Implementation of the Semantic Layer Management System and the Scripting Engine

Deliverable D3.2.2a

FascinatE identifier:	FascinatE-D322a-SES-ImplementationOfSLMSandSE-v06.docx
Deliverable number:	D3.2.2a
Author(s) and company:	R. Kaiser, W. Weiss, G. Kienast (JRS); G. Thomas (BBC), M. Masetti (SES);
Internal reviewers:	J.-F. Macq (ALU), O.A. Niamut (TNO)
Work package / task:	WP3
Document status:	Final
Confidentiality:	Public

Version	Date	Reason of change
1	2011-07-20	Document created
2	2012-01-14	SMLS component first draft specifications added
3	2012-01-18	Text highlighting
4	2012-02-10	Draft version (almost) ready for internal review.
5	2012-02-14	Changes to chapter 4, to be finalised by JRS
6	2012-02-17	Final version based on internal review

Acknowledgement: The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 248138.

Table of Contents

1 Executive Summary

This deliverable provides a detailed description of the architecture and design of the Semantic Layer Management System (SLMS) and the Production Scripting Engine (PSE) components.

Details about the internal software architecture for this real-time system are presented and interfaces to other production components are described. Details of subcomponents of the distributed engine, design decisions and technology choices are discussed.

These components are core outcomes of Work Package 3 and will interact with the A/V analysis tools described in D3.3.1a.

The SLMS is the core semantic data store of the FascinatE system. The semantic data store has to provide a fast (online) classification of A/V features and indexing support for multidimensional queries (along time, space, a mix of both, media type, and other characteristics).

In the realm of a format agnostic live event broadcast system FascinatE, the Production Scripting Engines are software components that automate taking decisions on what is visible and audible at each playout device and prepare the audiovisual content streams for display. Essentially, they act together as a *Virtual Director* with the production team possibly steering it via a backend user interface.

2 Introduction

2.1 Purpose of this Document

This deliverable contains the detailed specifications of the Semantic Layer Management System (SLMS) and the Production Scripting Engine (PSE) components of the FascinatE system.

The two components represent the major output of Work Package 3 which has the aim to provide automatic support for the editing process in both live and offline scenarios by integrating metadata and knowledge about the content, domain, production context, target formats/devices and users.

The PSE is a *Virtual Director*, an intelligent software component, that automates content selection from live streams ("camera switching") to a high degree and it does this in parallel for a large number of viewers. The PSE implements a set of cinematographic principles and takes decision on camera selection within real-time constraints. Besides physical cameras, it also handles *virtual* cameras, dynamic (moving) crops from an ultra-high definition view such as the one provided by the OMNICAM.

For the PSE to be able to take decisions, a continuous load of information (in terms of A/V features) has to be extracted online and sent to the PSE. On the other hand the PSE has no memory, in the sense that holds only the last few frames features and quickly decides what to do mostly depending on information regarding the *present*.

The SLMS provides a layer for holding the memory of all the metadata information regarding the broadcasted event (including a segmentation of the end users). Moreover the SLMS holds details of the event domain and sensor profiles that be used by the PSE. Storing all metadata extracted from an event can enable services (statistical services for example) not currently envisaged.

The design of the semantic layer and the scripting engine follows the canonical waterfall workflow. Starting from requirements, formats and interfaces (production requirements have been outlined in D1.1.1, the scene structure and coding scheme has been described in D2.1.2 and finally tools and models to represent metadata have been analysed in D3.1.2), the different design and implementation decisions are presented and discussed. The architecture of the components (at high and medium level) is presented. Finally hints on the implementation at sub-component level are provided.

The design and implementation details are referred to the interim version, used for the interim FascinatE system that will be presented at second Project Review. In describing a feature or design detail that will be present in the final system prototype, this will be clearly reported.

A final statement on the status of maturity of the components: These components (especially the Semantic Layer Management System) largely depend on other system components. This constraint heavily impacted on the design process.

2.2 Scope of this Document

This document is connected with D3.2.1 where the architecture of the Semantic Layer Management System and the Production Scripting Engine components were introduced.

The implementation of the final version of the SLMS and the PSE will be described in Deliverable 3.2.2b, which will be released at project month 36.

2.3 Status of this Document

This is the final version of D3.2.2a.

2.4 Related Documents

This deliverable is based on the following documents:

- D1.4.2 *Interim system specification* gives an overview of the main parts of the FascinatE system, explaining their functions and the kind of data that moves between them.
- D2.1.2 *Interim specification of generic data representation and coding scheme* describes some metadata elements that are an integral part of the layered scene representation.
- D3.1.2 *Metadata and knowledge models and tools.*

- D3.2.1 *The Semantic Layer Management System and Scripting Engine Design*
- D5.1.2 *AV Renderer with Arbitrary Sparse Loudspeaker Setups & Simple Interactivity* summarizes the modules and features of the initial version of the AV rendering prototype.

Further, relevant research papers are listed in the reference section.

3 Semantic Layer Management System (SLMS)

3.1 Introduction

The Semantic Layer Management System (SLMS from now on) is the software component responsible for parsing, storing and querying metadata describing multimedia streams.

Metadata will come in a variety of formats, at different rates, conveying different information.

The aim of the component is to harmonize, normalize and drill in different dimensions (mainly time and space and media object) all the multimedia information stored.

A/V metadata will be mostly encoded in MPEG-7, using basically the semantic entities exposed by MPEG-7 (StillRegion, MovingRegion, TextAnnotation, Usage descriptors among the others). Camera metadata are encoded in a format used by BBC in their production environment.

The component will interface mainly with the multimedia analysis components and the scripting engine and is basically triggered by input calls. The next section outlines a list of stories that the component should implement.

3.2 Stories

Stories can be used to describe the functionalities of a component. Each story describes a given functionality, providing also details about a simple testing procedure.

The following table lists the stories tackled by the SLMS, identifying in which milestone of the FascinatE system (M5/M7) they will be implemented, reporting test scenarios and open issues still to be fixed.

M5 refers to the system ready for second project review (project month 27), M7 refers to the final version of system components, ready to be integrated into the final demonstrator (as depicted in D1.4.2 – Interim System Specification).

ID	Name	Mile-stone	Test	Main Client Module	Open issues
1	Register camera	M5	Call setCameraProfile and check database to see if proper data was inserted. Camera profile format is based on BBC format. Interface is file based.	Sensors	
2	Register microphone	M7	Call setMicProfile and check database to see if proper data was inserted Mic profile format is still to be fixed.	Sensors	How are they sending data (stream or file)?
3	Update camera profile	M5	Call updateCameraProfile and check database to see if proper data was inserted. Camera profile format is internal BBC (ASCII file). Camera profile is linked with camera frame (profile valid for that frame).	Sensors	

5	Get camera profile	M7	Call getCameraProfile providing camera id and compare received data with corresponding record in database. Optionally the frame id can be passed to have the profile valid at that frame. Profile metadata is based on BBC format.	PSE	
6	Get microphone profile	M5	Call getMicProfile providing microphone id and compare received data with corresponding record in database	PSE	Which MPEG-7 attributes is PSE expecting to receive?
7	Receive detected moving objects	M7	Call setMovingRegions and check database to see if proper data was inserted	AM	
8	Receive detected audio objects	M7	Call setAudioObject and check that proper data is inserted.	AM	Which attributes are we going to receive for audio?
9	Receive relevant region	M7	Call setRelevantRegion and check database to see if proper data was inserted	Editor UI	How is a region expressed?
10	Receive event	M7	Call setEvent and check database to see if proper data was inserted	Editor UI	How is an event expressed? In which format?
11	Receive object annotation	M7	Call updateMovingRegion and check database to see if proper data was inserted	Editor UI	How is an annotation expressed? (e.g. how to express that a MovingRegion represents Lionel Messi?). Annotations should be context aware
12	Register new group of viewers	M7	Call setGroup and check database to see if proper data was inserted	?	Who is responsible for registering new user groups? How are subscriptions performed? How is the user profile expressed? In which format?
13	Get group of viewers	M7	Call getGroup providing group id and compare received data with corresponding record in database	PSE	How is the user profile expressed? In which format?

14	Get all MovingRegions describing a moving object in time interval	M5	Call getObject providing object id and time interval. All MovingRegions describing the motion of the object in given time interval are returned.	PSE	Should we provide an iterator to have all MovingRegions? Alternatively, should we provide output data as MPEG-7 file or is it better to pass a data structure?
15	Set canvas size	M7	Call setCanvasSize (width, lenth). This sets the size of the canvas and is used in conjunction with setGridRes/getObjInCell to compute the objects present in a cell in a given time frame.	PSE	
16	Set grid resolution	M7	Call setGridRes (rows, cols) to set the number of rows and columns of the grid. Along with setCanvasSize it has to be called once at event beginning. All computations regarding the number of stored objects per cell are based on these parameters.	PSE	
17	Get objects in a given cell at a given frame	M7	Call getObjsInCell (row, col, frame) to get the list of A/V objects found in that cell at given frame	PSE	Should we provide an iterator?
18	Get main action area in time interval	M7	Call getMostDenseCell (frame) providing time interval to get back coordinates of most dense cell.	PSE	This method is domain dependent (makes sense in a football match, not in a concert) Should we provide output data as MPEG-7 file or is it better to pass a data structure?

3.3 SMLS Architecture

The following figure shows high level system architecture

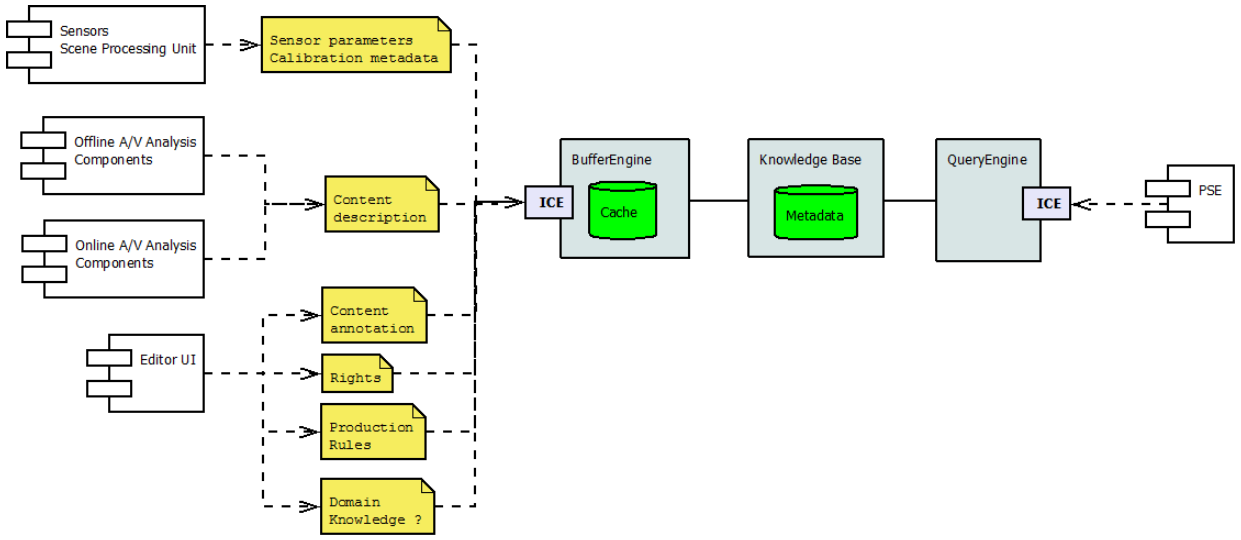


Figure 1: The SMLS architecture

The knowledge store handles metadata associated with multimedia streams (regarding stream technical attributes, detected objects information and access rights).

Since we are going to handle real-time metadata, performance will be a key issue: both data processing and transferring need to be as fast as possible. What's more important is to keep up with the data transmission rate of online analysis modules, so that no information is lost. Similarly, when querying knowledge store, results must be delivered according to time constraints given by scripting engines.

According to these considerations, the Knowledge Store can be subdivided in following components:

- **Buffer Engine** - it is responsible for providing other FascinatE modules (namely Sensors, Editor UI, Analysis Modules) with input API for sending new data to Knowledge Store. Moreover, it also performs caching of incoming metadata in case Knowledge Store is not fast enough to process them in real time, thus preventing loss of data. The bufferEngine does not only provide an API but can actively fetch metadata exposed by the other components.
- **Knowledge Base** - this is the core component: it parses incoming metadata, processes and stores them. Since metadata are likely to arrive in XML format, relevant attributes must be extracted and stored inside a DBMS (Data Base Management System). Since Knowledge Store is supposed to extract meaningful information from metadata, this component will perform all needed operations.
- **Query Engine** – this component will provide other FascinatE components (namely the Scripting Engines) with API for querying the content of the KB. Moreover, it will be responsible for formatting output data in the right format required by querying module.

