Requirements Analysis Document

Chair for Applied Software Engineering, TUM

Table of Contents

1 Introduction

Sections:

- Purpose of the system
- Scope of the system
- Objectives and success criteria of the project
- Definitions, acronyms, and abbreviations
- References

1.1. Purpose of the system

The goal of the "Virtual Symphony Orchestra" system is to increase the interest of children in classical music by giving them an understanding of the variety and beauty of music in a timely and playful manner. For this purpose the children slip into the role of an orchestra conductor. They can share in the exciting process of an orchestra performance to find out, that classical music can be a lot of fun.

The main goal of the project is to build a virtual symphony orchestra, which can be conducted by professional musicians and musical laymen alike. Digital audio and video recordings of a concert piece of the Bavarian Radio Symphony Orchestra are produced for this purpose. The recordings are prepared on a computer in such a way that they can be manipulated by the conductor, allowing him to control characteristic aspects of music like tempo and volume using different forms of interaction, such as gestures, facial expressions and movements of the baton.

1.2. Scope of the system

1.3. Objectives and success criteria of the project

Objectives

? provide an environment for increasing the intrest of children in classical music.

? provide a multimedia infrastructure for creating impressive audiovisual experiences.

? provide a learning environment for introducing children to musical instrument technology and harmonics.

? provide an infrastructure for enabling users to interact with the system by changing facial expressions, moving a conductor's baton and gesturing.

Success Criteria

1.4. Definitions, acronyms, and abbreviations

1.5. References

2 Proposed system

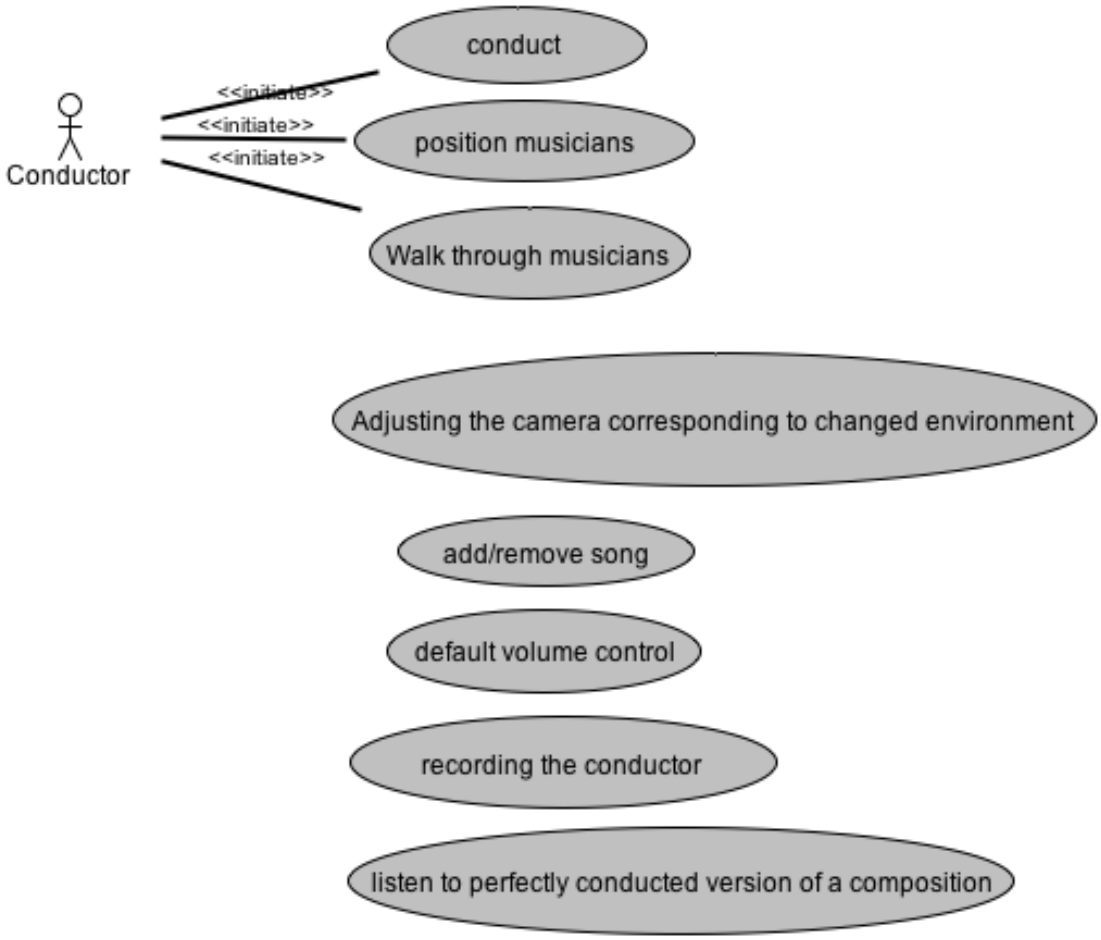
Sections:

- Overview
- Functional requirements
- Nonfunctional requirements
- System models

2.1. Overview

2.2. Functional requirements

Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts.



Sections:

- Actors
- User tasks

2.2.1. Actors

These are the different types of users the VSO system supports.

Actor: Administrator

The Administrator sets the audio and video configurations: add or removes songs,sets the bass, the volume, prepares the recording of the conductor while conducting etc.

Actorinstances	Heinz (Administrator)
Initiated UserTasks	add/remove song, default bass control, default volume control, recording the conductor

Initiated UseCases	Creating Default Settings, Start Program, Switching to AdministrationMode, Mute a part of the orchestra (AM), Moving a musician (AM), Changing the volume (AM), Changing the tempo (AM), Install the System

Open Issues	Description
What does "Preparing the recording of the conductor" mean?	

Actor: Conductor

The Conductor is the End User of the system. In general he can be anything from a mucial layman up to a professional conductor.

He conducts virtual orchestra, removes instruments, posistions musicians and moves through the virtual room

Super-Actor	ExhibitionVisitor
Actorinstances	Wolle (Conductor)
Initiated UserTasks	position musicians, conduct, Walk through musicians
Initiated UseCases	Conducting, Start Conducting, Stop Conducting, Prepare Conducting, Changing the tempo (CM), Changing the volume (CM), Changing the tempo of a part of the orchestra (CM), Changing the volume of a part of the orchestra

Actor: ExhibitionVisitor

Sub-Actor	Conductor, Listener
Initiated UseCases	Browsing the menu

Actor: Listener

A music listener.

Super-Actor	ExhibitionVisitor
Actorinstances	Horatio (Listener)

Initiated UseCases	Moving a musician (CM), Walking through the orchestra, Selecting a part of the orchestra (LM)

Actor: User Input Evaluation System

Actorinstances	Referee

Open Issues	Description
Find a better name for this Actor	Actors and use cases are used to model the interaction between the users of a system and the system itself. Therefore, System is a very bad name for an actor. In some cases, we use an actor to model independent behavior of a system. The name must be very clear for this case.

2.2.2. User tasks

User Task: add/remove song

The Administrator adds new audio and video material, so the conductor can choose between different songs

Initiating Actor	Administrator

User Task: Adjusting the camera corresponding to changed environment

User Task: conduct

The conductor conducts the musicians. He can put a focus on certain musicians or groups of musicians while conducting. He changes speed, volume of the focussed musicians.

Initiating Actor	Conductor

User Task: default bass control

The Administrator sets a default bass control, that the conductor can always modify.

Initiating Actor	Administrator

User Task: default volume control

The Administrator sets a default volume control, that the conducter can always modify.

Initiating Actor	Administrator

User Task: listen to perfectly conducted version of a composition

The conductor listens to an interpreted version in order to get a beter understanding how the composition should sound

User Task: position musicians

The conductor positions the musicians in the virtual room in the way he wants, in order to listen to the music. He can also remove musicians.

Initiating Actor	Conductor

User Task: recording the conductor

The Administrator records the conductor while he is conducting the musicians.

Initiating Actor	Administrator

User Task: Walk through musicians

The conductor walks through the virtual room while the musicans are playing.

Video and Sound are changing according to his position.

He controls the walk via gestures.

Initiating Actor	Conductor

Open Issues	Description
Walking through the orchestra: is it possible?	To walk through the orchestra, we need - 3d-models of each musician and - 3d-model of the concert-hall But we can not render the musician-models from different points of view using standard video cameras. So what?

2.3. Nonfunctional requirements

Nonfunctional requirements describe aspects of the system that are not directly related to its functional behavior.

Sections:

- Usability
- Reliability
- Performance
- Supportability
- Implementation
- Interface
- Operation
- Packaging
- Legal

2.3.1. Usability

Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of the system or one of its components.

Nonfunctional Requirement: Ease of Use

Every user older than 7 years of age should be able to conduct the orchestra.

Constrained Issues	Description
How great is the latency between the conducting gesture of the Conductor and the reaction of the virtual orchestra?	
How to realize the training scenario?	
How many different conducting gestures should be available in the system?	This issue affects conducting gestures representing measures: How many differen measures should be recognized by the system? (2/4,3/4,4/4,3/8,6/8.....)

Constrained Elements	Description
Browsing the menu	The Conductor wants to choose one of the several menus or wants to interact with an user interface component

Constrained Elements	Description
Changing the tempo (CM)	The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.
Changing the volume (CM)	The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish
Conducting	
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Prepare Conducting	
Start Conducting	
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.
Stop Conducting	
Selecting a part of the orchestra (AM)	

Nonfunctional Requirement: FullScreen conductor interface

The Conductors interface should run in fullscreen mode.

Constrained Elements	Description
Browsing the menu	The Conductor wants to choose one of the several menus or wants to interact with an user interface component
Changing the tempo (CM)	The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.
Changing the volume (CM)	The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish
Conducting	
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Prepare Conducting	
Start Conducting	
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.

Constrained Elements	Description
Start Program	
Stop Conducting	
Selecting a part of the orchestra (AM)	

Nonfunctional Requirement: Interaction Forms

The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Constrained Elements	Description
Browsing the menu	The Conductor wants to choose one of the several menus or wants to interact with an user interface component
Changing the tempo (CM)	The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.
Changing the volume (CM)	The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish
Conducting	
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.
Start Program	
Stop Conducting	
Selecting a part of the orchestra (AM)	

Nonfunctional Requirement: Short Learning Curve

The learning curve for assimilating the baton's movements, gestures and facial expressions supported by the tracking system should be smooth and short.

Nonfunctional Requirement: Windowed administrator interface

The Adminstrators Interface should be a normal window.

Constrained Elements	Description
Creating Default Settings	
Mute a part of the orchestra (AM)	The Administrator can mute a Musician
Switching to AdministrationMode	
Start Program	
Moving a musician (AM)	The Administrator should not need to move a musician by gestures with the baton. Instead he should be able to use the keyboard to achieve much higher accuracy
Changing the volume (AM)	Changing the volume in AdministrationMode

2.3.2. Reliability

Reliability is the ability of the system or a component to perform its required functions under stated conditions for a specified period of time.

Nonfunctional Requirement: Prevention of Audio Dropouts

High processor load conditions should never lead to severe audio dropouts.

Nonfunctional Requirement: Stable Application

The Admin Interface and the Conductor Interface should never quit without an explicit quit command.

