonons in Solids

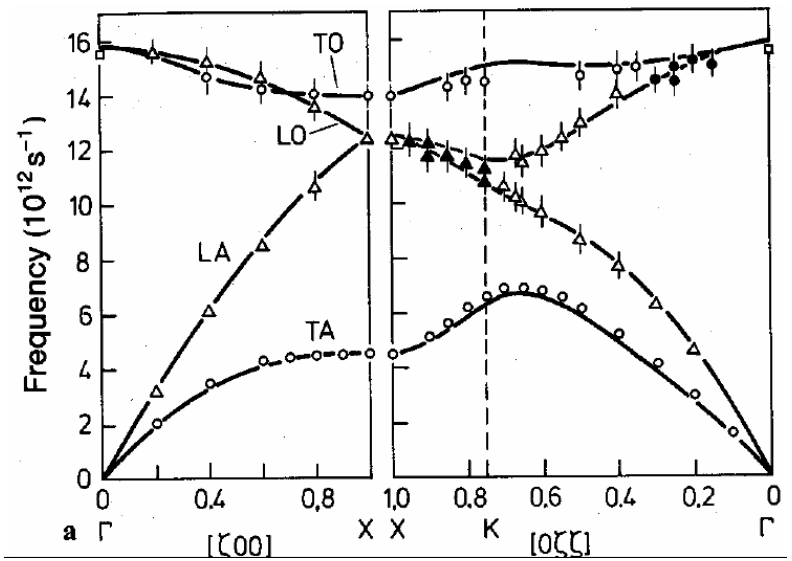
- Dispersion Relation:

$$\omega_k = \sqrt{2\omega^2(1 - \cos(ka))} \qquad (1)$$



- for small k:

$$\omega_k = ck \qquad (2)$$



Outline

1 Common Architecture Pattern

2 Solid

3 Phonon

Outline

1 Common Architecture Pattern

2 Solid

3 Phonon

Outline

1 Common Architecture Pattern
- Problem to Solve
- Architecture Principles
- Interface Based Approach
- Introspection Based Approach
- Q_INTERFACES

Requirements (1/2)

Cross-project collaboration

- Maximize reusability (freedesktop.org anyone?)
- Multimedia is not our business

Release cycles and Binary compatibility

- Other project teams work for fun too
- Don't want to force our own cycles and rules

Requirements (2/2)

Flexibility

- Provide choice to users and distributors
- Switch subsystems on the fly

Portability

- New porting concerns...
- ... allow it

Frontend/Backend Split

- Loose coupling with other projects
- Portability
- Flexibility
- Binary Compatibility ... with extra care!

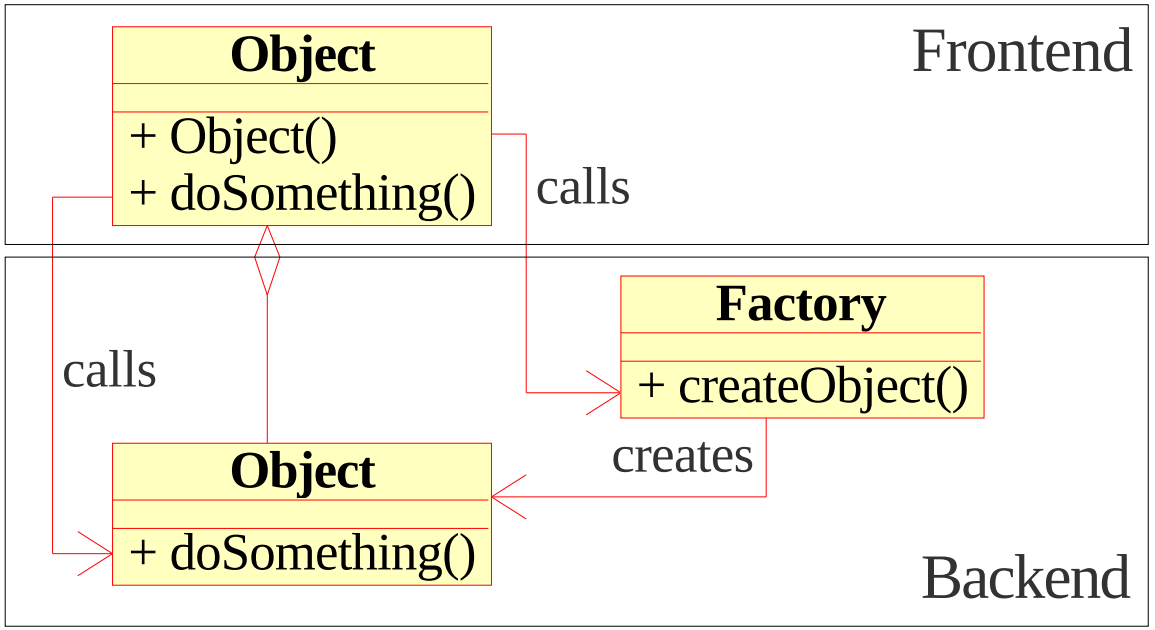
Frontend/Backend Split

- Loose coupling with other projects
- Portability
- Flexibility
- Binary Compatibility ... with extra care!

Outline

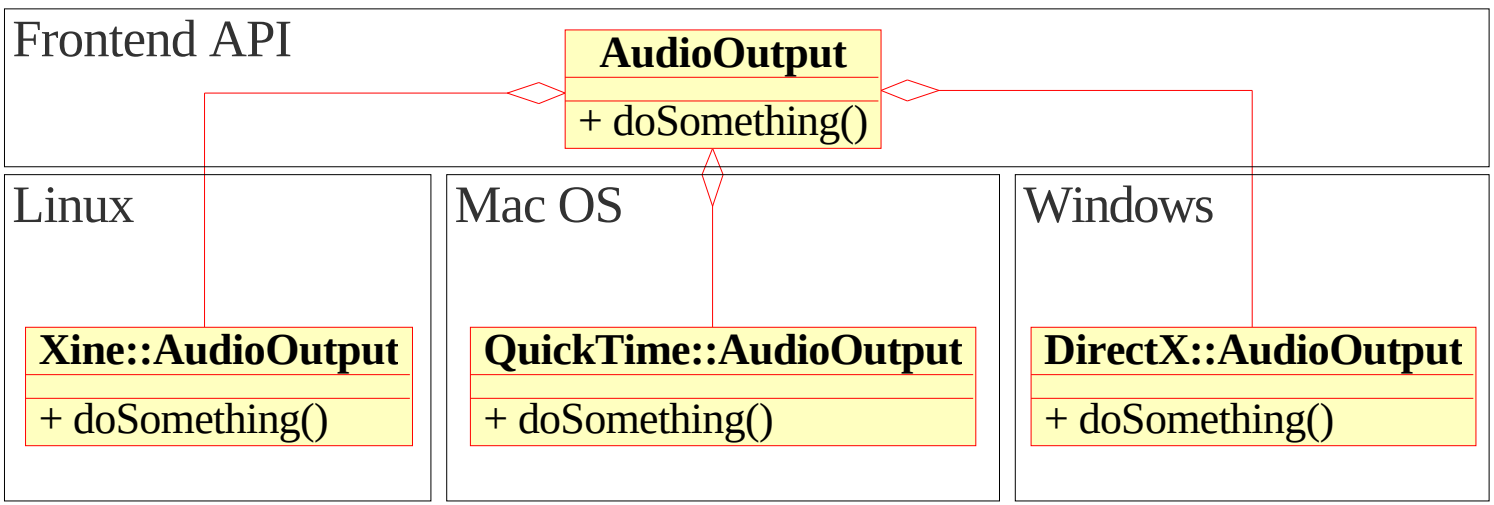
1 Common Architecture Pattern
- Problem to Solve
- Architecture Principles
- Interface Based Approach
- Introspection Based Approach
- Q_INTERFACES