Consider that the Buffer Engine and Query Engine will provide an ICE interface: this will make communication between remote modules easier and will implement a robust mechanism for realizing load balancing and a first step of data caching. ICE[1] is an object-oriented middleware platform. Fundamentally, this means that ICE provides tools, APIs, and library support for building object-oriented client-server applications. ICE applications are suitable for use in heterogeneous environments: client and server can be written in different programming languages, can run on different operating systems and machine architectures, and can communicate using a variety of networking technologies. The source code for these applications is portable regardless of the deployment environment.

[1] http://www.zeroc.com

The **Buffer Engine** will run as a service, waiting for incoming calls. When new data is sent, it must implement a caching mechanism to retain data until the KB is ready to process it. Here follows the detailed workflow:

- Wait for write method to be called
- Write data to cache
- Deliver new data to Knowledge Base

Since it is necessary to process incoming data as fast as possible, different aspects must be taken into consideration:

- Many instances of the Buffer Engine can be active at the same time, using load balancing services provided by ICE architecture.
- Caching mechanism can rely on ICE dispatcher architecture and ICE persistence features

The **Knowledge Base** will not provide any interface to FascinatE modules and it will be used as a library by the Buffer Engine and Query Engine components.

This component is responsible for:

- Parsing incoming metadata
- Store metadata
- Process metadata in order to extract relevant information

Knowledge Base will provide methods for both writing and reading data that will be used by the Buffer Engine and Query Engine respectively.

The Knowledge Base features a data repository built on top of a DBMS.

In order to make query operations as fast as possible, all relevant attributes will be extracted from incoming data and inserted in ad hoc tables: so there will be one table for MovingRegions, one for StillRegions, and so on, covering all attributes that can be queried

When Knowledge Base is notified by the Buffer Engine that new data have arrived, it will retrieve them from cache and then process them. When processing is done, data will be deleted from cache.

Finally, one last word regarding **Query Engine**: This module is mainly conceived to relief Knowledge Base from processing query results. Moreover, if modules need data to be queried and provided as fast as possible, many instances of the Query Engine can be active at the same time, using load balancing services provided by ICE architecture.

The Query Engine will run as a service, waiting for incoming calls. Its main tasks are:

- Parse incoming queries and pass them to Knowledge Base
- Format returned data to meet needs of module that made the query

3.3.1 SLMS Performance requirements

The SLMS component has to meet high performance requirements. It is expected that A/V analysis modules (which work in parallel and are able to analyse each single video frame) will extract features at a very high rate. The ICE framework provides the IceGrid[2], a set of features to implement several load balancing client-server solutions. The following is a list of the major IceGrid features used by the SLMS:

- **Replication and load balancing**. IceGrid supports replication by grouping services deployed on several servers into a single virtual service. Furthermore, IceGrid monitors the load on each computer and can use that information to decide which of the endpoints to return to a client.
- **Automatic failover**. Ice supports automatic retry and failover in any proxy that contains multiple endpoints. When combined with IceGrid's support for replication and load balancing, automatic failover means that a failed request results in a client transparently retrying the request on the next endpoint with the lowest load.

It is envisaged that the use of the IceGrid services will let the SLMS component to scale up to meet the target performance requirements. No performance tests have been conducted so far as the M5 prototype has not been integrated yet.

[2] http://doc.zeroc.com/display/Ice/IceGrid

3.4 SLMS Software Packages implemented at M5

All packages will be included in com.fascinate.softeco namespace as Java libraries. Relevant packages are:

- com.fascinate.softeco.bufferengine – contains all classes needed when receiving new data
- com.fascinate.softeco.queryengine – contains all classes needed when querying and outputting data
- com.fascinate.softeco.kb – contains all classes needed when parsing, storing and processing data

The Buffer Engine exposes these modules:

- a service, waiting for incoming calls. When new data is sent, it will keep it in a local cache until the KB is ready to process it. Here follows the detailed workflow:
- a polling process that will polls a configured folder for camera profiles.

The KB (Knowledge Base) sub-component is responsible for:

- Parsing incoming metadata
- Store metadata
- Process metadata in order to extract relevant information

KB will provide methods for both writing and reading data, which will be used by BufferEngine and QueryEngine respectively:

- setMovingRegions

The KB features a data repository built on top of a DBMS (Data Base Management System).

In order to make query operations as fast as possible, all relevant attributes will be extracted from incoming data and inserted in ad hoc tables: so there will be one table for MovingRegions, one for StillRegions, and so on, covering all attributes which we can query.

When KB is notified by BufferEngine that new data have arrived, it will retrieve them from cache and then process them. When processing is done, data will be deleted from cache.

The Query Engine sub-component will run as an IceGrid service, waiting for incoming calls. Its main tasks are:

- Parse incoming queries and pass them to KB
- Format returned data to meet needs of module that made the query

4 Production Scripting Engine (PSE)

The Production Scripting Engine (PSE) is a *Virtual Director*, an intelligent software component that automates content selection from live streams ("camera switching") to a high degree. It can do so in a personalized manner in parallel for a large number of viewers. Its behaviour, incrementally becoming more sophisticated and pleasing to watch, implements a set of pragmatic rules (capture what's important) and aesthetic rules (when and how to cut between cameras). The PSE implements a set of cinematographic principles and takes decision on camera selection within real-time constraints. Besides physical cameras, it also, or mainly, handles *virtual* cameras, dynamic (moving) crops from an ultra-high definition view such as the OMNICAM.

4.1 Virtual Director

4.1.1 Introduction

The FascinatE project aims to create an innovative end-to-end system for immersive and interactive TV services. It allows users to navigate in an ultra-high resolution video panorama, showing live or recorded content, with matching accompanying audio. The output is adapted to the viewing device, covering anything from a mobile handset to an immersive panoramic display with surround sound, delivering a personalized multi-screen experience. For lean-back consumption, the PSE automatically selects camera views per viewing group and decides when to cut, pan and zoom, so that the most important action is shown.

FascinatE is using a panoramic camera with high enough resolution for cropping interesting regions, the OMNICAM, depicted in Figure 2. It is a collection of 6 HD cameras sharing a single optical centre for obtaining a 180° panoramic video sequence stitched together in real-time from the 6 tiles. The vertical field of view is 60 degrees. The HD cameras are placed on their side and point upwards to a reflecting mirror to maximize the resolution such that when the video sequences are stitched together, the resolution of the final panorama is usually 6984 x 1920 pixels. This resolution allows capturing even distant objects in good quality so that e.g. persons at the other end of a sports field can be detected automatically. The minimum distance of objects to the camera depends on the accuracy of the camera mounting and is roughly two meters. This special camera allows the PSE to frame virtual cameras as moving crops of the panoramic video stream, as it captures the whole scene in good resolution. Viewers might want to see different parts of the scene and PSE will automatically take generic preferences into account based on a limited user profile (viewing groups).

Besides the OMNICAM, a range of broadcast cameras such as the ALEXA[3] are used.

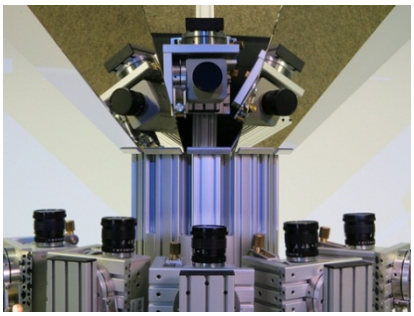


Figure 2: The OMNICAM [4].

In the following we report on design and implementation (ongoing) of an automatic view selection component, the FascinatE Production Scripting Engine (PSE). The PSE takes decisions on what is visible and audible at each playout device and prepares the audiovisual content streams for display. Such components are often referred to as a *Virtual Director*. In order to take reasonable decisions, the engine needs knowledge about what is currently happening in the scene and which camera streams are capturing that action.

[3] http://www.arri.com/camera/digital_cameras/cameras.html

[4] http://www.hhi.fraunhofer.de/en/departments/image-processing/applications/2d-3d-omnicam/

4.1.2 Features enabled by the PSE

The FascinatE system supports a set of features that allow the viewer to go beyond traditional broadcast viewing (e.g. multi-language audio) and achieve a higher degree of personalization through interaction. The FascinatE system is format agnostic which means that it produces live content streams for different playout device types in parallel, effectively reducing production cost. In technical terms, that means that instead of the traditional production of a single stream, a lot of different streams are produced in parallel based on the LSR. That in turn necessitates automation in view selection.

Viewers are given increased freedom to interact with the system in order to individually select what they want to see – directly, via more abstract options or implicitly as the system learns about their preferences without specific interaction. In one extreme case, a viewer would lean back and not interact with the system at all and watch the default coverage as suggested by the PSE. On the other side heavy interaction goes as far as free view navigation within the high-resolution content on dedicated devices with sophisticated interaction modes such as gesture control (Kinect). In the latter case the viewer completely leaves the PSE's suggested path and navigates freely until she/he decides to get back to automatic framing. Free viewer navigation is directly handled by the renderer in the current FascinatE architecture.

As a human production team could not cater for a large audience in parallel, the PSE is needed to automate content selection as far as possible. The format agnostic production system reasons for different viewer groups in parallel. There is at least one group per playout device type, and possibly more, implied by the options available in the viewer interface.

One key feature is the ability to closely follow an object such as a person, or to follow groups such as athletes from a certain country. The PSE's intelligent behaviour aims to ensure that, despite these selections, actions of a high priority are overruling specific user preferences to the benefit of the viewer experience. As an example, in a football situation where a touchdown (very high priority event) is likely to happen next, the close-up-view of a player apart from the scene will be left to make sure the viewer doesn't miss the likely occurrence of a more important action. This of course requires basic prediction functionality that can also lead to improper behaviour at times. To be able to follow a certain object, a certain quality and stability of the tracking component is required. Alternatively, this information could also come through manual annotation from the EUI.

Another advanced feature will be the viewer-specific replay generation. Based on information derived from the user profile, the PSE selects events in which individuals are specifically interested. Saving on production cost and taking into account the limitations and delay of content analysis, the duration, i.e. the length of the replay before and after the low-level annotation, is determined automatically based on the type of the event, according to the scenario specific event (ontology) model.

4.1.3 Definitions

The following brief definitions are relevant for the rest of the document.

Viewer group

FascinatE produces audiovisual content streams for a large number of viewers in parallel. Viewers receive different content because their devices have different playout capabilities (audio speakers, video resolution, aspect ratio…), a different connection bandwidth or they actively (interaction) or passively (derived from a user profile) influenced the PSE's content selection process to personalize the experience. Still, groups of viewers might work with the same parameters.

The SLMS keeps a hierarchical taxonomy of viewers called *Viewer Groups* which the PSE uses for (dynamically) creating/influencing parallel sub-processes on viewer group specific decision making. At the simplest level, at least one group per device type exists.

Shot (View)

A shot is rectangular area selected as a view of the current scene. It could be a crop within the OMNICAM panorama or the whole view (or also a crop) from a broadcast camera. The PSE usually defines shots as the outer bounding box of a certain area in which the concrete view (resolution, aspect ratio) rendered for playout has to be fit. Shots computed by the PSE have a basic priority according to their type in the domain specific Shot Type Ontology. Shots have a list of high-level events attached to them, which are assigned a priority value themselves, e.g. a list of persons detected in that area.

Production Script

Scripts are messages that are sent by the PSE and include instructions and metadata information. The core content are prioritized options of (virtual/physical) camera views – ultimately decision that instruct the renderer what to show. Scripts can be sent quite frequently, at most one per frame containing updates for

each viewing group. Information is sent from PSE instances to subsequent instances along the production chain (further refining of decision through re-prioritisation of options), to the DSE (preparation and optimisation of content streams) and of course the renderer. Receivers of scripts parse those parts which are relevant to them but usually do not send information back to the sender in a direct response. In FascinatE scripts are encoded as XML snippets and mostly transmitted via ICE interfaces.

4.1.4 Sources of Knowledge

The PSE uses different kinds of information on which it bases its decision making. Without any information from the outside, the PSE could only take random decisions. An *understanding* of the scene is necessary. However, it is never assumed that the information at the engine's dispense is perfect or complete. The following gives a brief overview of its potential sources:

Content Analysis

So far, AV content analysis relevant for the PSE has mainly been applied to the panoramic image. Details on the PSE's usage of analysis results from the broadcast cameras and audio are yet to be defined. The content analysis modules process a real-time content stream and emit a real-time stream of cues. It detects low-level cues in the content. Independent of the scenario, a person tracking module as described in [Kaiser, 2011a] is informing the PSE of the location of persons in the scene. Additional audiovisual analysis components include the extraction of a saliency measure for regions and the detection of scenario-specific events, implemented using a Machine Learning based classification approach. Results are transmitted using MPEG-7 (for details please refer to D3.1.2) via an ICE interface. MPEG-7 for that purpose implied certain disadvantages, however, none of the many other metadata standards that exist today is perfect, either.