Constrained Elements	Description
Browsing the menu	The Conductor wants to choose one of the several menus or wants to interact with an user interface component
Changing the tempo (CM)	The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.
Changing the volume (CM)	The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish
Conducting	
Creating Default Settings	
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Mute a part of the orchestra (AM)	The Administrator can mute a Musician
Prepare Conducting	
Start Conducting	

Constrained Elements	Description
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.
Switching to AdministrationMode	
Start Program	
Stop Conducting	
Moving a musician (AM)	The Administrator should not need to move a musician by gestures with the baton. Instead he should be able to use the keyboard to achieve much higher accuracy
Selecting a part of the orchestra (AM)	
Changing the volume (AM)	Changing the volume in AdministrationMode

Nonfunctional Requirement: State of the Art Audio/Video Synchronization

The audio video synchronization should be accurate when changing the music tempo.

2.3.3. Performance

Performance requirements are concernd with quantifiable attributes of the system, such as response time. availability, accurcy, etc.

Nonfunctional Requirement: Good Response Time

The Response between an action and a noticable reaction should be less than 100 ms

Constrained Elements	Description
Browsing the menu	The Conductor wants to choose one of the several menus or wants to interact with an user interface component
Changing the tempo (CM)	The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.
Changing the volume (CM)	The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish
Conducting	
Creating Default Settings	
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Mute a part of the orchestra (AM)	The Administrator can mute a Musician

Constrained Elements	Description
Prepare Conducting	
Start Conducting	
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.
Switching to AdministrationMode	
Start Program	
Stop Conducting	
Moving a musician (AM)	The Administrator should not need to move a musician by gestures with the baton. Instead he should be able to use the keyboard to achieve much higher accuracy
Install the System	
Selecting a part of the orchestra (AM)	
Changing the volume (AM)	Changing the volume in AdministrationMode

Comments	Description
A static latency does not match the different skills of the Users	A professional conductor who wants to use our system wants a lower latency, whereas a novice user might profit from a higher latency.

Nonfunctional Requirement: High accuracy

95% of the different gestures should be recognized correctly

Constrained Elements	Description
Browsing the menu	The Conductor wants to choose one of the several menus or wants to interact with an user interface component
Changing the tempo (CM)	The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.
Changing the volume (CM)	The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish
Conducting	
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Start Conducting	

Constrained Elements	Description
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.
Stop Conducting	
Selecting a part of the orchestra (AM)	

Open Issues	Description
How many different conducting gestures should be available in the system?	This issue affects conducting gestures representing measures: How many differen measures should be recognized by the system? (2/4,3/4,4/4,3/8,6/8.....)

Nonfunctional Requirement: Scalability

The VSO system should support an orchestra of up to 80 musicians

Nonfunctional Requirement: Short startup time

The startup time should be less than 1 minute.

Constrained Elements	Description
Creating Default Settings	
Prepare Conducting	
Start Conducting	
Switching to AdministrationMode	
Start Program	

2.3.4. Supportability

Supportability requirements are concerned with the ease of changes to the system after deployment, including for example adaptability and maintainability.

Nonfunctional Requirement: Support of new media

Every media should be replaceable.

Constrained Elements	Description
Install the System	
Changing the volume (AM)	Changing the volume in AdministrationMode

2.3.5. Implementation

Implementation requirements are constraints on the implementation of the system.

Nonfunctional Requirement: Implementation languages

The Implementation languages should be Objective-C, C and C++ as they can be mixed at will.

Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.

Constrained Elements	Description
Moving a musician (CM)	The Conductor can move a musicianto the place where he wants him to be
Mute a part of the orchestra (AM)	The Administrator can mute a Musician
Prepare Conducting	
Start Conducting	
Verifying ConductingGestures	While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.
Switching to AdministrationMode	
Start Program	
Stop Conducting	
Moving a musician (AM)	The Administrator should not need to move a musician by gestures with the baton. Instead he should be able to use the keyboard to achieve much higher accuracy
Install the System	
Selecting a part of the orchestra (AM)	
Changing the volume (AM)	Changing the volume in AdministrationMode

Nonfunctional Requirement: Target Environment

The VSO system runs on MacOS X.

Open Issues	Description
Should we specify Mac OS X 10.4 or later as target environmet?	CoreVideo and CoreAudio are components available since 10.4.x and better.

2.3.6. Interface

Interface requirements are constraints imposed by external systems, including legacy systems and interchange formats.

Nonfunctional Requirement: Audio Surround System Requirement

The VSO system requires an audio surround system to work properly with full functionality.

2.3.7. Operation

Operations requirements are constraints on the administration and management of the system in the operational setting.

Nonfunctional Requirement: System Administration

For the VSO Museum version an administrator team is needed for install and maintenance.

2.3.8. Packaging

Packaging requirements are constraints on the actual delivery of the system.

Nonfunctional Requirement: One DMG file includes the whole application

The application should be installable by dragging it to the Application folder.

Constrained Elements	Description
Install the System	

2.3.9. Legal

Legal requirements are concerned with licensing, regulation, and certification issues.

Nonfunctional Requirement: No legal restrictions for the used media

There should be no payment fee or restriction for the media we use.

Constrained Elements	Description
Creating Default Settings	
Prepare Conducting	
Start Conducting	
Start Program	
Install the System	

2.4. System models

Sections:

- Scenarios
- Use case model
- Object model

2.4.1. Scenarios

In this section, single features of the VSO system will be described from the viewpoint of a single actor in a concrete, focused and informal way.

Sections:

- Actor Instances
- Scenarios

2.4.1.1. Actor Instances

Actor Instance: Heinz (Administrator)

Heinz ist Administrator des VSO Systems und spielt regelmäßig Updates und Patches ein

Instanciated Actor	Administrator
Initiated Scenarios	Add Mozart's Magic Flute to song base, Conductor creates a recorded session

Actor Instance: Horatio (Listener)

A fan of classic music.

Instanciated Actor	Listener
Initiated Scenarios	Home Orchestra Mode
Participating Scenarios	Conductor walks through orchestra

Open Issues	**Description**
Should Wolle conduct while walking through the orchestra?	How shall it be possible to conduct the orchestra while walking through it? How could Wolle control the walking direction if his hands and gestures are already in use for conducting?

Actor Instance: Referee

Instanciated Actor	User Input Evaluation System
Initiated Scenarios	Referee gives feedback.
Participating Scenarios	Conductor changes focus, Conductor initiates conducting session, Conductor aborts a conducting session., Conductor changes tempo

Actor Instance: Wolle (Conductor)

Wolle Petri, der grösste Schlagerstar aller Zeiten!

Instanciated Actor	Conductor
Initiated Scenarios	Conductor walks through orchestra, Configure Settings for Beethoven's ninth symphony, Conductor changes volume, Conductor changes tempo, Get song info for Beethoven's ninth symphony, Training, Conductor changes focus, Conductor initiates conducting session, Conductor aborts a conducting session.
Participating Scenarios	Conductor creates a recorded session, Referee gives feedback.

2.4.1.2. Scenarios

Scenario: Add Mozart's Magic Flute to song base

1. Heinz receives a dvd with new audio and video material of Mozart's Magic Flute. He logs into the administrator section of the vso-program.

2. Heinz the activates the update multimedia function. There he selects add song and dvd as the update source.

3. He inserts the dvd and waits until the files are copied and then sees the default position of the musicians and default audio settings. He confirms the default settings and selects finish update.

4. Heinz gets a confirmation that the update was successful.

Scenario: Conductor aborts a conducting session.

1.Wolle feels unhappy about his conducting and he wants to abort the session.

2.Wolle stops making a new measure gestures.

3.Referee recognizes the lacking of conducting acticities.

4.Referee pauses the session and asks for further procedure.

5.Wolle choose the stopping procedure.

Scenario: Conductor changes focus

1.Wolle wants to refocus his acticities on a part of the orchesta.

2.Conductor changes his focus on the orchesta.

3.Referee recognises the new focus of Wolle.

4.Wolle continues his conducting on the new focuses orchesta part.

5.Referee recognises Wolle's acitivities and applies them on the focused orchesta part.

Scenario: Conductor changes tempo

1. While conducting Wolle gets the feeling the song sounds too fast/slow. Thus he slows down/speeds up the movement of the conducting baton.

2. The referee decreases/increases the tempo and Wolle likes it.

Scenario: Conductor changes volume

1. While conducting Wolle realizes that the song sounds too gentle/loud. He starts making bigger/smaller gestures.

2. The system increases/decreases volume and Wolle is happy with the sound.

Scenario: Conductor creates a recorded session

1. After a long practise session Wolle is very proud of his conducting skills and informs Heinz that he wants to record his version of the song.

2. Heinz logs into the admin section and starts the record funcion. Heinz presses start and the system confirms that record is started.

3. Wolle conducts until the song is finished.

4. Heinz stops recording. The system confirms that the recording session was successful.

5. Heinz burns a cd with the recorded files for Wolle.

Open Issues	Description
Is an intervention by Heinz necessary?	

Scenario: Conductor initiates conducting session

1.Wolle feels ready and wants to start a new conducting session (Training/Live).

2.Wolle initiates the session type and if needed he chooses a difficulty level.

3.Referee creates the requested session and gives Ready-feedback to Wolle.

4.Wolle conducts the fist measure.

5.Wolle conducts the second measure.

6.Referee recognises the second measure and initiates the orchestra to play.

7.Referee gives feedback to Wolle concerning his activities.

Comments	Description
Session types	Session types: 1.Live session: No feedback during conducting, only a review after the session 2.Training session: During this session the referee gives feedback based on the analysed conducting activities. In this session you can choose different training levels.

Scenario: Conductor walks through orchestra

1. Wolle decides to walk around the orchestra to get a better impression how the different instruments sound. He starts making gestures while conducting.

2. Wolle first visits the first violine for a while. Then he walks to the back and listens to the drummers. And in the end he has a closer look at cello.

Open Issues	Description
Should Wolle conduct while walking through the orchestra?	How shall it be possible to conduct the orchestra while walking through it? How could Wolle control the walking direction if his hands and gestures are already in use for conducting?

Scenario: Configure Settings for Beethoven's ninth symphony

1. Wolle chooses Beethovens's ninth symphony frome the list of available songs and selects the quartet version.

2. Wolle now positions the musicians in the room. He does to focus on the first violin and thus places the first violinist very close and moves the others back. He even mutes the second violin in order to be not distracted from the first violin.

3. Because Wolle is used to loud music he turns up the volume and bass level and then confirms his

stettings.

Scenario: Get song info for Beethoven's ninth symphony

1. Wolle wants to conduct Beethovens's ninth symphony, but he does not know how the song should sound. Thus he activates the "listen to perfectly version" function of the song.

2. After listening to the "perfect version" Wolle decides to have a closer look at the orchetra score. He activates "show orchetra score" and studies it.

3. After a while he is happy with it and quits the song info section.

Scenario: Home Orchestra Mode

The system is used as a sound system to playback an selected orchstra in normal speed and selected volume without user interaction.

The user starts the application and selects "Home Orchestra Mode". He's now presented a list of available songs and presets.

He chooses a setting and sits back on his sofa and enjoys his favourite orchestra playing songs while watching the musicians in a virtual concert hall.

Scenario: Referee gives feedback.

1.Referee recognizes each conducting activity of Wolle.

2.Referee analyzes each conducting activity of Wolle

3.Referee decides the feedback type (Tempo-, Volume-, Measure-Feedback).

4.Referee gives feedback

5.Wolle gets feedback

Comments	Description
Description too abstract	Scenarios should be concrete, focused descriptions of a single feature of the system from the viewpoint of a single actor.

Scenario: Training

Wolle wants to train his skills in conducting. He starts VSO and selects the traning mode. After selecting a difficulty level the orchestra plays at a given speed and volume.