General Pattern



- BC Frontend Classes
- Frontend objects hold a reference to backend objects
- Internally use a factory for backend objects creation

Portability



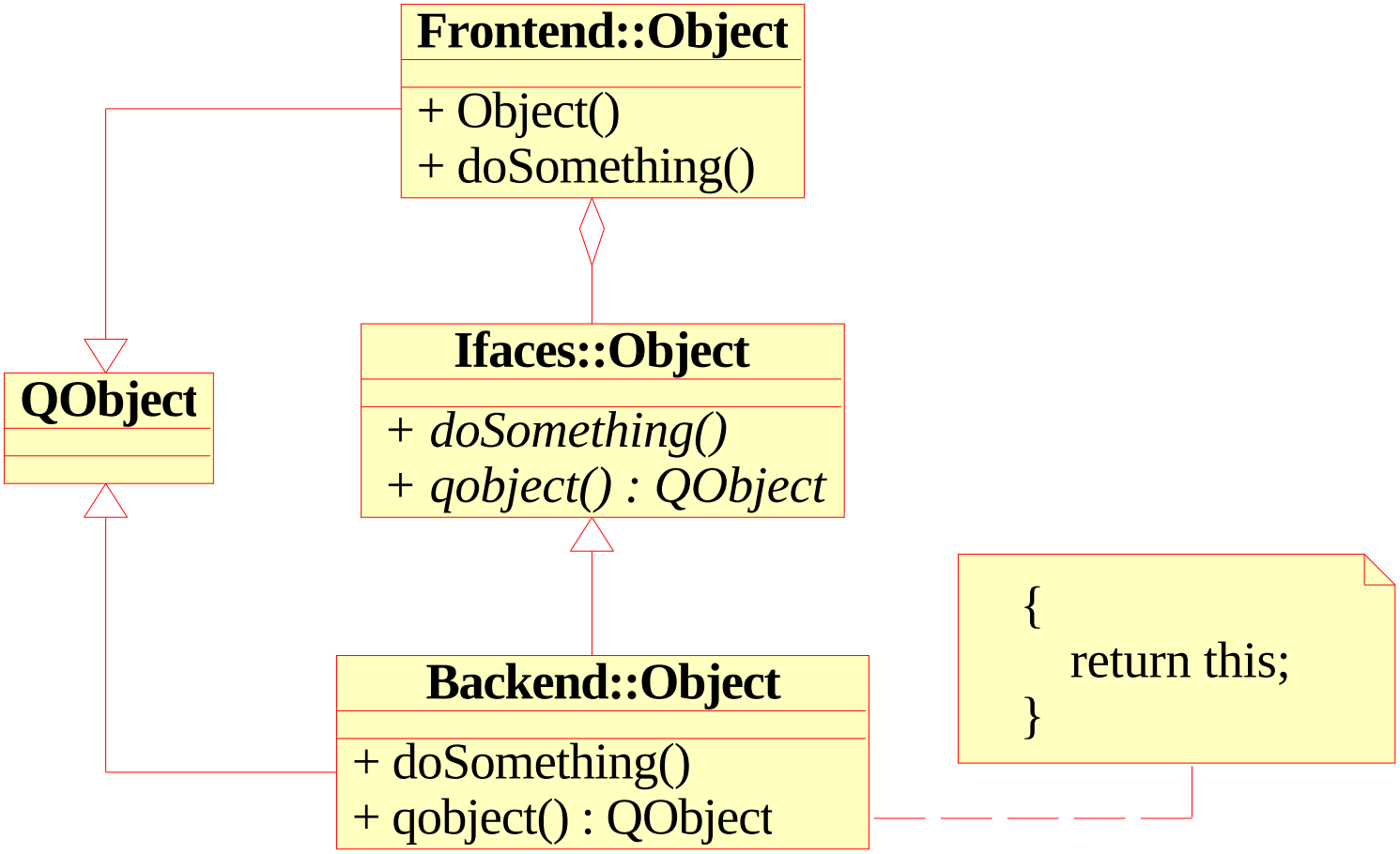
Backend Switching

1. Frontend factory is signalled to switch the backend
2. Frontend factory tells all frontend objects to save state and delete their backend object
3. If all backend objects are deleted
   - Frontend factory unloads the old backend
   - Frontend factory loads the new backend
4. Frontend objects are told to restore their state (recreating the backend objects)

Outline

1 Common Architecture Pattern
- Problem to Solve
- Architecture Principles
- Interface Based Approach
- Introspection Based Approach
- Q_INTERFACES

Interface, Overview



Interface, Pros & Cons

Pros

- Enforce compile time checking
- Easy to document backend writing
- Fast method calls (almost no overhead)