Manual annotation

Further, the PSE is tightly integrated with a user interface for the professional production team, the Editor UI tools[5] (see screenshot in Figure 3). They allow creating live annotations for concepts not covered by AV content analysis. The main tasks are identity assignment for a subset of the person tracks (e.g. most interesting persons in a concert, sports match), live action annotation, steering the PSE through commands and selection between shot options provided by the PSE (effectively a re-prioritization). They could further serve as a tool for validating and correcting the results of content analysis. Manual annotation is expensive even for professional broadcast productions.

[5] To be designed by TII, implemented by JRS.

Figure 3: Design of the Editor UI, by TII. The tool will support several tasks of the broadcast team, such as live annotation of high-level events. For details, please consult D3.5.1.

Viewer profiles

Viewer groups are stored in the SLMS. Additionally, for each group or individual user, user profiles are needed. They will be stored in the SLMS and are queried by the PSE. Note that they are different from the dynamic user profiles that manage interaction and sit at the rendering component (for details please refer to deliverables of WP5 (especially D5.1.2). The interplay and synchronization mechanism between the two is yet to be defined.

Relevance feedback

The PSE could interpret viewer preferences that are implicitly expressed through interaction[6]. That channel however is currently not targeted and remains scope for future extension.

The Web

For now, utilizing personal information about the viewers to infer special interests and preferences and to use this information to further automate content selection personalization is out of scope for FascinatE. However, it would be a logical extension to the project especially for consumption on personal/mobile devices with a single or very few viewers. The FascinatE consortium is aware of the state-of-the-art and might consider the option of enhancing scripting automation via Social Network Analysis (SNA), public Web content analysis (e.g. information about the event that is broadcast) and extended user profiling. Such features might require explicit authorization by the viewers, and privacy issues must be respected. The information gathered could also be used to inform the viewer in the user interface.

[6] http://en.wikipedia.org/wiki/Relevance_feedback

4.2 Decision-Making Approach

This section will briefly explain the decision to implement a rule-based approach for a number of the PSE's subcomponents (details in subsequent Sections). We discuss pros and cons of approaches based on Description Logics, Machine Learning, and rule-engines/CEP. Section 4.2.4 summarizes related work regarding various aspects of the PSE's functionality. This Section neither intends to state a comprehensive list nor a comprehensive pro/contra comparison of alternate approaches; only main issues are highlighted.Description Logics based

Description Logics (DL) are a family of knowledge representation languages widely used for modelling ontologies and also for ontologies on the Semantic Web, the Web Ontology Language (OWL). Description Logics are decidable fragments of first order logics and allow representing the knowledge of our application domain in a formal and precise way by using concepts, individuals and related properties. Furthermore, it allows to formally reason over the facts to infer new knowledge. These features make the use of DL ontologies interesting for modelling the behaviour of the PSE. However, in the past we ran into scalability issues using DL for processing dynamic input to a knowledge graph of increasing size [Kaiser, 2011b]. Hence, we suggest using DL only for static models of the PSE.

4.2.1 Machine Learning based

Using Machine Learning techniques allows to evolve the behaviour based on empirical data e.g. from a ground truth. It would allow implementing a general algorithm that is applicable in several different scenarios. But, one of the problems is to create useful test data for learning the algorithms. An example test data for the PSE in this case would be to annotate a recorded content by different directors that state when they want to select which camera. As pointed out by [Al-Hames, 2007], selecting cameras and cutting videos can be done in many different ways, that are all reasonable and no objectively correct behaviour can be defined. Further problems come up when it is necessary to modify the behaviour; thereby the ground-truth must be adapted to reflect the new desired behaviour. This is a research challenge on its own, because such changes are not really deterministic and hard to foresee. Fine-tuning however needs to be fast so that adapted behaviour can be tested. We do not recommend Machine Learning approaches for the decision taking component of the PSE, however, it might be suited for sub-processes.

4.2.2 Rule-based Event Processing

The recent development of event processing systems [Etzion, 2010] which are based on the idea of processing events that include logic for filtering, transforming or detecting patterns in events, brought a lot of powerful and scalable systems. Furthermore, the combination with rule based systems brought a lot of expressivity in this domain. Rule based systems are used to store and manipulate knowledge to interpret information in a useful way. Event-Condition-Action (ECA) rules are well suited because they often fit with human thinking. Directors/experts are usually able to express their behaviour in natural language rules (ECA) which can be transformed into a formal language. Rules/principles might refer to domain concepts (real-life actions) that are not directly observable by content analysis (higher semantic level). Not all rules are general; most depend on a domain, a production, or a style. It is difficult to regard rules in isolation because the PSE's behaviour results only from their interplay. Rule based systems allow us to formalise the behaviour of selecting cameras in a way that is both executable by the PSE and flexible enough to be modified and fine-tuned by humans. These systems are fast, allow us to define temporal patterns over time windows and are our recommendation for the PSE.

The continuous queries within CEP modules are proven to be very fast, and most query languages allow to define powerful triggers on (spatio-)temporal patterns. It remains a research question how to effectively model larger corpora of rules (e.g. via layering), and, foremost, how to effectively resolve decisions based on contradicting/competing principles.

Several key decision making subcomponents of the PSE will be implemented using a rule-based approach.

4.2.3 Stream Reasoning

Another interesting technology is Stream Reasoning which combines the ideas of reasoning over Description Logic Ontologies and processing event streams like in complex event processing. Stream reasoning is the basis for a higher level decision making process that requires complex and real time reasoning over noisy data streams and rich background knowledge. Techniques of machine learning and existing reasoning mechanisms extended for continuous processing are used. Currently, this is an ongoing research topic with a lot of open challenges and there are currently only few experimental implementations that can be used [Stuckenschmidt, 2010], [Hoecksema, 2010].

4.2.4 Related Work Examples

A number of automatic view selection approaches using recorded content have been proposed, e.g. the interactive storytelling system proposed by [Kim, 2011] or the narrative structure language (NSL) [Ursu, 2008] developed within the NM2 project[7]. However, our research problem here is a significantly different one due to the real-time decision making requirement, the lack of comprehensive high level annotations and the lack of a thoroughly authored story frame. In the following, we discuss and compare a number of innovative systems.

The Production Scripting Engine processes a stream of events and makes decisions based on the events utilizing temporal reasoning. Event processing has been identified as a feasible solution to implement this as it can be integrated well with a rule engine. A comprehensive overview of this research field has recently been compiled in [Etzion, 2011]. A survey of methods for automatic interpretation and understanding video events is presented in [Lavee, 2009]. Methods of abstracting video data and how to model events are investigated. For event modeling, the authors cite Finite-State machines, Bayesian Networks, Hidden Markov Models, Dynamic Bayesian Networks, Conditional Random Fields, Grammar models, Petri Nets, Constraint Satisfaction and Logic approaches.

A Virtual Director commonly refers to an intelligent software system that attempts to automatically frame and select between multiple video camera views, essentially replacing a human director and camera operator crew. The Virtual Director in the TA2 [Falelakis, 2011] system[8] aims at taking such decisions in a different domain. The *Orchestration Engine*, as it is called in this FP7 project, selects between a range of streams in an attempt to support group-to-group communication in scenarios of social video conferencing. Thereby low-level event streams from audiovisual content analysis are processed based on a rule set that defines how to abstract from it for the detection of higher-level events. Both this and the decision making process are realized using the JBoss Drools event processing engine. A number of publications are listed on the project website.

The aim of the AMI and AMIDA Projects[9] is it to develop a browser for recorded business meetings and to enable remote business meetings. A typical meeting room in this project consists of a table, a whiteboard and a projector screen. In one room are four people where each person is captured by a fixed close up camera and a separate microphone. Additionally, there are three overview cameras (left, right and centre) and two microphone arrays for far-field recordings.

A Virtual Director is developed that permanently evaluates and selects the best suitable camera either for the remote location or for the recorded content to create a summary of the meeting. [Al-Hames, 2007] formulates this task as a pattern recognition problem and therefore they apply machine learning techniques using Hidden Markov Models and the Viterbi algorithm. The first layer of the Hidden Markov Model is fed with a global motion feature that preserves the major characteristics of the motion, a skin blob feature representing the activities of the participants hand and head movements and acoustic features including Mel frequency cepstral coefficients and the energy. The second layer models individual actions such as standing up, sitting down, nodding or shaking the head. Finally the Viterbi algorithm is used to segment the video stream into a sequence of camera switches.

The machine learning algorithms have been trained based on previously recorded and annotated content. The recorded meetings where annotated by different people. The authors note that there is a rather low agreement of the annotations between the annotators and therefore not consistent enough. They then decided to use only two sets of annotations to ensure a consistent training set. In an experiment the desired output was compared with the actual result of the system, with the result that the system selects the wrong camera (frame error rate) in 27% of the cases.

The APIDIS[10] project [Chen, 2010], [Chen, 2011] aims at automatically producing personalized multimedia content in the field of team sport. Personalization in this context means that the user's preferences such as the preferred team or player as well as the user's profile, history and device capabilities are considered for camera selections. The system builds on computer vision tools to automate the production of video summarisations. A multi view analysis is implemented that detects and tracks players automatically as well as a monitoring the scoreboard to recognise the main actions and the status of the sport game.

[7] http://www.ist-nm2.org

[8] http://www.ta2-project.eu

[9] http://www.amiproject.org/

[10] http://www.apidis.org

To produce semantically meaningful and visually appealing content the system balances between following three principles: (i) Completeness: that is the integrity of selecting a view and that of storytelling in summarisation. (ii) Fineness refers to the amount of provided details. Spatially it prefers close views and temporally it implies redundant storytelling. (iii) Smoothness is the graceful displacement of the virtual camera viewpoint and temporally appropriate switching between cameras.

The generic architecture for creating personalised video summaries consists of an online and an offline processing block. The first step of the online processing is to segment the videos based on game states as well as to identify salient objects and detect highlighted events. Then for each segment the production strategy is planned by determining the temporal boundaries, determining the view-type of each shot and inserting necessary replays. The last step is the camerawork planning where a viewpoint of a certain camera is selected by an optimisation function that considers completeness and fineness. The smoothness is applied afterwards with a two-layer Markov chain. The online processing block covers the video summarisation which includes the user preferences. It is modelled as a resource allocation problem that evaluates all optimal combinations of clips by their benefits and costs under the given user preferences.

Another algorithm for automated video production from multiple cameras is presented in [Daniyal, 2011] where the authors describe their object and frame-level feature ranking implementation. The problem of selecting cameras is modelled as a decision process during which information is only partially visible. Therefore the approach estimates object visibility scores and employs an optimal control policy for maximising the visibility over time and minimising the number of camera switches.

The implementation consists of a Partially Observable Markov Decision Process where the video content such as object size, location or scene activity is a Markov Process and the camera scheduling process is based on recursively estimating and updating the belief state. An additional reward model allows the approach to control the number of camera switches.

The proposed method was evaluated on a basketball match monitored by 5 cameras with partially overlapping fields of view. The outcome was compared to a manually generated ground truth with an overlap of 95,42%. The result of the subjective test is that 26 out of 31 people considered the automatically generated video is as good as the manually generated one.

The European FP6 project “LIVE: Live Staging of Media Events” [Jiang, 2011] allows the real-time involvement of users in live and interactive TV productions and it also changes the role for the traditional live TV director. In this project a director becomes a video conductor that communicates in real-time with his audience and therefore must direct in real-time the actors on the stage according to the audience’s mood to ensure a highest possible quality of entertainment. Collecting feedback of users enables modelling and tracking of the TV viewers within the TV production and allows the production of personalised content for the target viewer groups.

Viewers have the possibility in the interactive use case to give explicit feedback by voting, e.g. they choose one from the set of provided answers. Implicit feedback is automatically collected by tracking channel switches and watching times of channels. Based on user feedback and on other data following modalities are analysed: number of viewers per channel, preferences of the channel audience, analysis of the user groups on an observed channel, trends of viewers and voting statistics. The analysed data is an input for the production team which can then adapt the content. The developed system for live interactive TV production consists of following four main parts: A consumer’s IPTV application to display notifications to a specific group and allows the users to send feedback back to the director. The real-time notification and feedback channel transfers the viewer’s responses of the votes and channel switch information to the TV production team as well as the notifications to the IPTV application. The services for real-time analysis of the audience, including their preferences and responses are called feedback collection and analysis services. The adjustments and the results of the analysis for the production team are configured and displayed by the feedback application.

The live intelligent TV production use case aims at developing tools to more efficiently produce multi-channel live TV broadcasts. These include managing media assets, generating metadata of media items, conducting videos and provide intelligent decision support. One production team is then able to track and synchronize individual content items on up to five output channels in parallel. Therefore a knowledge based middleware is used to support and manage the production process in preparing and staging of media events. The knowledge model is enriched with low-level annotations extracted by an audiovisual analysis and high-level annotations generated manually by users. Then, a recommender system processes the knowledge model for on-the-fly content selection from the TV archives during live production for personalisation of TV channels according to the preferences of the target group.

A field trial was done at the Olympic Games Beijing 2008 with 489 users resulting that the interactive multi-channel TV format was positively rated by the consumers. Also the director stated in an interview that “the new real-time communication and feedback options became very important for him during the trail”.