The speed and volume of the orchestra changes during playback and the conductor has to adapt the speed. In lower diffcult levels an instant feedback is presented to Wolle, to show him the speed and volume he is conducting.

After the song is finished or if Wolle wants to quit, an information panel is displayed, presenting information about accuracy and timing of the conductor.

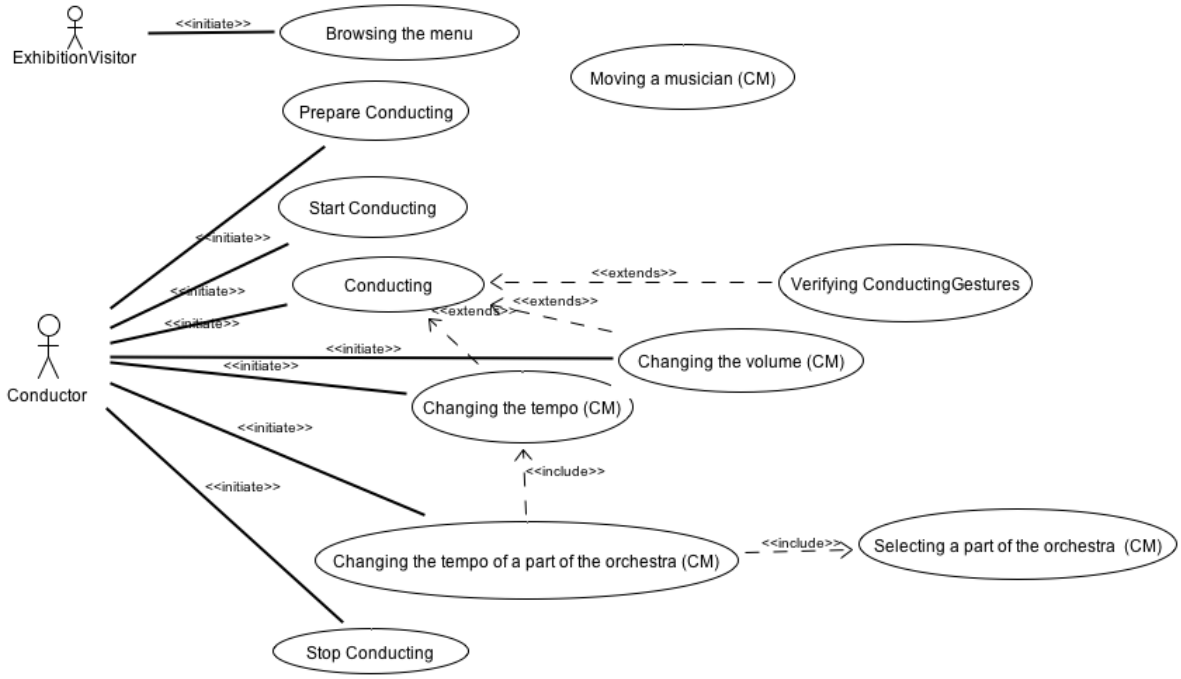
Open Issues	Description
How to realize the training scenario?	

2.4.2. Use case model

Sections:

- Conducting Use Cases
- Administrative Use Cases
- Listening Use Cases

2.4.2.1. Conducting Use Cases



Use Case: Browsing the menu

Type	Conducting

The Conductor wants to choose one of the several menus or wants to interact with an user interface component

Initiating Actor	ExhibitionVisitor

Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Conductor indicates his wish to leave the conducting mode and to interact with one of the GUI elements.	
		The System recognizes the Conductors intention and shows the menu items.
	The Conductor points to the wanted GUI element and "beats" the menu item to select it.	
		The selected menu item gets activated by the System
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Changing the tempo (CM)

Type	Conducting

The conductor wishes to change the speed of the orchestra and changes his conducting gestures to indicate his wish.

Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The conductor slows down his conducting gestures or changes his facial expression	
		The system analyzes the user input to decide how much the tempo should change.
		The System changes the tempo.
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Changing the tempo of a part of the orchestra (CM)

Type	Conducting

Initiating Actor	Conductor
Participating Actors	
Realized User Task	

Preconditions	- System is in CM - Conducting started	
Flow of events	*Actor steps*	*System steps*
	The Conductor selects a part of the ochestra	
	and changes the tempo	
Postconditions		
Exceptions		

Open Issues	**Description**
Is this functionality really desirable?	Changing the tempo (BPM) of several musical tracks and than rendering all them simultaneously would lead in most cases to a quite unpleasant sound. (State of the art audio sequencers like ableton live, cubase, logic? etc do not provide such functionality)

Use Case: Changing the volume (CM)

Type	Conducting

The conductor wishes to change the volume of the orchestra and changes his conducting gestures to indicate his wish

Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The conductor makes bigger gestures	
		The System detects the changes in the gesture based on previous gestures
		It changes the volume of the orchestra
Postconditions		
Exceptions		

Nonfunctional Requirements	Description
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Changing the volume of a part of the orchestra

Type	Conducting

The conductor focuses his gestures towards a certain part of the orchestra. Volume changes now only apply to this selected part.

Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions	- System is in CM - conducting started	
Flow of events	*Actor steps*	*System steps*
	The conductor selects a part of the orchestra.	
	The conductor changes the volume by making bigger/smaller gestures.	
Postconditions		
Exceptions		

Use Case: Conducting

Type	Conducting

Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	the conductor makes gestures to conduct the virtual orchestra	
		the system tracks the conductor's gestures and performs adequate actions like adjust volume or tempe to give feedback to the user
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Moving a musician (CM)

Type	Conducting

The Conductor can move a musicianto the place where he wants him to be

Initiating Actor	Listener	
Participating Actors		

Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	Via a special gesture the Conductor selects a musician.	
		The gesture is recognized and the musician is higlighted to indicate the selection.
	The Conductor waves his baton around.	
		The Gestures are mapped on the changing position of the musician.
	Via another special gesture the Conductor indicates his satisfaction with the achieved position.	
		System saves the position.
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Prepare Conducting

Type	Conducting	
Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions	The conductor starts a new conducting session	
Flow of events	*Actor steps*	*System steps*
	The conductor chooses a song from a drop down menu	
		The system loads the song with the default volume and bass settings
	The conductor chooses a orchestra schema (quartet, big orchestra...)	
		The system loads the default orchestra schema and the video for it
	The conductor repositions or deletes some of the musicians if he wants	
		The system shows the new schema of the musicans
	The Conductor sets new settings of bass and volume if he wants	
		The Systems saves the new settings
Postconditions	New audio, video and orchestra settings are saved	
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms

Nonfunctional Requirements	Description
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
No legal restrictions for the used media	There should be no payment fee or restriction for the media we use.
Short startup time	The startup time should be less than 1 minute.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.

Use Case: Selecting a part of the orchestra (CM)

Type	Conducting	
Initiating Actor		
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The conductor focusses on a special part of the orchestra by obviously looking in its direction or by pointing at them with his left hand	
		The System recognizes the movement of the eyes or the orientation of the arm and applies all coming actions on the selected part of the orchestra
		It constantly checks if the focus remains on the selected part.
Postconditions		
Exceptions		

Use Case: Start Conducting

Type	Conducting

Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The conductor tells the system to start.	
		The System presents the video of the Orchestra and waits for user input so that it can start audio and video playback.
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
No legal restrictions for the used media	There should be no payment fee or restriction for the media we use.
Short startup time	The startup time should be less than 1 minute.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.

Use Case: Stop Conducting

Type	Conducting

Initiating Actor	Conductor	
Participating Actors		
Realized User Task		
Preconditions	The conductor has begun a conducting session	
Flow of events	*Actor steps*	*System steps*
	The conductor tells the system to stop	
		The system saves or saves not the conductor's settings, as the conductor wishes
Postconditions	The conducting session is over	
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Verifying ConductingGestures

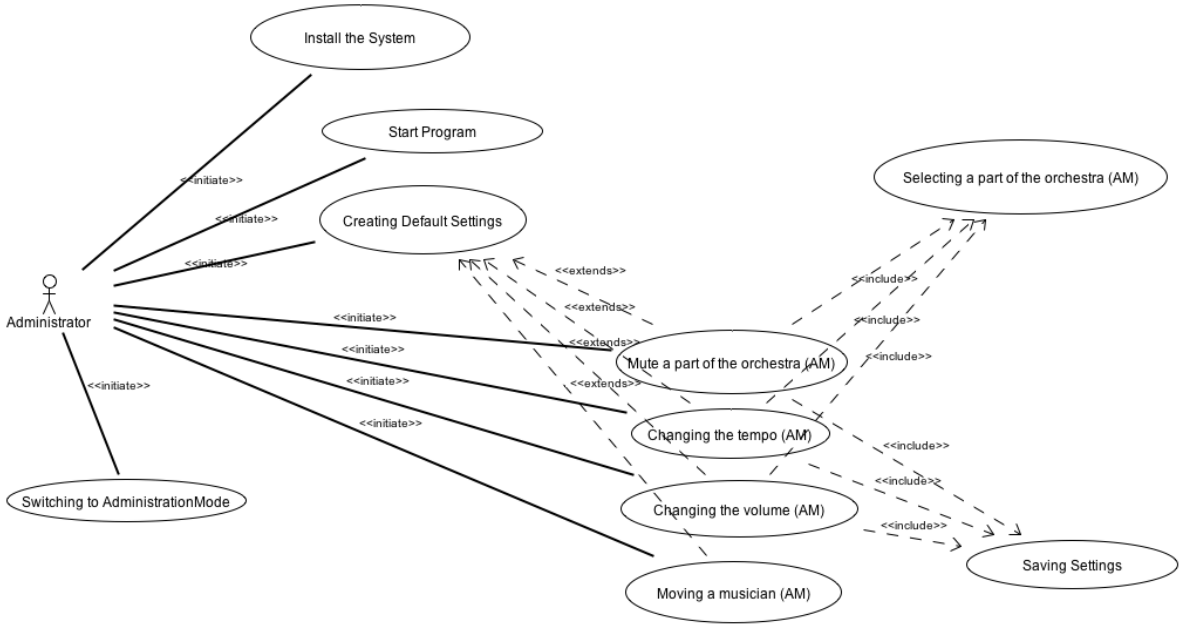
Type	Conducting

While the User is conducting the orchestra, the system gives him feedback about the correctness of his conducting gestures. Depending on system settings, the system stops music playback if the user makes incorrect gestures too often.

Initiating Actor		
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The User makes conducting gestures that should match a specific measure	
		The System gives feedback to the User about the correctness of his gestures. If the gesture has been incorrect too often (or for a too great amount of time), the system stops playback of audio and video.
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

2.4.2.2. Administrative Use Cases



Use Case: Changing the tempo (AM)

Type	Administrative

Changing the tempo of an Orchestra-Part in Administrator Mode

Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Administrator selects a part of the orchestra	
		The System shows the detail view of t selected part
	The Administrator changes the tempo of the selected part by moving sliders or manually entering values	
	He saves his changes	
Postconditions		
Exceptions		

Use Case: Changing the volume (AM)

Type	Administrative

Changing the volume in AdministrationMode

Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Administrator selects a part of the orchestra	
		The System shows the detail view of the part of the orchestra
	The Administrator changes the volume of the orchestra by moving sliders or manually entering values	
	and saves his changes	
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Support of new media	Every media should be replaceable.
Windowed administrator interface	The Adminstrators Interface should be a normal window.

