3 Challenges in Multimedia Programming

3.1 Frameworks & Media Integration

3.2 Time Synchronization

3.3 Interactive and Event-Driven Programs

Literature:

P. Ackermann: Developing Object-Oriented Multimedia Software
based on the MET++ Application Framework, dpunkt 1996

http://java.sun.com/products/java-media/jmf/

H. M. Eidenberger, R. Divotkey: Medienverarbeitung in Java, dpunkt 2004

Frameworks

- **Definition** (Taligent): “A ***framework*** is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions.”
- **Definition** (nach Pomberger/Blaschek): “A ***framework*** (Rahmenwerk, Anwendungsgerüst) is a collection of classes which provides an abstract design for a family of problems”
- Goals:
  - Reuse of code, architecture and design principles
  - Reuse of schematic behaviour for a group of classec
  - Homogeneity among different application systems for a problem family (e.g. similar usability concept)

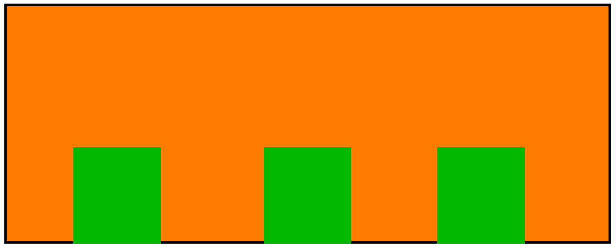
Classification of Frameworks

- Architecture driven framework:
  - Adaption by inheritance and method override
  - Complex class hierarchies and patterns
  - Adaption requires excellent programming skills and steep learning curve
  - Examples: Java Media Framework (JMF), MET++
- Data driven framework:
  - Adaption by object creation and setting of object properties
  - Delegation mechanisms (chaining of objects, events as objects)
  - Easier to learn but less flexible
  - Example: Pygame
- Compromise: Two-Level architecture:

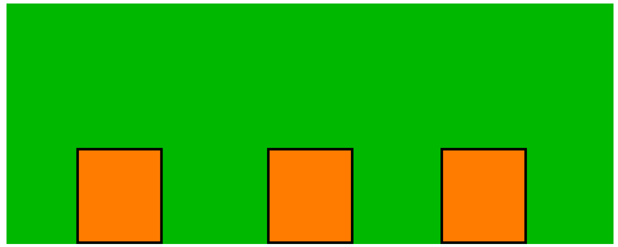
Data driven
Architecture driven

Class Library vs. Framework

Class library



Framework



Application specific parts

Prefabricated parts

"Don't call us, we call you"

("Hollywood Principle")

Adaptation by instantiation mainly

Control flow not pre-defined

Adaptation includes specialization

Predefined control flow

Base Part of Multimedia Framework: Stage

- Multimedia application as visual interface
  - Integration into interface/window framework
  - Root for time and space containment hierarchy
- Examples:
  - *Display* in Pygame
  - *Layout* in SMIL
  - *Canvas* in OpenLaszlo
  - *Stage* in JavaFX, Flash/AS
- Functions:
  - Define size of display area
  - Define general properties of display area (color space etc.)
  - Set window caption

Media Input/Output

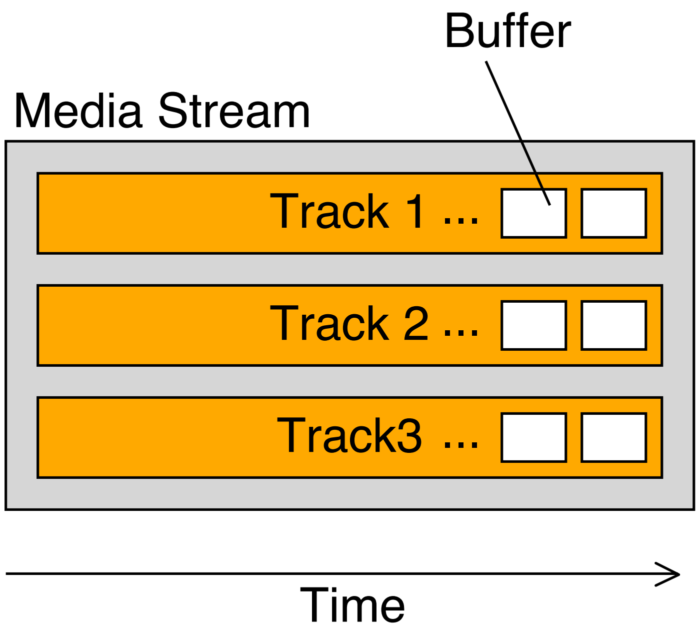
- Media data exist in external files
  - Various file formats
  - Sometimes rather complex (compressed file formats)
- Generic input/output
  - Provides functions to read and write various file formats
  - Provides homogeneous internal data type for image, sound etc.
  - Supports media file lifecycle:
    - » Check for existence, buffering, accessing
- Streaming support
  - Opening URL instead of local file
  - Dynamic buffering and loading
- Extensibility
  - Plugin architecture may enable easy extension with additional codecs