The My eDirector 2012[11] [Patrikakis, 2010] project aims to create context-aware and personalized media but in real-time streaming environments for large scale broadcasting applications. This allows end users to direct their own coverage of large athletic events and to create their own personal Virtual Director. Raw video content is enriched with annotations generated by scene analysis, person tracking and other sensor data. This annotated media stream is used to make camera decisions based on the user's profile.

NoTube[12] [Schopman, 2010] brings TV to a community enhanced platform by connecting TV content and user data using Semantic Web technologies. A user profile models user activities is generated by aggregating data from Social Network activities, e.g. on Facebook or delicious. TV metadata is enriched with structured data from the Linked Data cloud. User recommendations are then generated with the average semantic distance between the interests of the user and potentially identified TV program items.

Within BBC's Automated Coverage project[13] another example of an approach to automate camera selection was investigated. A generic spring model was applied for camera framing such that moving objects were covered with smooth pans. Details about a shot framing algorithm will be described later in Annex B: Shot Framing Algorithm.

Overall, there are a number of activities researching into *beyond HD* which will increase the necessity of automatic camera selection behaviour. NHK's Super Hi-Vision[14] system (as exhibited at IBC 2011) for example offers an even larger resolution than the OMNICAM, though a flat image. NHK targets several features such as program customization, recommendation and Social TV services. However, it's not only the higher resolution but the wide range of potential new features that are noteworthy. A high degree of production automation is not only interesting for economic reasons but enables a range of features for the benefit of the viewer, parallelizing personalization capabilities and more.

4.3 Architecture

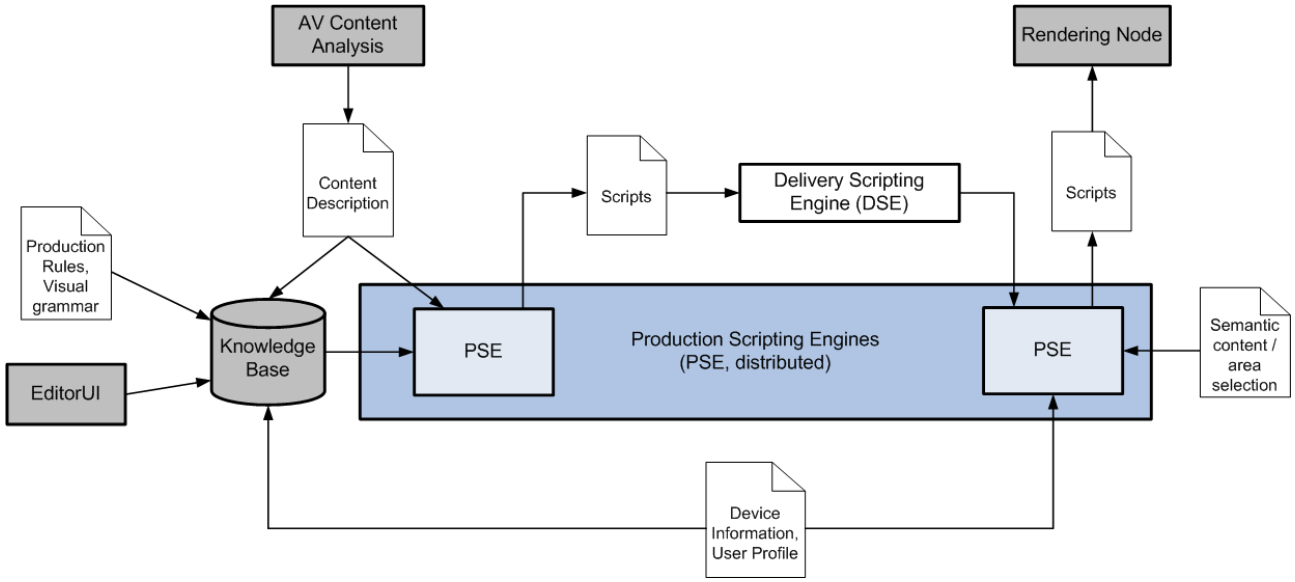


Figure 4: Simplified excerpt of the FascinatE system architecture.

The PSE is a distributed component. There is at least one instance at the production end, the primary PSE which consumes direct feeds from content analysis and the EUI, and also directly works with the SLMS knowledge base. It further has an interface to the DSE to inform it about which parts of the scene need to be transmitted with priority in order to render a specific Production Script. There’s also a PSE instance at the terminal end which has a direct interface to the rendering node. While the primary PSE basically emits a list of prioritized options, the PSE at the terminal end is responsible for taking the final decision. Depending on

[11] http://www.myedirector2012.eu

[12] http://notube.tv

[13] http://www.bbc.co.uk/rd/projects/virtual/automated-coverage

[14] http://www.nhk.or.jp/digital/en/super_hi/index.html

the scenario, more instances of the PSE can be added between them with the purpose of handling specific influences to content selection: an instance could check content rights and drop view options that certain viewers are not allowed to see. Another example would be to check potential privacy issues (no high zooms on the live audience).

The general role in the FascinatE architecture and role/purpose of the PSE has been discussed at length in previous deliverables. While there have been updates and refinements to the software design discussed in D3.2.1a, a lot of the general principles and thoughts still hold.

The PSE will be a Java application, available for other components as a service running on the servlet container Apache Tomcat[15]. JBoss Drools[16] is the event processing and rule engine which processes the main part of the business logic of the PSE that will be tested with JUnit[17] test cases to ensure a correct and stable behaviour. XML processing is done mainly with Apache XMLBeans[18]. Input and output interfaces are realized with the Internet Communication Engine (ICE)[19] (unless experiments suggest a different channel) and dynamic Web pages are used to monitor the PSE.

4.3.1 Renderer

The decisions and options emitted by the PSE can be effectively visualized by the FRN renderer as semi-transparent coloured overlay rectangles. This makes the FRN a powerful both for the production team (e.g. integrated into the Editor UI) and the developers of the PSE's automatic behaviour. It will be used for debugging, fine-tuning the camera selection behaviour, and subjective evaluations. The FRN offers a file based and a TCP/IP based interface. The latter will soon be extended to fully meet the PSE's requirements. For details on the FRN player see section 3.4 (Video Rendering) of Deliverable 5.1.2

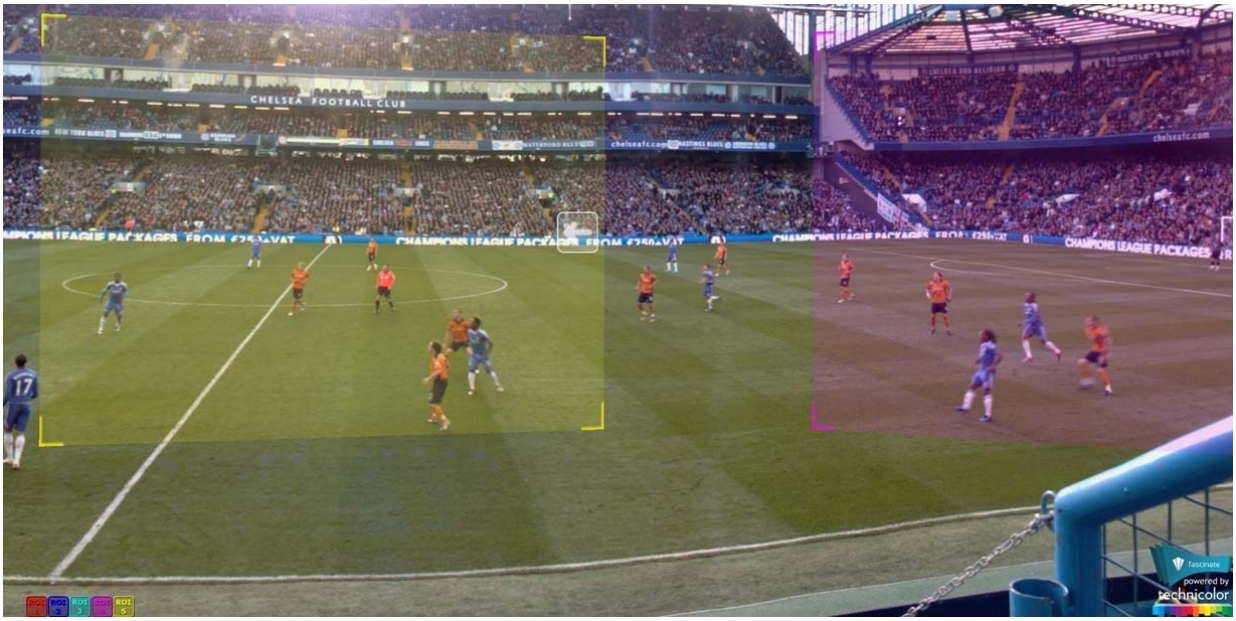


Figure 5: The FRN player showing virtual cameras as colour overlays.

[15] http://tomcat.apache.org/

[16] http://www.jboss.org/drools/

[17] http://junit.org/

[18] http://xmlbeans.apache.org/

[19] http://www.zeroc.com/ice.html

4.3.2 Rule-based behaviour modelling

We chose JBoss Drools[20] which is an open source business rule management system. It provides comprehensive functionalities for forward chaining rules and integrates event processing capabilities in an intelligent way. Drools uses an extended version of the Rete algorithm, called ReteOO, for efficient pattern matching. A pseudo example is given below:

```
rule "detect foul situation - rule 1"
  agenda-group "evaluation"
  salience 200

when
 $persA : GameRunningFact($gameTimestamp : timestamp);
 $persB : FoulEventReceived(type == START, xCoord == $xCoord, xCoord == $xCoord,
timestamp ==
 $gameTimestamp, $playerBID : playerBID) from entry-point "LowLevelCueStream"
 not(GameEvent(type == START, this after[0, 3000ms]) from entry-point "
LowLevelCueStream ")
then
  //send high priority event with coordinates to next level stream
end
```

The behaviour is a result of the interplay of *all* rules. Aesthetic behaviour for example cannot be added by a separate rule but has to be realized by a well-thought-out balance. This requires an iterative process where rules have to be added/modified and thresholds have to be adjusted. Decision making highly depends on its input knowledge given mainly by the AV content analysis and the Editor UI. The rule-base will consist of both general and scenario-specific rules. However, it has to be expected that all rules have to be configured, as it is the nature of such rule sets. To what extent this can be avoided remains to be investigated.

The shot framing and shot selection guidelines (see D3.1.1 section 4.4) will be represented as a set of rules, parameters and formulae that can be used by the rule processing engine to generate one (*decision*) or more (*options*) recommended framed shots.

For the purposes of defining the kind of metadata needed for shot selection and framing, here we consider only how to frame a shot, given the type of shot to use and the region(s) of interest that should be shown. The choice of shot type and region(s) to show may be made by a human operator via the Editor UI, or could be subject of a higher-level automated decision process within the PSE, tailored to a particular kind of programme.

4.3.3 Interfaces to other FascinatE components

The interfaces of the Production Scripting Engine have been described in Deliverable D3.1.2, mainly in section 4.3. Details on their implementation (using MPEG-7, ICE, Production Scripts, etc.) are discussed throughout Section 4 of this document.

4.3.4 Configuration

Most of the PSE's configuration will be stored in the Domain Ontology. It may be read locally (for implementation, testing) or retrieved via the SLMS (live).

Shot type ontology

The shot type ontology defines shot types and some metadata such as their basic priorities, basic size around enclosed high-level ROIs, etc. The OWL2 model will be loaded by the PSE at start-up as domain-specific configuration. It is essential for a number of sub-processes, most notably the Semantic Lifting which continuously executes rules for detecting high-level concepts, and the Shot Candidate Identification components, which decides when shots should be created/changed to cover a certain action. The development team will experiment to find out if a graph structure is beneficial for that purpose or a hierarchical tree of shots is more advantageous.

The shot type model will be part of the Domain Ontology.

[20] http://www.jboss.org/drools

Static shots

There is a list of fixed predefined static shots for a domain. They are shared amongst viewing groups, though some may only be selectable for specific device types. They include the full image of the broadcast (side) cameras and certain regions of the OMNICAM panorama (e.g. goal area for soccer). These shots are available for the whole duration.

The predefined static shots will be part of the Domain Ontology.

Dynamic shots

For each domain a set of dynamic shots is defined which are triggered on specific occasions. An example would be an audience pan of three seconds that can be shown after an action of type “slam dunk” has been detected. It may be triggered when certain concepts are detected and are assigned with certain priority.

The predefined dynamic shots will be part of the Domain Ontology.

Sensor information

Basic information about the cameras and microphones (resolution, aspect ratio, etc.) is needed for the PSE’s content selection processes. The information is considered fixed for a broadcast will be loaded from the SLMS at start-up time.