Use Case: Creating Default Settings

Type	Administrative

Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The administrator loads or removes recordings of a musical piece with audio and video.	
	If the administrator wants to, he positions the musicians in the virtual room in the way he wants.	
	He sets the audio characteristics like reverb, bass and volume.	
	After his changes he saves the configuration as presets.	
		The System saves the configuration in the default settings, so they can be chosen by the conductor.
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
No legal restrictions for the used media	There should be no payment fee or restriction for the media we use.
Short startup time	The startup time should be less than 1 minute.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Windowed administrator interface	The Adminstrators Interface should be a normal window.

Use Case: Install the System

Type	Administrative

Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
No legal restrictions for the used media	There should be no payment fee or restriction for the media we use.
One DMG file includes the whole application	The application should be installable by dragging it to the Application folder.
Support of new media	Every media should be replaceable.

Use Case: Moving a musician (AM)

Type	Administrative

The Administrator should not need to move a musician by gestures with the baton. Instead he should be able to use the keyboard to achieve much higher accuracy

Initiating Actor	Administrator
Participating Actors	
Realized User Task	

Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Administrator opens the "Settings and Preferences" Dialog	
		A two-dimensional view of the orchestra from above is shown.
	The Administrator selects a virtual musician.	
		The musician is highlighted.
	Either by dragging the musicians representation or by manually entering the musicians position textually the administrator changes the position.	
		The system moves the musician around according to the Conductors input.
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Windowed administrator interface	The Adminstrators Interface should be a normal window.

Use Case: Mute a part of the orchestra (AM)

Type	Administrative

The Administrator can mute a Musician

Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Administrator selects a part of the orchestra	
		The System shows the detail view for the selected part
	The Administrator mutes the selected part	
	and saves the settings	
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Windowed administrator interface	The Adminstrators Interface should be a normal window.

Use Case: Saving Settings

Type	Administrative

Initiating Actor	
Participating Actors	
Realized User Task	

Preconditions		
Flow of events	*Actor steps*	*System steps*
Postconditions		
Exceptions		

Use Case: Selecting a part of the orchestra (AM)

Type	Administrative

Initiating Actor		
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
Ease of Use	Every user older than 7 years of age should be able to conduct the orchestra.
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
High accuracy	95% of the different gestures should be recognized correctly
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Use Case: Start Program

Type	Administrative	
Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Administrator starts the system by double-clicking its application icon	
		The System starts up and presents the AdministratorView.
	The Adminstrator chooses a defined preset and chooses to switch to UserMode	
		The system switches to UserMode and goes fullscreen
Postconditions		
Exceptions		

Nonfunctional Requirements	**Description**
FullScreen conductor interface	The Conductors interface should run in fullscreen mode.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
No legal restrictions for the used media	There should be no payment fee or restriction for the media we use.
Short startup time	The startup time should be less than 1 minute.
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.
Interaction Forms	The Conductor should be able to interact with the sytem with gestures, facial expressions and movements of his baton.

Nonfunctional Requirements	Description
Windowed administrator interface	The Adminstrators Interface should be a normal window.

Use Case: Switching to AdministrationMode

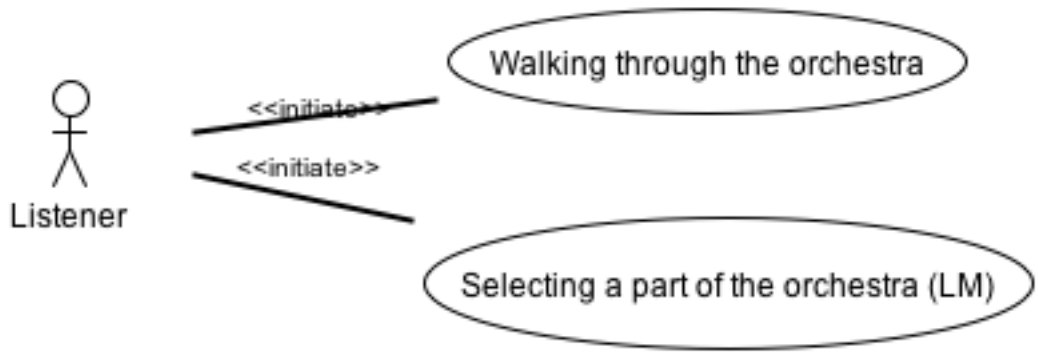
Type	Administrative

Initiating Actor	Administrator	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	The Administrator tells the System to change to AdministrationMode	
		The System verifies the identity of the Administrator. No other person should be able to change to AdministratorMode.
	The Administrator identifies himself.	
		After identification the System switches to AdministrationMode.
Postconditions		
Exceptions		

Nonfunctional Requirements	Description
Windowed administrator interface	The Adminstrators Interface should be a normal window.
Good Response Time	The Response between an action and a noticable reaction should be less than 100 ms
Implementation languages	The Implementation languages should be Objective-C, C and C++ as they can be mixed at will. Objective-C is the prefered language, C and C++ should only be used whenever it is necessary, for example when working with CoreAudio.
Short startup time	The startup time should be less than 1 minute.

Nonfunctional Requirements	Description
Stable Application	The Admin Interface and the Conductor Interface should never quit without an explicit quit command.

2.4.2.3. Listening Use Cases



Use Case: Selecting a part of the orchestra (LM)

Type	Listening	
Initiating Actor	Listener	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
	With his baton the Conductor points to a part of the orchestra	
		The System highlights the affected part
	The Conductor finalizes his selection by "knocking" on the part	
		The System highlights the selected part in a special way to indicate the selection.
Postconditions		

Exceptions	

Use Case: Walking through the orchestra

Type	Listening

Initiating Actor	Listener	
Participating Actors		
Realized User Task		
Preconditions		
Flow of events	*Actor steps*	*System steps*
Postconditions		
Exceptions		

Open Issues	**Description**
Walking through the orchestra: is it possible?	To walk through the orchestra, we need - 3d-models of each musician and - 3d-model of the concert-hall But we can not render the musician-models from different points of view using standard video cameras. So what?

2.4.3. Object model

Sections:

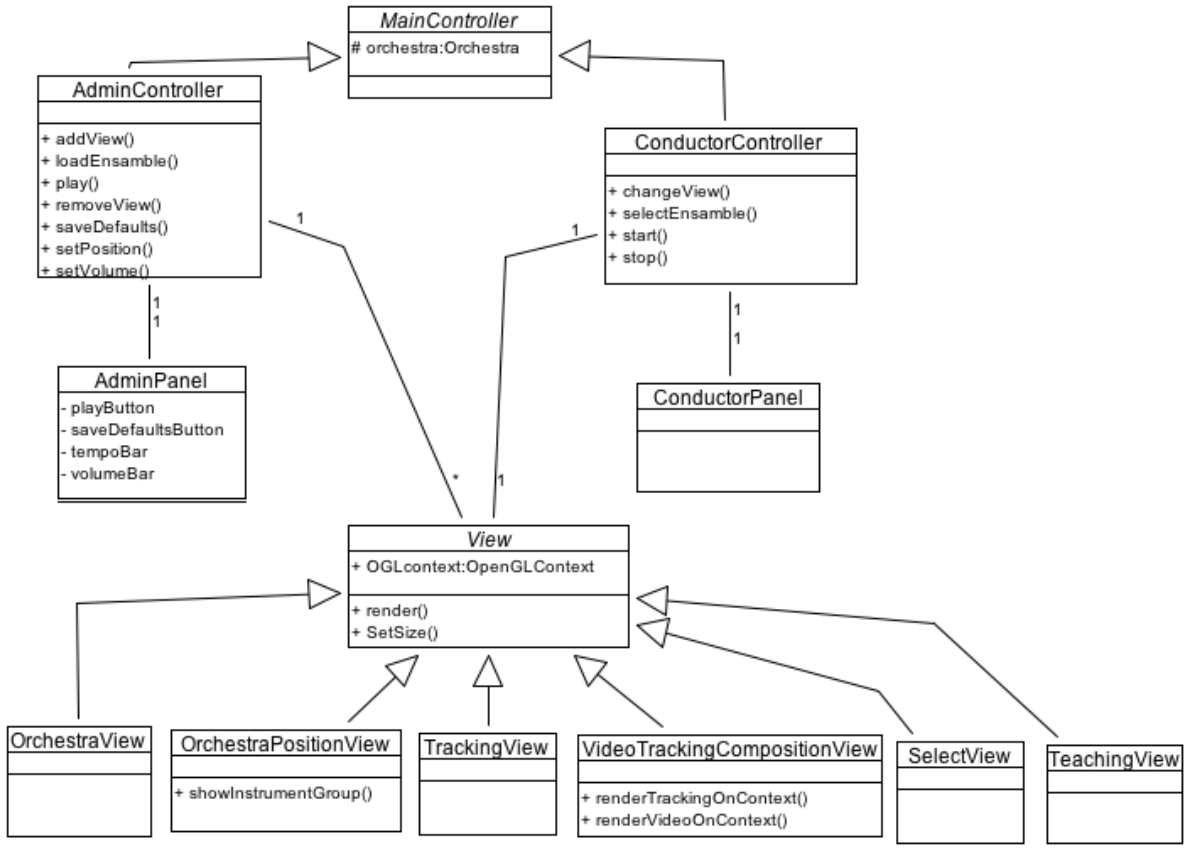
- Packages
- Classes

2.4.3.1. Packages

Package: vso

Subpackages	UserInterface, Orchestra, Architecture, Tracking, Video.Solution.Model, Audio, Video.Application.Model

Package: UserInterface



Classes	MainController, AdminController, AdminPanel, View, TrackingView, OrchestraView, VideoTrackingCompositionView, ConductorController, ConductorPanel, TeachingView, OrchestraPositionView, SelectView
Related Components	UI

Open Action Items	Description
Associate the user interface classes to the related components	Map the classes or packages of user interface subsystem to the related user interface components. The 'Components' field of the classes or the 'Object Model Elements' field of the components can be used to create the mapping.

Class: MainController

MainController has contains the common functionality of the AdminController and the ConductorController

Subclasses	AdminController, ConductorController
Related Components	UI

Attributes	Description
# orchestra:Orchestra	

Class: AdminController

The AdminController class is responsable for all activities, which can happen in the admin mode. For example - load and remove views, give the default settings to the orchestra, set the positions of the musicians, play a video and audio file etc.

Superclasses	MainController
Related Components	UI

Operations	Description
+ addView()	adds a new view to the admin panel
+ loadEnsamble()	loads an ensamble
+ play()	plays audio and video file for the selected ensamble
+ removeView()	
+ saveDefaults()	saves the default settings from the Admin Interface and give them to the orchestra
+ setPosition()	sets the positions for the musicians
+ setVolume()	sets the volume for the selected musicians

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AdminController		1	AdminPanel
*	View		1	AdminController

Class: AdminPanel

implements the Admin Interface

Related Components	UI

Attributes	Description
- playButton	
- saveDefaultsButton	
- tempoBar	
- volumeBar	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AdminController		1	AdminPanel

Class: View

View has contains the common functionality of all Views

Subclasses	SelectView, OrchestraView, TrackingView, VideoTrackingCompositionView, OrchestraPositionView, TeachingView

Attributes	Description
+ OGLcontext:OpenGLContext	

Operations	Description
+ render()	gives the graphical context to the orchestra modules so they say the audio and video module to start the work with it
+ SetSize()	set the size of the view (context of the view)

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
*	View		1	AdminController
1	View		1	ConductorController

Comments	Description
Revise View Taxonomy	The view is not well thought out. For example, what is the difference between TeachingView and VideoView. Should I not be able to use a video view when teaching? Also, the missing attributes and operations for the subclasses of View indicate that you have not really analyzed the view abstraction.