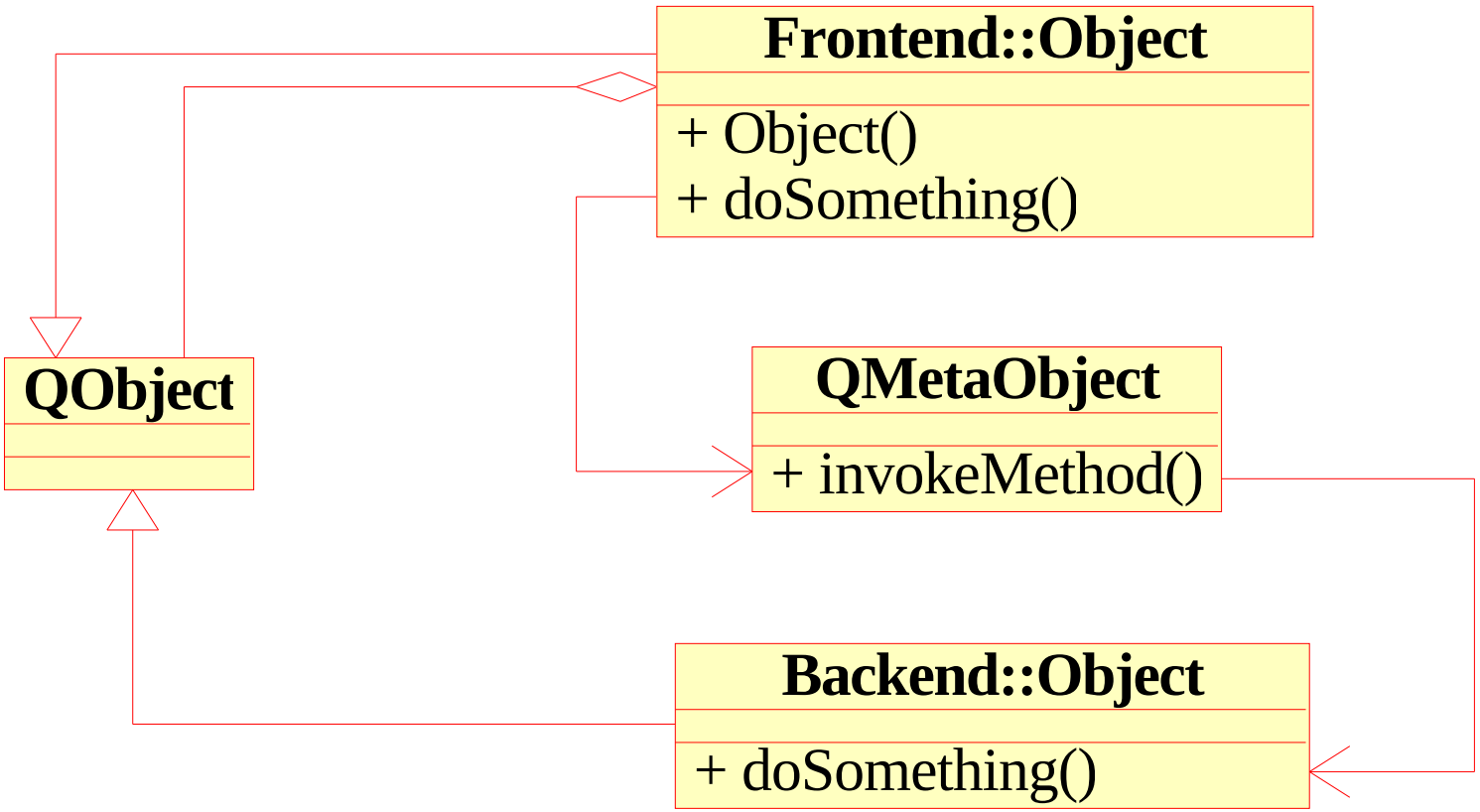
Cons

- BC concerns on the backends side
- Multiple inheritance
- Several pointers to the backend class
- You can't have a QWidget in the inheritance tree

Outline

1 Common Architecture Pattern
- Problem to Solve
- Architecture Principles
- Interface Based Approach
- Introspection Based Approach
- Q_INTERFACES

Introspection, Overview



Introspection, Pros & Cons

Pros

- No BC concerns
- No need to maintain two sets of classes (frontend & interfaces)
- Free to partially implement a backend class

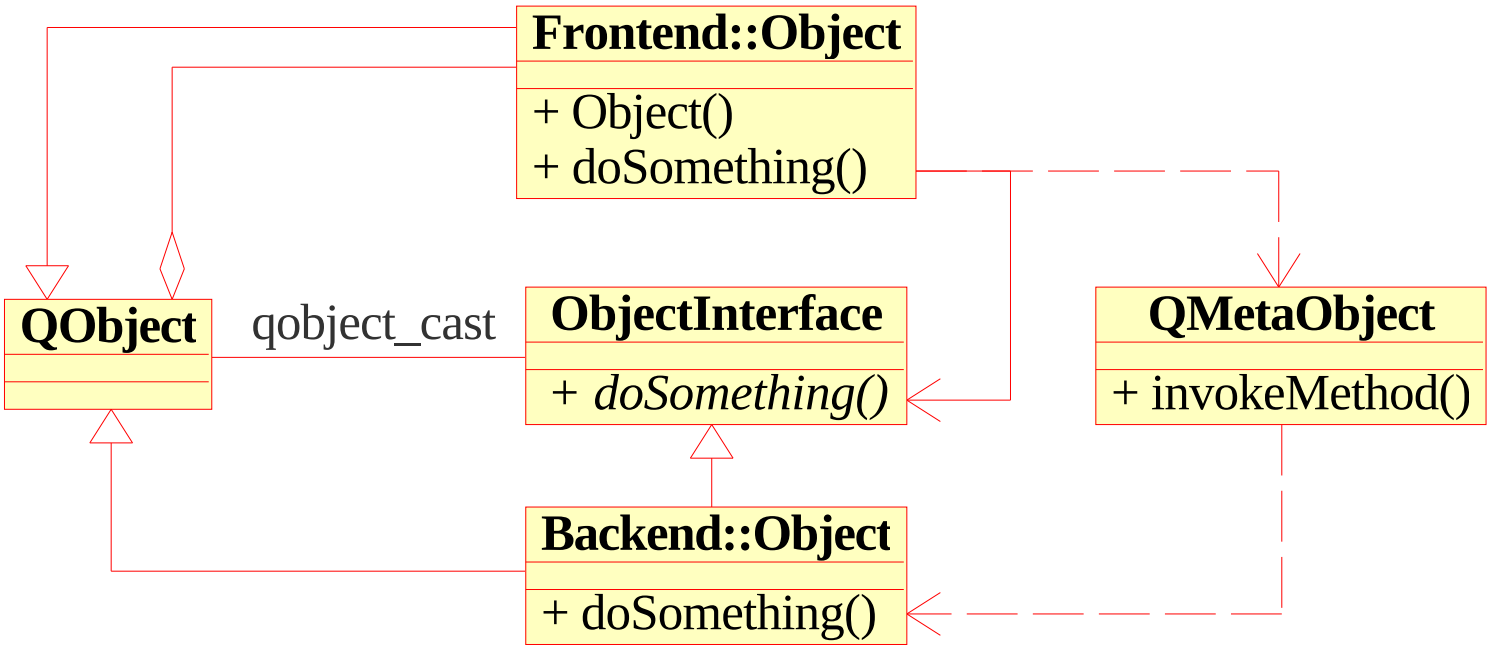
Cons

- No compile-time checking, needs specific validation tools
- Requires more work for explaining backend writing
- Slower method calls (`invokeMethod()` overhead)

Outline

1 Common Architecture Pattern
- Problem to Solve
- Architecture Principles
- Interface Based Approach
- Introspection Based Approach
- Q_INTERFACES

Qt to the Rescue



Qt to the Rescue

Declaring and Implementing Q_INTERFACES

```
class ObjectInterface {
  public:
    virtual void doSomething() = 0;
};
Q_DECLARE_INTERFACE(ObjectInterface,
    "org.kde.ObjectInterface/1.0")

class ObjectImpl : public QObject,
                   public ObjectInterface {
  Q_OBJECT
  Q_INTERFACES(ObjectInterface)
  public:
    void doSomething();
};
```

Qt to the rescue

Using Q_INTERFACES

The frontend class has a `QObject*` member `m_iface` pointing to the backend object:

```
ObjectInterface *foo =
    qobject_cast<ObjectInterface*>(m_iface);
foo->doSomething();
```

Normally you should check whether `foo` is `NULL`. For Phonon this check is done when certifying backends using (ui)methodtest.