4.4 PSE Sub-Components

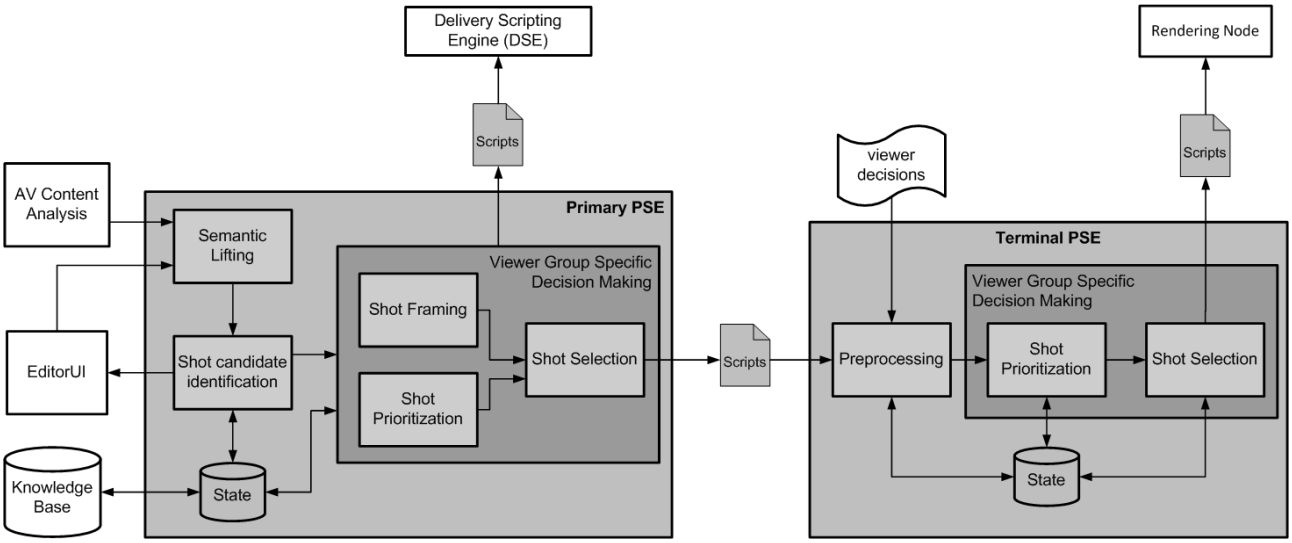


Figure 6: Internal software design of the PSE, illustrating the subcomponents’ interplay (grey boxes). The PSE is a distributed component – the left block depicts the primary PSE at the production end with access to the content analysis live cue streams and the EUI. The right area depicts the PSE instance at the terminal end which sends final instructions to the renderer.

The Production Scripting Engine (PSE) is responsible for decision making on content selection. Its internal architecture is depicted in Figure 6. Subcomponents and the information flow between them are indicated. To repeat, the key feature is to automatically select a suitable area within the OMNICAM panorama image, in addition to cuts to humanly operated broadcast cameras. Selection behaviour is based on pragmatic (cover most interesting actions) and cinematographic (ensure basic aesthetic principles) rules, comparable to the approach proposed by [Falelakis, 2011]. The decision making process is not always fully automatic but can involve supervision by a human-in-the-loop, a production team member deciding between prepared options using the Editor UI tools. The PSE is a distributed component with at least one instance at the production site and one at the terminal end. More complex scenarios will require additional levels of decision making in between, taking into account licensing and content rights, targeted advertisements, event audience privacy issues and such. This is also reflected in the architecture diagram in Figure 4.

The primary PSE is consuming real-time low-level event streams as extracted by AV analysis and generated by the Editor UI (near real-time annotations). Low-level means that the information bits are per se not directly usable for the decision making process, as they are very narrow facts/statements about details of the scene. What the PSE really needs to have in order to take quality scripting decisions is a more abstract understanding of the current situation, though. As an example, a sequence of coordinates of players and a ball are relatively meaningless in isolation. However, a certain constellation might indicate a very high

probability that a basket/goal is to be scored in the upcoming seconds, and the PSE might want to react to such a situation in a specific way.

The output of the PSE is called a *script*, which consists of a combination of content selection options and decisions, renderer instructions, user interface options a.s.o. Scripts (custom XML format) are passed to subsequent PSE components from the production site towards the terminal, where final instructions are given to a device-specific renderer. The Terminal PSE processes scripts received from previous PSE instances. When the rules allow doing so[21], it determines a single shot update per viewing group and sends final decisions as instructions to the renderer.

All PSE instances keep a local state to keep track of their own decisions, which is required for a number of features, e.g. a certain shot variety as an aesthetic principle. Different shots (fixed, dynamic as e.g. following an object) and shot types will be modeled as an OWL2 ontology. Event types of this controlled vocabulary are specific to a scenario. In the example of soccer content they include for example different zoom level side views, player closeups, audience cheer pans, fixed goal area shots etc.

Most of the PSE's subcomponents will use JBoss Drools[22] which is a unified and integrated platform for rules, workflows and event processing. It is a hybrid chaining rule engine implementing a forward and a backward chaining inference engine. The forward chaining rules are processed using an extended version of the Rete matching (see [Forgy, 1982]). The business logic, the rules, can be expressed in a declarative way by using the native Drools Rule language, a XML rule language or a self-defined domain specific language.

Summarizing, the Editor UI and the AV content analysis are the main metadata (knowledge) sources for decision making within the PSE. The following sections will describe the individual subcomponents of the PSE.

We chose to separate the sub-processes of Semantic Lifting (abstracting from low-level input), Shot Candidate Identification (find suitable shots), Shot Framing (smoothly follow ROIs), Shot Prioritization (weigh a number of factors) and Shot Selection (decision making). The following explains each sub-process in more detail.

4.4.1 Semantic Lifting

Input: real-time low-level cue stream (multiple cues per frame); high-level events annotated by the Editor UI. (Soccer content: list of player bounding boxes, tracked; ball: manually annotated for offline tests)

Output: events of a higher, more abstract semantic level, as defined in the ontology (Soccer content: FocusArea; FocusAreaDirection; GlobalDirection; etc.)

Frequency: whenever detection happens, for movement observation updates in constant intervals of n frames;

Approach: use JBoss Drools CEP-enabled rule-engine; prepare incoming cues, add to event stream, map to internal coordinate system where necessary; use an artificial raster[23] over the image; spatial query in which area most people are; detect global direction of persons over last n frames, local area direction of persons over last n frames, etc.

The Semantic Lifting component is designed to receive low-level events from both AV content analysis and the Editor UI component. It is the *frontal* interface of the PSE subcomponents and processes event streams, therefore it must be able to handle a high volume of input events. Low-level events from the AV analysis are sent in MPEG-7 format[24] and include bounding boxes of detected persons.

The purpose of the Semantic Lifting component is to filter and enrich (metadata such as confidence values) the incoming information. Further, higher-level events are extracted from the low-level event stream. This can be achieved in various forms, e.g. by observing trends and sudden changes, by detecting predefined patterns in the event stream, by fusing audio and video events, by doing fuzzy classification, or by a combination of these methods. As a result the Semantic Lifting component outputs events of a higher semantic level that can be directly used to determine shot candidates.

[21] Aesthetic rules ensure e.g. that cuts between shots don't happen too often.

[22] http://www.jboss.org/drools

[23] Optimal size to be identified through experiments.

[24] Multimedia Content Description Interface, ISO/IEC 15938:2001

After defining the first set of rules and sketching the system's architecture we identified following requirements - the Semantic Lifting component must be able to:

- Process event streams in real-time.
- Handle possible out of order events.
- Reason with uncertain information (i.e. confidence values of low-level events).
- Express and reason on First Order Logic statements, as predicate logic is not sufficient.
- Perform spatial, spatiotemporal and temporal reasoning including Allen's temporal operators [Allen, 1983].

4.4.2 Shot Candidate Identification

Input: high-level events as defined in the ontology

(Soccer content: player bounding boxes (tracks), ball bounding box (track, manually annotated))

Output: list of ROIs (extent based on shot type ontology)

(Soccer content: nr of shots depends on situation, nr of viewing groups)

Frequency: updated every frame

Approach: manage list of currently used shots; analyse scene actions as indicated by high-level events; decide if shots should be added or removed based on priorities and predefined min/max number of concurrent shots allowed;

The shot candidate identification aims to determine suitable candidates based on the high level event information as provided by Semantic Lifting. The output are not final decisions, but options that subsequent components use to take decisions for each individual user group based on a range of factors prioritizing different types of shots and the event/action types assigned to them. The component determines at least one candidate; the actual number depends on the scenario. We have yet to determine the optimal number given further parameters such as the Editor UI integration. In that frontend for the production team, options are likely to be visualized as rectangular colour overlays on the panoramic image. This seems to work well for visualization and to assist decision, but only for a limited number of views at the same time.

For the identification itself a number of domain-specific rules are executed by JBoss Drools to immediately and properly react to actions in the scene. As previously mentioned, we are using our event/action ontology for classification of interesting actions, so that we can determine the most interesting ones. Even though the dynamic occurrence of unexpected events might require additional views to be added at times, one strong design principle is to ensure continuity, to work with a rather constant number of views. The initial design of this component will be revisited as soon as practical lessons learned allow deriving its ideal behaviour.

4.4.3 Shot Framing

Input: list of ROIs (Soccer content: player bounding boxes (tracks); initially use ball track for debugging.)

Output: list of shots - bounding box that will contain the actual camera framing independent of the viewing device where possible (Soccer content: virtual camera smoothly following player, ball, group of players etc.)

Approach: smoothing based on a spring model; for non-fixed shots determine current position each frame

For each shot candidate the Shot Framing component computes a suitable framing bounding box. Size and aspect ratio differ per viewing device and the decision gives the coordinates of the surrounding box. The component deals with both fixed shots and such covering moving objects for which *smooth* camera pans are needed. The calculation employs a spring model for smoothing out minor movements and avoiding rough stops. It takes the object type, direction and speed of movement into account. As an example, a horizontally moving athlete is positioned side of the image centre so that more of the running direction area is seen. Further, the bounding box size depends on the distance of the object to the camera, i.e. more distant objects are covered by smaller boxes so that they appear larger.

The PSE is closely working together with another type of Scripting Engine, the Delivery Scripting Engine (DSE). The DSE can access the rendering instructions from the PSE. The DSE may for instance decide to pre-render or not some specific shots before delivery. In general it decides how to prepare content streams and makes sure needed content is available for the renderers, optimizing bandwidth management.

4.4.4 Shot Prioritization

Input: list of views (options) assigned with metadata and priorities, per viewing group; re-prioritization of views via the Editor UI (manual decisions)

Output: list of views (options) assigned with metadata and (recalculated) priorities, per viewing group

Approach: (re-)calculations based on configuration, Shot Type Ontology

Naturally, shot prioritization and selection have to operate per viewer group as well. In our current architecture, the Shot Prioritization component is a preparation step within the workflow which continuously (re-)assigns priorities to shots based on a range of factors. One example would be a PSE instance executing content rights – any view that is disallowed would be set to priority 0 – which ensures that the viewer will never be able to see this part/view of the scene through the PSE (by whatever it would be multiplied).

4.4.5 Shot Selection

Input: list of views (preliminary decision plus options) assigned with metadata and priorities, per viewing group

Output: updated list of preliminary decision and options; list of options per viewing group may become shorter as invalid options might be identified for example (e.g. content rights); the decision at the terminal PSE will become an instruction for the renderer (final decision).

Approach: eliminate shot options based on user input, right restrictions etc; desired behaviour has to be captured (rule-based approach means knowledge elicitation from experts rather than learning from test data) and formalized (rule-engine specific); selection component has to be built that implements those principles – modelling the behaviour is a non-trivial research problem;

The Shot Selection component consumes that input and either takes a final decision or computes a set of suitable options to be sent to the next PSE instance in the network. The terminal PSE for example has to take final decisions; other PSE instances may pass on a list of options. Options might but not necessarily have to correspond to preferences in the viewer's menu. Shot selection takes the model from the viewer profile and of the shot's associated metadata into account. The recent viewing history is also a factor e.g. to ensure variety in types of shots ensured by cinematographic rules. At the terminal end final decisions have to be made, therefore device specific transition commands are included in the scripts as well. Decision scripts are passed to the renderer.

4.4.6 Utility components

The following highlights the features of the most important utility components.

MPEG-7 Binding Component

To bind the MPEG-7 XML document to an object oriented hierarchical class tree we use the Apache XMLBeans[25] that creates a binding based on the XML Schema. In our case we use the MPEG-7 Audiovisual Description Profile (AVDP) from the upcoming ISO/IEC 15938-9:2005/PDAM standard. The main functionality of the AVDP is to describe the results of automatic media analysis with low-, mid- and high-level features for audio and visual content. Typical application scenarios would be shot/scene detection, face detection/tracking, speech recognition or summarization. Processing the MPEG-7 Document with Apache XMLBeans can be compared to the C++ MPEG-7 Library[26] which also allows to serialize and deserialize a MPEG-7 XML Document in an object oriented way.

