PSE 2012

OQAT

Objective Quality Assessment Toolkit

Praxis der Softwareentwicklung
WS 2012

E n t w u r f s d o k u m e n t

Auftraggeber
Karlsruher Institut für Technologie
Institut für Technische Informatik
CES - Chair for Embedded Systems
Prof.Dr.J.Henkel

Betreuer: S. Kobbe

Auftragnehmer

Name	E-Mail-Adresse
Eckhart Artur	artur.eckhart@gmail.com
Ermantraut Georg	georg.ermantraut@gmail.com
Leidig Sebastian	sebastian.leidig@gmail.com
Monev Alexander	bcclan@mail.bg
Sailer Johannes	johsailer@gmail.com

Karlsruhe, 20.5.2012

Inhaltsverzeichnis

1 Einleitung

Die Anwendung *Objective Quality Assessment Toolkit* , welche im Auftrag des CES hergestellt wird, wird wie im Pflichtenheft spezifiziert nach dem *Model View ViewModel* Entwurfsmuster angefertigt. Hierbei ist das Ziel ein möglichst lose gekoppeltes System der Aufgaben zu erreichen.

Das ViewModel fungiert unter anderem als Koordinationsmodul zwischen View und Model, so werden z.b. Databindings, welche sich nicht deklarativ (zur Entwicklungszeit im XAML-code) konstruieren lassen, von einem ViewModel einer jeweiligen View an ein bestimmtes ModelElement gebunden.

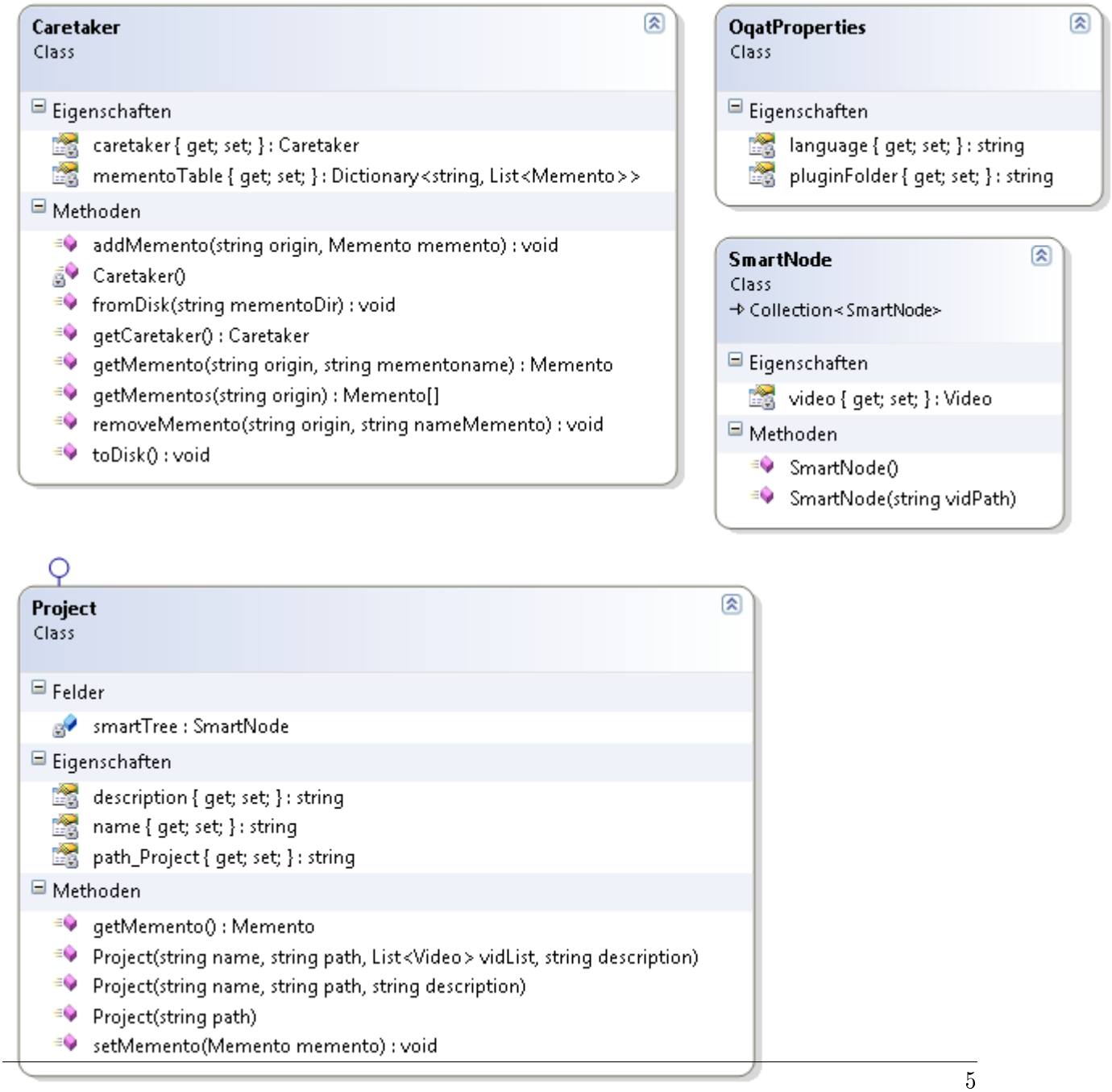
Das Model übernimmt neben der Datenhaltung und Organisation auch das Sichern und Wiederherstellen der Daten auf der Festplatte. Wobei Daten OQAT -Anwendungseinstellungen (Sprache, bestimmte Standardpfade) von OQAT sind oder von einem Plugin erstellte Informationen.

Da die View für OQAT mit Hilfe von WPF (Windows Presentation Foundation) entwickelt wird, ist die Verwendung von deklarativen Mitteln (XAML) den imperativen (C#) vorzuziehen. Durch solches Vorgehen wird Codedopplung bei der GUI-Entwicklung vermieden und eine gewisse Robustheit erreicht. Ein Nachteil der weitestgehend deklarativen Programmierung der View ist der Verlust einer sinnvollen UML-Diagramm-Darstellungen für diese, ohne mit der Implementierung der View anzufangen. Daher wurde auf das Erstellen einzelner View-Klassen in diesem Entwurf verzichtet. Um dennoch eine Vorstellung für die zu entwickelnde GUI zu bekommen, kann das Pflichtenheft zu Rate gezogen werden.

2 Gesamtentwurf

2.1 Model

2.1.1 Private Klassen



Project : Klasse
Alle für das Projekt relevanten Daten befinden sich hier. Dies beinhaltet einen Pfad, einen Namen und eine Beschreibung. IMemorizable wird implementiert.

OqatProperties : Klasse
Dies sind die globalen Einstellungen der Anwendung: Sprache sowie Pfad zum Pluginverzeichnis.

Caretaker : Klasse
Der Caretaker ist zuständig für das Laden und Speichern von Mementos von und auf die Festplatte. Er stellt Methoden bereit, damit andere Klassen Zugriff auf ihre Mementos haben und wird als Singleton realisiert. Der Caretaker verwaltet eine Tabelle von Strings und die dazu gehörigen Listen von Mementos.

SmartNode : Klasse
Videos eines Projektes werden durch SmartNodes in einer Baumstruktur zusammengestellt, um sie dann dem SmartTree der View zur Verfügung zu stellen. Eine SmartNode kann mehrere Kinder-SmartNodes haben.

2.1.2 Oqat Public Ressources

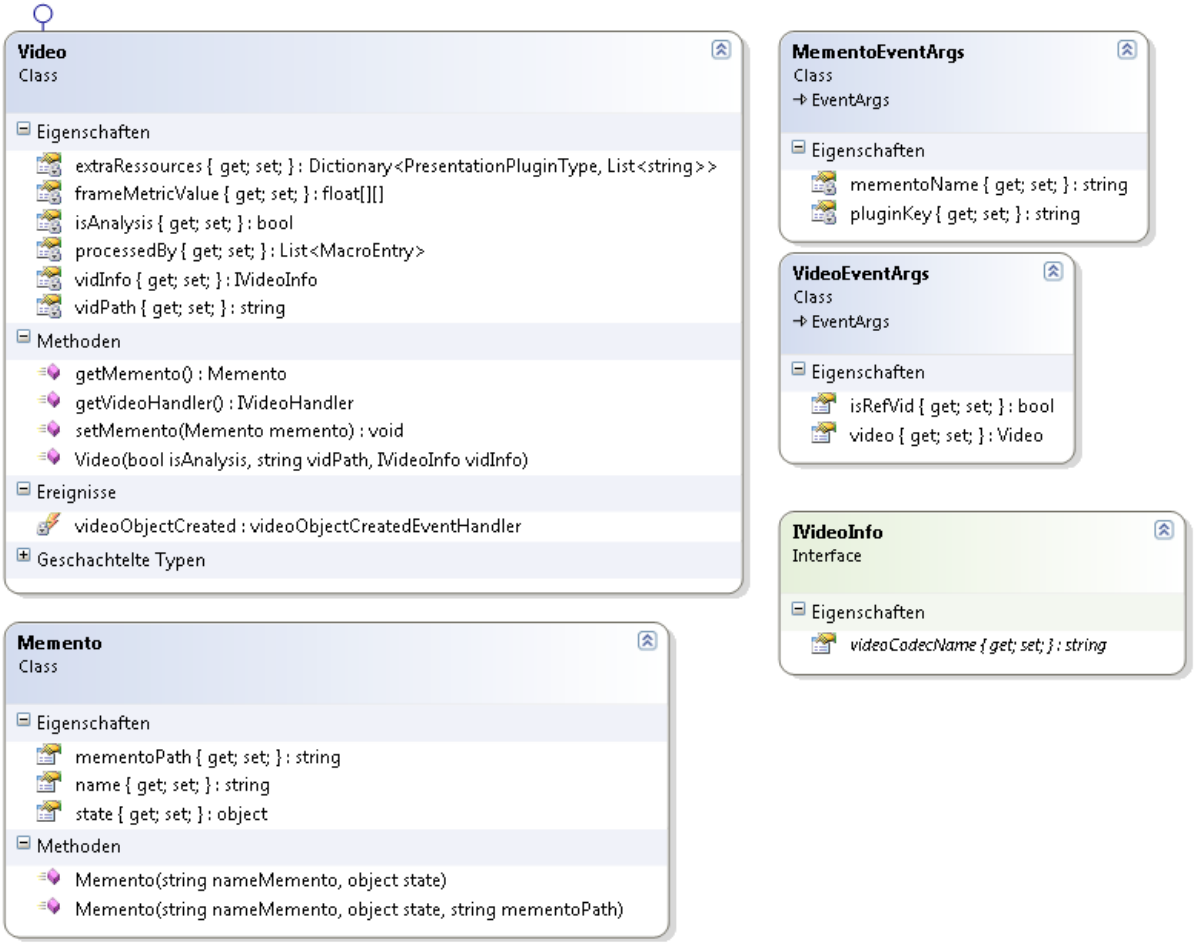


Abbildung 2.2: Model B

Video : Klasse
Im Video werden ein Pfad zu einer Videodatei, ein VideoInfo-Objekt sowie mögliche »Extraressourcen«, wie eine Datei zugehöriger Motionvektoren, gelegt. Falls es sich um ein Analysevideo handelt, werden zusätzlich entsprechende Analysedaten hier abgelegt. Ein Video kann außerdem auch einen VideoHandler zurück geben, mit dem auf die Video-Datei zugegriffen wird. Implementiert IMemorizable.

VideoEventArgs : Klasse
Enthält ein Video und ein Boolean, der wahr ist, falls es sich um ein Referenzvideo handelt. Wird von Events als Argument benutzt.

Memento : Klasse

Ein Memento speichert den Zustand eines Objektes. Es werden ein Name und gegebenenfalls auch ein Pfad zum Speichern auf die Festplatte bereitgestellt. Der Memento-Benutzer legt außerdem seine zu speichernden Daten ab.

MementoEventArgs : Klasse
Enthält zwei Strings: Name eines Mementos und Referenz zu einem Plugin. Wird von Events als Argument benutzt.

IMemorizable : Interface
Es wird ermöglicht, den Zustand einer Klasse als Memento zu speichern und vorherige Zustände zu laden.

IVideoInfo : Interface
Falls ein Videoformat Voreinstellungen braucht bietet IVideoInfo die Möglichkeit diese abzulegen.