Outline

1 Common Architecture Pattern

2 Solid

3 Phonon

Outline

2 Solid
- Motivations & Goals
- Application Development
- Backend development
- Current State

Fix the Current Situation

What's available right now?

- "Hardware discovery": mediamanager, medianotifier...
- Network management: knetworkmanager, kwifimanager...
- Power management: kpowersave, sebas' powermanager...

Why does it suck?

- Only cover a partial set of devices and features
- Information hardly accessible to other applications
- Tied to a particular system

Allow Future Improvements

New use cases

- "Device nomadism"
- Bluetooth is already here
- Hardware/Software collaboration

New platforms

- Windows
- Mac OS X
- *BSD (current support is far from perfect)

The Whole Picture

Several domains

- Hardware Discovery, Power & Network Management
- Each domain corresponds to a group of classes
- Each domain will have at least one policy agent

Policy agents, what? why?

- Already there: knetworkmanager, mediamanager...
- Responsible for:
  - Interacting with the user
  - Enforcing his settings
- The library is only about interacting with the system

Outline

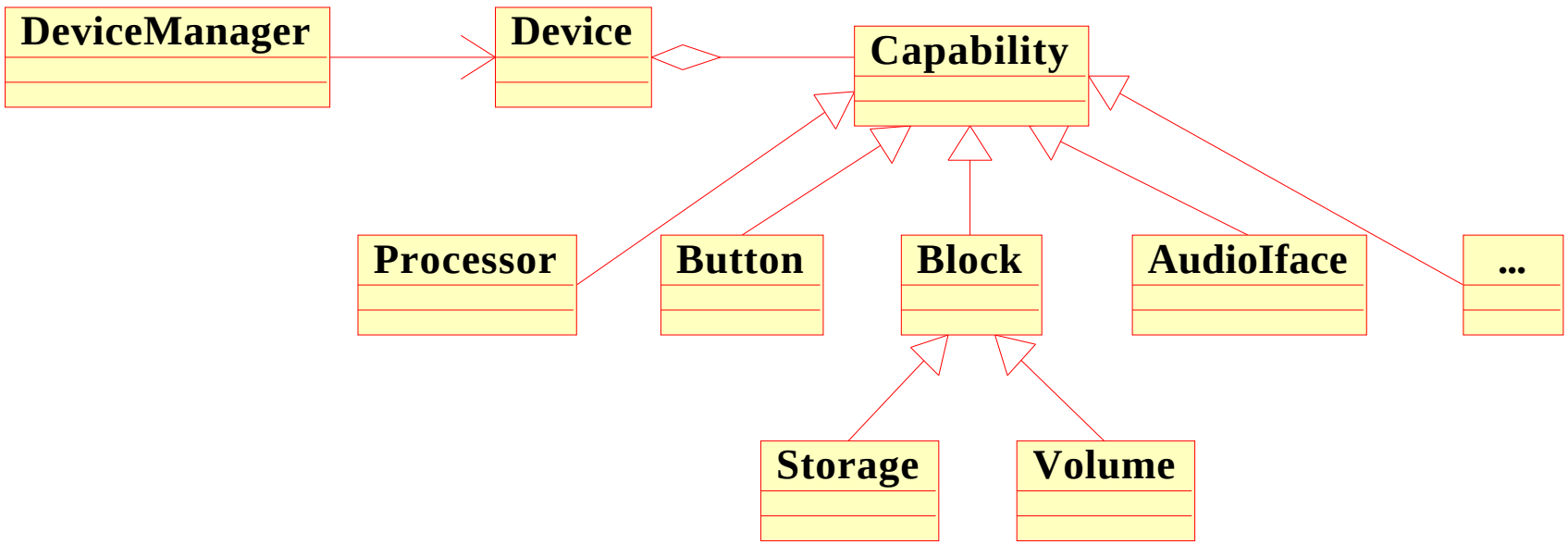
2 Solid
- Motivations & Goals
- Application Development
- Backend development
- Current State

Hardware Discovery

Point of view

- Report as many hardware facts as possible
- No device interaction, storage is the only exception
- Domain specific interaction should be done in domain specific code
  - Playing music $\longrightarrow$ Phonon
  - Printing $\longrightarrow$ kdeprint
  - ...
- Solid provides:
  - Information needed to access devices
  - Not the mechanisms to access devices

Hardware Discovery



Solid::DeviceManager (1/2)

List devices

```
DeviceList all =
    DeviceManager::self().allDevices();
```

Get device plug notifications

```
DeviceManager *dm = &DeviceManager::self();
connect( dm, SIGNAL(deviceAdded(QString)),
         obj, SLOT(slotDoSomething(QString)) );
```

Solid::DeviceManager (2/2)

Complex queries

```
const DeviceManager &dm = DeviceManager::self();
DeviceList matched = dm.findDeviceFromQuery(
        QString(),
        Capability::Unknown,
        "Processor.canThrottle␣==␣true"
        "␣OR␣"
        "Volume.isMounted␣==␣true"
);
```

Solid::Device (1/2)

Basics

```
Device *d;
// ...
QString vendor = d->vendor();
QString product = d->product();
```

Query capability

```
Device *d;
// ...
if (d->is<Volume>()) {
    // Do something
}
```

Solid::Device (2/2)

Using capabilities

```
Device *d;
// ...
if (d->is<Volume>()) {
    Volume *v = d->as<Volume>();

    QString mountPoint = v->mountPoint();
    connect( v, SIGNAL(mountStateChanged(bool)),
             obj, SLOT(slotDoSomething(bool)) );

    KJob *job = v->eject();
    job->start();
}
```

Power Management

Goals

- Make power management easy
- Provide a set of simple actions
- Allow state change notifications

Central class

- `PowerManager`
- Implemented as a singleton
- Exposes all the powermanagement features
  - Power Schemes
  - Power Sources
  - Suspend / Resume
  - CPU power

Solid::PowerManager (1/2)

Scheme management

- `PowerManager::self().supportedSchemes();`
- `PowerManager::self().setScheme("powersave");`

Power sources

- `PowerManager::self().batteryState();`
- `PowerManager::self().acAdapterState();`

Hibernating

```
KJob *job =
    PowerManager::self().suspend(PowerManager::ToDisk);
job->start();
```

