Entwurfsbericht

Synchronisation und Auswertung von Roboterfußballvideos

tj18b

Mitglieder

Dan Häßler
Robert Wagner
Sirk Petzold
Alexander Eichhorn
Erik Diener
Jonas Wrubel
Tomas Daetz Chacon

Betreuer

Hans-Gert Gräbe
Tobias Wieprich
Tobias Jagla
André Köhler

Inhaltsverzeichnis

1 Visionen und Ziele

/LV10/ Das System soll für das NAO-Team HTWK die Bearbeitung und Auswertung von Videos einzelner Roboterfußballspiele vereinfachen und diverse Bearbeitungsmöglichkeiten bereitstellen.

/LZ10/ Das System soll bis zu vier Videos mit festem Dateiformat und festgelegter FPS-Zahl und die dazugehörigen Logs automatisch synchronisieren. Dabei sollen möglichst viele Schritte editierbar und transparent gehalten werden.

2 Rahmenbedingungen und Produktübersicht

Die Applikation gliedert sich aktuell in 5 wesentliche Bereiche. Als zentrale Einheit dient die **Video-Area**, in der alle geladenen Videos abgespielt und alle Operationen abgebildet werden.

Die **Kontrolleinheit** enthält den Play-Button zum Abspielen der Videos, eine Stop und Skip-Funktion für 1 Frame, sowie eine Skip-Funktion für eine ausgewählte Anzahl von Sekunden. Die Skipfunktionen wurden jeweils vorwärts und rückwärts implementiert. Deren Frameweite wird über das Setupmenü einstellbar sein. Darüber hinaus gibt es einen Loadbutton für jedes Video, mit dem das geladene Video verändert werden kann. Eine Drag&Drop Funktion soll zusätzlich noch eingeführt werden. Darüber hinaus folgt noch eine Fortschrittsanzeige, mit der sowohl Videos als auch die Logs entsprechend vor- und zurückgespult werden können.

In der **Track-Area** werden die einzelnen Spuren für die Videos dargestellt, in denen entsprechende Markierungen gesetzt werden können die in der Speicherung im MLT-Format berücksichtigt werden. Die Spuren sind untereinander responsive und erlauben keine Überlappungen. Vom Nutzer erzeugte Überschneidungen werden demnach vom Programm automatisch korrigiert.

Die **Log-Area** dient der Darstellung der geparsesn Logfiles und selektiert stets das Element das zur aktuellen Videoposition passt. Darüber hinaus kann hier per Klick ein Element ausgewählt werden, wodurch alle Videos zu der entsprechenden Stelle gespult werden.

In der **Menüleiste** werden zudem einige zusätzliche Funktionen implementiert. Das Laden von Logfiles, Videos und Bilddateien wird hier ebenso möglich sein, wie das Aufrufen verschiedener Hilfsdokumente, das Auswählen der gewünschten Audiospur und das Rückgängig machen der letzten Aktion. Hiefür wurden bereits die ersten Bedienungsanlagen implementiert, welche allerdings noch ohne Wirkung bleiben.

Damit Komponenten die sich nicht aneinander gekoppelt sind sauber miteinander kommunizieren können, wurde ein Publish-Subscripe System in Form eines Eventbus implementiert. Dieser könnte allerdings noch durch den Einsatz einer Library ersetzt werden. Die Anwendung ist in ihrer Größe dynamisch veränderlich, läuft aktuell unter Windows

und Linux und wird im Laufe der Entwicklung noch einige Verbesserungen bezüglich der Benutzeroberfläche erhalten. Die Darstellung der Videos wird in einem Anwenderfreundlicheren Kontext erweitert. Abdockbare Fenster sind hierbei aktuell im Gespräch.

3 Grundsätzliche Struktur- und Entwurfsprinzipien

3.1 Programmiersprachen/Markup

Die ausschließlich verwendete Programmiersprache ist Java. FXML wird als markup language für das UI verwendet.