2.2 ViewModel

2.2.1 Oqat Organisation

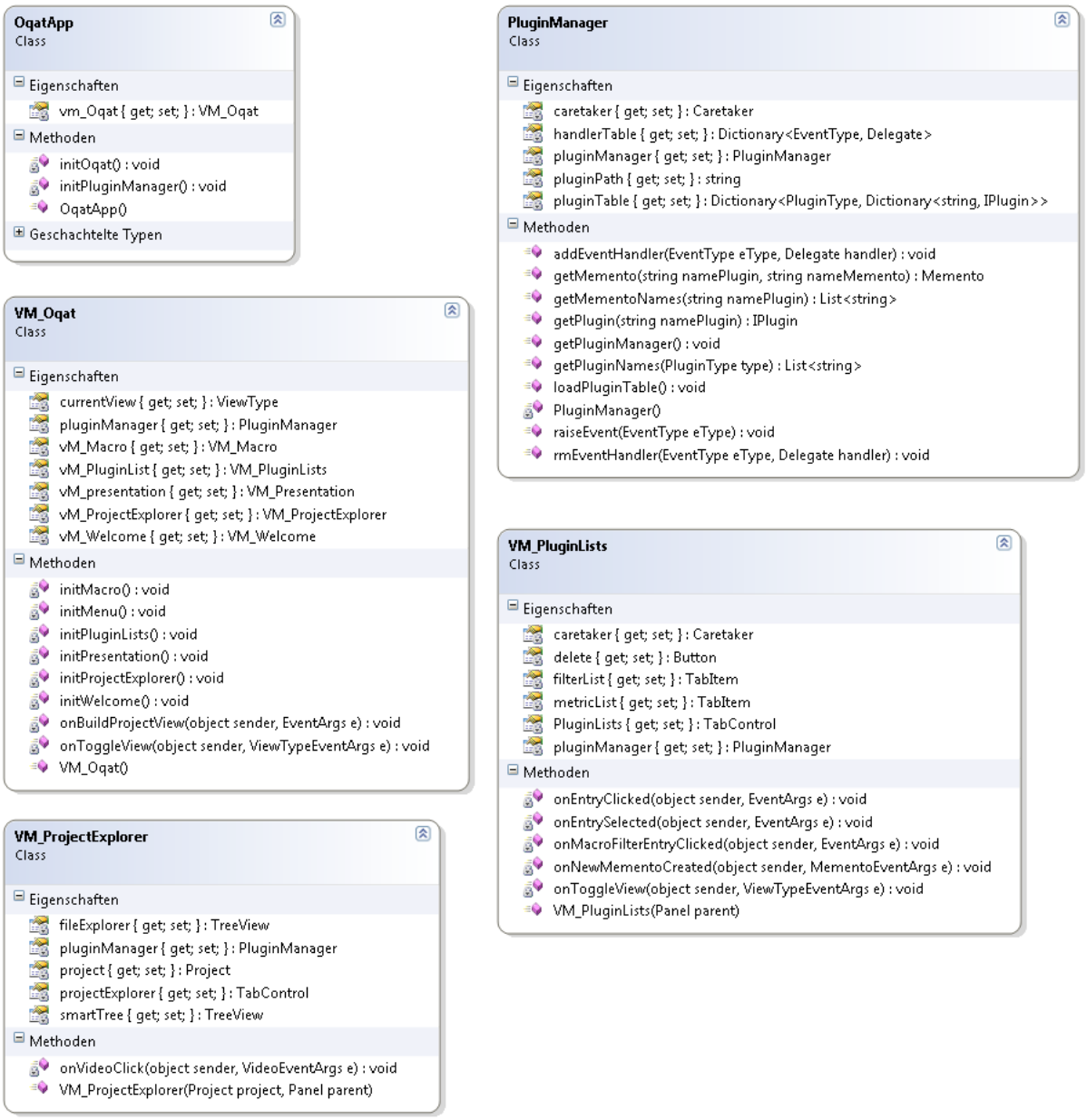


Abbildung 2.3: Oqat Organisation

Alle sichtbaren VM (ViewModel) implementieren eine Methode onToggleView, die für den Ansichtswechsel zuständig ist und eventgesteuert funktioniert.

OqatApp : Klasse

Die Klasse wird beim starten des Programms als erstes angesprochen und initialisiert die Anwendung und den Pluginmanager.

VM Oqat : Klasse
Diese initialisiert die ViewModel Komponenten. Dazu gehören Willkommensbildschirm, das Projekt, Menü, Presentation und die Pluginlisten. Die Klasse stellt auch eine Delegate-Methode zum Ansichtswechsel bereit.

VM ProjectExplorer : Klasse
Besteht aus einem SmartTree und einem Dateiexplorer, um diese in der GUI bereitzustellen. Er kann durch den Dateiexplorer Videos laden, diese werden im SmartTree angezeigt.

PluginManager : Klasse
Der Pluginmanager ist die Schnittstelle zwischen OQAT und den Plugins. Er stellt Methoden zum Abrufen von Plugins und den zugehörigen Mementos. Außerdem werden Events mit deren Handlern bei ihm registriert. Er ist als Singleton realisiert. Der Pluginmanager enthält einen Pfad zum Pluginordner sowie eine Tabelle, die den Plugintypen die zugehörigen Plugins zusammen mit deren Namen zuordnet.

VM Pluginlists : Klasse
Die Pluginlist verwaltet die hinzugefügten Metriken und Filter, sowie die anderen Plugins. Er kann die Events entryClick und entrySelect abfangen. Bei entryClick wird ein Filter oder eine Metrik vom Container ausgewählt und die zugehörigen Einstellungen angezeigt. Bei entrySelect wird der ausgewählte Filter oder Metrik zu einer Makrowarteschlange hinzugefügt. VM Pluginlists stellt eine Delegatemethode bereit, die neu erstellte Mementos nach dem Abfangen des jeweiligen Events zu dem jeweiligen Plugincontainer hinzufügt.

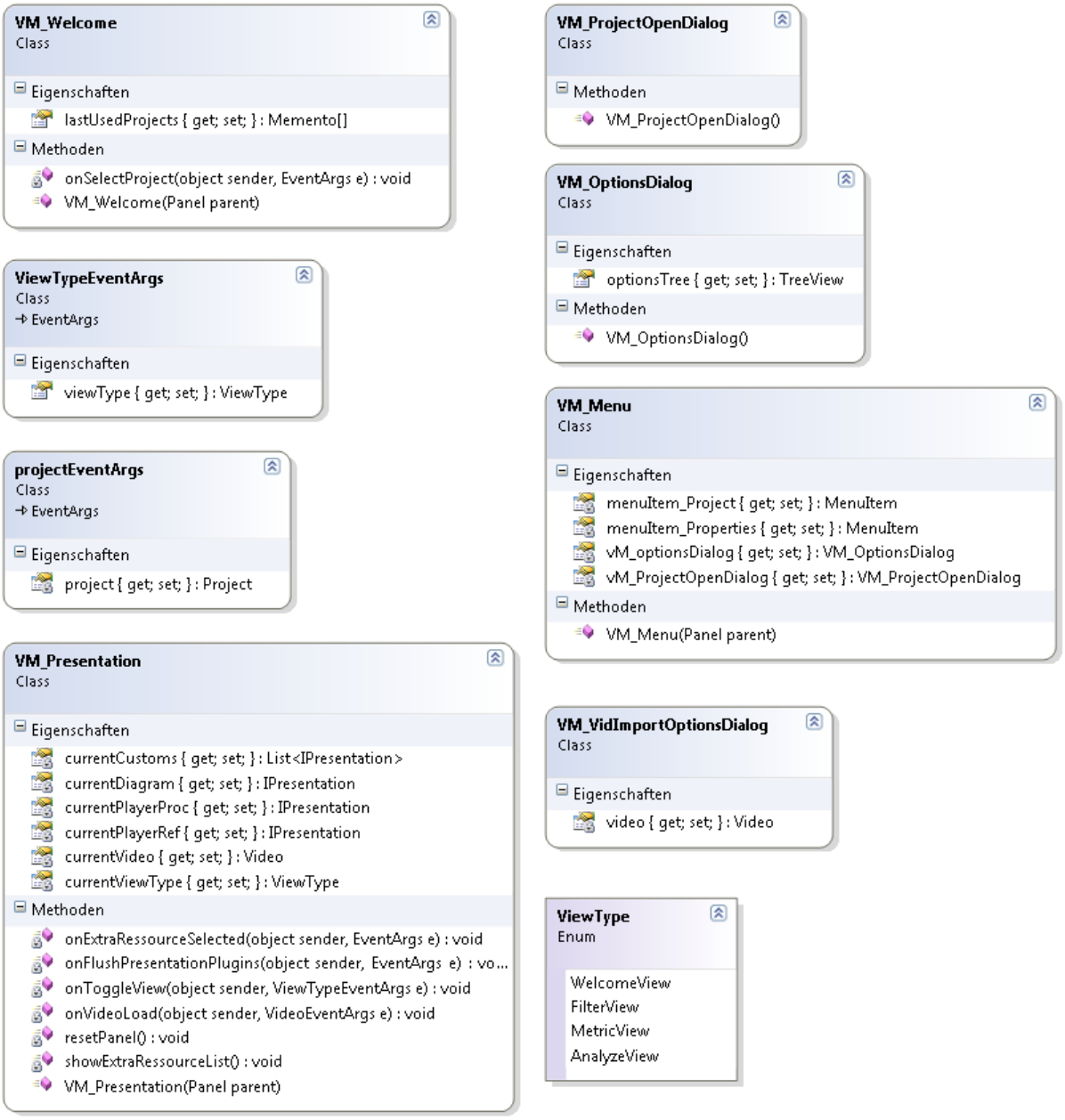


Abbildung 2.4: Oquat Organisation

VM Menu : Klasse

Dieses ViewModel ist für die Verwaltung der Menüleiste und der Menüitems zuständig.

VM Welcome : Klasse

Ist der Willkommensbildschirm, der beim Programmstart die zuletzt benutzten Projekte anzeigt.

VM OptionsDialog : Klasse

Optionsfenster für allgemeine Einstellungen von OQAT .

VM Presentation : Klasse
Der zentrale Visualisierungsbereich wird hiermit verwaltet. Dazu nutzt es die Presentationplugins um möglichst treffende Darstellungen zu erreichen.

ViewTypeEventArgs : Klasses
Enthält ein Objekt vom Typ ViewType. Wird von Events als Argument benutzt.

projectEventArgs : Klasse
Enthält ein Objekt vom Typ Projekt. Wird von Events als Argument benutzt.

ViewType : Enumeration
In dieser Enum sind die verschiedenen Typen von Ansichten aufgelistet.

VM ProjectOpenDialog : Klasse
Steuert das Fenster, das zum Öffnen eines Projektes verwendet wird.

VM VidImportOptionsDialog : Klasse
Steuert das Fenster, das zum Importieren eines Videos verwendet wird.