Solid::PowerManager (2/2)

Managing CPU power

```
PowerManager::self().setCpuFreqPolicy(
        PowerManager::Powersave );
PowerManager::self().setCpuEnabled(1, false);
```

Useful signals

```
PowerManager *pm = &PowerManager::self();
connect( pm, SIGNAL(batteryStateChanged(int)),
         obj, SLOT(doSomething(int)) );
connect( pm, SIGNAL(acAdapterStateChanged(int)),
         obj, SLOT(doSomething(int)) );
connect( pm, SIGNAL(buttonPressed(int)),
         obj, SLOT(doSomething(int)) );
```

Network Management

Censored

Tomorrow, 17:15

Network Status support in KDE and how to use it

Will Stephenson

This guy rocks... I really mean it

Outline

2 Solid
- Motivations & Goals
- Application Development
- Backend development
- Current State

Hardware Discovery

First: `DeviceManager`

- Subsystem initialization
- Device listing, signals
- Complex queries can wait
- Will be factory for `Device`

Second: `Device`

- Parent / Child relationship
- Basic informations (product and vendor names)
- Will be factory for capabilities

Third: `Capability` and children

- Implement them in the order you want

Power Management

First: basics & scheme management

- Subsystem initialization
- Methods and signals for power scheme management

Second: suspending

- Listing supported suspend methods
- Implementing `KJob *suspend(SuspendMethod);`

Third: other features

- Power Sources
- CPU power management

Network Management

Censored

Tomorrow, 17:15

Network Status support in KDE and how to use it

Will Stephenson

Did I tell you that this guy rocks?

Outline

2 Solid

- Motivations & Goals
- Application Development
- Backend development
- Current State

Current State

Library

- Hardware discovery, Power & Network management
- A few features for system statistics (not covered here)
- Needs more applications using it

Policy agents

- Porting medianotifier, knetworkmanager and friends
- That's a target for aKademy Coding Marathon

Backends

- Everything required for Linux like systems is done
- To support your favorite platform, just write a backend!

Outline

1 Common Architecture Pattern

2 Solid

3 Phonon

Outline

3 Phonon
- Introduction
- Core Classes
- Phonon for Application Developers
- Phonon for Backend Developers
- State of Development

Architecture

you already know this. . .

- frontend/backend separation
- backend dynamically loaded
- exchanging the backend on the fly should be possible
- no BIC breakage if the Phonon API evolves in KDE 4.x times

Goals

- task-oriented design
- easy multimedia development
- no framework for video editor or pro-audio apps
- no “competition” for GStreamer/NMM like media frameworks

Motivation (1/2)

the user's perspective

- A user should be able to playback any media without configuration steps
- "power users" want great flexibility
- additional multimedia hardware should be available to all applications without any further steps
- users need to decide what device to use for what purpose/program

Motivation (2/2)

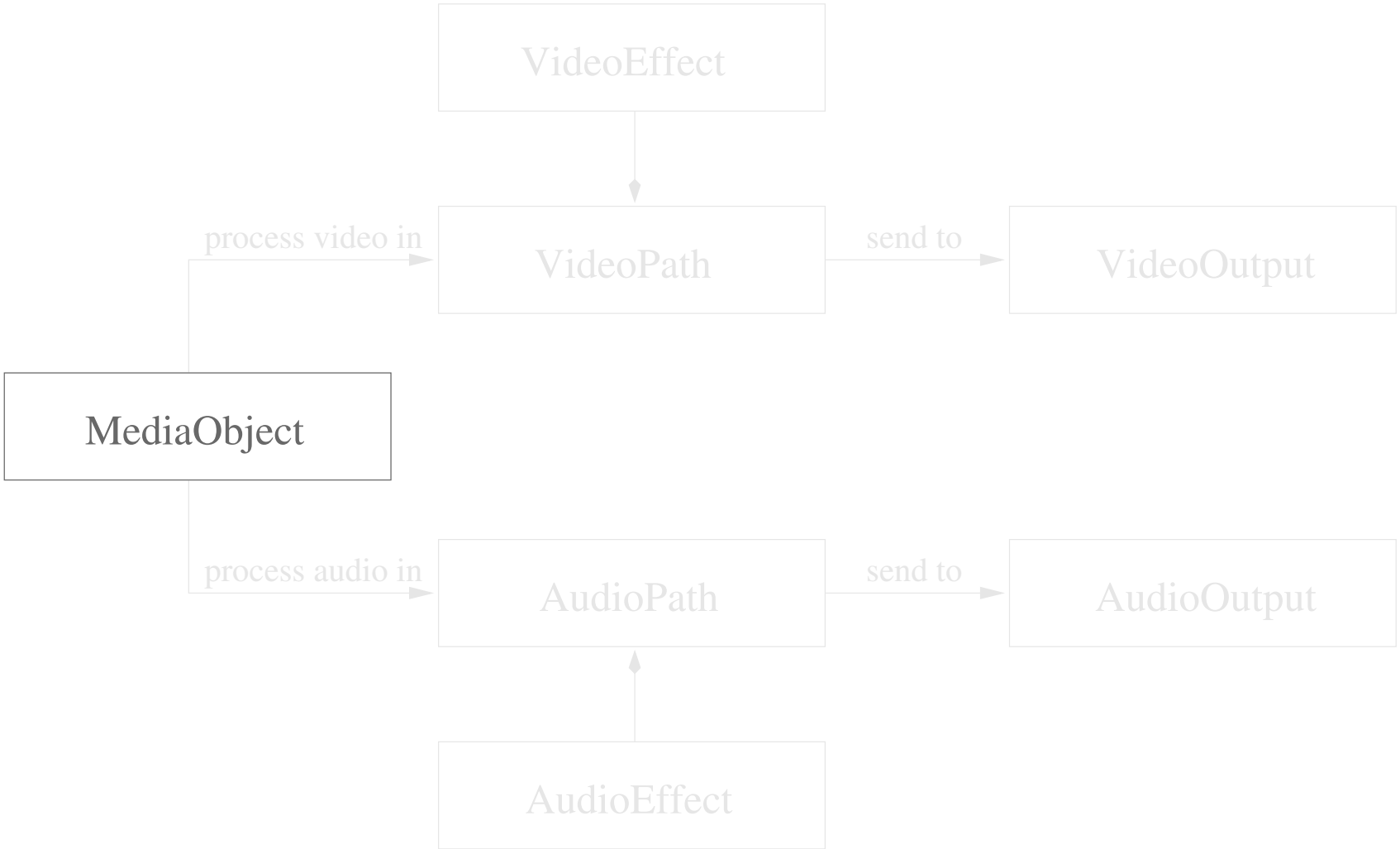
the developer's perspective

- Qt/KDE style API
- developers need APIs that are straightforward, easy to use and understand
- applications need a multimedia API that works on UNIX systems (including OS X) and Windows
- ABI/API changes should not hinder KDE from using the newest version of some media framework