Classification of Media Sources

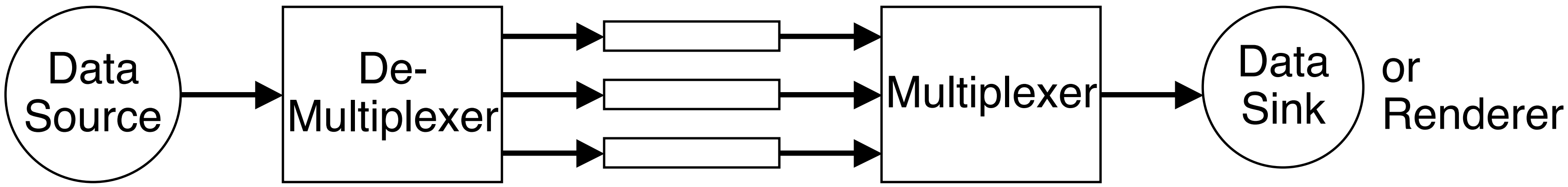
- Timing requirements:
  - Real time vs. Non-real time
    - » Real time: Defined frequency for arrival of media data
- Buffering:
  - Unbuffered vs. Buffered (buffer size)
  - Buffering safeguards against jitter, but introduces delay
- Control flow:
  - Push model: Source determines time of data transmission
  - Pull model: Consumer determines time of data transmission
- Distribution:
  - Source local or remote to consumer
  - File vs. network stream
- Processing chain configuration:
  - Source may be a *transformer* connected to another source

Media Packaging

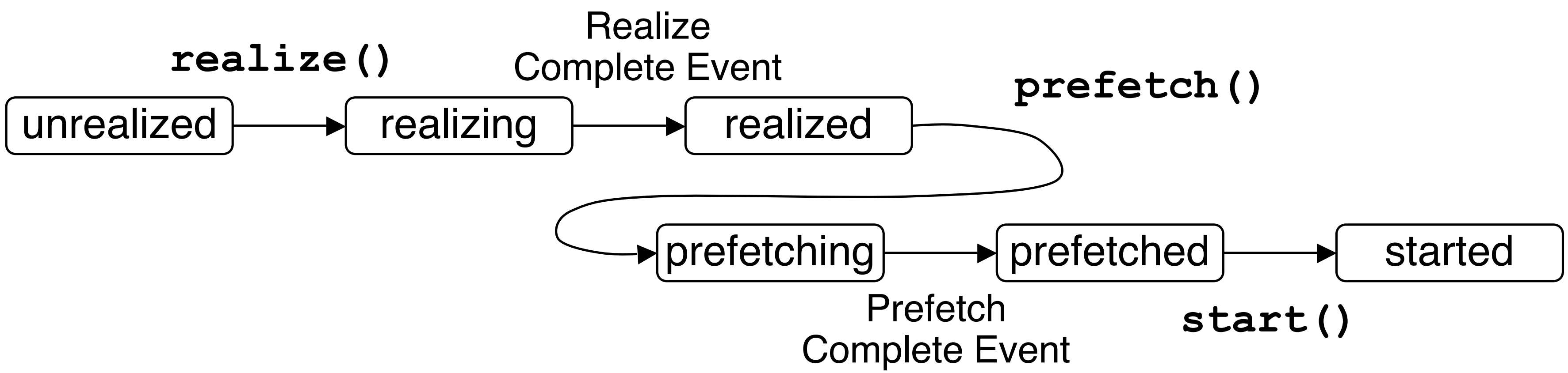
- Media source (file or stream)
  - May define more than one data stream
  - Possibly of different media types
- Example: QuickTime movie
  - Video track
  - Possibly separate sound tracks
  - Text (caption) track
  - Annotation track



Processing chain model of JMF:

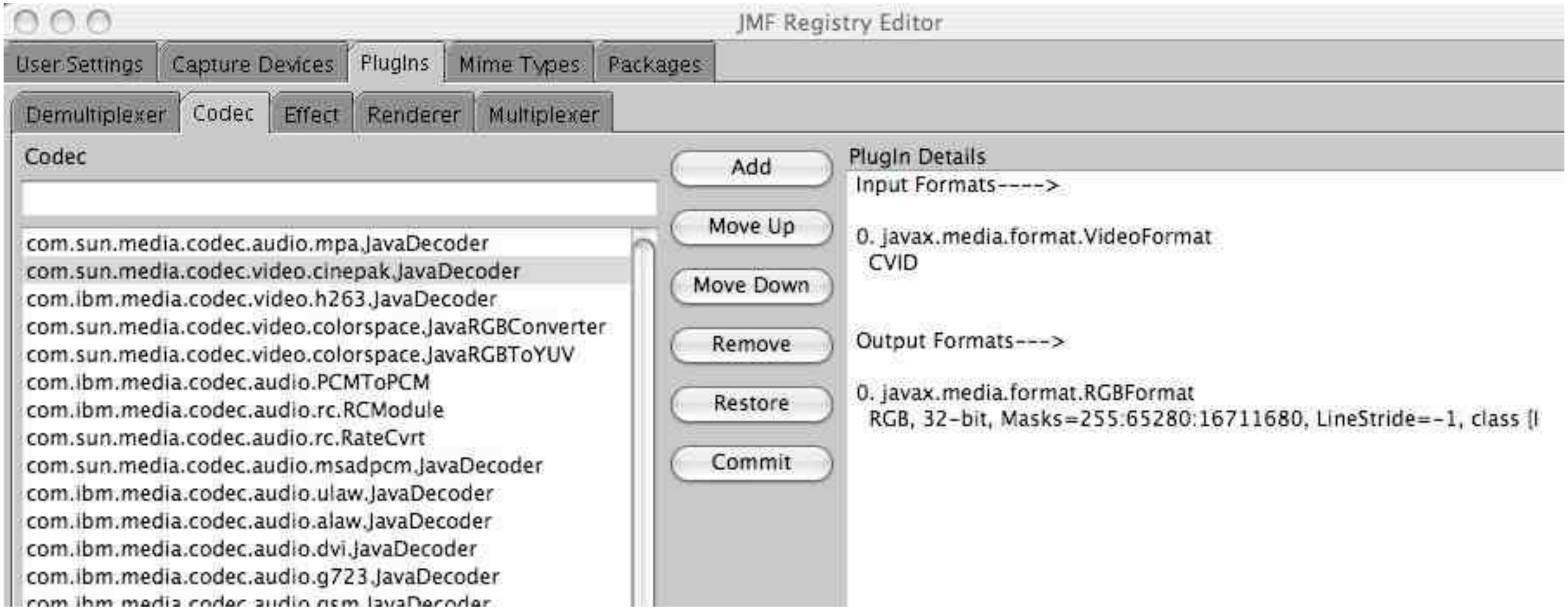


Example: State Model of JMF `Player`



- *Unrealized:*
  - Start state
- *Realizing:*
  - Media dependent parts of *player* are accessed
- *Prefetching:*
  - Input stream is read to fill buffer
- *Started:*
  - Processing is being executed

Example: Codec Plugin Architecture in JMF



3 Challenges in Multimedia Programming

3.1 Frameworks & Media Integration

3.2 Time Synchronization

3.3 Interactive and Event-Driven Programs

Literature:

P. Ackermann: Developing Object-Oriented Multimedia Software based on the MET++ Application Framework, dpunkt 1996

Synchronization Levels

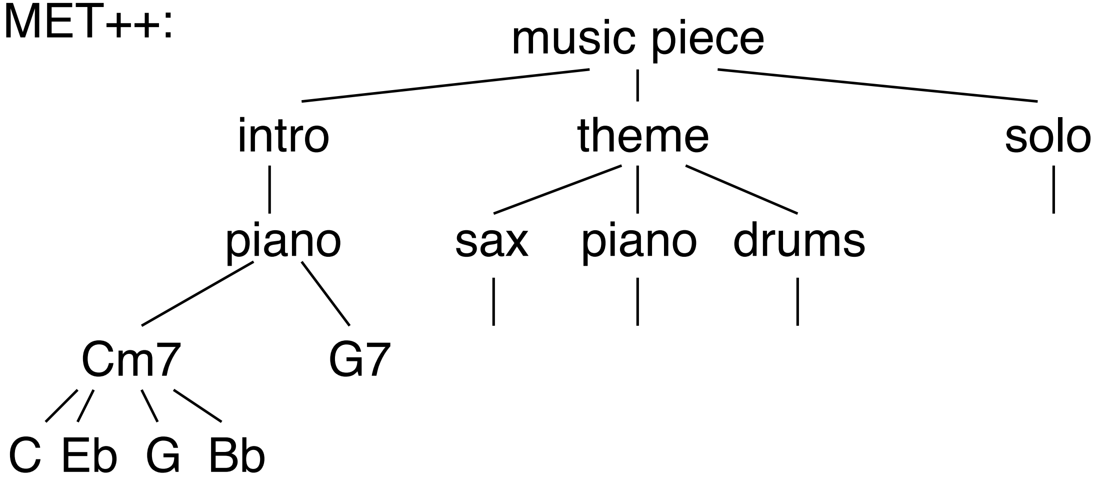
- Intramedia synchronization
  - Low-level synchronization
  - Ensures continuity of playback in a single media stream
  - Should be dealt with in media-specific classes of the framework
- Intermedia synchronization
  - Guarantees synchronization between different media streams
  - All media streams are synchronized according to a *global clock*
  - Is the key goal of the time synchronization mechanisms in the framework