3.2 Architektur/Struktur

Die Software kommt ohne eine Server-Anwendung aus und ist eigenständig als Stand-Alone Application funktional. Als Build-Tool wird Maven verwendet.

3.3 Frameworks/Libraries

Die grafische Benutzeroberfläche wird mit JavaFX und dem internen Tool SceneBuilder modelliert. Da aber JavaFX im Bereich Audio- und Videomanipulation einigen unserer Anforderungen nicht gerecht wird, binden wir das Java Framework vlcj ein.

3.4 Vorgehensweise

Zuerst wurde ein Grundgerüst erstellt, welches bereits alle bisher für notwenig erachteten Klassen enthält und in dem der Großteil der geplanten Methoden deklariert ist (siehe Abschnitt „Datenmodell“). Die einzelnen Funktionalitäten werden nacheinander - wie im Releaseplan beschrieben - entwickelt und zusammengeführt. Leichte Abweichungen aufgrund von schwer vorhersehbaren Problemen waren eingeplant und sind bereits eingetreten. Darüber hinaus konnte bereits ein Vorsprung im Entwicklungsfortschritt gegenüber der Planung erwirkt werden. Dieser soll möglichst beibehalten werden, um ein Zeitfenster zur Lösung eventuell größerer Probleme zu gewährleisten.

4 Struktur- und Entwurfsprinzipien einzelner Pakete

4.1 Package: Source Packages

Dieses Package beinhaltet die entwickelten Java Module, welche sich aus java Dateien zusammensetzen. Dabei ist weitestgehend jedes Package als Modul implementiert.

robokicker.config	Parameter/Konfigurierbare Werte
robokicker.pubsub	EventBus Implementierung des Publish-Subscribe Pattern
robokicker.log	Log file parse, LogPane als GUI-Controller zur zugehörigen FXML, Log Entry als Datenstruktur
robokicker.menu	Menu, SettingsMenu als GUI-Controller
robokicker.track	TrackRange als Datenstruktur, TrackPane als GUI-Controller und die TrackBox als Spurelement
robokicker.scene	Robokicker als main class, sowie Szene als GUI-Controller
robokicker.videoplayer	VideoPlayer, VideoPane als GUI-Controller

4.2 Package: Test Packages

Gleicher Aufbau wie source packages. Beinhaltet JUnit Tests zu jeder testrelevanten Klasse.

4.3 Package: Resources

In diesem Package liegen folgende hierarchisch untergeordneten Packages:

fxml Hier liegen alle FXML-Dateien, die die hierarchische Struktur und das Layout des GUI beschreiben. Dabei besitzt jede größere Komponente eine eigene fxml-Datei.

Das macht den Code übersichtlicher und bringt dadurch Vorteile in der Wartbarkeit und Wiederverwendbarkeit.

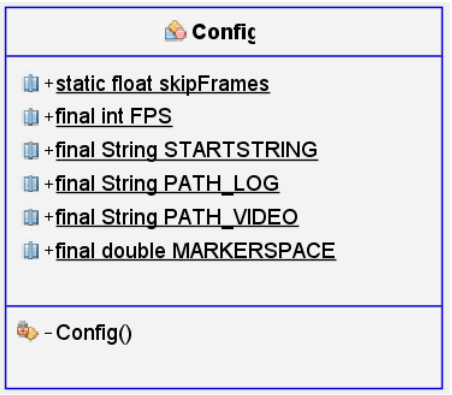
icons In diesem Package liegen Icons.

styles In diesem Package liegt die CSS Datei für das GUI.