Coordinate Transformation Component

Different coordinate systems are used in the FascinatE system. The content analysis of the OMNICAM for example processes the 6 tiles independently before fusing results to information about the whole panorama. The FRN renderer on the other hand regards the panorama as 6 independent projections into the spherical space and the interface accepts vectors that are basically offsets to the centers of the tiles.

Internally, the PSE uses a coordinate system over the OMNICAM panorama with ‘0,0’ in the left top corner. Same holds for the broadcast cameras. The Coordinate Transformation Component merges the different coordinate systems into one big coordinate covering a complete video scene. This enables it to apply spatial reasoning in one video scene. When using the FRN renderer for debugging and other reasons, coordinates from the internal format have to be converted to the FRN coordinate system. Details about the coordinate system of the renderer can be found in the deliverable 2.1.1, section 3.2.

[25] http://xmlbeans.apache.org/

[26] http://mpeg7.joanneum.at/

Communication Interfaces

For communicating with other FascinatE components such as content analysis, we use the Internet Communications Engine (ICE)[27] that is a highly efficient middleware, an alternative to CORBA and SOAP. ICE is available under a dual-licensing scheme including the GNU General Public License and a commercial license. It supports a variety of programming languages such as Java and C++. A detailed description about the incoming and outgoing interfaces can be found in Deliverable 3.1.2, section 4. XML-RPC is another suitable candidate.

4.5 Production Scripts

Production Scripts are information snippets that are sent from the PSE to further PSE instances or other components, such as the DSE, the SLMS or the renderer. They contain information/metadata and instructions for their intended receivers.

For the representation of Production Scripts, we created a custom XML format to describe the diverse decisions and information bits. Reflecting the iterative development process, the script content will be extended as scenarios and component capabilities grow throughout the lifetime of the project.

The PSE scripts contains most notably:

- Configuration and metadata on cameras, viewer groups
- Decisions on views (camera, region) to be selected for a certain playout device type, group of viewers or individual viewer.
- Options for view selection (plus priorities, metadata, ROIs contained) – to be decided by subsequent PSE instances.
- Knowledge necessary for further PSE instances in the network to take decisions (e.g. ROIs contained).
- Cutting instructions (none, fade, rough cut, …).
- Information on view priorities and saliency, which allow the DSE to optimize the preparation of the content streams for delivery and decide if new content streams, e.g. closeup-view following a certain person, need to pre-rendered before delivery..

The question if instructions/options for the playout device user menu are embedded in the production scripts or sent as a separate stream remains to be decided depending on the scenario, the script frequency (every *n* frames or event-based) and the viewer menu design. One of the scripts' purposes is to inform the DSE about relevant regions so that bandwidth optimization can be achieved.

A script example is available in Annex A

4.6 Status & Research Challenges of the PSE

We have discussed the detailed architecture for a *Virtual Director* implementation that supports a format-agnostic live event broadcast production system and is able to frame shots within a high-resolution panoramic video stream. The architecture design and technology choice has been informed by lessons learned in a number of related activities (cp. [Falelakis, 2011]). We are in an early stage of the implementation of the Production Scripting Engine and are working on several subcomponents in parallel. We soon expect a working demo to be available that uses the FRN renderer to visualize the automatically framed camera views. We plan to test and evaluate event processing and rule evaluation performance, system delay and viewer satisfaction but are yet to define those experiments in detail.

Developing a rule-base that realizes the desired behaviour is not a straightforward engineering task. There is little related work on the formal representation of the principles that broadcast teams operate on. Such principles are typically expressed as event-condition-action (ECA) rules that consider their context in the condition, but are still expressed independent from each other. However, the challenge is to achieve a sound behaviour by the *sum* of the rules, their interplay. As we target different scenarios (mainly sports events and music concerts), effective re-use of a subset of rules is desired. One example in that regard should be the ability to smoothly follow an object on close-up moving through the panoramic image.

These issues will mainly influence the implementation of the Semantic Lifting and Shot candidate identification components and likely reveal interesting research questions. FascinatE further aims for intelligent audio orchestration. As an example for sports broadcast, if there is an audience shot after a

[27] http://www.zeroc.com/ice.html

successful score, the audio should correspond (loud cheers). More general, a viewer may want to hear the fans of his/her favourite team more than those of the opposing - the visibility of objects should correspond to their audibility. Objects that are currently not visible may not be audible at all or at a dimmed level. Audio close-ups might become problematic though, as they may interfere with the privacy of a player-coach discussion, or may cover inappropriate language of players, etc.

4.6.1 Possible issues going forward

To get useful output of the event processing component it is necessary to process the events in the order they occur ([Etzion, 2010]). But, due to the reason that we have several distributed audiovisual analysis machines we must assume that we get the detected events out-of-order. This is because every machine has its own clock that is used to assign a timestamp to the generated event and the clock might be out of synch in comparison the other machines. Furthermore, the detection of an event may be delayed due to thread scheduling of the operating system and the transmission over the network takes also some time that is varying.

JBoss Drools for example is able to process events retrospectively, meaning when it receives a delayed event it processes it in the correct timely order but it is not aware of the correct sequence of the events. Solutions to create a correct event order in a distributed event-based system would be:

- To synchronize the clock of the machines with a time server. This reduces the possible delay but might not entirely solve the problem which depends on the temporal granularity of the rules and queries. The task is not as trivial though as one might assume.
- To implement a buffer and a timeout mechanism. The buffer incorporates a delay and the timeout mechanism ignores events over a certain threshold. This is only applicable where delays and ignoring events can be accepted.
- To compensate a wrong decision in retrospect. This approach executes the desired behaviour immediately but corrects it possibly afterwards with undo and redo operations. This is not always possible.

In our situation it is sufficient if the clocks of the analysis machines are synchronized with a time server. If we register problems with the order of events we could implement a buffer with a small delay.

4.6.2 Research challenges

The research problem of a *Virtual Broadcast Director* for automated coverage of live events consists of several core challenges (*subjective selection and classification*):

Formal representation of desired behaviour

Ideally, the knowledge and sense of human directors, camera operators and other staff of the broadcast team would be *downloaded* to a database for automatic execution – that is of course not possible as such. However, visual production grammar, sets cinematographic principles, to a certain extent can be captured and modelled.

The elicitation of desired production behaviour is an elaborate process. The behaviour can be categorized into several types. As an example, there will be *pragmatic* principles making sure that the most interesting actions should be selected. On another level, *aesthetic* principles will ensure that the visual output obeys basic rules. *Cinematographic* principles will state how a certain shot should be framed a.s.o. We decided to capture the behaviour in the form of ECA (event - condition - action) type rules because it is both an intuitive format for video production experts (directors, camera operators, experts for camera framing, …) to state their principles and, further, such rules can be processed by machines. The rule definition process will be driven with the help of domain experts who together with rule developers state an initial set of rules in natural language. Rule developers will then use a semi-formal intermediate notation as a basis for crafting the rules in a notation specific to a rule engine. It can be evaluated to a certain degree how well the desired behaviour of the experts has been implemented in the rule-engine's specific logic.

In a nutshell, the research community is currently lacking standards and design patterns on how to model and formally represent complex behaviour such as targeted in FascinatE.

Pre-processing of low-level event streams

While sensors typically emit low-level information bits, directors think in more abstract concepts. A (limited) scene understanding can be achieved by abstracting the input to a higher semantic level. The PSE aim to bridge the *Semantic Gap* through detection of certain situations, inference, spatiotemporal patterns, etc. On a real-time stream of events, higher-level concepts relevant for production grammar have to be detected via continuous queries on the recent time window. Complex Event Processing (CEP) technology is well-suited

for that task. Processing the stream, the queries detect certain occurrences and changes/trends, identify spatiotemporal patterns of low-level events or semantically infer high-level concepts. The task is challenging because it operates on fuzzy, uncertain knowledge.

Real-time decision making

This challenge is to take real-time decisions by continuously evaluating the former two against each other – the desired behaviour versus the *understanding* of the current situation. To take decisions, competing & contradicting principles have to be solved. Immediate reaction to high-priority events is needed (maximum overall system delay), however, to keep a visually pleasing balance over time, aesthetic principles might prevent the system from deciding to do so.

Because a purely reactive system might not be able to create good enough viewer satisfaction (events missed due to delay etc.), a basic level of prediction intelligence is desired.

Evaluating a Virtual Director

Evaluation of the Production Scripting Engine's performance is inherently difficult. There is no objectively *right* sequence of decisions – there is an infinite number of appropriate ways how to broadcast an event. Some decisions can be regarded as objectively wrong in isolation; however, in a sequence of decisions they might be the best of a set of imperfect options.

Evaluation of the PSE will be done in several ways:

- Evaluate the performance and scalability of individual subcomponents.
- Compare different approaches regarding such indicators (delays, precision/recall of decision etc.).
- Evaluate the user experience with real test viewers as subjective evaluations (experiment details yet to be defined).

The PSE's evaluation, much like the evaluation of the overall FascinatE system, has to take delays of individual components and overall (not everything is on a critical path) into account. On the system level, it has to be insured that delays don't get components out of sync. Further, it has to be ensured that delays are not unnecessarily propagated through the engine or have detrimental effect on the overall user experience – a potential threat especially with the rule-based approach we chose for several sub-processes.

On the software level, the PSE will continuously be JUnit tested for all implemented rules, testing both the correctness and speed of execution, with recorded dummy data. Stress tests (e.g. extremely high number of low-level input cues) will also be performed. Making general performance statements over different event processing implementations is difficult due to different requirements of applications and outside influences on performance metrics. In a rule-based environment the performance metrics are mainly influenced by the number of events per time unit, the number of attributes of an event, the number of rules in the rule base, the complexity of the rules, the current number of entities in the working memory, etc.

Important performance objectives within the PSE's real-time requirements are the average latency and the maximum latency of making decisions based on incoming cues as well as the deviation in latency. However, we cannot straightforwardly measure the latency of the whole processing chain. The reasons are (i) temporal operators in the rules and (ii) rules which deliberately wait for other decisions, e.g. rules that ensure that at least two seconds are passed between two cuts in order to prevent odd screen grammar.

To improve the development process, more support from IDE tools is desired. We will develop additional tools assisting development and monitoring, e.g. cue event stream visualizations for basic visual analytics. They will help speed up the implementation processes, especially fine-tuning the overall behaviour. In general, there are no well-established standards for expressing desired camera selection behaviour.

For subjective evaluations we plan to directly use the FRN renderer. Options can nicely be visualized as semi-transparent boxes, for viewing a concrete decision; any of them can be followed full-screen.

5 Conclusions & Summary

This document clarifies how the Production Scripting Engine and the Semantic Layer Management System have been designed in order to fulfil the project requirements. Part of their features will be implemented for the final version of the system (milestone M7), therefore this is a living document, finalised in the D3222 deliverable.

Regarding the Semantic Layer Management System we have introduced the component architecture, describing the sub-components and explaining their functionalities.

Regarding the PSE we have discussed the rationale behind implementation decisions for subcomponents of the PSE using event processing algorithms and expressing camera framing principles as a set of interdependent event-condition-action rules. We chose to logically separate the steps of Semantic Lifting (abstracting from low-level input), Shot Candidate Identification (find suitable shots), Shot Framing (smoothly follow ROIs), Shot Prioritization (weigh a number of factors) and Shot Selection (final decision making, decide when to cut). The PSE takes decisions on the level of Viewer Groups. The knowledge driving these decisions is based on semantically lifted low-level cues originally extracted by content analysis and on manual annotation through the Editor UI.

The core challenge of designing and developing the PSE is to process an event stream in real-time conditions and take decisions based on a complex, predefined and exchangeable set of rules (production grammar, behaviour). Several delay factors have to be respected and affect the experience. Beyond the pragmatic aim to show the most relevant actions the rules aim to ensure aesthetically pleasing camera selection behaviour. Automation in that regard is a multifaceted research challenge. It has to be kept in mind that the information available to the PSE, in effect the level of *understanding* about the current situation, is rather limited compared to what human operators are able to grasp. The advantage of the Production Scripting Engine is that is can help produce a large number of parallel, individualized streams instead of only one single video stream, as typical for most TV broadcasts.

Automated view selection behaviour (*Virtual Director*) is inherently difficult to evaluate. The central issue with evaluation is that there's no ground truth, no *ideal* camera selection that could be compared against on the level of individual decision. Setting up qualitative evaluation experiments is an elaborate and expensive process where a lot of effort to limit outside factors is necessary.

6 Annex A: Production Script Format

The following XML-code depicts a production script example (with comments explaining what the individual sections are containing).