Class: TrackingView

Superclasses	View

Class: OrchestraView

Shows the video with the musicians

Superclasses	View

Class: VideoTrackingCompositionView

Superclasses	View

Operations	Description
+ renderTrackingOnContext()	renders the context to the audio
+ renderVideoOnContext()	renders the context to the video

Class: ConductorController

The ConductorController class is responsable for all activities, which can happen in the conductor mode. For example - switsching between the different views (TeachingView, VideoTrackingCompositionView...), loads the admin mode if it is required by the orchestra etc.

Superclasses	MainController

Operations	Description
+ changeView()	change the view, depending on what the conductor wants: to conduct or to learn (to be teached)
+ selectEnsamble()	selects the ensamble which the conductor wants to play
+ start()	calls the render() operation from the view class, witch render the selected view to the orcestra
+ stop()	stop method can be called by the orchesttra when the conductor stops conducting. Using that method the view and the modus can be selected

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	View		1	ConductorController
1	ConductorPanel		1	ConductorController

Class: ConductorPanel

implements the fullscreen interface for the conductor

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorPanel		1	ConductorController

Class: TeachingView

Superclasses	View

Class: OrchestraPositionView

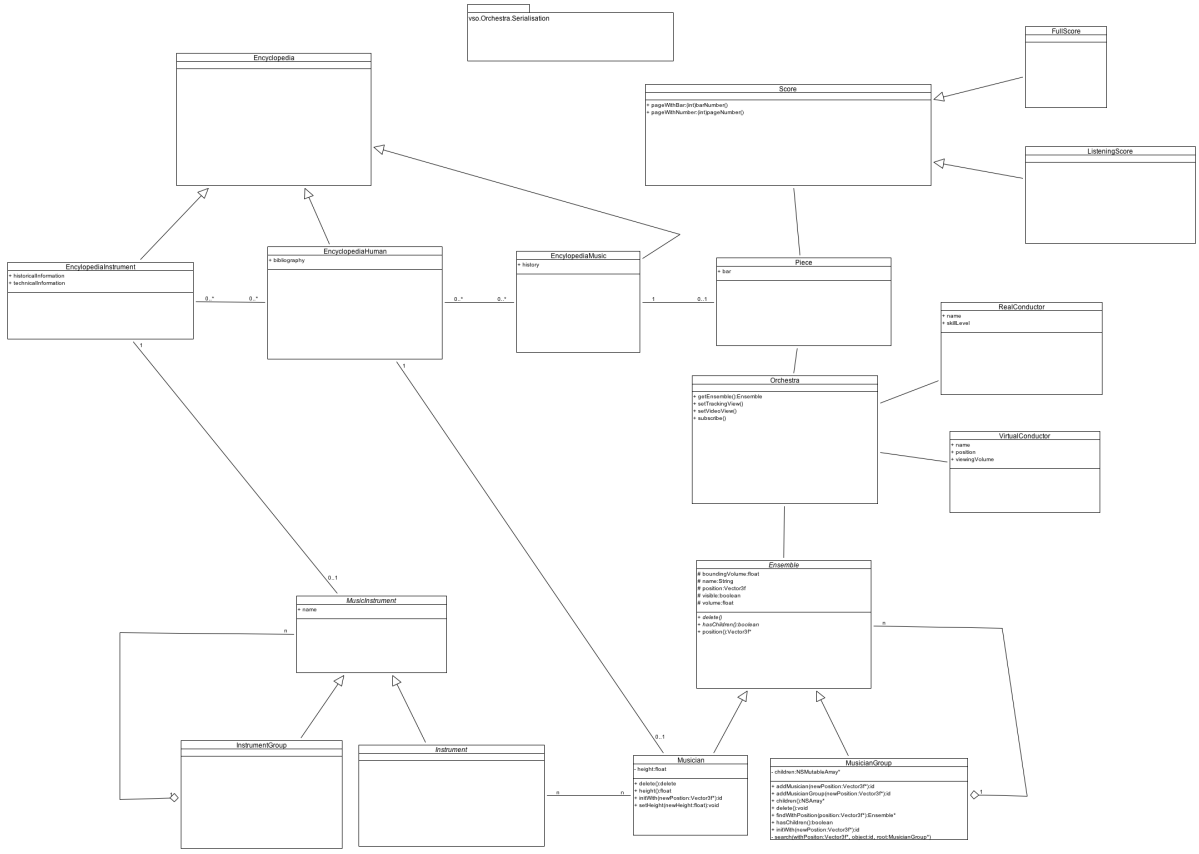
Superclasses	View

Operations	Description
+ showInstrumentGroup()	shows the snstrument group. so the admin can change the positions of the single instruments

Class: SelectView

Superclasses	View

Package: Orchestra



Subpackages	Serialisation
Classes	Piece, Orchestra, Ensemble, Musician, MusicianGroup, Encyclopedia, EncyclopediaHuman, EncylopediaMusic, MusicInstrument, InstrumentGroup, Instrument, VirtualConductor, RealConductor, Score, EncylopediaInstrument, FullScore, ListeningScore

Open Action Items	Description
Associate the orchestra classes to the related components	Map the classes or packages of orchestra subsystem to the related orchestra components. The 'Components' field of the classes or the 'Object Model Elements' field of the components can be used to create the mapping.

Class: Piece

Attributes	Description
+ bar	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Piece			Orchestra
1	EncylopediaMusic		0..1	Piece
	Piece			Score

Comments	Description
Please rename Piece into Score	

Class: Orchestra

Operations	Description
+ getEnsemble():Ensemble	returns the ensemble for further use in other packages
+ setTrackingView()	This Operation forwards the OpenGLContext to the Tracking Package
+ setVideoView()	This Operation forwards the OpenGLContext to the Video Package
+ subscribe()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Orchestra			Ensemble
	Piece			Orchestra
	Orchestra			VirtualConductor
	Orchestra			RealConductor

Comments	Description
Orchestra is missing an assocation to Audio/Video recording from here	I am missing an association to the Audio/Video recorded for the Score (the end must 0..*, because there should be 0 or more recordings for the Score)

Class: Ensemble

Subclasses	Musician, MusicianGroup

Attributes	Description
# boundingVolume:float	Volume containing the MusicianGroup and it's Musicians
# name:String	
# position:Vector3f	
# visible:boolean	depicts wether the MusicianGroup or Musician is visible to the conductor
# volume:float	

Operations	Description
+ delete()	
+ hasChildren():boolean	
+ position():Vector3f*	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	MusicianGroup		n	Ensemble
	Orchestra			Ensemble
	Renderer			Ensemble

Class: Musician

Superclasses	Ensemble

Attributes	Description
- height:float	Height of the musician

Operations	Description
+ delete():delete	
+ height():float	
+ initWith(newPostion:Vector3f*):id	
+ setHeight(newHeight:float):void	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
n	Instrument		n	Musician
0..1	Musician		1	EncyclopediaHuman

Class: MusicianGroup

Superclasses	Ensemble

Attributes	Description
- children:NSMutableArray*	

Operations	Description
+ addMusician(newPosition:Vector3f*):id	
+ addMusicianGroup(newPosition:Vector3f*):id	
+ children():NSArray*	
+ delete():void	
+ findWithPosition(position:Vector3f*):Ensemble*	
+ hasChildren():boolean	
+ initWith(newPostion:Vector3f*):id	
- search(withPositon:Vector3f*, object:id, root:MusicianGroup*)	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	MusicianGroup		n	Ensemble

Class: Encyclopedia

Subclasses	EncyclopediaHuman, EncylopediaMusic, EncylopediaInstrument

Class: EncyclopediaHuman

Superclasses	Encyclopedia

Attributes	Description
+ bibliography	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
0..*	EncyclopediaHuman		0..*	EncylopediaMusic
0..1	Musician		1	EncyclopediaHuman
0..*	EncylopediaInstrument		0..*	EncyclopediaHuman

Class: EncylopediaMusic

Superclasses	Encyclopedia

Attributes	Description
+ history	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
0..*	EncyclopediaHuman		0..*	EncylopediaMusic
1	EncylopediaMusic		0..1	Piece

Class: MusicInstrument

Subclasses	Instrument, InstrumentGroup

Attributes	Description
+ name	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	InstrumentGroup		n	MusicInstrument
1	EncylopediaInstrument		0..1	MusicInstrument

Open Issues	Description
What is the difference between MusicInstrument and Instrument?	Asumming there is a design reason for having these two different classes. Why are both abstract? What about just using the 1 to * aggregation, which is already defined in the model, without the composite association?

Class: InstrumentGroup

Superclasses	MusicInstrument

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	InstrumentGroup		n	MusicInstrument

Class: Instrument

Superclasses	MusicInstrument

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
n	Instrument		n	Musician

Open Issues	Description
What is the difference between MusicInstrument and Instrument?	Asumming there is a design reason for having these two different classes. Why are both abstract? What about just using the 1 to * aggregation, which is already defined in the model, without the composite association?

Class: VirtualConductor

The Conductor class holds information about the parameters of the virtual conductor, like position and viewing volume.

Attributes	Description
+ name	
+ position	
+ viewingVolume	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Orchestra			VirtualConductor

Open Issues	Description
What is the role of the virtual conductor?	What does the virtual conductor do?

Class: RealConductor

The RealConductor represents the user of the system and stores information like his name or skills.

Attributes	Description
+ name	
+ skillLevel	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Orchestra			RealConductor

Class: Score

Subclasses	FullScore, ListeningScore

Operations	Description
+ pageWithBar:(int)barNumber()	
+ pageWithNumber:(int)pageNumber()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Piece			Score

Class: EncylopediaInstrument

Superclasses	Encyclopedia

Attributes	Description
+ historicalInformation	
+ technicalInformation	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
0..*	EncylopediaInstrument		0..*	EncyclopediaHuman
1	EncylopediaInstrument		0..1	MusicInstrument

Class: FullScore

A full score is a large book showing the music of all instruments. It is large enough for a conductor to use in rehearsals and performance.

Superclasses	Score

Class: ListeningScore

The listeningscore is a simplified and often graphical represented form, which makes the reading and understanding easier for a novice.

Superclasses	Score

Package: Serialisation

This package is responsible for the archiving of settings and the orchestra configuration

Classes	XMLProcessor

Class: XMLProcessor

Operations	Description
+ conductor():Conductor	
+ orchestraConfiguration():InstrumentGroup	
+ piece():Piece	
+ preferences()	
+ readOrchestraConfiguration:(NSString*)filePath():void	
+ readPreferencesFile:(NSString*)filePath():void	
+ writeOrchestraConfiguration:(NSString*)filePath()	
+ writePreferences:(NSString*)filePath()	

Package: Architecture

Classes	VSOAppControl, NSNotificationCenter, VSONotificationCenter, Vector3f

Class: VSOAppControl

VSOAppControl is the StartupItem and the main controller of the VSO Application.

It asks for the mode of operation, server or client, and sets the applications parameters accordingly.

Forthermore it detects offered services on the network via bonjour.

Attributes	**Description**
- serviceMode:int=0	Stores the users choice of service. Can be either server or client. Possible Values and their meanings: 0 undefined 1 server 2 client

Operations	**Description**
- searchServices():void	Searches for bonjour services on the network matching the
- startServerService():void	
- stopServerService()	

Class: NSNotificationCenter

Subclasses	VSONotificationCenter

Comments	Description
NSNotificationCenter is not a typed object	Which is the type: entity, boundary or control? Also, the class seems to be a solution domain class.