2.2.2 Macro

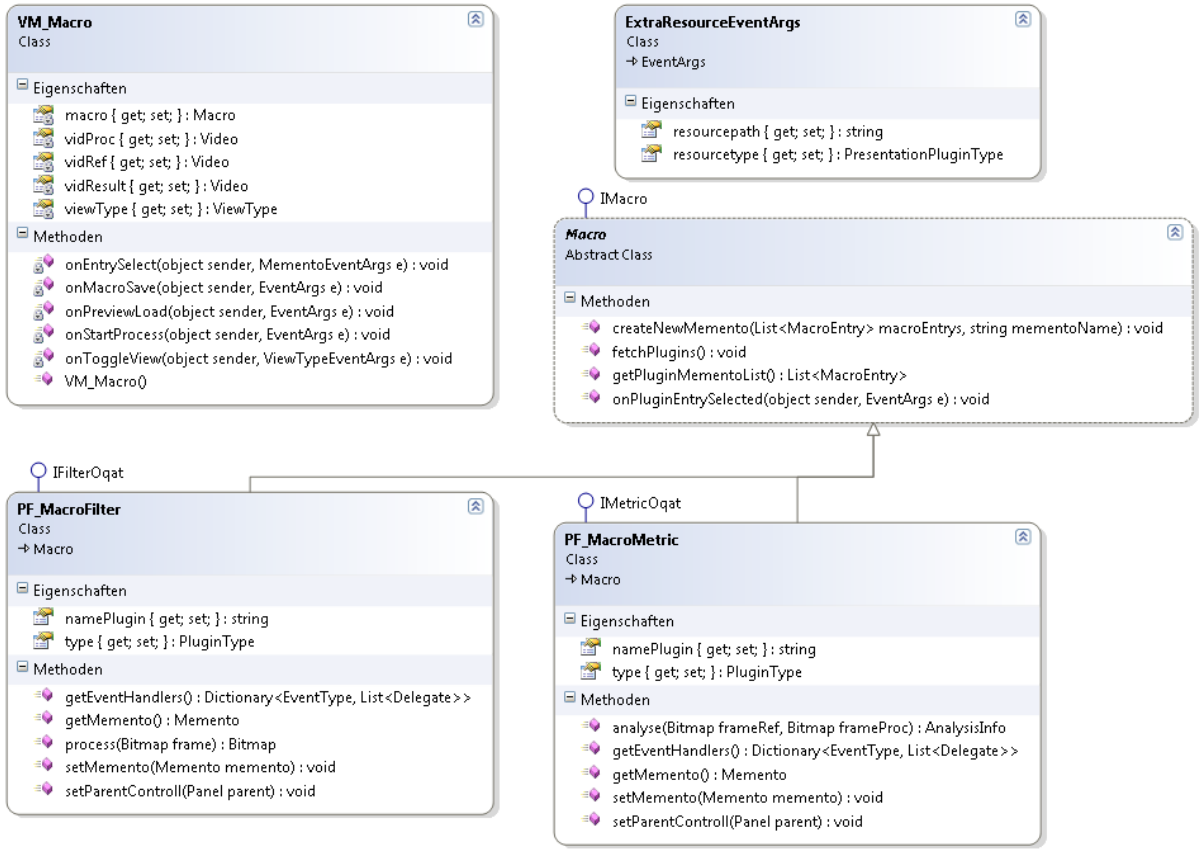


Abbildung 2.5: Macro

VM Macro : Klasse
Legt fest, was bei einem Macro-bezogenen Event passiert. VM Macro kann Events vom Typ onEntrySelect, onMacroSave, onStartProcess und onToggleView abfangen. OnEntrySelect dient zur Auswahl von Filtern oder Metriken, die zum Macro zugefügt werden. OnMacroSave dient zum Speichern des Macrofilters. OnStartProcess startet den Macrofilter oder –analysevorgang. VM Macro enthält als Attribute ein Objekt vom Typ Macro, ein Objekt vom Typ Viewtype sowie Referenzen zu Videoobjekten.

Macro : Klasse
Enthält eine Liste von MacroEntry-Objekten und implementiert IMacro.

PF MacroMetric : Klasse
Erbt von Macro, implementiert IMacro und IMetricOqat. Diese Klasse bietet Methoden zur Analyse von Frames bzw. Videos an. Die Liste von MacroEntry-Objekten enthält in diesem Fall Referenzen zu Metriken, die durch diese Methoden angewandt werden können.

PF MacroFilter : Klasse
Erbt von Macro, implementiert IFilterOqat. Die Liste von MacroEntry-Objekten enthält in diesem Fall Referenzen zu Filtern, die von einem MacroFilter durch die jeweiligen Methoden der Klasse PF MacroFilter auf Bilder bzw. ganze Videos angewandt werden können.

ExtraResourcesEventArgs : Klasse
Enthält Pfad zu Extraressourcen sowie deren Typ. Wird von Events als Argument benutzt.

2.3 Plugins

2.3.1 Präsentation

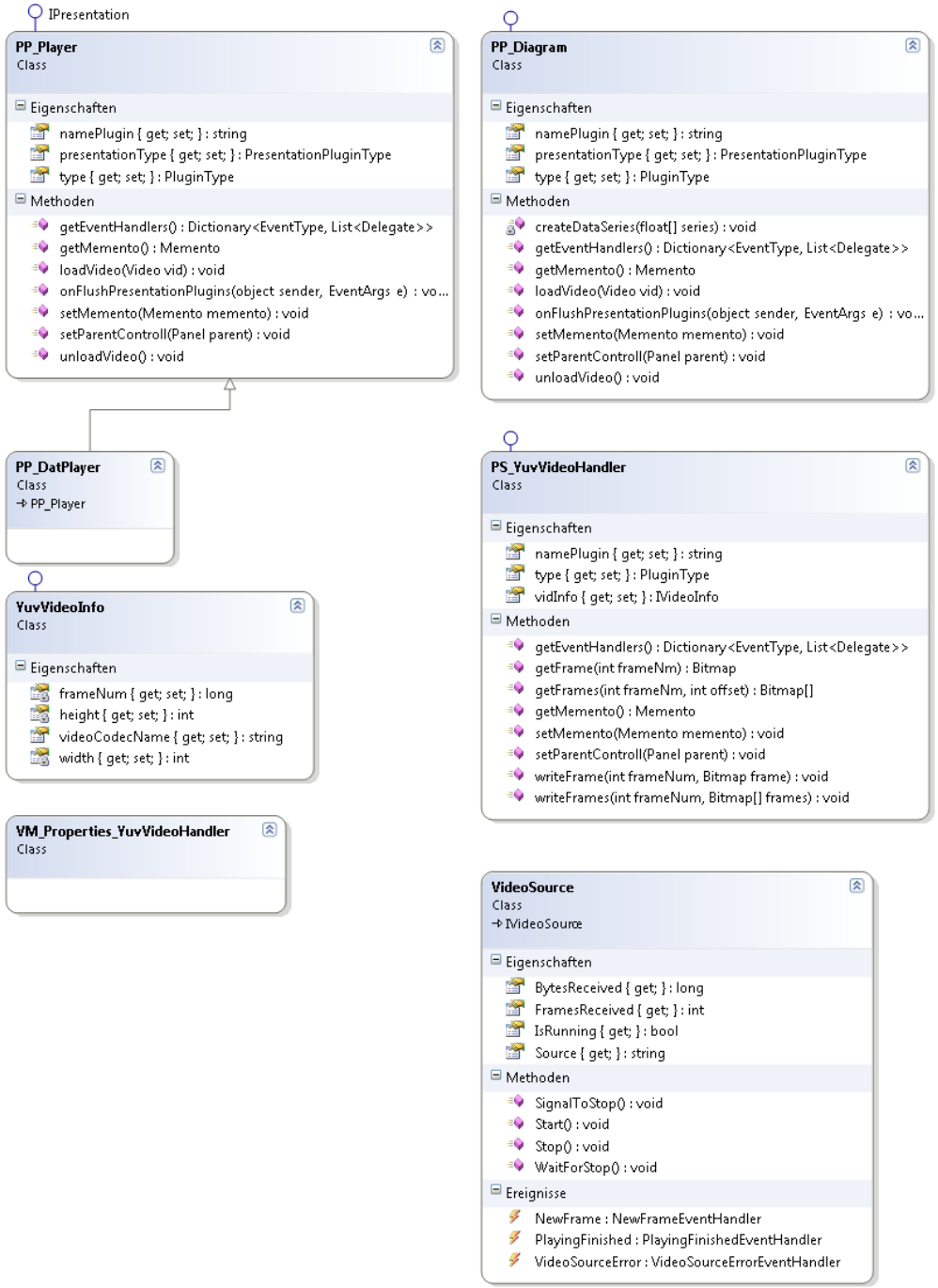


Abbildung 2.6: Presentation

PP Player : Klasse
Stellt einen Container dem VideoSourcePlayer zur Verfügung. Lädt und entfernt die Videos, die von dem VideoSourcePlayer gezeigt werden sollen.

PP Diagramm : Klasse
Stellt Oxyplot einen Container zum Zeichnen von Diagrammen zur Verfügung. Lädt und entfernt Videos, deren Attributen dann im Diagramm dargestellt werden können.

VideoSource : Klasse
Die VideoSource dient als Adapter zwischen einem VideoHandler, der Videodateien liest und der vom AForge-VideoSourcePlayer benötigten VideoSource. Implementiert IVideoSource von AForge. Implementiert Methoden zum Abspielen eines Videos.

PP DatPlayer : Klasse
Erbt von PP Player und bietet dem VideoSourcePlayer die Möglichkeit, MotionVektoren (Dat-Dateien) als Overlays darstellen.

PS YuvVideoHandler : Klasse
Erbt von IVideoHandler. Der YuvVideoHandler holt sich die Frames von einer YUV-Video-Datei bzw. schreibt sie wieder.

YuvVideoInfo : Klasse
Enthält Informationen, die benötigt werden, um ein YUV-Video abzuspielen: Anzahl Bilder und Auflösung.

VM Properties YuvVideoHandler : Klasse

2.3.2 Filter

Die Plugins vom Typ Filter enthalten die jeweilige Arithmetik für den jeweiligen Filter. Spezielle Filter wie z.B. Blur werden als Memento abgelegt. Die dazugehörige VM kontrolliert das Einstellungsfenster der Mementos. Es wird IFilterOqat implementiert.

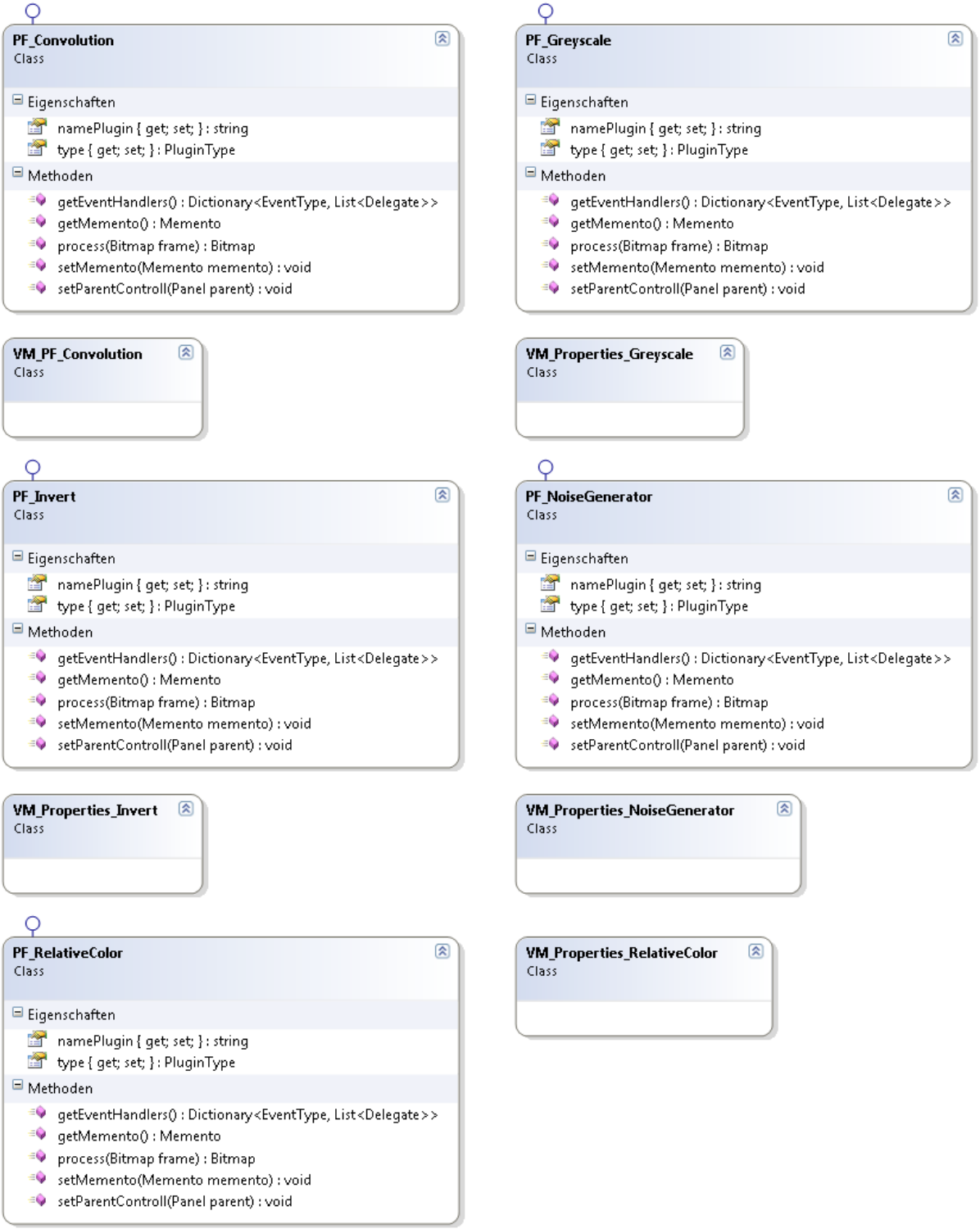


Abbildung 2.7: Filter

2.3.3 Metric

Die Plugins vom Typ Metric enthalten die jeweilige Arithmetik für die jeweilige Metrik. Es wird IMetricOqat implementiert.