This format will be extended as the components' requirements increase.

```
<?xml version="1.0" encoding="UTF-8"?>
<PSEScript>

  <!-- Definition of camera sources. -->
  <cameras>
    <camera id="17" type="http://www.fascinate-
project.eu/domainOntologies/soccer#OmniCam" resolutionX="6984" resolutionY="1920"
fps="60"/>
    <camera id="17" type="http://www.fascinate-
project.eu/domainOntologies/soccer#broadcastCam" resolutionX="1920" resolutionY="1080"
fps="25"/>
  </cameras>

  <!-- Definition of the current viewer groups. -->
  <viewerGroups>
    <viewerGroup id="1" type="http://www.fascinate-project.eu/domainOntologies/soccer#TV"
deviceResX="1920" deviceResY="1080"/>
    <viewerGroup id="2" type="http://www.fascinate-
project.eu/domainOntologies/soccer#TimeLab" deviceResX="6984" deviceResY="1920"/>
  </viewerGroups>

  <!-- List of generic shot options. Outer bounding boxes are given. Concrete
resolution/aspect ratio depends on device capabilities per viewer group. -->
  <genericViewOptions>
    <viewOption id="23" camID="17" dynamic="yes" basicShotPriority="48" topLeftX="0"
topLeftY="0" bottomRightX="400" bottomRightY="300">
        <ROIsContained>
          <PSEROI id="54" type="http://www.fascinate-
project.eu/domainOntologies/soccer#person" priority="75" topLeftX="14" topLeftY="12"
bottomRightX="54" bottomRightY="35"/>
        </ROIsContained>
      </viewOption>
  </genericViewOptions>

  <!-- List of shots that are currently available for a certain viewer group. Decision is
found via priorities which are continuously re-evaluated. -->
  <decisions>
    <decision viewerGroup="1" action="keepShot">
      <prioritizedOption>
          <concreteOption viewOptionID="54" priority="91" topLeftX="0" topLeftY="0"
bottomRightX="400" bottomRightY="300"/>
            <concreteOption viewOptionID="23" priority="78" topLeftX="200"
topLeftY="150" bottomRightX="600" bottomRightY="450"/>
            <concreteOption viewOptionID="33" priority="50" topLeftX="354"
topLeftY="650" bottomRightX="754" bottomRightY="950"/>
        </prioritizedOption>
      </decision>
      <decison viewerGroup="2" action="roughCut" offset="0" cutToViewOption="54">
          <prioritizedOption>
          <concreteOption viewOptionID="54" priority="91" topLeftX="0" topLeftY="0"
bottomRightX="400" bottomRightY="300"/>
```



```
            <concreteOption viewOptionID="23" priority="78" topLeftX="200"
topLeftY="150" bottomRightX="600" bottomRightY="450"/>
            <concreteOption viewOptionID="33" priority="50" topLeftX="354"
topLeftY="650" bottomRightX="754" bottomRightY="950"/>
        </prioritizedOption>
    </decison>
  </decisions>

  <!-- Which shot has been shown for how long. Keeping track of the recent history
enables to execute aesthetic grammar. -->
  <viewHistories>
      <history viewingGroup="12">
     <cut order="1" timeCode="01:35:45:00" viewOption="1" transitionType="slowFade"/>

     <cut order="2" timeCode="01:30:12:14" viewOption="3" transitionType="roughCut"/>
      </history>
  </viewHistories>

</PSEScript>
```


7 Annex B: Shot Framing Algorithm

The shot framing and shot selection guidelines (see D3.1.1 section 4.4.2) will be represented as a set of rules, parameters and formulae that can be used by the rule processing engine to generate one (*decision*) or more (*options*) recommended framed shots.

For the purposes of defining the kind of metadata needed for shot selection and framing, here we consider only how to frame a shot, given the type of shot to use and the region(s) of interest that should be shown. The choice of shot type and region(s) to show may be made by a human operator via the Editor UI, or could be subject of a higher-level automated decision process within the PSE, tailored to a particular kind of programme.

It is proposed that the shot framing process be broken down into two steps: (1) given a bounding box containing an object to be framed, determine the key point-of-interest on the object and the size of the object, and (2) choose the location in the image at which this point of interest (or multiple points of interest) should appear, and the size at which the object(s) will appear (i.e. degree of zoom).

7.1 Determination of Point-of-Interest from Bounding Box

It is assumed that each object of interest is defined by a bounding box that contains the object, for example generated from the content analysis tools that detect groups of moving feature points, or from an approach such as foreground/background segmentation. An approach that was used successfully in a previous project[28] gives an indication of the kind of rules and parameters that may be used. The point-of-interest was determined by considering a bounding box to be a projection into the image of a 3D bounding cuboid. The location of the point-of-interest within the bounding cuboid was determined based on parameters that were a function of the type of object being followed. The bounding cuboid size and location were represented in 3D world coordinates, derived from image analysis from one or more calibrated cameras, using additional assumptions or constraints such as the known height of an object or a constraint that its base should touch the ground plane. These constraints were also a function of the object type; for example an object of type "person" could be constrained to be in contact with the ground whereas an object of type "ball" need not be. The point of interest was determined from parameters specifying the relative location within the bounding box (e.g. the central point of the top of the cuboid for a person, the centre of the bounding cuboid for a ball), with an optional fixed offset (e.g. 15cm below the top for a person – to correspond to their eyes, but no offset for a ball).

The metadata likely to be needed to specify the point of interest therefore comprise:

- a three-component vector to specify a point within the bounding cuboid, expressed as an offset from the lower near-left-hand corner in terms of the fraction of the lengths on each axis
- a three-component vector providing specifying the offset of the point-of-interest from this point expressed in metres.

Each of these can be specified to be a function of the type of object, e.g. ball or person.

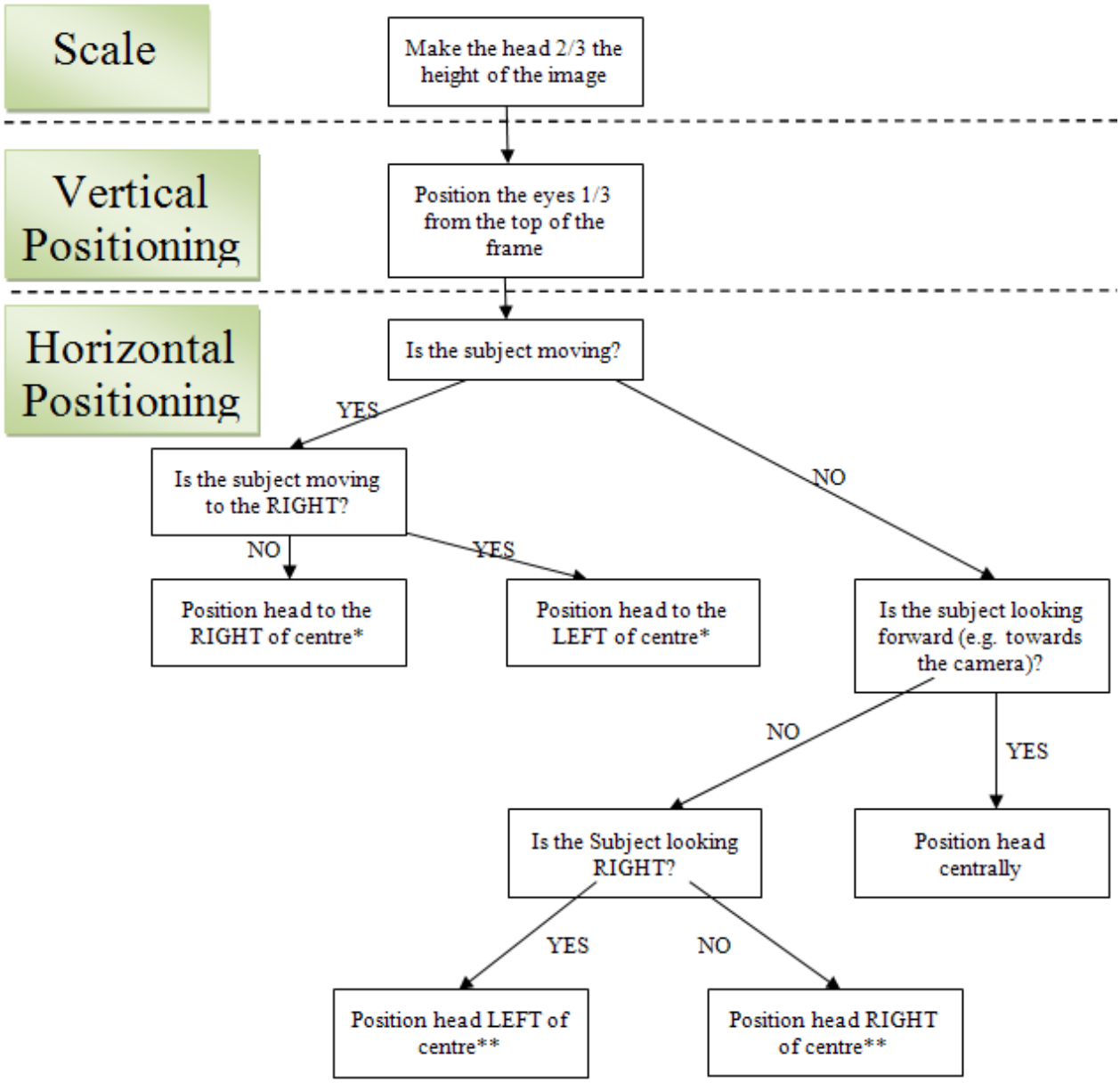
7.2 Framing the Point-of-Interest within the Image

Broadly speaking there are two main stages to framing a shot. Firstly, the type of shot needs to be chosen; this could be a close up, framing the action, a wide-shot etc. Secondly, once the type of shot is chosen, the shot needs to be composed (i.e. the actual settings of pan/tilt/zoom for the virtual camera need to be determined). In this section, an algorithm and associated parameters are defined with the assumption that the (virtual) camera can be panned, tiled and zoomed but that its position cannot be altered, corresponding to the reframing that can be applied to an image from an OMNICAM or a broadcast camera mounted on a fixed pan/tilt head.

Each different type of shot can be expressed as an algorithm that defines how it is framed based on the position of the subject(s). Below we consider four types of shot that are used to cover one or more people (close-up, ultra close-up, mid shot, wide shot), and a shot to frame an area of "action" that contains multiple objects and/or people.

[28] The BBC R&D project "AMICOS" on automated camera control: www.bbc.co.uk/rd/projects/virtual/automated-coverage

7.2.1 Close-Up



We assume that the close-up covers a single person..

There are two values that require calculating. One is for looking room, one is for walking / running room. These values may need combining if the subject is both moving and not looking at the camera.

***Looking Room**. If the subject is not looking directly towards the camera the positioning of the head should be offset from the centre by the following proportion of the frame:

$$\delta = \alpha \left\| \frac{\pi - \theta}{\pi} \right\|$$

where:

θ is the difference between the angle the face is looking and the plane perpendicular to the line of sight of the camera (i.e. the direction if the head is facing directly to the side / profile shot).

α is a constant where ½ > α >0 (user defined)

Angles are defined in radians.

****Walking / Running Room**. If the subject is moving along an axis that is not directly towards or away from the camera the positioning of the head should be off centred by the following proportion:

$$\varepsilon = \beta\upsilon\left\|\frac{\pi - \xi}{\pi}\right\| \qquad \varepsilon \le 1/2$$

where:

ξ is the difference between the angle the subject is moving along and the plane perpendicular to the line of sight of the camera.

β is a constant where ½ > β > 0 (user defined such that ε < ½

υ is the velocity of the subject[29]

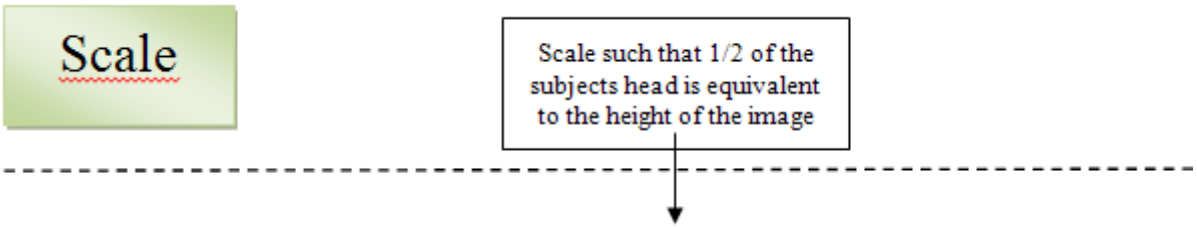
The user-defined values may be different for different image aspect ratios (for example, δ, ε may have slightly smaller values for wider aspect ratios).

If a subject is both moving and looking off-centre the movement offset can be combined, for example using *Max* (δ, ε) or a linear combination of the values given by

$$A\delta + B\varepsilon \qquad \text{where } A < 1;\ B < 1;\ A+B \le 1$$

7.2.2 Ultra Close-Up

The ultra close-up can also be referred to as a choker shot (some people use these two terms to refer to subtly different things whereas for others they mean the same thing). This shot frames a single subject. This shot is defined as the close up but with the following scale:



7.2.3 Mid-Shot

For a mid-shot the framing rules are similar to the close-up and ultra-close-up, but with a different scale factor.