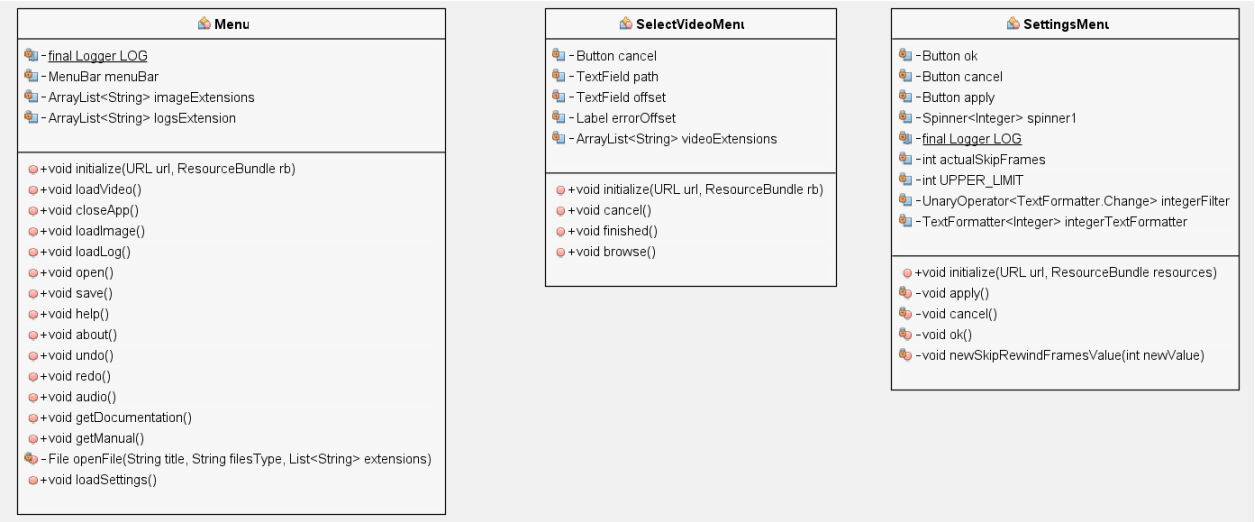
test In diesem Package liegen Videodateien und Textdateien zum Testen während des Entwickelns.