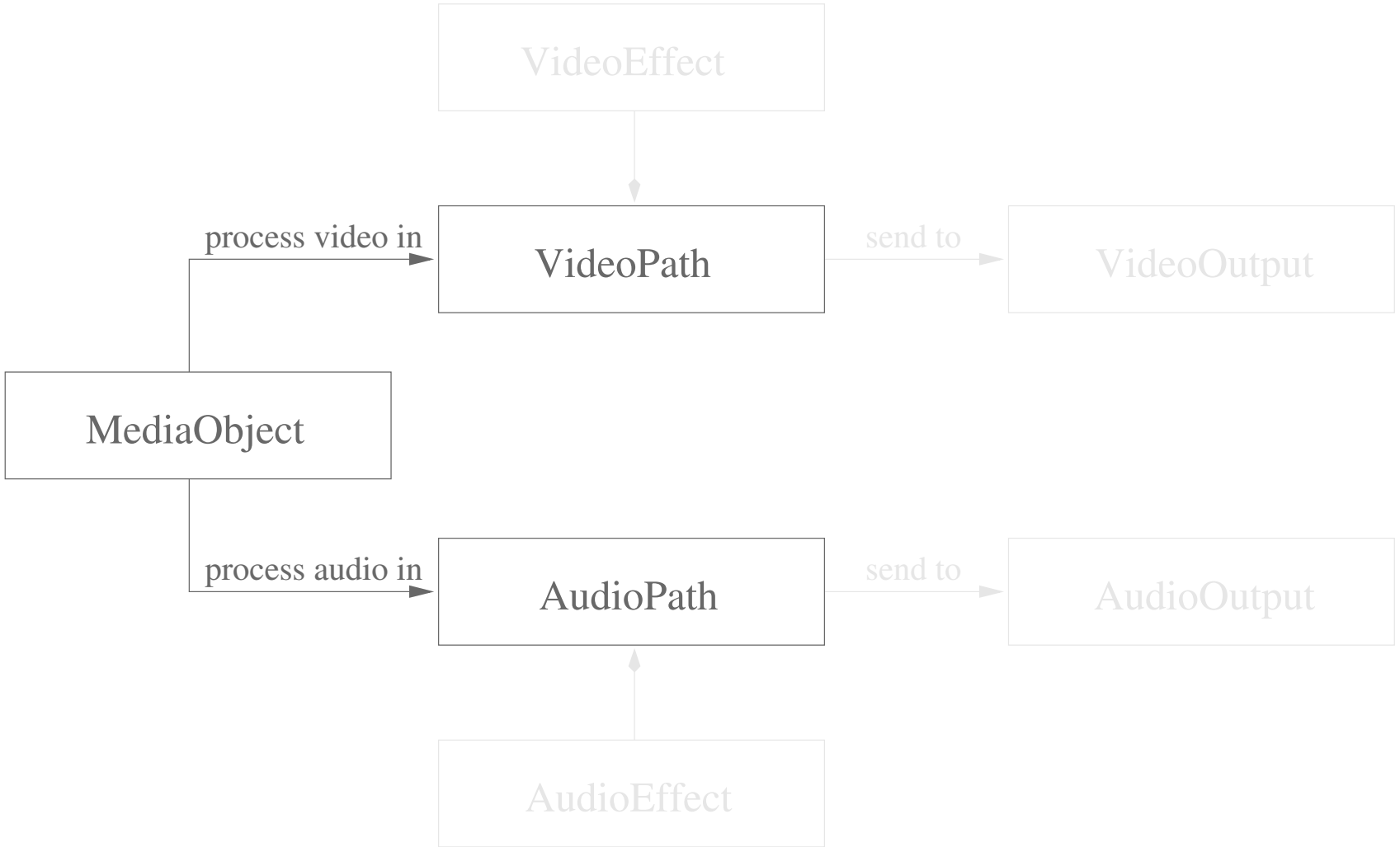
Outline

3 Phonon
- Introduction
- Core Classes
- Phonon for Application Developers
- Phonon for Backend Developers
- State of Development