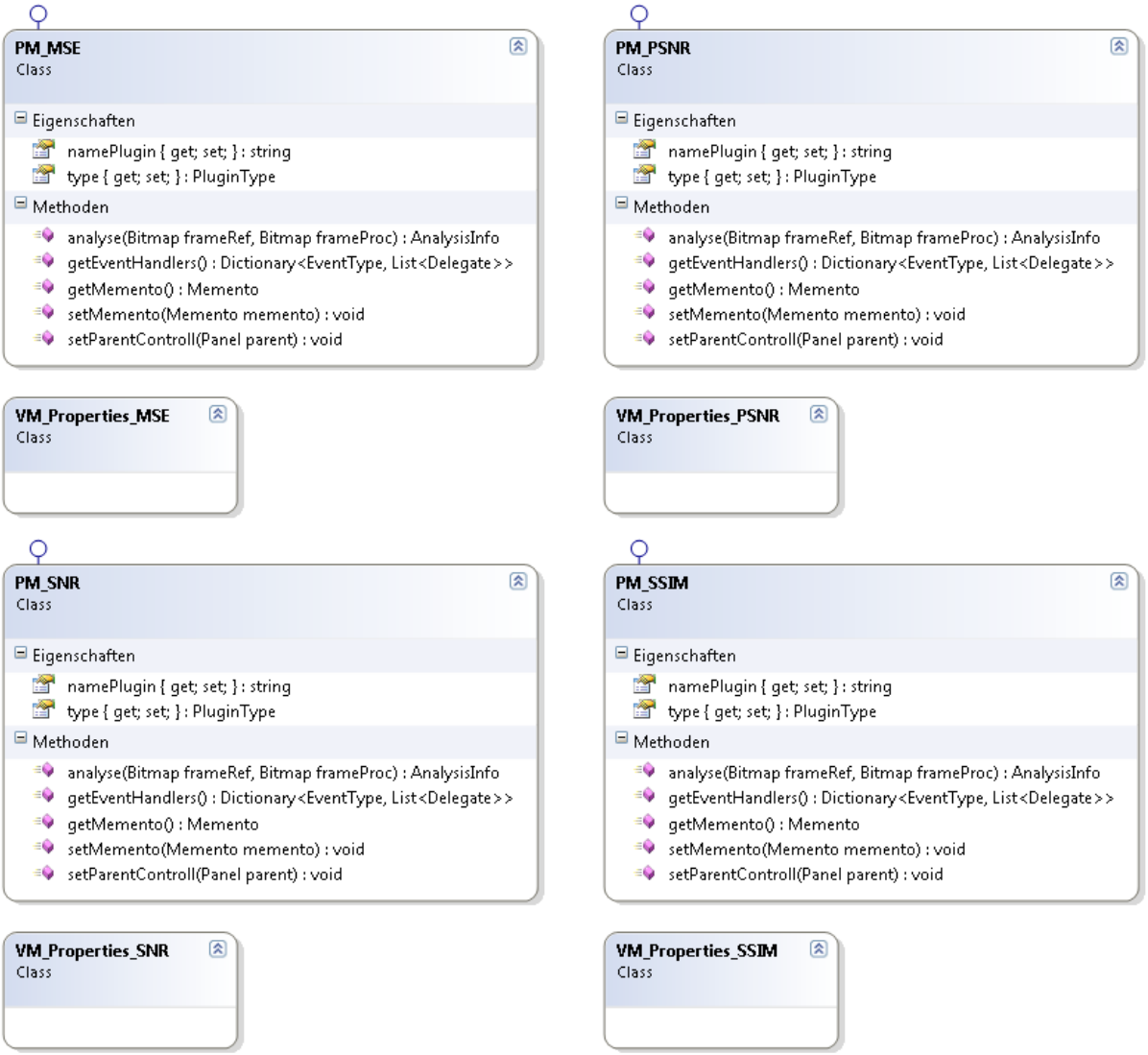


Abbildung 2.8: Metric

2.3.4 Oqat Public Ressources

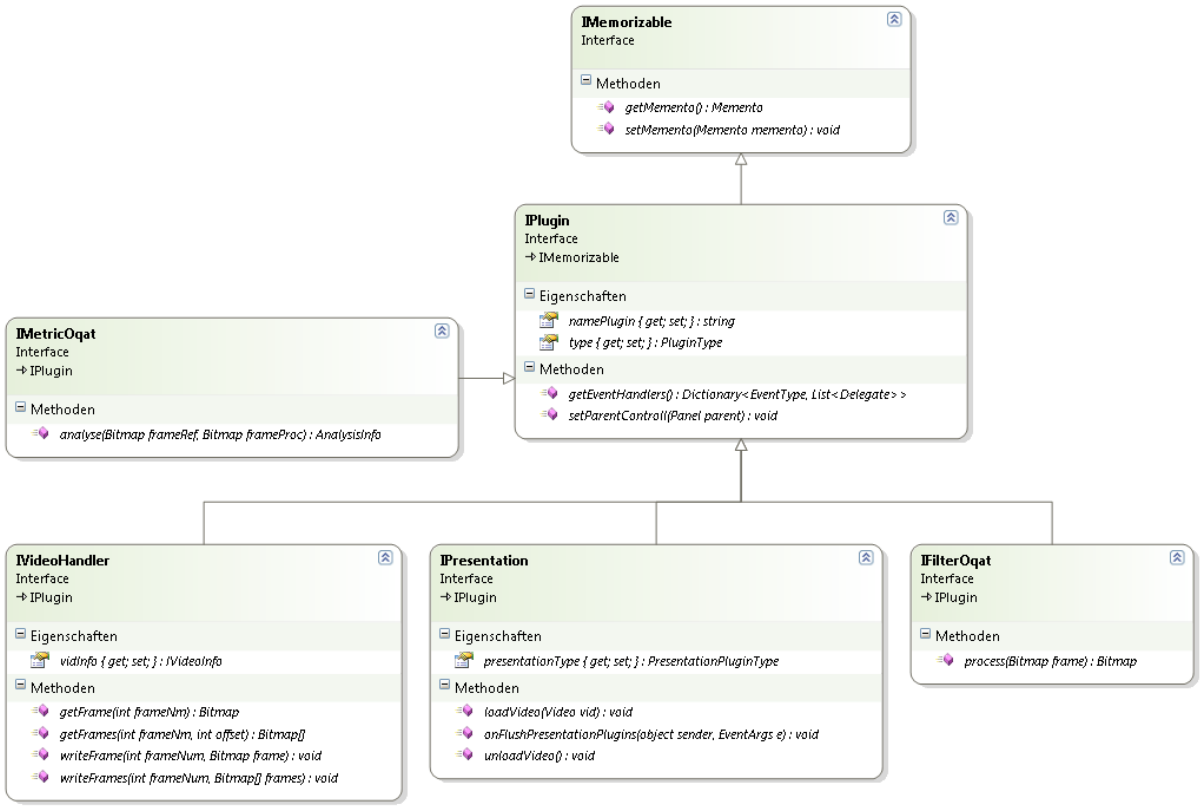


Abbildung 2.9: Public Plugins

IPlugin : Interface
Jede Klasse, die ein Plugin ist, muss dieses Interface implementieren. Namenattribut, Plugintypattribut und Methoden zur Verwaltung von Events (mittels Tabellen von EventTypen und den zugehörigen Delegatemethoden) werden dann von IPlugin geerbt. IPlugin implementiert IMemorizable.

IPresentation : Interface
Implementiert das IPlugin Interface. Plugins vom Typ Presentation müssen dieses Interface implementieren. IPresentation bietet Methoden, um ein Video zu laden oder entfernen, und legt fest, was beim Entladen von Presentationplugins passiert.

IMetricOqat : Interface
Implementiert das IPlugin Interface. Metriken müssen dieses Interface umsetzen. Es bietet eine Methode an, die für zwei Bitmap-Objekte ein IAnalysisInfo-Objekt zurückgibt.

IFilterOqat : Interface
Implementiert das IPlugin Interface. Filterplugins müssen dieses Interface umsetzten. Es bietet

eine Methode zur Bearbeitung von Bildern an, die ein Bitmap annimmt und wieder ein Bitmap zurückgibt.

IVideoHandler : Interface
Implementiert das IPlugin Interface. Das Interface stellt Methoden zur Verfügung, die mit Frames in einem Video umgehen, indem man den Methoden eine Framenummer übergibt. Es bietet eine Methode an, mit der man sich einen bestimmten Frame holen kann, sowie eine Methode, an die man eine Framenummer und ein Offset übergibt und sich somit einen Array von Frames (Bitmaps) aus einem Video holt. Analog funktionieren die Methoden zum überschreiben von Frames.

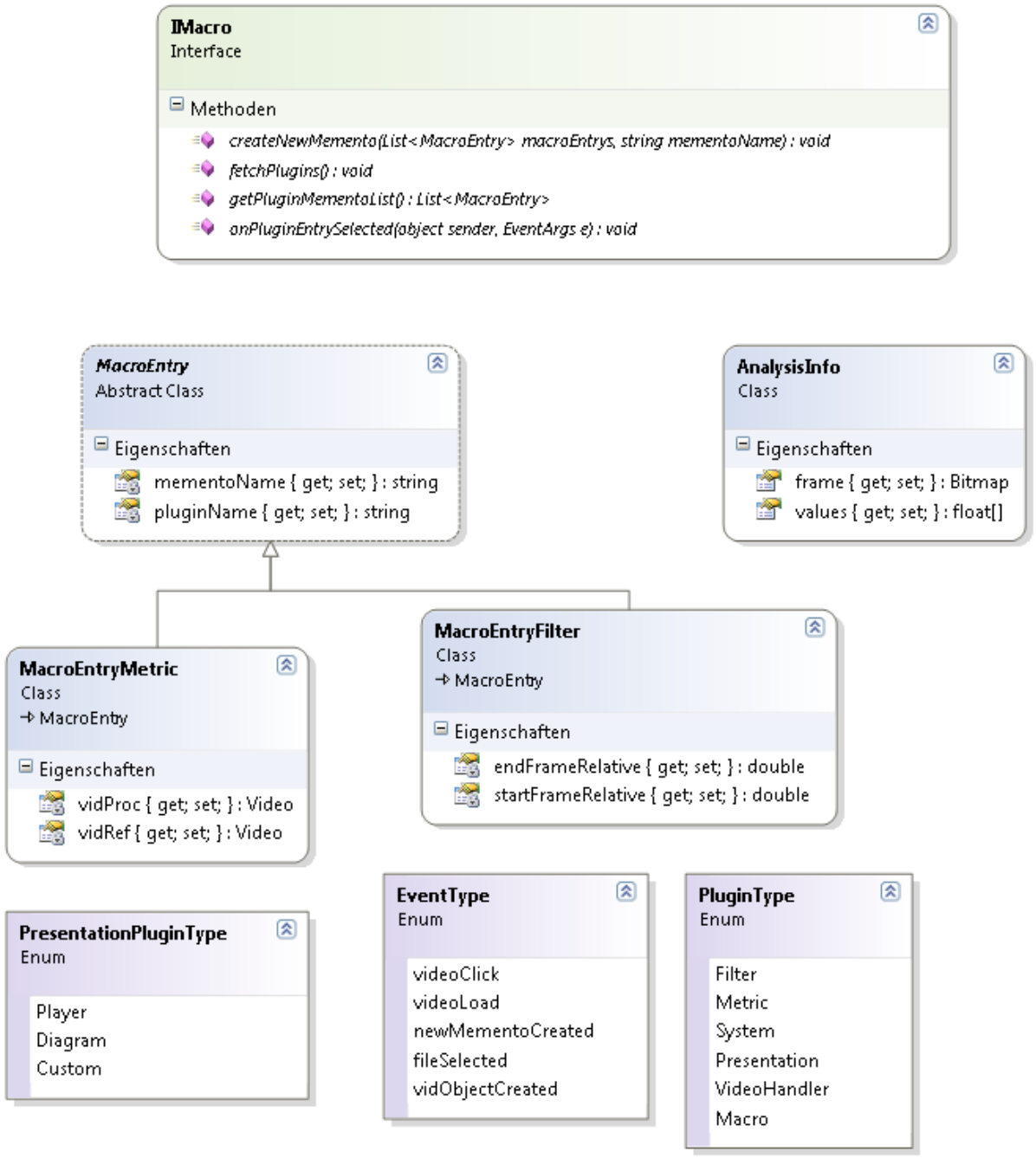


Abbildung 2.10: Public Plugins

IMacro : Interface

Implementiert das IPlugin Interface. Ein Objekt vom Typ Macro muss dieses Interface implementieren. Es verwaltet eine Liste von Objekten vom Typ MacroEntry und bietet eine Methode, die aus so einer Liste und einem beliebigen Namen ein Memento-Objekt erstellen kann.