5 Datenmodell

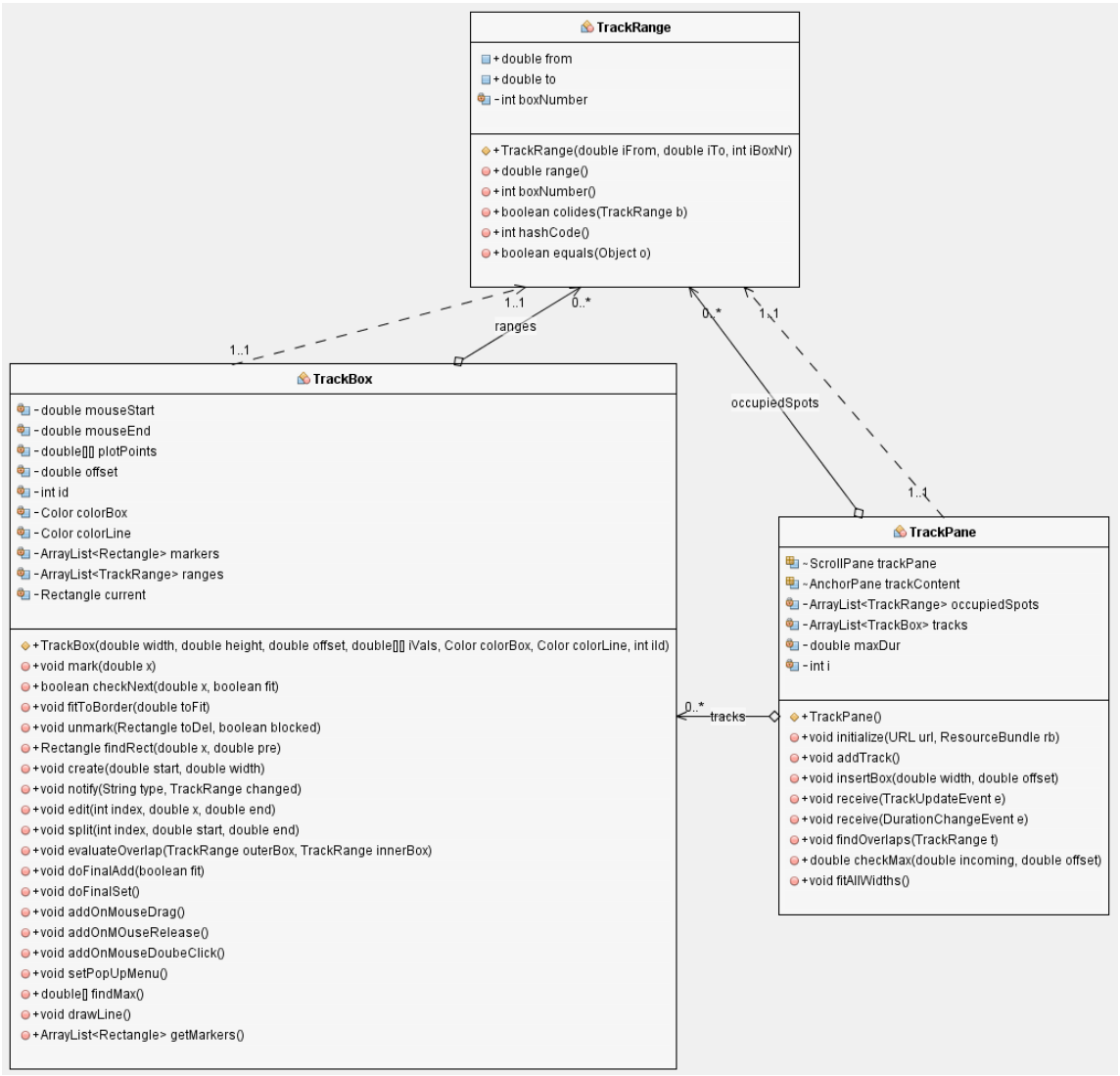
5.1 Config



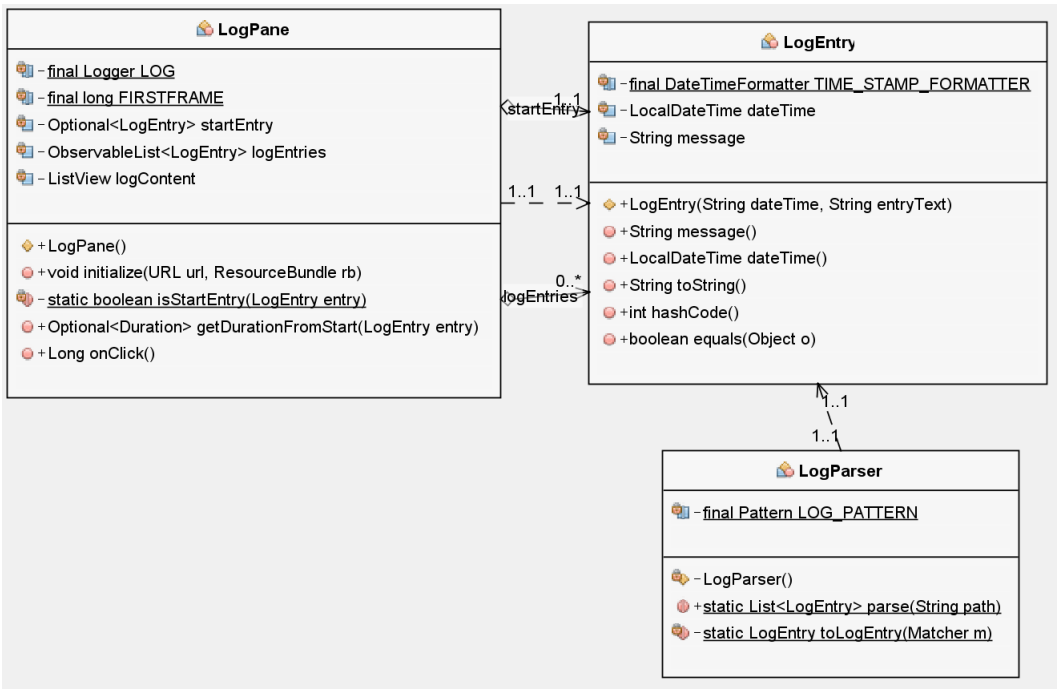
5.2 Menu


Menu
- final Logger LOG
- MenuBar menuBar
- ArrayList<String> imageExtensions
- ArrayList<String> logsExtension
+void initialize(URL url, ResourceBundle rb)
+void loadVideo()
+void closeApp()
+void loadImage()
+void loadLog()
+void open()
+void save()
+void help()
+void about()
+void undo()
+void redo()
+void audio()
+void getDocumentation()
+void getManual()
- File openFile(String title, String filesType, List<String> extensions)
+void loadSettings()
SelectVideoMenu
- Button cancel
- TextField path
- TextField offset
- Label errorOffset
- ArrayList<String> videoExtensions
+void initialize(URL url, ResourceBundle rb)
+void cancel()
+void finished()
+void browse()
SettingsMenu
- Button ok
- Button cancel
- Button apply
- Spinner<Integer> spinner1
- final Logger LOG
- int actualSkipFrames
- int UPPER_LIMIT
- UnaryOperator<TextFormatter.Change> integerFilter
- TextFormatter<Integer> integerTextFormatter
+void initialize(URL url, ResourceBundle resources)
-void apply()
-void cancel()
-void ok()
-void newSkipRewindFramesValue(int newValue)