Specification Paradigms for Timing

- Formal language
  - Programming language:
    - » Control flow defines timing
    - » Expressiveness achieved through constructs for concurrency: Threads, active or passive waiting (Example: Python/Pygame)
  - Declarative specification language:
    - » E.g. temporal logic expression (“X is repeated until Y” etc.)
- Time functions (time line)
  - Basic principle: Function from time value to parameter value
  - Parallel *tracks* to express concurrency (Example: Flash)
- Event composition
  - Implicit ordering given by event processing
  - May include temporal relations for events (like *before, meets, overlaps, during, after, …*)

Time Containment Hierarchy

Example from MET++:



- Media presentations have an inherent hierarchy of sub-parts
  - Far beyond simple parallel tracks!
- Time container concept:
  - Part of the containment hierarchy enhanced with *time layout* specification
    - » E.g. parallel, sequential, individual (relative) event specifications
  - *Glue* objects and strategies fill gaps in layout (e.g. logo, freeze, silence, …)

Variations of Time Functions

- *Time function*:
  - Maps a time value onto a parameter determining the audio/visual presentation (Concept from MET++)
  - Various *interpolation strategies* are used to compute intermediate values
    - » May affect performance of individual media elements (e.g. *local time warping* in MET++)
- Time line in JavaFX:
  - General mechanism to compute parameter values
  - Playable sub-presentation (time container)
- Time line in Flash:
  - Using parallel tracks (from visual authoring metaphor)
  - Time lines may be nested (objects having their own time line)

Time Events

- *Rule:* Timing in general is relative to presentation time (enables fast forward etc. by changing presentation *speed*)
- Absolute timing:
  - Clock event: “Tick” after certain time interval
  - Timer: Event fired after a certain time has elapsed
- Media-specific timing:
  - E.g. “new frame” event for video/animation
- Sub-element relative timing:
  - Start and end of presentation of a sub-element
  - May include delay specification (“3 seconds after end of clip 2”)
  - *Cueing* events (reaching a certain point in a time-dependent presentation)

3 Challenges in Multimedia Programming

3.1 Frameworks & Media Integration

3.2 Time Synchronization

3.3 Interactive and Event-Driven Programs

Literature:

P. Ackermann: Developing Object-Oriented Multimedia Software
based on the MET++ Application Framework, dpunkt 1996

Event-Driven Programming

```
while True:
   for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        if event.type == pygame.KEYDOWN:
            if event.key in [K_SPACE,K_RIGHT]:
                ...
            if event.key == K_LEFT:
                ...
```

Pygame

- There is no classical “main control flow”
- Main program structure:
  - Set up configuration of objects
  - Enter infinite loop:
    - » Ask for new event(s)
    - » Process event

Listener-Style Event-Driven Programming

```
while True:
      for event in pygame.event.get():
         for listener in listeners[event.type]:
            listener.processEvent(event)
```

- Core Idea:
  - Event loop is part of the framework (main control flow)
  - Application programs just *register listener objects*
- Listener object:
  - Variation of *Observer* pattern
  - Implements a defined interface
  - Registers with event processing framework

(A)Synchronous Event Processing

- Synchronous event processing:
  - Event processing is like a procedure call
  - Control is given to listener when event arrives
  - Control is given back to main event loop after event is processed
  - Danger: Blocking main event loop
- Asynchronous event processing:
  - Event processing is a concurrent/parallel *thread*
  - Event processing thread is informed of relevant events
  - Execution of main event loop is not blocked by event processing
  - More flexible, safer, more difficult to program