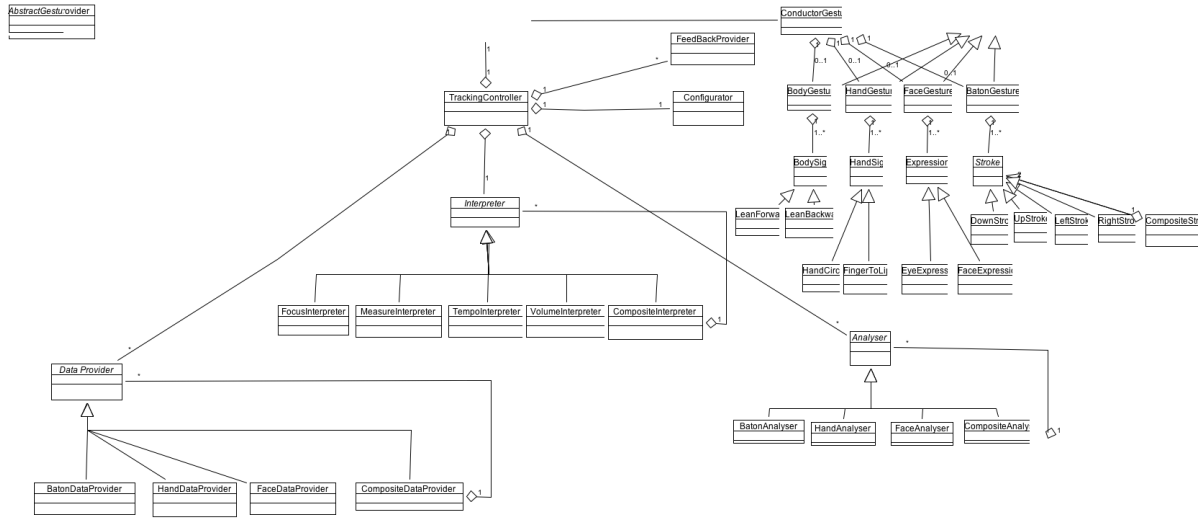
Class: VSONotificationCenter

Superclasses	NSNotificationCenter

Class: Vector3f

Operations	Description
+ add()	
+ cross()	
+ div()	
+ dot()	
+ mul()	
+ sub()	

Package: Tracking



Classes	TrackingController, Data Provider, BatonDataProvider, HandDataProvider, FaceDataProvider, CompositeDataProvider, Interpreter, TempoInterpreter, MeasureInterpreter, BatonAnalyser, FocusInterpreter, GestureDBProvider, VolumeInterpreter, CompositeInterpreter, HandAnalyser, FaceAnalyser, CompositeAnalyser, Analyser, FeedBackProvider, Configurator, AbstractGesture, LeanForward, BodyGesture, HandGesture, FaceGesture, BatonGesture, ConductorGesture, LeanBackward, CompositeStroke, LeftStroke, UpStroke, RightStroke, DownStroke, FaceExpression, EyeExpression, FingerToLips, HandCircle, Stroke, Expression, HandSign, BodySign

Open Action Items	Description
Associate the tracking classes to the related components	Map the classes or packages of tracking subsystem to the related tracking components. The 'Components' field of the classes or the 'Object Model Elements' field of the components can be used to create the mapping.

Class: TrackingController

Related Components	TrackingController

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		1	GestureDBProvider
	TrackingController		1	Interpreter
1	TrackingController		*	FeedBackProvider
1	TrackingController		*	Data Provider
1	TrackingController		*	Analyser
1	TrackingController		1	Configurator

Class: Data Provider

Subclasses	BatonDataProvider, HandDataProvider, FaceDataProvider, CompositeDataProvider

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		*	Data Provider
1	CompositeDataProvider		*	Data Provider

Class: BatonDataProvider

Superclasses	Data Provider

Class: HandDataProvider

Superclasses	Data Provider

Class: FaceDataProvider

Superclasses	Data Provider

Class: CompositeDataProvider

Superclasses	Data Provider

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeDataProvider		*	Data Provider

Class: Interpreter

Interprets the Data provided by the Analysers

Subclasses	TempoInterpreter, MeasureInterpreter, FocusInterpreter, VolumeInterpreter, CompositeInterpreter

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeInterpreter		*	Interpreter
	TrackingController		1	Interpreter

Class: TempoInterpreter

Superclasses	Interpreter

Class: MeasureInterpreter

Superclasses	Interpreter

Class: BatonAnalyser

Superclasses	Analyser

Class: FocusInterpreter

Superclasses	Interpreter

Class: GestureDBProvider

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		1	GestureDBProvider
	ConductorGesture			GestureDBProvider

Class: VolumeInterpreter

Superclasses	Interpreter

Class: CompositeInterpreter

Superclasses	Interpreter

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeInterpreter		*	Interpreter

Class: HandAnalyser

Superclasses	Analyser

Class: FaceAnalyser

Superclasses	Analyser

Class: CompositeAnalyser

Superclasses	Analyser

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeAnalyser		*	Analyser

Class: Analyser

Subclasses	BatonAnalyser, FaceAnalyser, HandAnalyser, CompositeAnalyser

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeAnalyser		*	Analyser
1	TrackingController		*	Analyser

Class: FeedBackProvider

Provides Feedback about the users conducting knowledge

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		*	FeedBackProvider

Class: Configurator

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		1	Configurator

Class: AbstractGesture

Subclasses	BatonGesture, BodyGesture, FaceGesture, HandGesture

Class: LeanForward

Superclasses	BodySign

Class: BodyGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	BodyGesture
1	BodyGesture		1..*	BodySign

Class: HandGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	HandGesture
1	HandGesture		1..*	HandSign

Class: FaceGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	FaceGesture
1	FaceGesture		1..*	Expression

Class: BatonGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	BatonGesture
1	BatonGesture		1..*	Stroke

Class: ConductorGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	BodyGesture
1	ConductorGesture		0..1	HandGesture
1	ConductorGesture		0..1	FaceGesture
1	ConductorGesture		0..1	BatonGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	ConductorGesture			GestureDBProvider

Class: LeanBackward

Superclasses	BodySign

Class: CompositeStroke

Superclasses	Stroke

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeStroke		2	Stroke

Class: LeftStroke

Superclasses	Stroke

Class: UpStroke

Superclasses	Stroke

Class: RightStroke

Superclasses	Stroke

Class: DownStroke

Superclasses	Stroke

Class: FaceExpression

Superclasses	Expression

Class: EyeExpression

Superclasses	Expression

Class: FingerToLips

Superclasses	HandSign

Class: HandCircle

Superclasses	HandSign

Class: Stroke

Subclasses	DownStroke, UpStroke, LeftStroke, RightStroke, CompositeStroke

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	BatonGesture		1..*	Stroke
1	CompositeStroke		2	Stroke

Class: Expression

Subclasses	EyeExpression, FaceExpression

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	FaceGesture		1..*	Expression

Class: HandSign

Subclasses	HandCircle, FingerToLips

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	HandGesture		1..*	HandSign

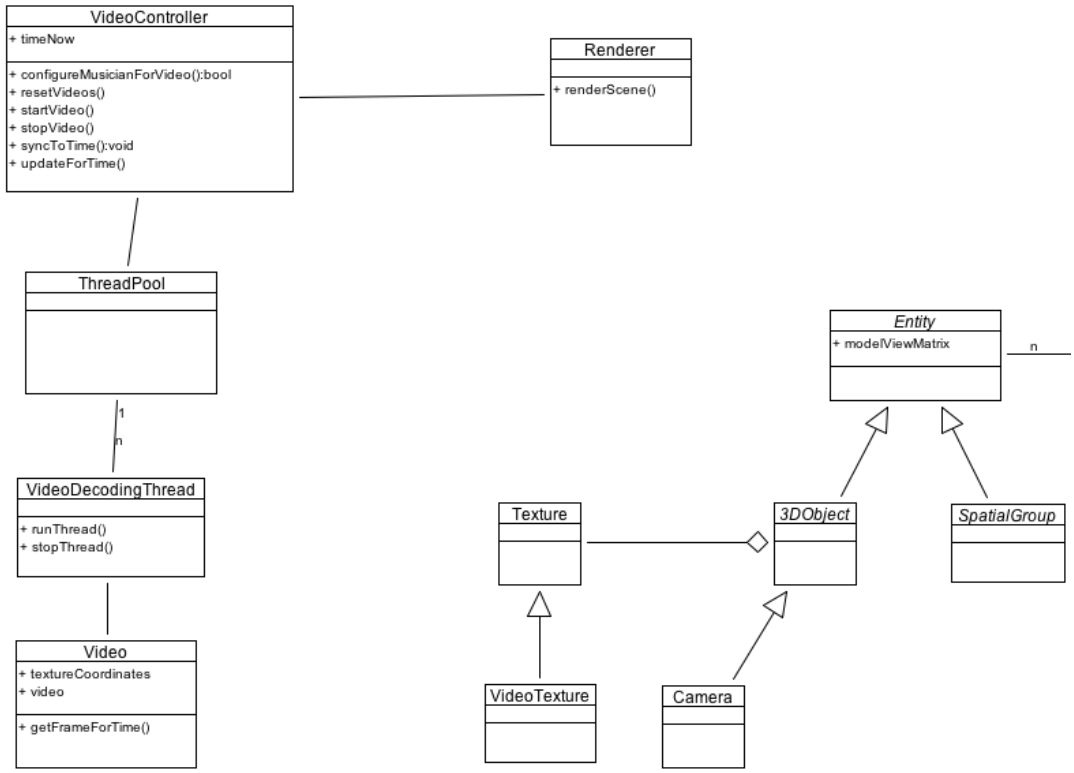
Class: BodySign

Subclasses	LeanForward, LeanBackward

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	BodyGesture		1..*	BodySign

Package: Video.Solution.Model

Video Subsystem Solution Model



Classes	VideoController, Video, Renderer, ThreadPool, VideoDecodingThread, Entity, 3DObject, SpatialGroup, Camera, Texture, VideoTexture

Open Action Items	**Description**
Associate the video classes to the related components	Map the classes or packages of video subsystem to the related video components. The 'Components' field of the classes or the 'Object Model Elements' field of the components can be used to create the mapping.