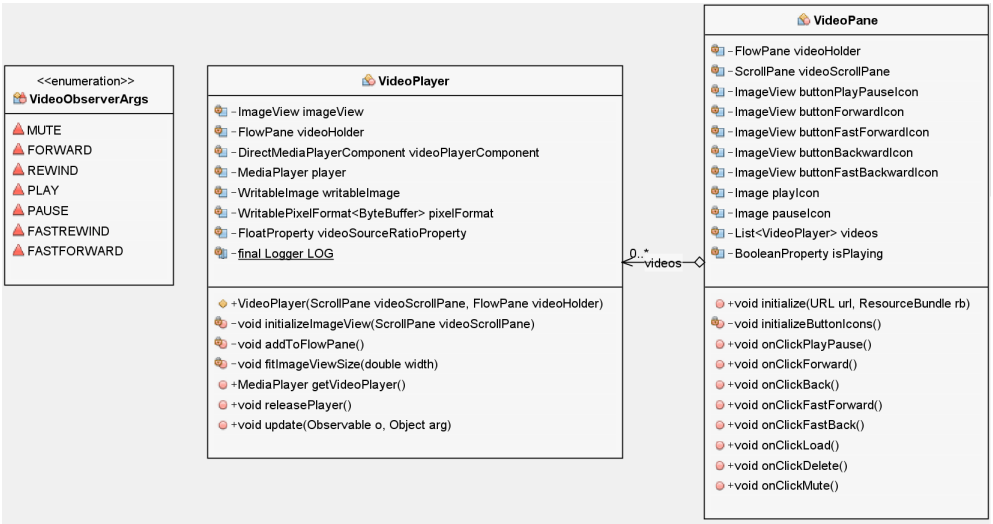
5.3 Trackpane



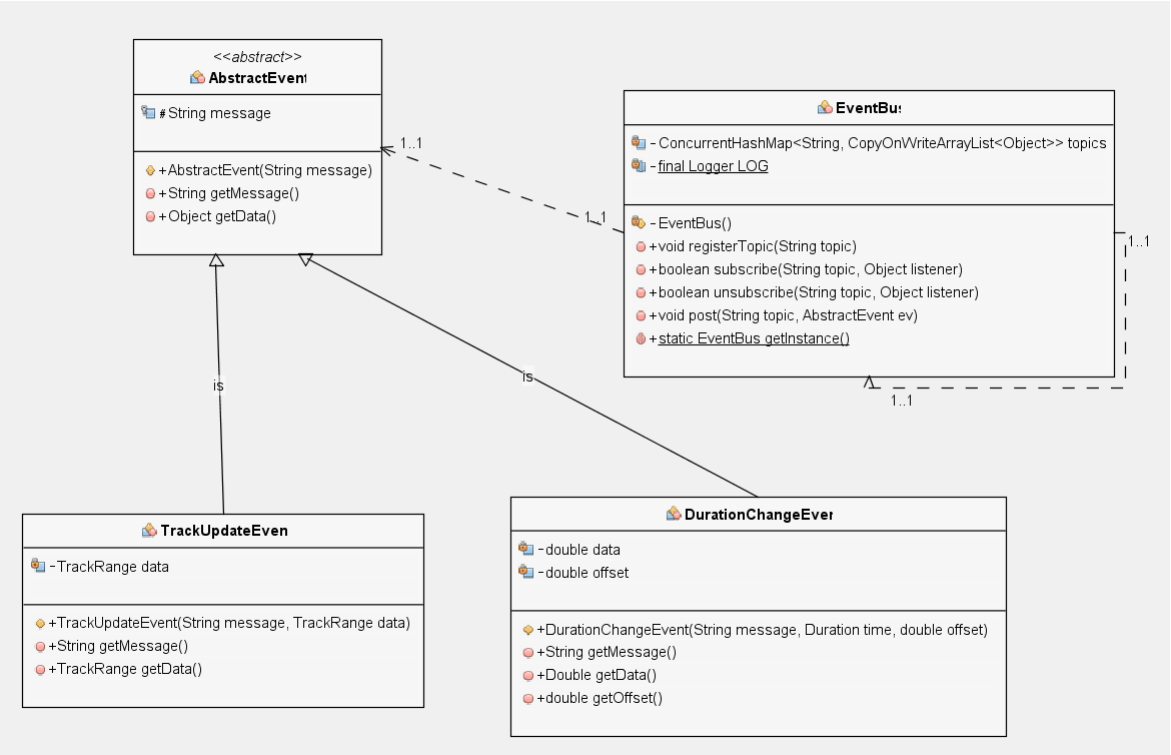
5.4 Logpane



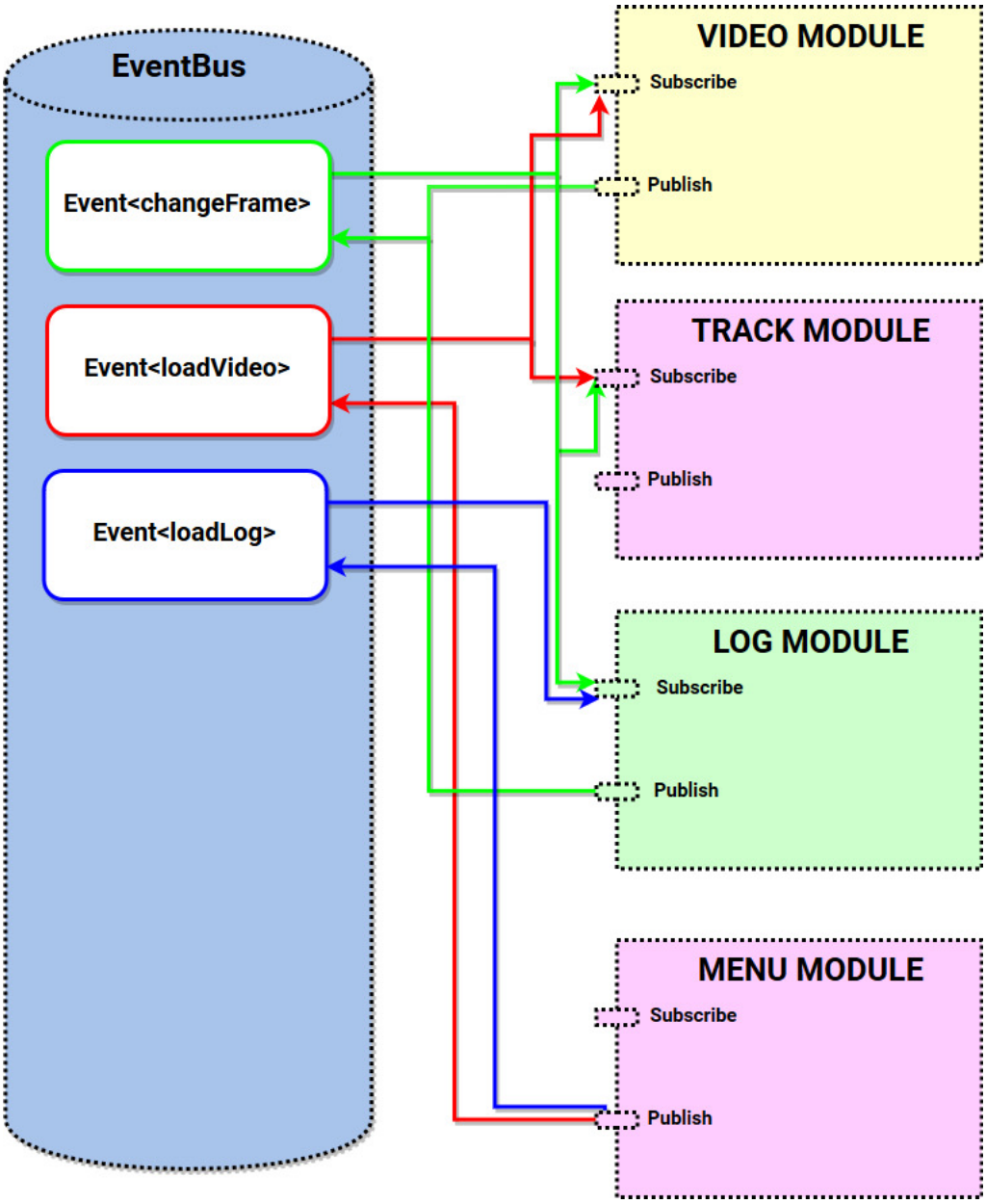
5.5 Videopane



5.6 Eventbus


<<abstract>>
AbstractEvent
String message
+AbstractEvent(String message)
+String getMessage()
+Object getData()
EventBus
- ConcurrentHashMap<String, CopyOnWriteArrayList<Object>> topics
- final Logger LOG
- EventBus()
+void registerTopic(String topic)
+boolean subscribe(String topic, Object listener)
+boolean unsubscribe(String topic, Object listener)
+void post(String topic, AbstractEvent ev)
+static EventBus getInstance()
1..1
1..1
1..1
1..1
is
is
TrackUpdateEvent
- TrackRange data
+TrackUpdateEvent(String message, TrackRange data)
+String getMessage()
+TrackRange getData()
DurationChangeEvent
- double data
- double offset
+DurationChangeEvent(String message, Duration time, double offset)
+String getMessage()
+Double getData()
+double getOffset()

6 Publish-subscribe



7 Glossar

Markup Language (dt.: Auszeichnungssprache) Mithilfe dieser Kategorie von Programmiersprachen werden Inhalte, Formatierungen und Datenaustausch beschrieben, oder weitere Auszeichnungssprachen definiert.

Stand-Alone Application Ist eine Software-Anwendung, welche auf jedem Client-System (Desktop Computer) installiert werden kann und ohne Abhängigkeit zu anderen Services lauffähig ist.

JavaFX Von Oracle entwickeltes Framework, zur professionellen Erstellung plattformübergreifender Java-Applikationen, mit interaktiven und multimedialen GUIs.

FXML XML-basierte Markup Language, dient der deklarativen Beschreibung von grafischen Oberflächen.