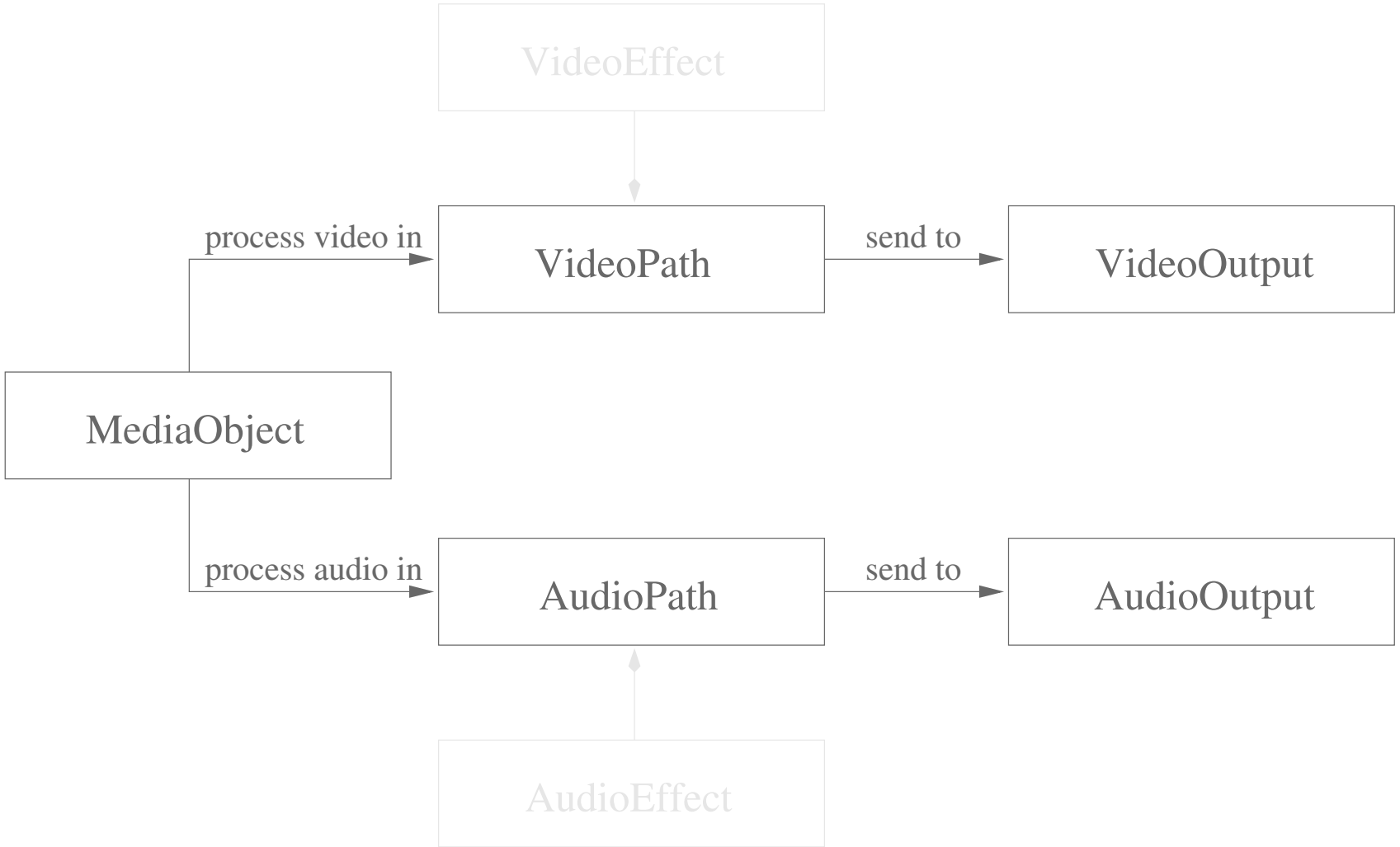
The Core Classes



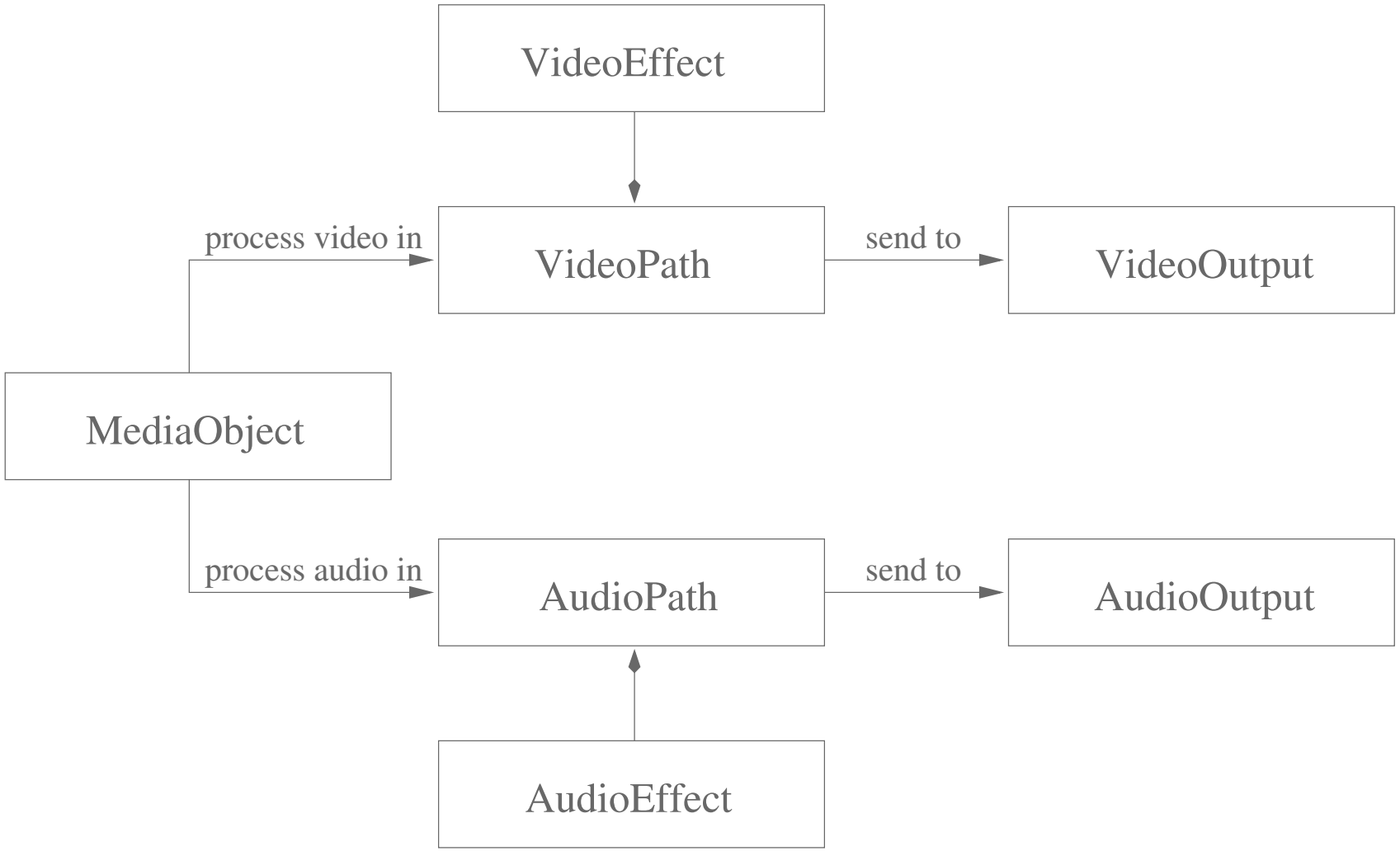
The Core Classes



The Core Classes



The Core Classes



Outline

3 Phonon

- Introduction
- Core Classes
- Phonon for Application Developers
- Phonon for Backend Developers
- State of Development

Phonon AudioPlayer

The class `AudioPlayer`

- provides enough features to use it for JuK (from KDE3 at least)
- inherits `QObject` for signals and slots
- start playing with
  - `play(KUrl);`
  - `load(KUrl); play();` for preloading
- `pause()`, `stop()`, `seek()`
- reports time info and signals when playback finished

AudioPlayer example

AudioPlayer

```
AudioPlayer *player = new
    AudioPlayer(Phonon::MusicCategory, this);
player->play(KUrl("file:///home/user/song.ogg"));
```

seek/pause/stop

```
player->seek(milliseconds);
player->pause(); player->stop();
```

volume

```
float volume = player->volume();
player->setVolume(0.5 * volume);
```

AudioPlayer example

AudioPlayer

```
AudioPlayer *player = new
    AudioPlayer(Phonon::MusicCategory, this);
player->play(KUrl("file:///home/user/song.ogg"));
```

seek/pause/stop

```
player->seek(milliseconds);
player->pause(); player->stop();
```

volume

```
float volume = player->volume();
player->setVolume(0.5 * volume);
```

AudioPlayer example

AudioPlayer

```
AudioPlayer *player = new
    AudioPlayer(Phonon::MusicCategory, this);
player->play(KUrl("file:///home/user/song.ogg"));
```

seek/pause/stop

```
player->seek(milliseconds);
player->pause(); player->stop();
```

volume

```
float volume = player->volume();
player->setVolume(0.5 * volume);
```

Phonon VideoPlayer

same as the AudioPlayer class except that it is a `QWidget`

example: play and forget [run]

```
int main() {
  ...
  VideoPlayer player(Phonon::VideoCategory);
  connect(&player, SIGNAL(finished()), &app,
      SLOT(quit()));
  player.show();
  player.play(KUrl("file:///home/user/video.ogm"));
  player.seek(player.totalTime() * 97 / 100);
  return app.exec();
  ...
}
```

Modern Mediaplayers Want More

- gapless playback
- (cross)fades
- equalizer
- video brightness and contrast controls
- audio visualizations
- ...