Class: VideoController

Attributes	**Description**
+ timeNow	

Operations	**Description**
+ configureMusicianForVideo():bool	

Operations	Description
+ resetVideos()	
+ startVideo()	
+ stopVideo()	
+ syncToTime():void	
+ updateForTime()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoController			Renderer
	VideoController			ThreadPool

Class: Video

Wrapper for Quicktime Videos

Attributes	Description
+ textureCoordinates	
+ video	

Operations	Description
+ getFrameForTime()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoDecodingThread			Video

Class: Renderer

Operations	Description
+ renderScene()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoController			Renderer
	Renderer			Ensemble

Class: ThreadPool

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ThreadPool		n	VideoDecodingThread
	VideoController			ThreadPool

Class: VideoDecodingThread

Operations	Description
+ runThread()	
+ stopThread()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ThreadPool		n	VideoDecodingThread
	VideoDecodingThread			Video

Class: Entity

Subclasses	3DObject, SpatialGroup

Attributes	Description
+ modelViewMatrix	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	SpatialGroup		n	Entity

Class: 3DObject

Superclasses	Entity
Subclasses	Camera

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	3DObject			Texture

Class: SpatialGroup

Superclasses	Entity

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	SpatialGroup		n	Entity

Class: Camera

Superclasses	3DObject

Class: Texture

Subclasses	VideoTexture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	3DObject			Texture

Class: VideoTexture

Superclasses	Texture

Package: Audio

Audio Subsystem

Classes	AudioController, AUGraph, FilePlayerNode, 3DMixerNode, TimeStretchingNode, GlobalEffectsNode, AudioStream, <<interface>> OrchestraObserver, <<interface>> MusicianObserver

Class: AudioController

Main Class of Audio Subsystem

Superclasses	<<interface>> OrchestraObserver
Related Components	AudioController

Attributes	Description
- au_globalEffects:AudioUnit	
- au_globalVolumeMixer:AudioUnit	
- au_timeStretching:AudioUnit	
- orchestra:Orchestra	

Operations	Description
+ <<constructor>> AudioController()	
+ loadOrchestraModel()	
+ pauseAudio()	
- registerMusician()	
+ resetAudioController()	
+ setGlobalEffects()	
+ setGlobalTempo()	
+ setGlobalVolume()	
+ startAudio()	
+ stopAudio()	
- traverseOrchesra()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AudioController		n	AudioStream
	AudioController			AUGraph

Class: AUGraph

Access point to the AudioUnit Framework

The AUGraph is a high-level representation of a set of Audio Units, along with the connections between them. These APIs may be used to construct arbitrary signal paths through which audio may be processed, that is, a modular routing system. The APIs deal with large numbers of Audio Units and their relationships to one another.

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AUGraph		n	FilePlayerNode
	AUGraph			3DMixerNode
	AUGraph			TimeStretchingNode
	AUGraph			GlobalEffectsNode
	AudioController			AUGraph

Class: FilePlayerNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AUGraph		n	FilePlayerNode

Class: 3DMixerNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	AUGraph			3DMixerNode

Class: TimeStretchingNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	AUGraph			TimeStretchingNode

Class: GlobalEffectsNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	AUGraph			GlobalEffectsNode

Class: AudioStream

Superclasses	<<interface>> MusicianObserver

Attributes	Description
- au_3DMixer:AudioUnit	
- au_filePlayer:AudioUnit	
- au_graph:AUGraph	
- musician:Musician	

Operations	Description
+ <<constructor>> AudioStream()	
- setFilePath()	
- setPosition()	
- setVolume()	
+ updateSettings()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AudioController		n	AudioStream

Class: <<interface>> OrchestraObserver

Subclasses	AudioController

Class: <<interface>> MusicianObserver

Subclasses	AudioStream

Package: Video.Application.Model

Video Subsystem Application Model

Classes	Time, VirtualTime, World, Entity, EntityGroup, 3D Object, Observer, Light, PointLight, SpotLight, Surface, View, VideoView

Open Action Items	Description
Associate the video classes to the related components	Map the classes or packages of video subsystem to the related video components. The 'Components' field of the classes or the 'Object Model Elements' field of the components can be used to create the mapping.

Class: Time

Subclasses	VirtualTime

Attributes	Description
+ time	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Time			World

Class: VirtualTime

Superclasses	Time

Attributes	Description
+ offsetToRealTime	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoView			VirtualTime

Class: World

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Entity			World
	Time			World
	Observer			World
	View			World
	Light			World

Class: Entity

Subclasses	EntityGroup, 3D Object

Attributes	Description
+ position	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Entity			World
1	EntityGroup		n	Entity

Class: EntityGroup

Superclasses	Entity

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	EntityGroup		n	Entity

Class: 3D Object

Superclasses	Entity
Subclasses	Observer, Light

Attributes	Description
+ orientation	Spatial position

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Surface			3D Object

Class: Observer

Superclasses	3D Object

Attributes	Description
+ fieldOfView	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Observer			World

Class: Light

Superclasses	3D Object
Subclasses	SpotLight, PointLight

Attributes	Description
+ intensity	Light Intensity

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Light			World

Class: PointLight

Superclasses	Light

Class: SpotLight

Superclasses	Light

Class: Surface

Subclasses	VideoView

Attributes	Description
+ color	
+ material	Surface Material

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Surface			3D Object

Class: View

Operations	Description
+ renderWorld()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	View			World

Class: VideoView

Superclasses	Surface

Attributes	Description
+ video	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoView			VirtualTime

2.4.3.2. Classes

Class: 3D Object

Superclasses	Entity
Subclasses	Observer, Light

Attributes	Description
+ orientation	Spatial position

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Surface			3D Object

Class: 3DMixerNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	AUGraph			3DMixerNode

Class: 3DObject

Superclasses	Entity
Subclasses	Camera

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	3DObject			Texture

Class: <<interface>> MusicianObserver

Subclasses	AudioStream

Class: <<interface>> OrchestraObserver

Subclasses	AudioController

Class: AbstractGesture

Subclasses	BatonGesture, BodyGesture, FaceGesture, HandGesture

Class: AdminController

The AdminController class is responsable for all activities, which can happen in the admin mode. For example - load and remove views, give the default settings to the orchestra, set the positions of the musicians, play a video and audio file etc.

Superclasses	MainController
Related Components	UI

Operations	Description
+ addView()	adds a new view to the admin panel
+ loadEnsamble()	loads an ensamble

Operations	Description
+ play()	plays audio and video file for the selected ensamble
+ removeView()	
+ saveDefaults()	saves the default settings from the Admin Interface and give them to the orchestra
+ setPosition()	sets the positions for the musicians
+ setVolume()	sets the volume for the selected musicians

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AdminController		1	AdminPanel
*	View		1	AdminController

Class: AdminPanel

implements the Admin Interface

Related Components	UI

Attributes	Description
- playButton	
- saveDefaultsButton	
- tempoBar	
- volumeBar	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AdminController		1	AdminPanel

Class: Analyser

Subclasses	BatonAnalyser, FaceAnalyser, HandAnalyser, CompositeAnalyser

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeAnalyser		*	Analyser
1	TrackingController		*	Analyser

Class: AudioController

Main Class of Audio Subsystem

Superclasses	<<interface>> OrchestraObserver
Related Components	AudioController

Attributes	Description
- au_globalEffects:AudioUnit	
- au_globalVolumeMixer:AudioUnit	
- au_timeStretching:AudioUnit	
- orchestra:Orchestra	

Operations	Description
+ <<constructor>> AudioController()	
+ loadOrchestraModel()	
+ pauseAudio()	
- registerMusician()	
+ resetAudioController()	
+ setGlobalEffects()	
+ setGlobalTempo()	
+ setGlobalVolume()	
+ startAudio()	
+ stopAudio()	
- traverseOrchesra()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AudioController		n	AudioStream
	AudioController			AUGraph

Class: AudioStream

Superclasses	<<interface>> MusicianObserver

Attributes	Description
- au_3DMixer:AudioUnit	
- au_filePlayer:AudioUnit	
- au_graph:AUGraph	
- musician:Musician	

Operations	Description
+ <<constructor>> AudioStream()	
- setFilePath()	
- setPosition()	
- setVolume()	
+ updateSettings()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AudioController		n	AudioStream

Class: AUGraph

Access point to the AudioUnit Framework

The AUGraph is a high-level representation of a set of Audio Units, along with the connections between them. These APIs may be used to construct arbitrary signal paths through which audio may be processed, that is, a modular routing system. The APIs deal with large numbers of Audio Units and their relationships to one another.

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AUGraph		n	FilePlayerNode
	AUGraph			3DMixerNode
	AUGraph			TimeStretchingNode
	AUGraph			GlobalEffectsNode
	AudioController			AUGraph

Class: BatonAnalyser

Superclasses	Analyser

Class: BatonDataProvider

Superclasses	Data Provider

Class: BatonGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	BatonGesture
1	BatonGesture		1..*	Stroke

Class: BodyGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	BodyGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	BodyGesture		1..*	BodySign

Class: BodySign

Subclasses	LeanForward, LeanBackward

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	BodyGesture		1..*	BodySign

Class: Camera

Superclasses	3DObject

Class: CompositeAnalyser

Superclasses	Analyser

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeAnalyser		*	Analyser

Class: CompositeDataProvider

Superclasses	Data Provider

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeDataProvider		*	Data Provider

Class: CompositeInterpreter

Superclasses	Interpreter

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeInterpreter		*	Interpreter

Class: CompositeStroke

Superclasses	Stroke

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeStroke		2	Stroke

Class: ConductorController

The ConductorController class is responsable for all activities, which can happen in the conductor mode. For example - switsching between the different views (TeachingView, VideoTrackingCompositionView...), loads the admin mode if it is required by the orchestra etc.

Superclasses	MainController

Operations	Description
+ changeView()	change the view, depending on what the conductor wants: to conduct or to learn (to be teached)
+ selectEnsamble()	selects the ensamble which the conductor wants to play
+ start()	calls the render() operation from the view class, witch render the selected view to the orcestra
+ stop()	stop method can be called by the orchesttra when the conductor stops conducting. Using that method the view and the modus can be selected

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	View		1	ConductorController

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorPanel		1	ConductorController

Class: ConductorGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	BodyGesture
1	ConductorGesture		0..1	HandGesture
1	ConductorGesture		0..1	FaceGesture
1	ConductorGesture		0..1	BatonGesture
	ConductorGesture			GestureDBProvider

Class: ConductorPanel

implements the fullscreen interface for the conductor

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorPanel		1	ConductorController

Class: Configurator

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		1	Configurator

Class: Data Provider

Subclasses	BatonDataProvider, HandDataProvider, FaceDataProvider, CompositeDataProvider

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		*	Data Provider
1	CompositeDataProvider		*	Data Provider

Class: DownStroke

Superclasses	Stroke

Class: Encyclopedia

Subclasses	EncyclopediaHuman, EncylopediaMusic, EncylopediaInstrument

Class: EncyclopediaHuman

Superclasses	Encyclopedia

Attributes	Description
+ bibliography	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
0..*	EncyclopediaHuman		0..*	EncylopediaMusic
0..1	Musician		1	EncyclopediaHuman
0..*	EncylopediaInstrument		0..*	EncyclopediaHuman

Class: EncylopediaInstrument

Superclasses	Encyclopedia

Attributes	Description
+ historicalInformation	
+ technicalInformation	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
0..*	EncylopediaInstrument		0..*	EncyclopediaHuman
1	EncylopediaInstrument		0..1	MusicInstrument

Class: EncylopediaMusic

Superclasses	Encyclopedia

Attributes	Description
+ history	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
0..*	EncyclopediaHuman		0..*	EncylopediaMusic
1	EncylopediaMusic		0..1	Piece

Class: Ensemble

Subclasses	Musician, MusicianGroup

Attributes	Description
# boundingVolume:float	Volume containing the MusicianGroup and it's Musicians
# name:String	
# position:Vector3f	
# visible:boolean	depicts wether the MusicianGroup or Musician is visible to the conductor
# volume:float	

Operations	Description
+ delete()	
+ hasChildren():boolean	

Operations	Description
+ position():Vector3f*	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	MusicianGroup		n	Ensemble
	Orchestra			Ensemble
	Renderer			Ensemble

Class: Entity

Subclasses	3DObject, SpatialGroup

Attributes	Description
+ modelViewMatrix	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	SpatialGroup		n	Entity

Class: Entity

Subclasses	EntityGroup, 3D Object

Attributes	Description
+ position	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Entity			World
1	EntityGroup		n	Entity

Class: EntityGroup

Superclasses	Entity

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	EntityGroup		n	Entity

Class: Expression

Subclasses	EyeExpression, FaceExpression

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	FaceGesture		1..*	Expression

Class: EyeExpression

Superclasses	Expression

Class: FaceAnalyser

Superclasses	Analyser

Class: FaceDataProvider

Superclasses	Data Provider

Class: FaceExpression

Superclasses	Expression

Class: FaceGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	FaceGesture
1	FaceGesture		1..*	Expression

Class: FeedBackProvider

Provides Feedback about the users conducting knowledge

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		*	FeedBackProvider

Class: FilePlayerNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	AUGraph		n	FilePlayerNode

Class: FingerToLips

Superclasses	HandSign

Class: FocusInterpreter

Superclasses	Interpreter

Class: FullScore

A full score is a large book showing the music of all instruments. It is large enough for a conductor to use in rehearsals and performance.