MacroEntry : Klasse
Enthält zwei Stringattribute: eine Referenz zu einem Memento sowie eine Referenz zu einem Plugin.

MacroEntryFilter : Klasse
Erbt von MacroEntry. Enthält zwei Double-Attribute: startFrameRelative und endFrameRelative. Ein MacroFilter kann auf Teile von Videos, die eine verschiedene Anzahl Frames besitzen, angewandt werden. Daher ist es sinnvoll, die Start- und Endframes für den Filtervorgang nur relativ zur gesamten Anzahl Frames anzugeben.

MacroEntryMetric : Klasse
Enthält zwei Attribute vom Typ Video. Das sind die Videos, auf die eine Macrometric angewandt wird.

AnalysisInfo : Klasse
Eine Metrik schreibt ihre Ergebnisse zuerst in ein AnalysisInfo-Objekt, damit dann später in das neue Video die Daten eingetragen werden können.

PluginType : Enumeration
Der Enum enthält die Typen von Plugins.

PresentationPluginType : Enumeration
Der Enum enthält die Typen von Presentationplugins.

EventType : Enumeration
Der Enum enthält die Typen von Events.

3 Sequenzdiagramme

3.1 Initialisierung von OQAT

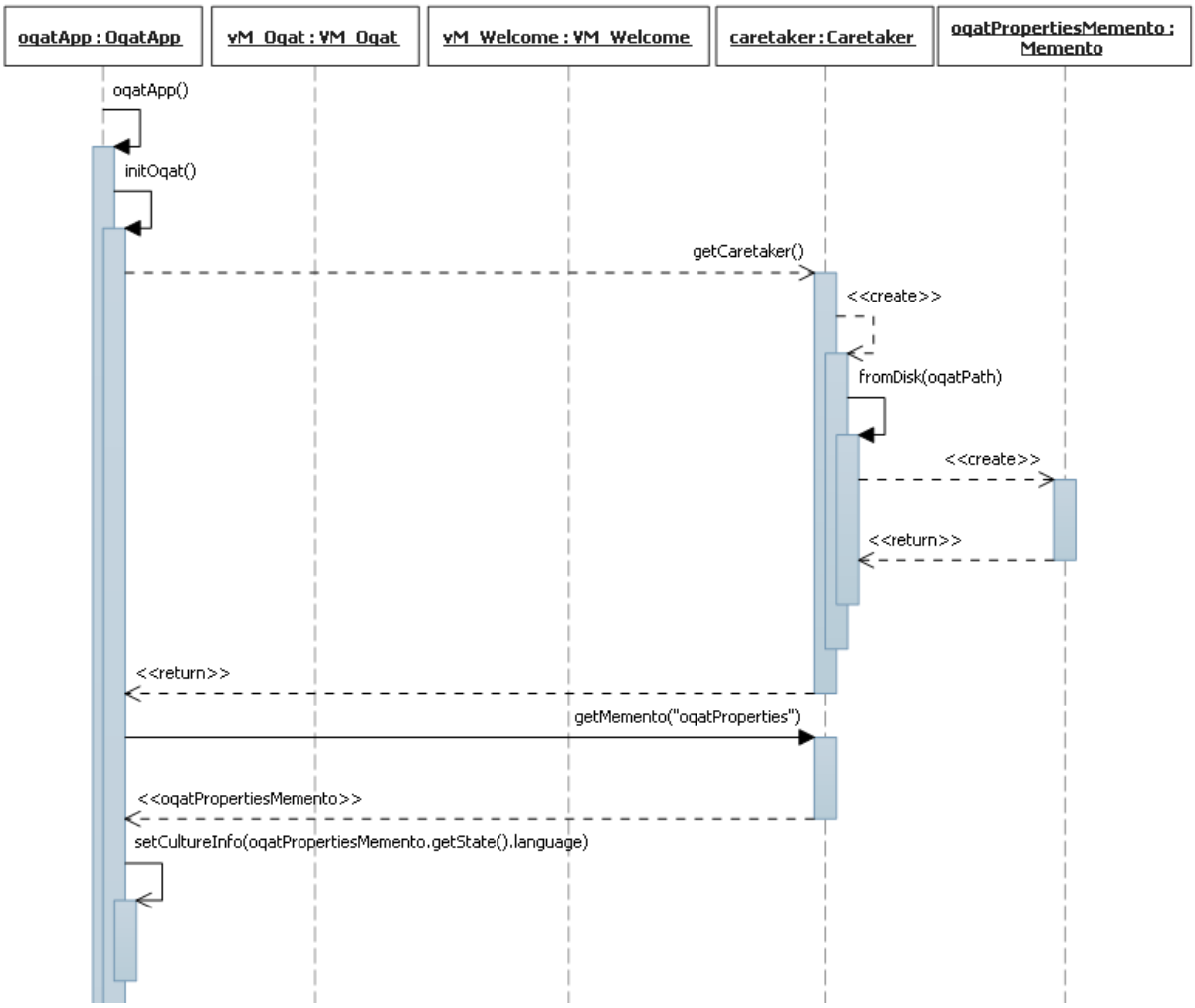


Abbildung 3.1: initOqat1

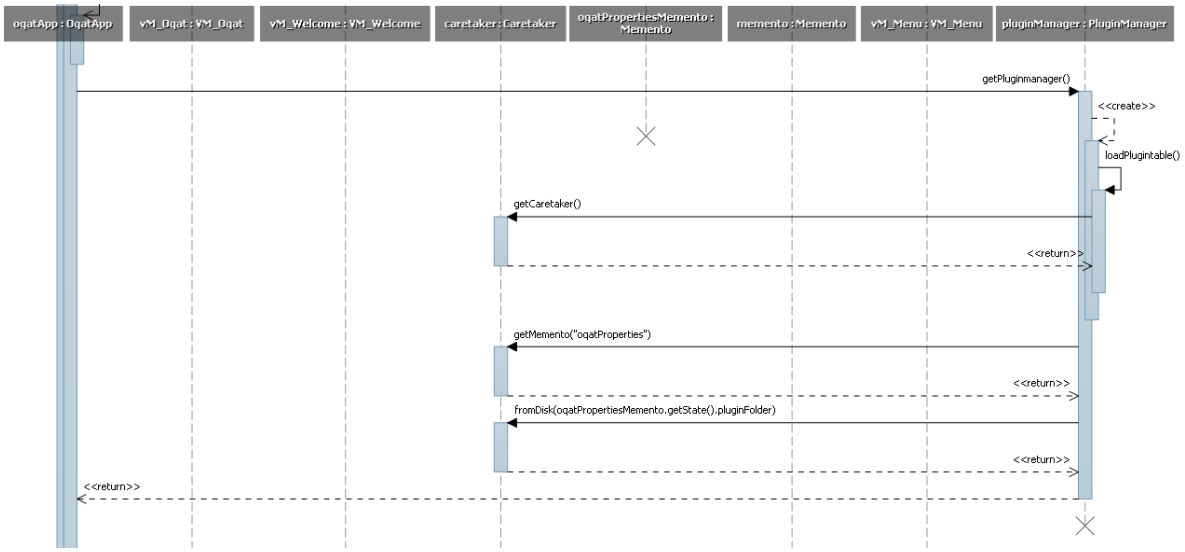