Phonon MediaQueue

define the output

```
output = new AudioOutput(Phonon::MusicCategory);
apath = new AudioPath;
apath->addOutput(output);
```

MediaQueue

```
media = new MediaQueue(this);
media->addAudioPath(apath);
media->setUrl("file:///home/user/song1.ogg");
media->setNextUrl("file:///home/user/song2.ogg");
media->play();
```

Phonon MediaQueue

define the output

```
output = new AudioOutput(Phonon::MusicCategory);
apath = new AudioPath;
apath->addOutput(output);
```

MediaQueue

```
media = new MediaQueue(this);
media->addAudioPath(apath);
media->setUrl("file:///home/user/song1.ogg");
media->setNextUrl("file:///home/user/song2.ogg");
media->play();
```

Phonon MediaQueue

the complete example [run]

```
output = new AudioOutput(Phonon::MusicCategory);
apath = new AudioPath;
apath->addOutput(output);
media = new MediaQueue(this);
media->addAudioPath(apath);
media->setUrl("file:///home/user/song1.ogg");
media->setNextUrl("file:///home/user/song2.ogg");
SeekSlider *slider = new SeekSlider;
slider->setMediaProducer(media);
media->play();
media->seek(media->totalTime * 97 / 100));
```

Outline

3 Phonon

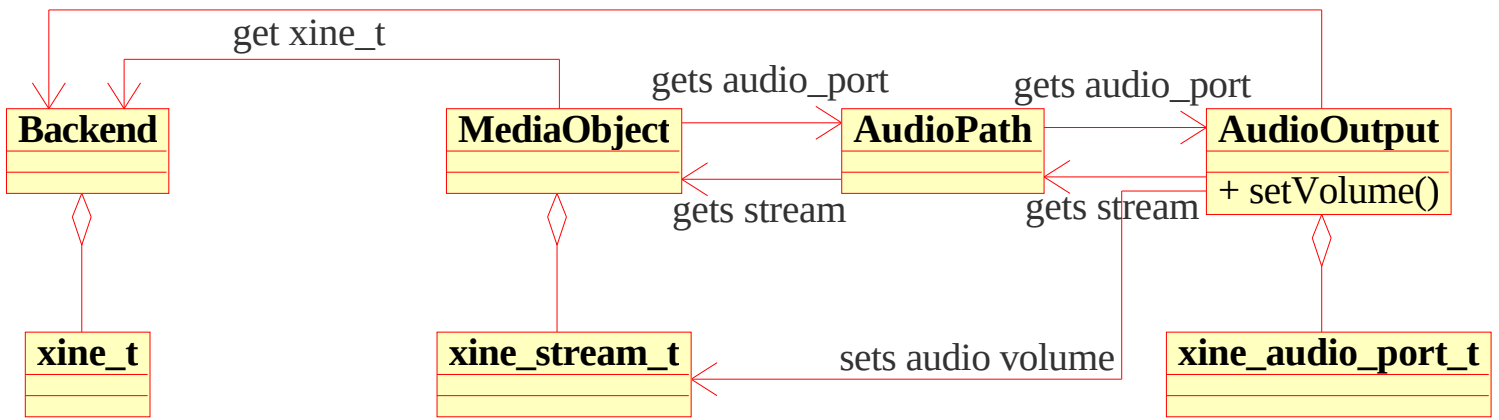
- Introduction
- Core Classes
- Phonon for Application Developers
- Phonon for Backend Developers
- State of Development

Overview

$3\frac{1}{2}$ backend implementations:

- NMM
- Xine
- avKode
- Fake

Basic Design



Optimizations

how to achieve instant reactions

example: Xine backend

- create `xine_stream_t` as early as possible
- `MediaObject::setUrl` → `xine_open`
- `AbstractMediaProducer::play()` → calls `xine_play`

Challenges

- missing functionality in a media framework: Xine only does playback
- Phonon is very flexible: needs a lot of thought for designing a backend

Backend ToDo List

existing backends can do (basic) playback, that's it

- implement `Xine::ByteStream` for KIO URLs
- Xine video breaks because it wants XThreads
- NMM needs a `VideoWidget` implementation
- `AvCapture`
- Effects (EQ, Fader, Compressor, Deinterlace, Contrast, Brightness, Saturation, ...)
- Phonon-GStreamer anyone?

Outline

3 Phonon
- Introduction
- Core Classes
- Phonon for Application Developers
- Phonon for Backend Developers
- State of Development

Open Tasks (1/2)

- KIO seeking in `MediaObject`
- (good) user interface for Phonon configuration
- Network API: VoIP
  - either: provide "lowlevel" audio/video I/O API only and let the application do the rest
  - or: provide API for SIP and audio transport over RTP

Open Tasks (1/2)

- KIO seeking in `MediaObject`
- (good) user interface for Phonon configuration
- Network API: VoIP
  - either: provide “lowlevel” audio/video I/O API only and let the application do the rest
  - or: provide API for SIP and audio transport over RTP

Open Tasks (2/2)

- look into standardizing the `AudioOutput` DBus interface to make it usable without Phonon
- finish API for recording
- DVD/TV support, chapters
- OSD (video overlays)
- *output device switching*
- *device listing → How to get the ALSA device list?*

Open Tasks (2/2)

- look into standardizing the `AudioOutput` DBus interface to make it usable without Phonon
- finish API for recording
- DVD/TV support, chapters
- OSD (video overlays)
- *output device switching*
- *device listing → How to get the ALSA device list?*

Devices and Solid

- Solid provides a list of available devices
- When a relevant device is (un)plugged Solid tells Phonon
  $\longrightarrow$ notifies all applications that should switch a device

Problem

- ALSA devices defined in asoundrc, not /sys
- virtual devices for mixing, resampling, ...
- parse .asoundrc?

Devices and Solid

- Solid provides a list of available devices
- When a relevant device is (un)plugged Solid tells Phonon
  $\longrightarrow$ notifies all applications that should switch a device

Problem

- ALSA devices defined in asoundrc, not /sys
- virtual devices for mixing, resampling, ...
- parse .asoundrc?

Summary

Hardware and Multimedia functionalities made easy for

- Application developers
- Users

- Phonon and Solid development is exciting
- Consider joining the projects: there's still lots to do

Resources

Phonon

`http://phonon.kde.org`

Solid

`http://solid.kde.org`

Questions ?

Matthias Kretz kretz@kde.org
Kevin Ottens ervin@kde.org