Superclasses	Score

Class: GestureDBProvider

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		1	GestureDBProvider
	ConductorGesture			GestureDBProvider

Class: GlobalEffectsNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	AUGraph			GlobalEffectsNode

Class: HandAnalyser

Superclasses	Analyser

Class: HandCircle

Superclasses	HandSign

Class: HandDataProvider

Superclasses	Data Provider

Class: HandGesture

Superclasses	AbstractGesture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ConductorGesture		0..1	HandGesture
1	HandGesture		1..*	HandSign

Class: HandSign

Subclasses	HandCircle, FingerToLips

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	HandGesture		1..*	HandSign

Class: Instrument

Superclasses	MusicInstrument

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
n	Instrument		n	Musician

Open Issues	Description
What is the difference between MusicInstrument and Instrument?	Asumming there is a design reason for having these two different classes. Why are both abstract? What about just using the 1 to * aggregation, which is already defined in the model, without the composite association?

Class: InstrumentGroup

Superclasses	MusicInstrument

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	InstrumentGroup		n	MusicInstrument

Class: Interpreter

Interprets the Data provided by the Analysers

Subclasses	TempoInterpreter, MeasureInterpreter, FocusInterpreter, VolumeInterpreter, CompositeInterpreter

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	CompositeInterpreter		*	Interpreter
	TrackingController		1	Interpreter

Class: LeanBackward

Superclasses	BodySign

Class: LeanForward

Superclasses	BodySign

Class: LeftStroke

Superclasses	Stroke

Class: Light

Superclasses	3D Object
Subclasses	SpotLight, PointLight

Attributes	Description
+ intensity	Light Intensity

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Light			World

Class: ListeningScore

The listeningscore is a simplified and often graphical represented form, which makes the reading and understanding easier for a novice.

Superclasses	Score

Class: MainController

MainController has contains the common functionality of the AdminController and the ConductorController

Subclasses	AdminController, ConductorController
Related Components	UI

Attributes	Description
# orchestra:Orchestra	

Class: MeasureInterpreter

Superclasses	Interpreter

Class: Musician

Superclasses	Ensemble

Attributes	Description
- height:float	Height of the musician

Operations	Description
+ delete():delete	
+ height():float	
+ initWith(newPostion:Vector3f*):id	
+ setHeight(newHeight:float):void	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
n	Instrument		n	Musician
0..1	Musician		1	EncyclopediaHuman

Class: MusicianGroup

Superclasses	Ensemble

Attributes	Description
- children:NSMutableArray*	

Operations	Description
+ addMusician(newPosition:Vector3f*):id	
+ addMusicianGroup(newPosition:Vector3f*):id	
+ children():NSArray*	
+ delete():void	
+ findWithPosition(position:Vector3f*):Ensemble*	
+ hasChildren():boolean	
+ initWith(newPostion:Vector3f*):id	
- search(withPositon:Vector3f*, object:id, root:MusicianGroup*)	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	MusicianGroup		n	Ensemble

Class: MusicInstrument

Subclasses	Instrument, InstrumentGroup

Attributes	Description
+ name	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	InstrumentGroup		n	MusicInstrument
1	EncylopediaInstrument		0..1	MusicInstrument

Open Issues	Description
What is the difference between MusicInstrument and Instrument?	Asumming there is a design reason for having these two different classes. Why are both abstract? What about just using the 1 to * aggregation, which is already defined in the model, without the composite association?

Class: NSNotificationCenter

Subclasses	VSONotificationCenter

Comments	Description
NSNotificationCenter is not a typed object	Which is the type: entity, boundary or control? Also, the class seems to be a solution domain class.

Class: Observer

Superclasses	3D Object

Attributes	Description
+ fieldOfView	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Observer			World

Class: Orchestra

Operations	Description
+ getEnsemble():Ensemble	returns the ensemble for further use in other packages
+ setTrackingView()	This Operation forwards the OpenGLContext to the Tracking Package
+ setVideoView()	This Operation forwards the OpenGLContext to the Video Package
+ subscribe()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Orchestra			Ensemble
	Piece			Orchestra
	Orchestra			VirtualConductor
	Orchestra			RealConductor

Comments	Description
Orchestra is missing an assocation to Audio/Video recording from here	I am missing an association to the Audio/Video recorded for the Score (the end must 0..*, because there should be 0 or more recordings for the Score)

Class: OrchestraPositionView

Superclasses	View

Operations	Description
+ showInstrumentGroup()	shows the snstrument group. so the admin can change the positions of the single instruments

Class: OrchestraView

Shows the video with the musicians

Superclasses	View

Class: Piece

Attributes	Description
+ bar	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Piece			Orchestra
1	EncylopediaMusic		0..1	Piece
	Piece			Score

Comments	Description
Please rename Piece into Score	

Class: PointLight

Superclasses	Light

Class: RealConductor

The RealConductor represents the user of the system and stores information like his name or skills.

Attributes	Description
+ name	
+ skillLevel	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Orchestra			RealConductor

Class: Renderer

Operations	Description
+ renderScene()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoController			Renderer
	Renderer			Ensemble

Class: RightStroke

Superclasses	Stroke

Class: Score

Subclasses	FullScore, ListeningScore

Operations	Description
+ pageWithBar:(int)barNumber()	
+ pageWithNumber:(int)pageNumber()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Piece			Score

Class: SelectView

Superclasses	View

Class: SpatialGroup

Superclasses	Entity

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	SpatialGroup		n	Entity

Class: SpotLight

Superclasses	Light

Class: Stroke

Subclasses	DownStroke, UpStroke, LeftStroke, RightStroke, CompositeStroke

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	BatonGesture		1..*	Stroke
1	CompositeStroke		2	Stroke

Class: Surface

Subclasses	VideoView

Attributes	Description
+ color	
+ material	Surface Material

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Surface			3D Object

Class: TeachingView

Superclasses	View

Class: TempoInterpreter

Superclasses	Interpreter

Class: Texture

Subclasses	VideoTexture

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	3DObject			Texture

Class: ThreadPool

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ThreadPool		n	VideoDecodingThread
	VideoController			ThreadPool

Class: Time

Subclasses	VirtualTime

Attributes	Description
+ time	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Time			World

Class: TimeStretchingNode

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	AUGraph			TimeStretchingNode

Class: TrackingController

Related Components	TrackingController

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	TrackingController		1	GestureDBProvider
	TrackingController		1	Interpreter
1	TrackingController		*	FeedBackProvider
1	TrackingController		*	Data Provider
1	TrackingController		*	Analyser
1	TrackingController		1	Configurator

Class: TrackingView

Superclasses	View

Class: UpStroke

Superclasses	Stroke

Class: Vector3f

Operations	Description
+ add()	
+ cross()	
+ div()	
+ dot()	
+ mul()	
+ sub()	

Class: Video

Wrapper for Quicktime Videos

Attributes	Description
+ textureCoordinates	
+ video	

Operations	Description
+ getFrameForTime()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoDecodingThread			Video

Class: VideoController

Attributes	Description
+ timeNow	

Operations	Description
+ configureMusicianForVideo():bool	
+ resetVideos()	
+ startVideo()	
+ stopVideo()	
+ syncToTime():void	
+ updateForTime()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoController			Renderer
	VideoController			ThreadPool

Class: VideoDecodingThread

Operations	Description
+ runThread()	
+ stopThread()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
1	ThreadPool		n	VideoDecodingThread
	VideoDecodingThread			Video

Class: VideoTexture

Superclasses	Texture

Class: VideoTrackingCompositionView

Superclasses	View

Operations	Description
+ renderTrackingOnContext()	renders the context to the audio
+ renderVideoOnContext()	renders the context to the video

Class: VideoView

Superclasses	Surface

Attributes	Description
+ video	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoView			VirtualTime

Class: View

Operations	Description
+ renderWorld()	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	View			World

Class: View

View has contains the common functionality of all Views

Subclasses	SelectView, OrchestraView, TrackingView, VideoTrackingCompositionView, OrchestraPositionView, TeachingView

Attributes	Description
+ OGLcontext:OpenGLContext	

Operations	Description
+ render()	gives the graphical context to the orchestra modules so they say the audio and video module to start the work with it
+ SetSize()	set the size of the view (context of the view)

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
*	View		1	AdminController
1	View		1	ConductorController

Comments	Description
Revise View Taxonomy	The view is not well thought out. For example, what is the difference between TeachingView and VideoView. Should I not be able to use a video view when teaching? Also, the missing attributes and operations for the subclasses of View indicate that you have not really analyzed the view abstraction.

Class: VirtualConductor

The Conductor class holds information about the parameters of the virtual conductor, like position and viewing volume.

Attributes	Description
+ name	
+ position	
+ viewingVolume	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Orchestra			VirtualConductor

Open Issues	Description
What is the role of the virtual conductor?	What does the virtual conductor do?

Class: VirtualTime

Superclasses	Time

Attributes	Description
+ offsetToRealTime	

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	VideoView			VirtualTime

Class: VolumeInterpreter

Superclasses	Interpreter

Class: VSOAppControl

VSOAppControl is the StartupItem and the main controller of the VSO Application.

It asks for the mode of operation, server or client, and sets the applications parameters accordingly.

Forthermore it detects offered services on the network via bonjour.

Attributes	Description
- serviceMode:int=0	Stores the users choice of service. Can be either server or client. Possible Values and their meanings: 0 undefined 1 server 2 client

Operations	Description
- searchServices():void	Searches for bonjour services on the network matching the
- startServerService():void	
- stopServerService()	

Class: VSONotificationCenter

Superclasses	NSNotificationCenter

Class: World

Client Multiplicity	Client	Label	Supplier Multiplicity	Supplier
	Entity			World
	Time			World
	Observer			World
	View			World
	Light			World

Class: XMLProcessor

Operations	Description
+ conductor():Conductor	
+ orchestraConfiguration():InstrumentGroup	
+ piece():Piece	
+ preferences()	
+ readOrchestraConfiguration:(NSString*)filePath():void	
+ readPreferencesFile:(NSString*)filePath():void	
+ writeOrchestraConfiguration:(NSString*)filePath()	
+ writePreferences:(NSString*)filePath()	

3 Glossary

The glossary contains a vocabulary and definitions for important or frequently encountered concepts, usually including idioms or metaphors.

Glossary Entry: AM

Administration Mode

Glossary Entry: CM

Conducting Mode

Glossary Entry: Conductor

The Conductor is the End User of the system. In general he can be anything from a mucial layman up to a professional conductor.

In the context of the RAD document, Conductor is an Actor.

Glossary Entry: LM

Listener Mode

Glossary Entry: Measure

Also known as bar (American English), time or Takt (German)

Glossary Entry: Renderer

Glossary Entry: Reverb

Colloquial abbreviation of reverberation, Hall (German)

Glossary Entry: Scene