Abbildung 3.2: initOqat2

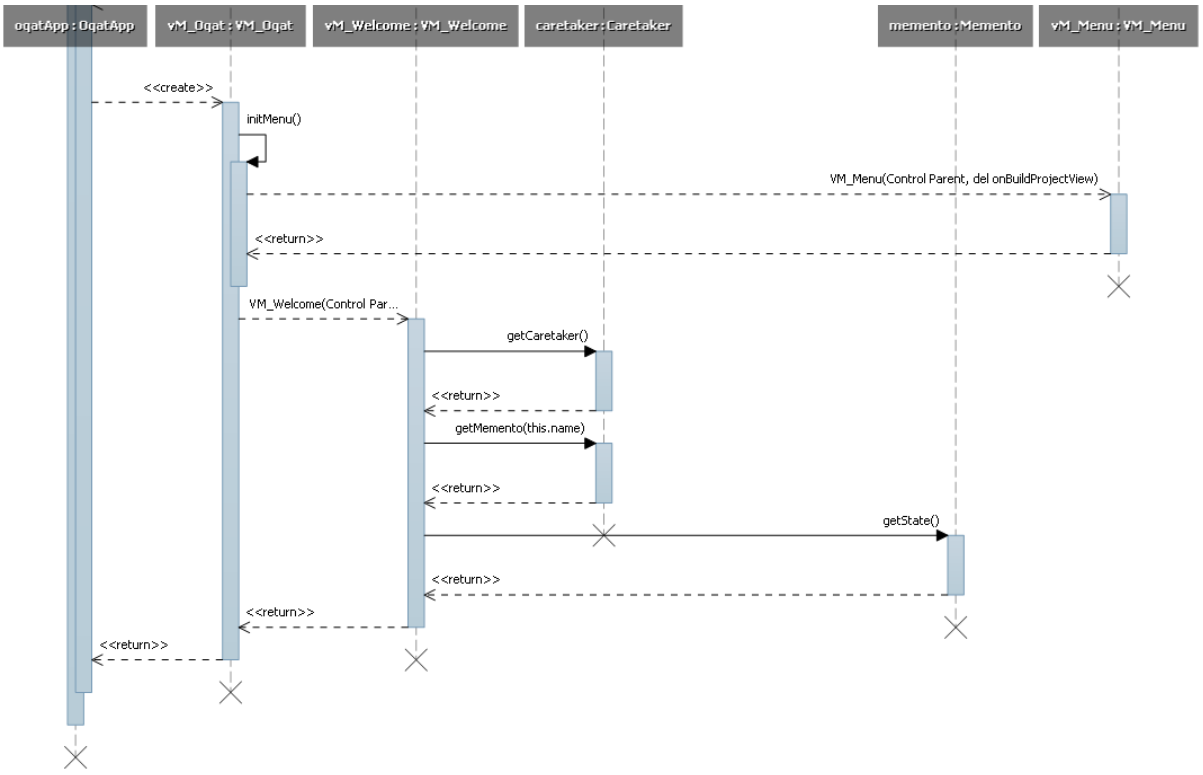


Abbildung 3.3: initOqat3name

3.2 Initialisierung eines Projekts

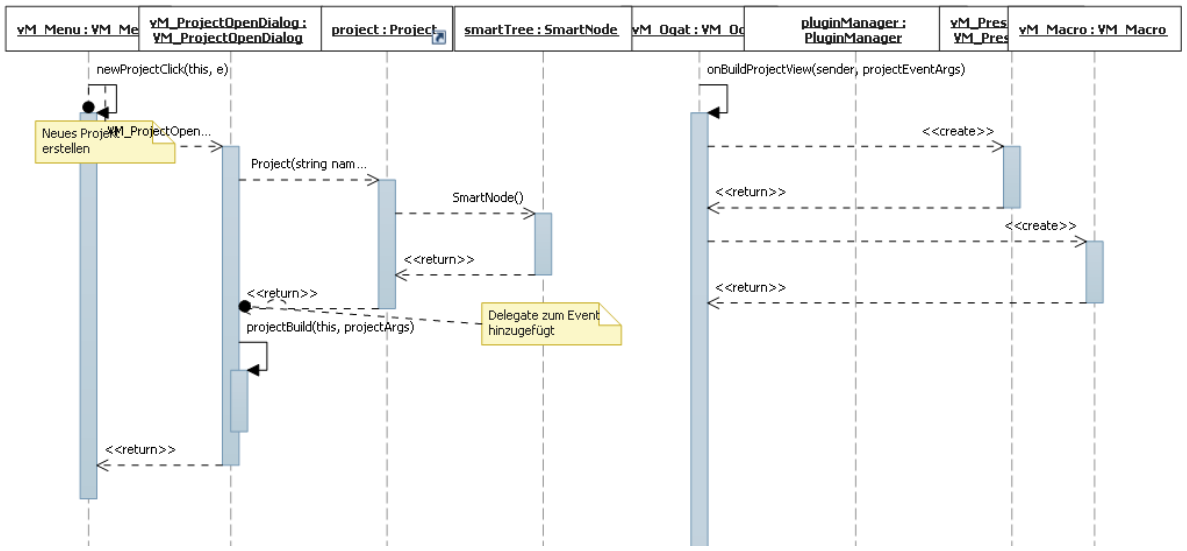


Abbildung 3.4: initProject1

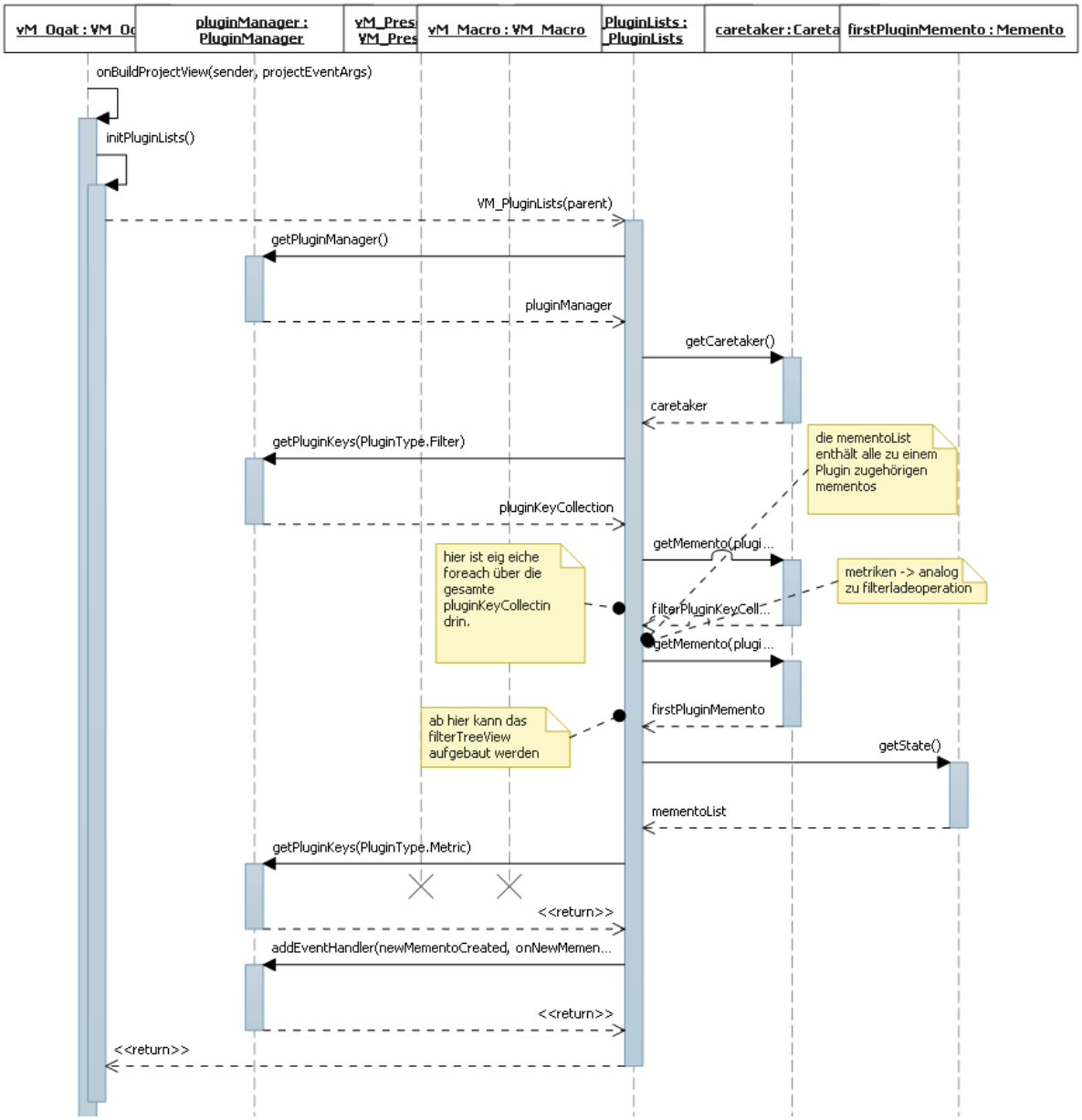


Abbildung 3.5: initProject2

3.3 Macro

3.3.1 Macro Filter

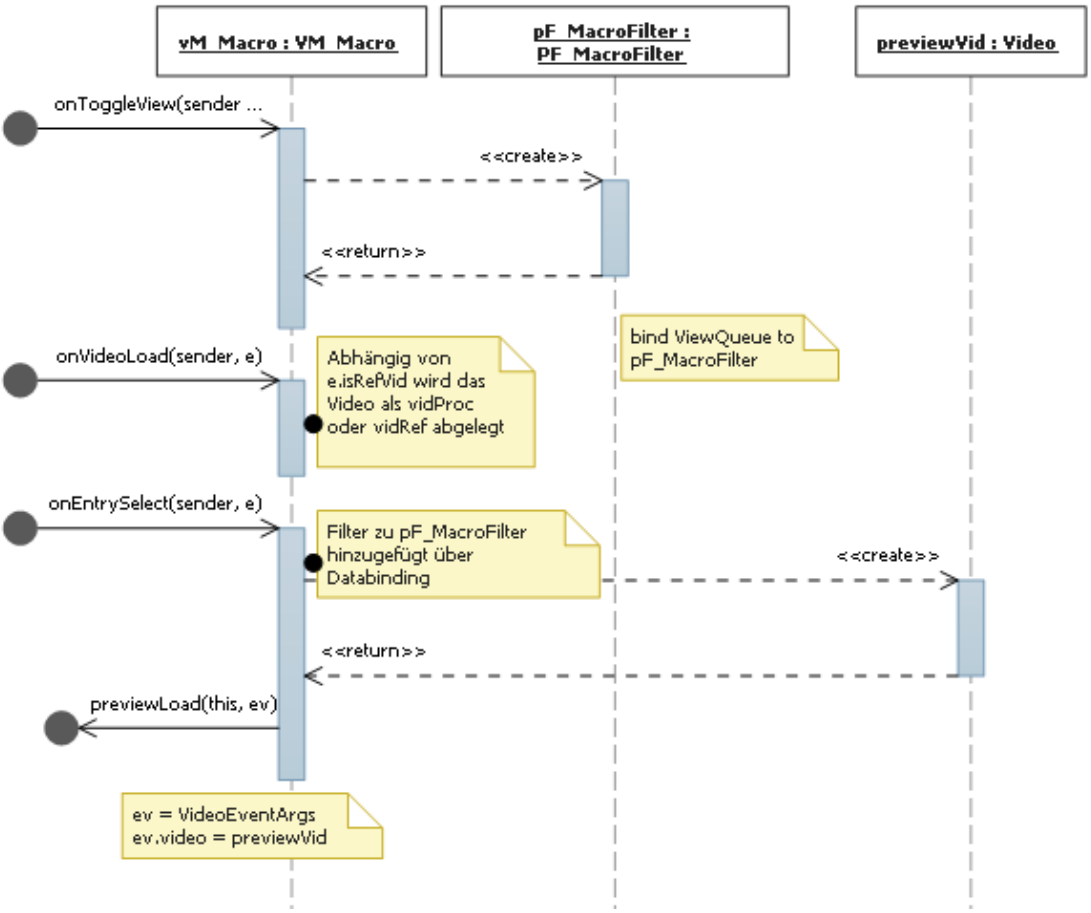


Abbildung 3.6: MacroFilter1

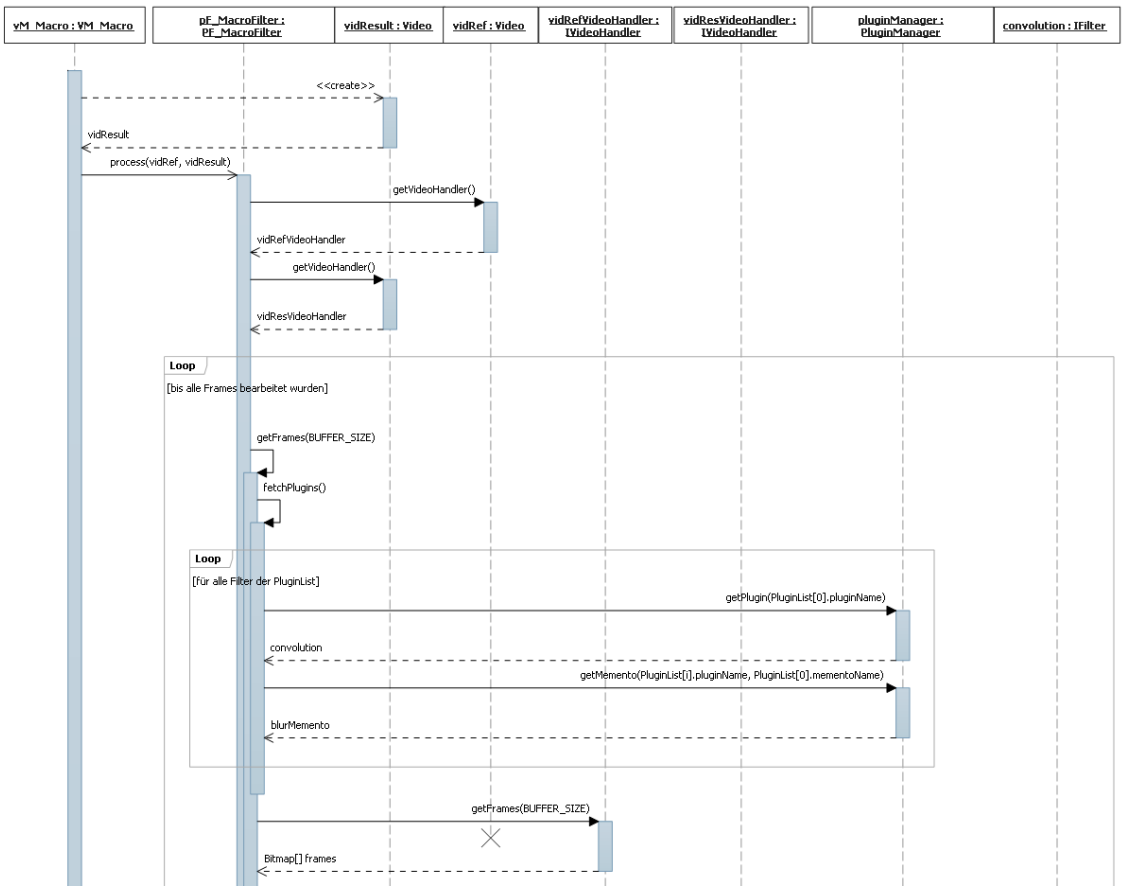


Abbildung 3.7: MacroFilter2

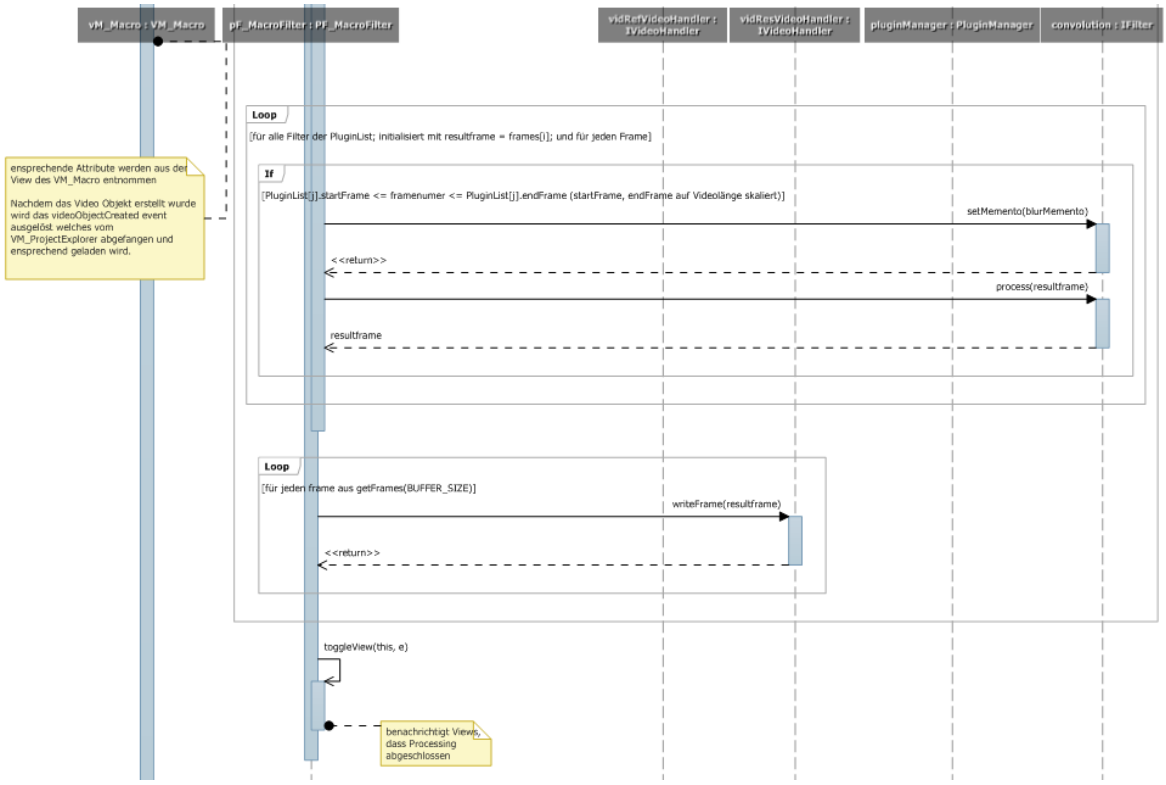


Abbildung 3.8: MacroFilter3

3.3.2 Macro Metric

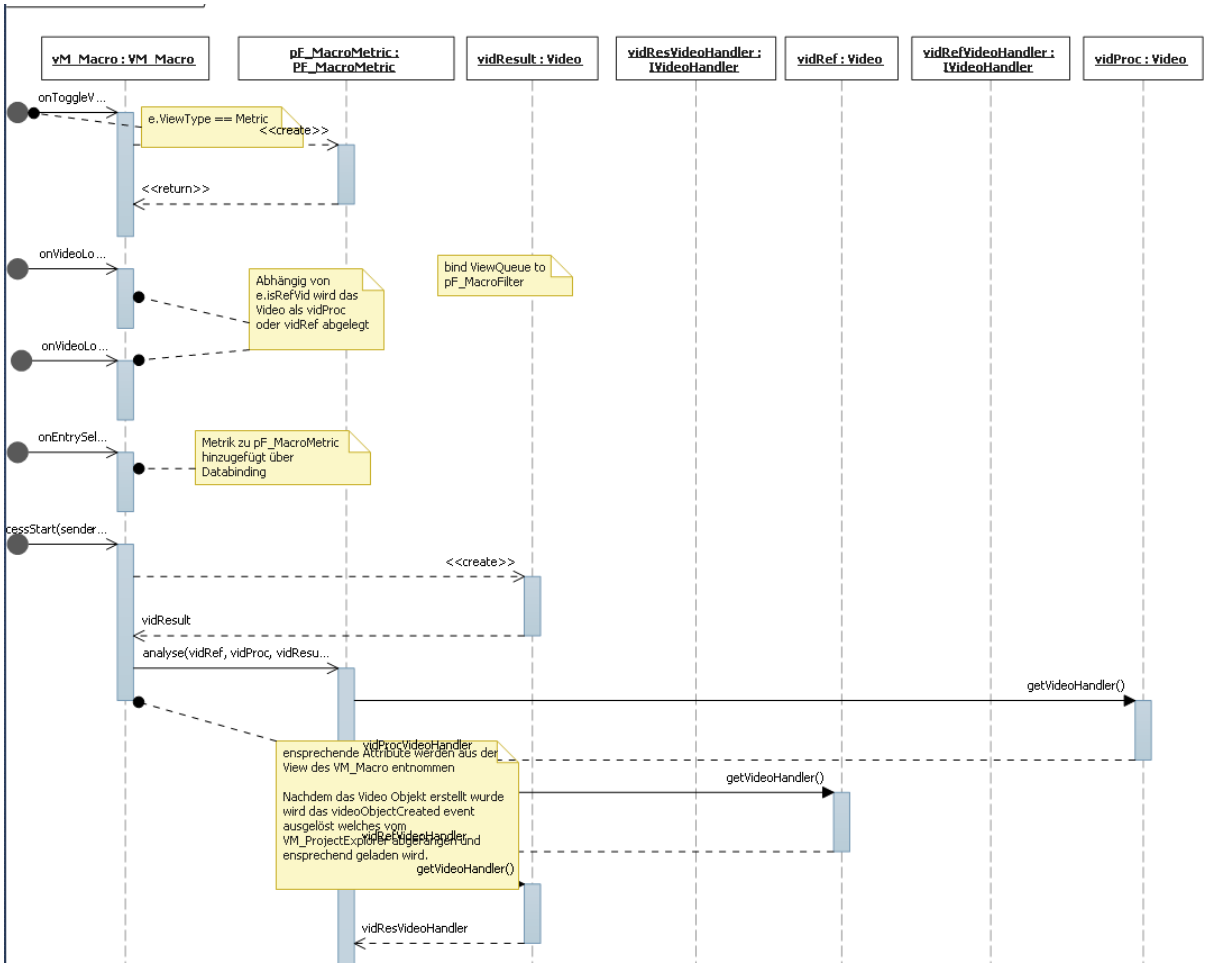


Abbildung 3.9: MacroMetric1

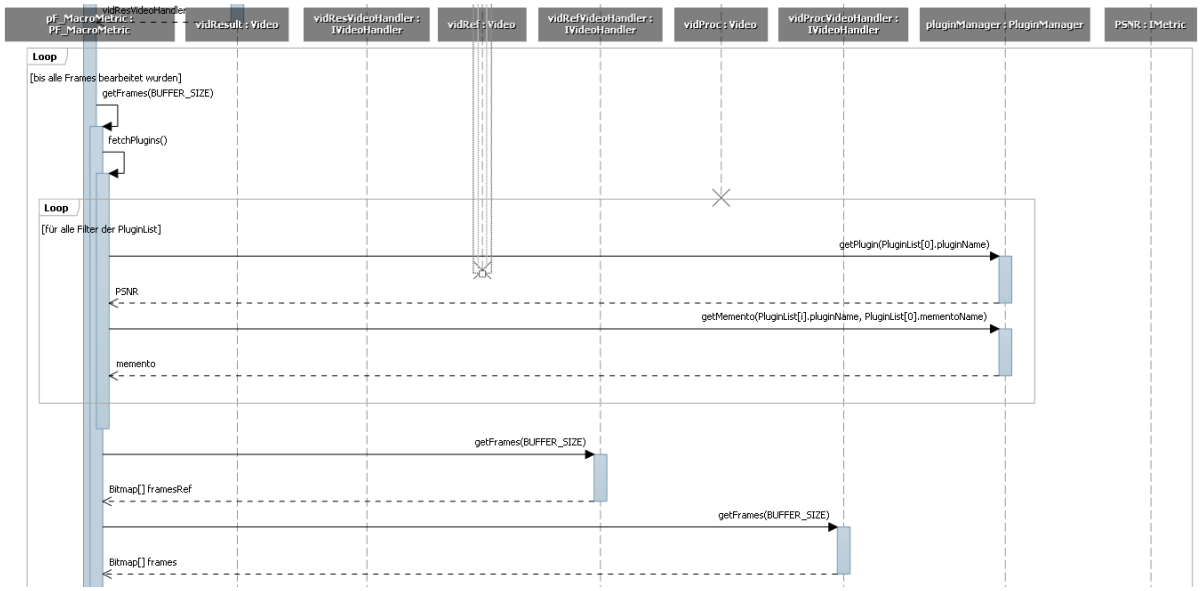


Abbildung 3.10: MacroMetric2

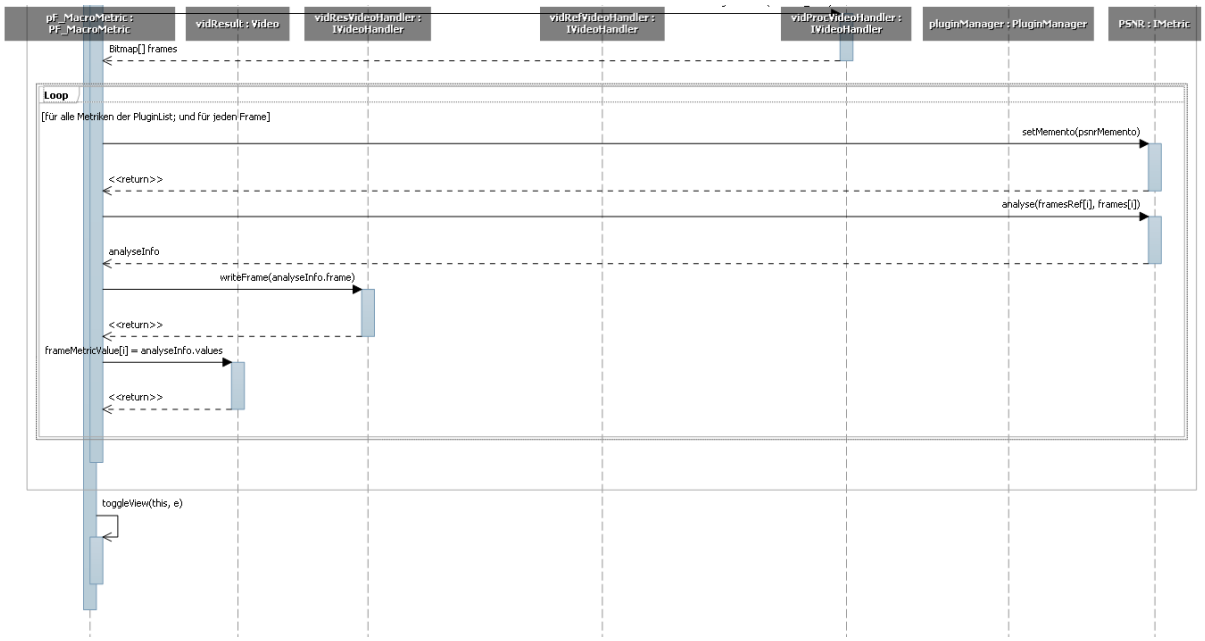


Abbildung 3.11: MacroMetric3

3.4 Video Load

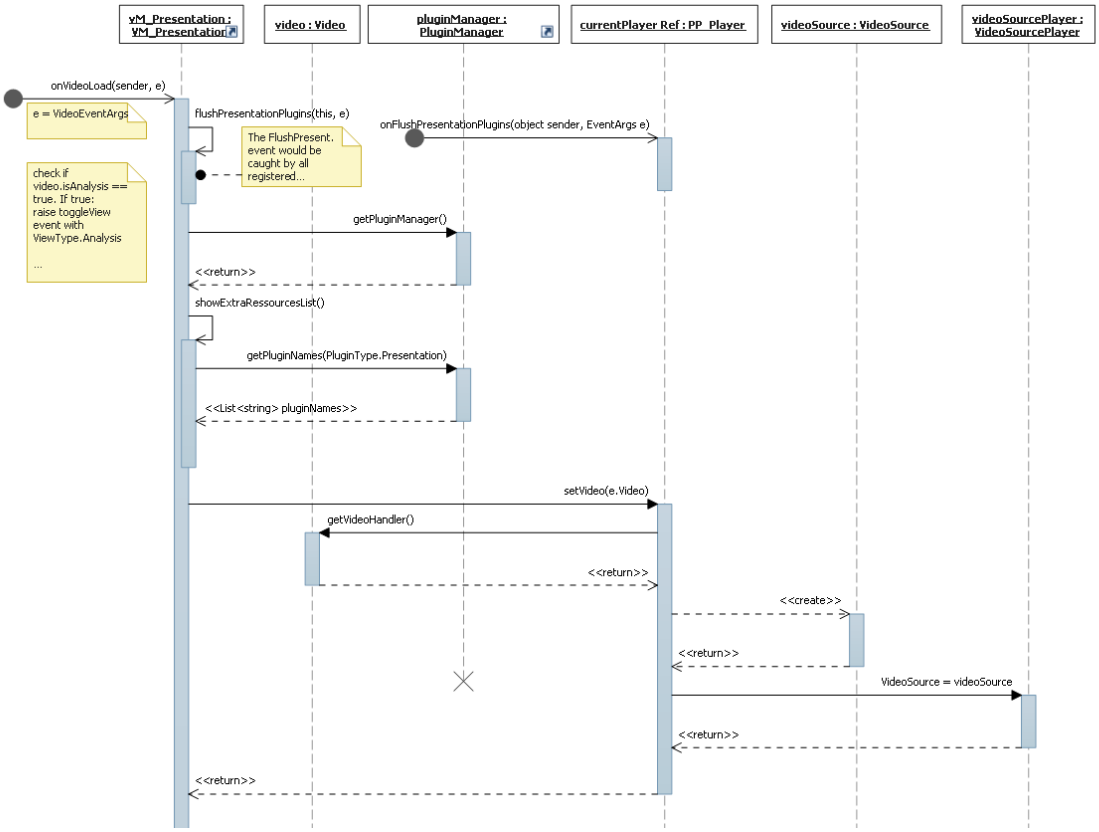


Abbildung 3.12: videoLoad1

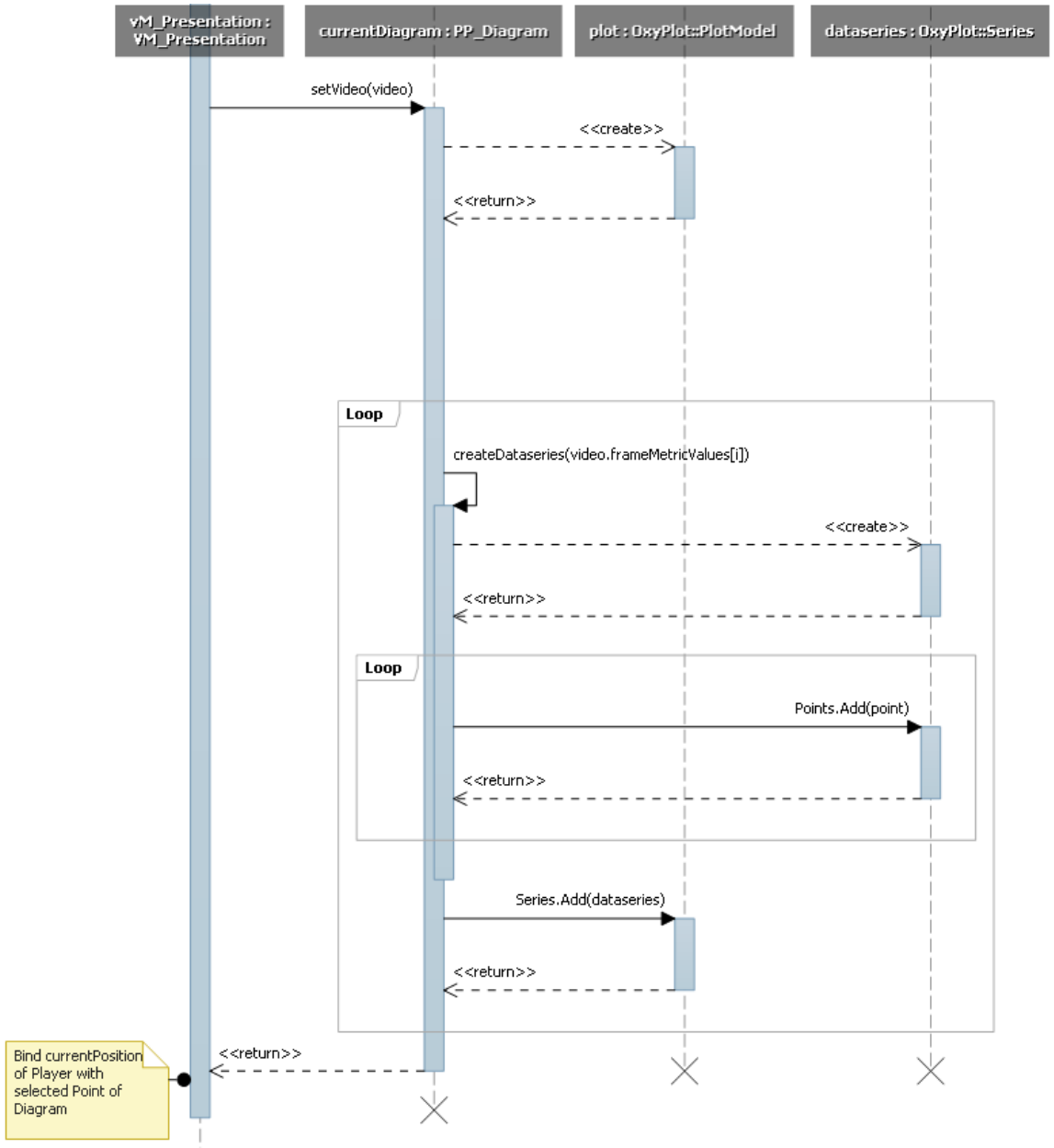


Abbildung 3.13: videoLoad2

3.5 Video Extra Ressource

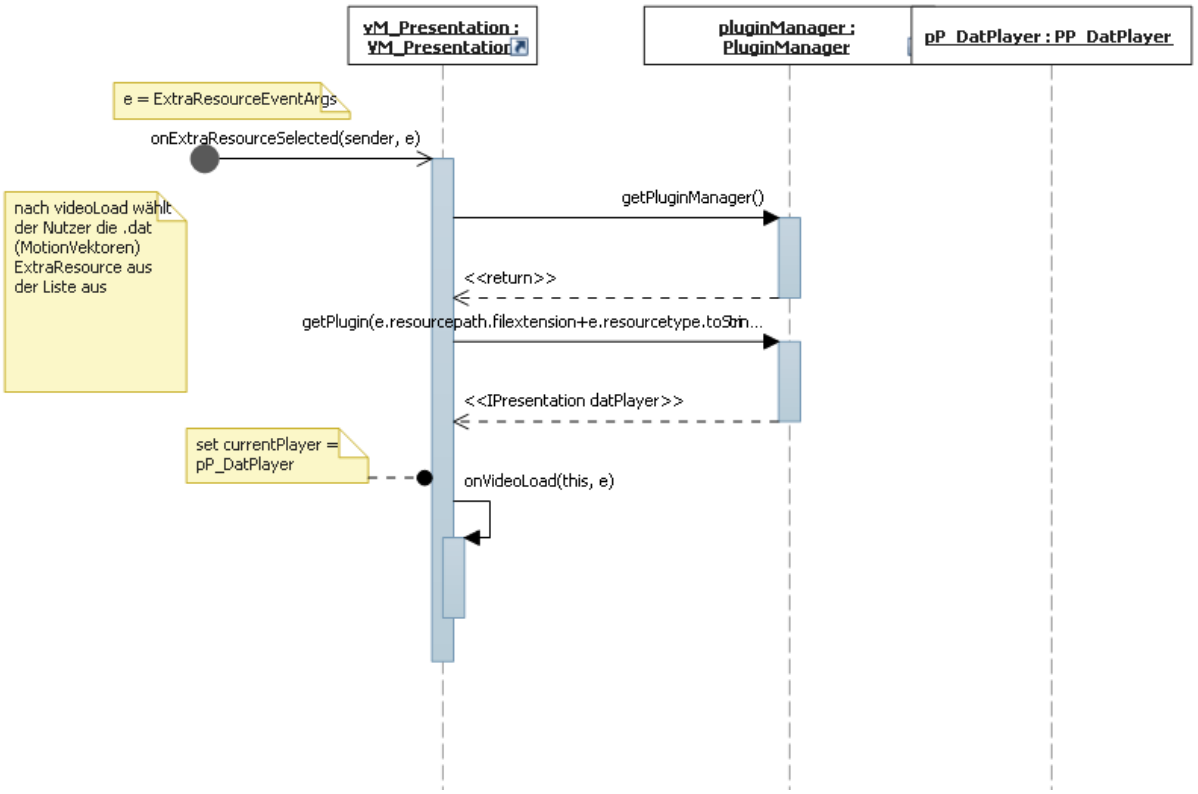


Abbildung 3.14: extraResourcen

3.6 vidImport

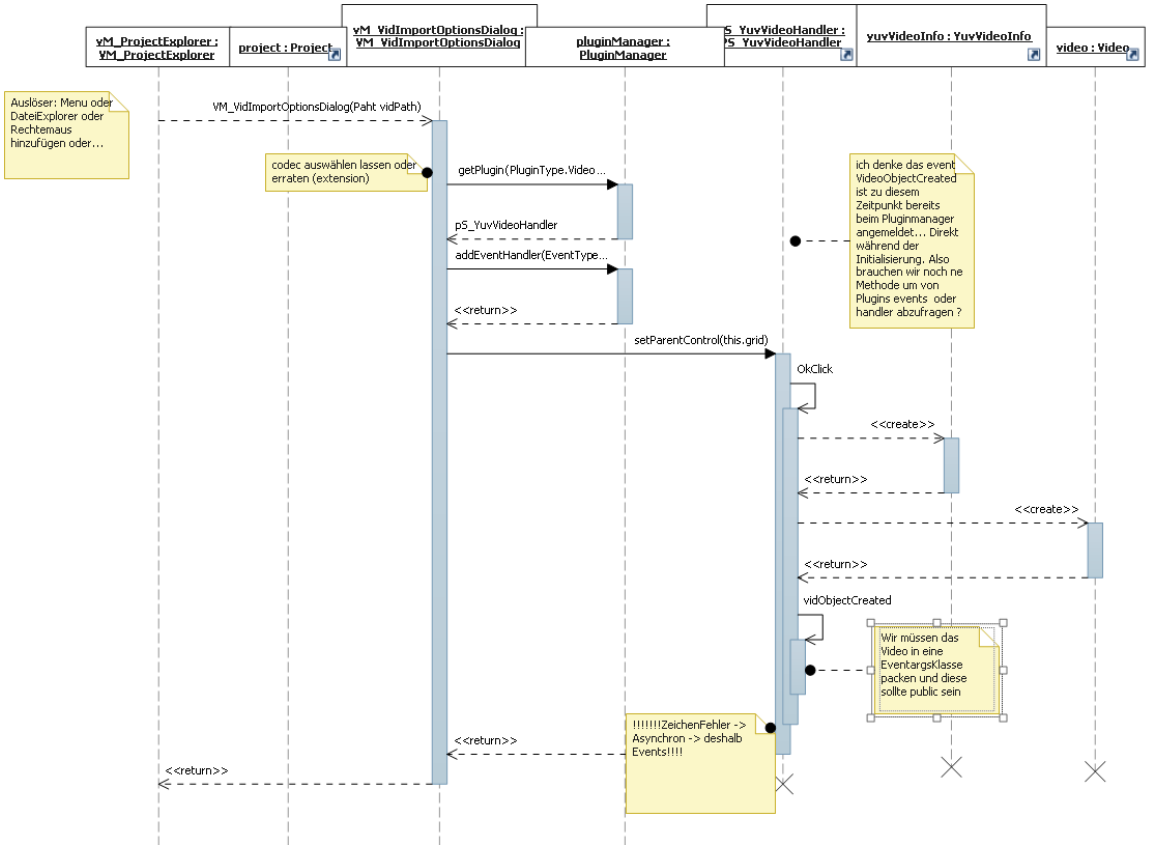


Abbildung 3.15: videoImport

Abbildungsverzeichnis