For wider types of shot such as this, it is more likely that there will be multiple subjects in the frame. For multiple subjects, the positioning could be based on either the 'average' location of the subjects, the location of the most important subject, or a combination of these. If there is no importance information or importance information is not relevant for the desired shot, the position of the collective subject can be defined by defining the 'average' position of N subjects as:

$$pos = \frac{\sum_{subject=1}^{N} subject_{eyesPosition}}{N}$$

If the importance of the subject is available then the shot may be framed simply on the most important subject. If multiple subjects are to be framed with their importance being taken into account the 'weighted average' position of N subjects can be described by:

$$pos = \frac{\sum_{subject=1}^{N} subject_{importance} \times subject_{eyesPosition}}{N \sum_{subject=1}^{N} subject_{importance}}$$

Note that the above position definitions may lead to subjects being partially in shot which may not be ideal (see discussion on framing the action, below).

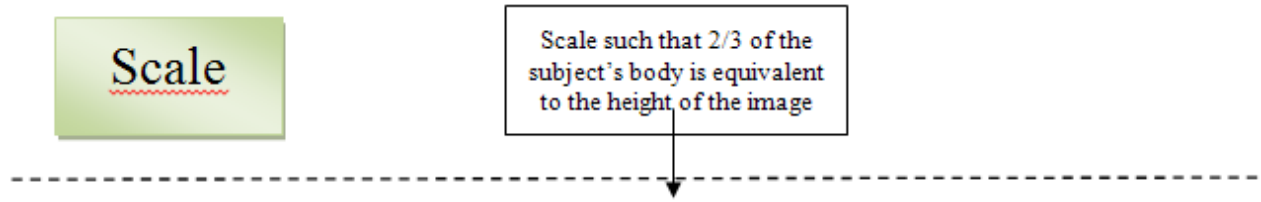
[29] υ maybe replaced by $f(\upsilon)$ to represent the proportion of looking room varying non-linearly with speed.

The shot framing algorithm for the mid-shot is then defined as it is for the close-up with the following differences:

- the subject's position being defined as the 'average' position as shown above

and

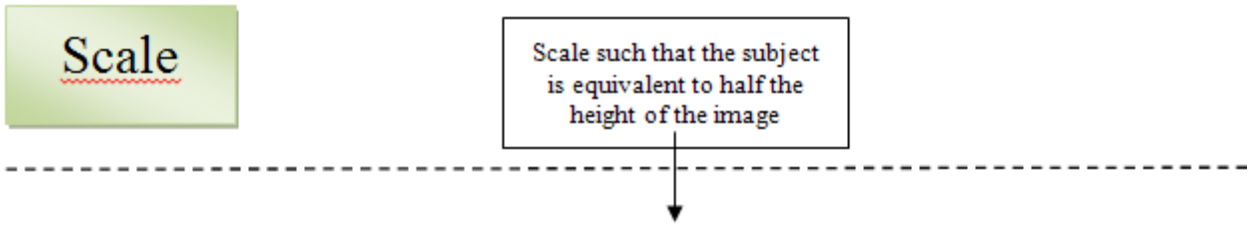
- the scale being defined by:



Note: there has been no attempt to say how the number, N, of subjects to be considered is derived. This is a figure that may change for different scenarios and requires further investigation.

7.2.4 Wide shot

Again this shot is based on the same fundamental algorithm as for the close-up shot. For multiple subjects to be framed these subjects can use the 'averaged' position as described above for the mid-shot. The scale of a mid-shot can change according to the kind of event or the individual preferences of the director, but would typically be 50% of the height of the image i.e.:



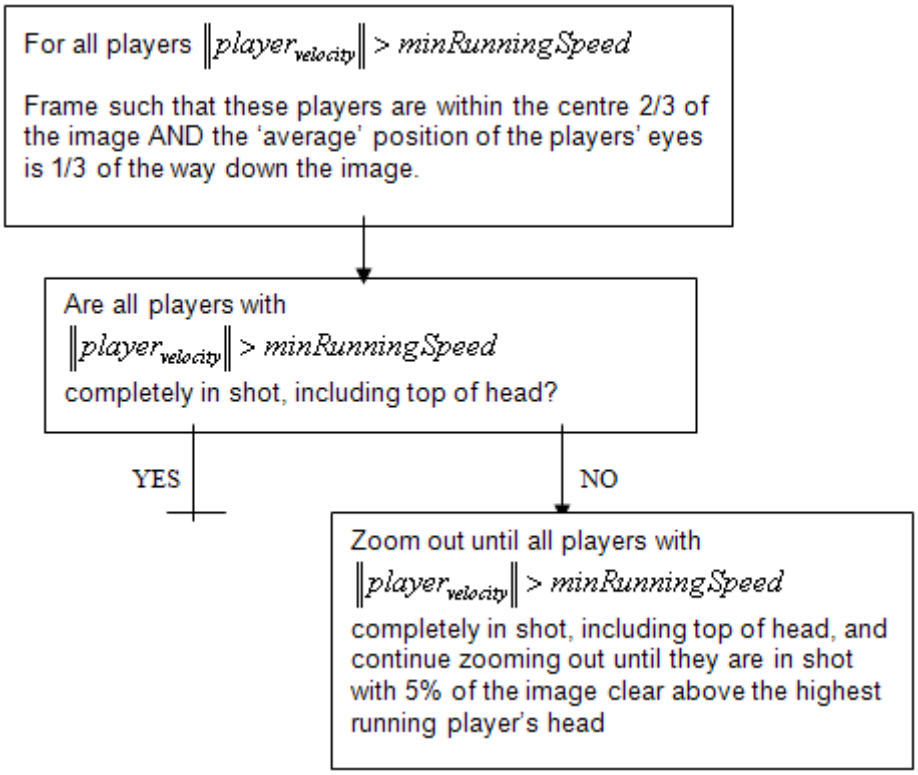
7.2.5 Framing the action

The action can be defined in many ways. The action may well be defined differently in different contexts. Here we will consider the context of football. One simple rule is proposed here as an example.

An example of a simple rule is to define the centre of the area of interest to include all the players that are running[30] (where running is defined as the player moving faster that a threshold speed: *minRunningSpeed*).

[30] Acceleration is another factor that may be considered. A rapidly accelerating player is likely to be involved in the point of action within a football game. This acceleration maybe straight line acceleration as the player moves towards the action or is the point of action as he runs with the ball or the acceleration could be a rapid change in direction such as in a tackle or dodging defenders.

Frame the shot such that:



The 'average' position is as defined in section 7.2.3 for the mid-shot.

7.2.6 Smoothing the shot framing

The approaches descried above could be applied independently to each frame in order to compose the shot. However, this can lead to very abrupt motion and un-natural shot framing, where the virtual camera responds to every slight change in the estimated position of the objects(s) being shown. In practice, a degree of temporal filtering needs to be applied. In previous work, this was achieved by specifying a tolerance in terms of a fraction of the image width and height, such that if the framing parameters from the previous frame would work for the current frame to within this tolerance, then the framing was left unchanged from the previous frame. This prevented the virtual camera from attempting to follow every tiny motion of a subject's head, for example. Temporal filtering can also be applied to the rate of change of the 'virtual' pan/tilt/field-of-view, so that changes happen smoothly.

7.2.7 Summary of framing metadata

In summary, the metadata required to specify the framing of the shot, based on the algorithms outlined above, comprise:

- A shot type (close-up, ultra close-up, mid shot, wide shot, action-framing shot)
- For each shot type, a set of pre-chosen values (not all being relevant for all shot types) that specify:
  - Scale (in terms of size of a head as a proportion of the frame height)
  - Height of point-of-interest in the image (as a proportion of the frame height)
  - "Looking room" scaling factor
  - "Walking/running room" scaling factor
  - An indication of the type of rule to apply when multiple subjects are in shot (average position, position of most important, etc.)
  - For action-framing shots, a definition of objects to be included in the 'action area' in terms of detected features (e.g. bounding box speed or acceleration) and of the percentage of the image height and width in which all selected objects must appear.
- Parameters to control the smoothness of the virtual camera motion, including two values representing the shot framing tolerance, expressed as fractions of the image width and height, and three temporal filter factors used to smooth the rate-of-change of the pan, tilt and field-of-view.

Note that all these values may be dependent on the screen aspect ratio and size. The choice of shot type (not explicitly discussed in this section) is also likely to depend on the screen size.

This set of metadata is likely to be changed or extended as shot framing algorithms are evaluated in the course of the project.

8 References

[Al-Hames, 2007] Marc Al-Hames, Benedikt Hornler, Ronald Müller, Joachim Schenk and Gerhard Rigoll (2007). Automatic Multi-Modal Meeting Camera Selection for Video-Conferences and Meeting Browsers, in Proceedings of IEEE International Conference on Multimedia and Expo, 2007, Beijing

[Allen, 1983] Allen, J. F. (1983). Maintaining knowledge about temporal intervals. (R. J. Brachman & H. J. Levesque, Eds.)*Communications of the ACM*, *26*(11), 832-843. ACM. Retrieved from http://portal.acm.org/citation.cfm?doid=182.358434

[Chen, 2010] Chen, F., Delannay, D. and De Vleeschouwer, C. (2010). "Multi-sensored Vision for Autonomous production of Personalized Video Summaries" 2nd International ICST Confrence on User Centric Media, Palma de Mallorca, Spain, September 2010.

[Chen, 2011] Chen, F., Delannay, D., and De Vleeschouwer, C. (2011). An Autonomous Framework to Produce and Distribute Personalized Team-Sport Video Summaries: A Basketball Case Study. In IEEE Transactions on Multimedia, Volume 13, December, 2011

[Daniyal, 2011] Daniyal, F., and Cavallaro, A. (2011). Multi-camera Scheduling for Video Production. In Conference for Visual Media Production, Los Alamitos, CA, USA, 2011

[Etzion, 2010] Opher Etzion and Peter Niblett, Event Processing in Action, Manning Publications, 2010 ISBN: 9781935182214. Proceedings of the 11th Interational Semantic Web Conference ISWC2011. Retrieved from http://iswc2011.semanticweb.org/fileadmin/iswc/Papers/Workshops/OrdRing/paper_8.pdf

[Etzion, 2011] Etzion, O. et al. (2011). The event processing manifesto Written by the participants of the 2010 Dagstuhl seminar on event processing. (K. M. Chandy, O. Etzion, & R. Von Ammon, Eds.)Processing, (10201), 1-60. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2011/2985

[Falelakis, 2011] Falelakis, M., Kaiser, R., Weiss, W., & Ursu, M. (2011). Reasoning for Video-mediated Group Communication. Proceedings of the IEEE International Conference on Multimedia Expo.

[Forgy, 1982] Forgy, C.L. Rete, "A fast algorithm for the many pattern/many object pattern match problem", Artificial Intelligence 19, 17-37, 1982

[Hoeksema, 2010] Jesper Hoeksema and Spyros Kotoulas. (2010). High-performance Distributed Stream Reasoning using S4. Proceedings of the 11th Interational Semantic Web Conference ISWC2011

[Hoeksema, 2011] Hoeksema, J., & Kotoulas, S. (2011). High-performance Distributed Stream Reasoning using S4.

[Kaiser, 2011a] Kaiser, R., Thaler, M., Kriechbaum, A., Fassold, H., Bailer, W., & Rosner, J. (2011). Real time person tracking in high-resolution panoramic video for Automated broadcast Production. Proceedings of the 8th European Conference on Visual Media Production CVMP 2011. Retrieved from http://www.cvmp-conference.org/dyn/1316428431644/CVMP11_Realtime-person-tracking-in-HiRes-Panoramic-video.pdf

[Kaiser, 2011b] Rene Kaiser, Claudia Wagner, Martin Höffernig, and Harald Mayer, "The interaction ontology model: supporting the virtual director orchestrating real-time group interaction," in Proceedings of the 17th international conference on Advances in multimedia modeling, 2011, MMM'11, pp. 263–273.

[Kaiser, 2012] Rene Kaiser, Wolfgang Weiss, Gert Kienast: The FascinatE Production Scripting Engine. MMM 2012: 682-692

[Kim, 2011] Kim, S., Moon, S., & Han, S. (2011). Programming the story: interactive storytelling system. Event London, 35, 221-229. Retrieved from http://www.freepatentsonline.com/article/Informatica/262037001.html

[Jiang, 2011] Jiang, J., Köhler, J., Mac Williams, C., Zaletelj, J., Güntner, G., Horstmann, H., Ren, J., Löffler, J., and Weng, Y. (2011). LIVE: An Integrated Production and Feedback System for Intelligent and Interactive TV Broadcasting, In Proceedings of IEEE Transactions on Broadcasting, Vol. 57, Nr. 3, pages: 646 – 661, DOI: 10.1109/TBC.2011.2158252, Sept. 2011

[Lavee, 2009] Lavee, G., Rivlin, E., & Rudzsky, M. (2009). Understanding Video Events: A Survey of Methods for Automatic Interpretation of Semantic Occurrences in Video. IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews, 39(5), 489-504. IEEE. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5075633

[Patrikakis, 2010] Patrikakis, C., Pnevmatikakis, A., Chippendale, P., Nunes, M. S., Cruz, R. S., Poslad, S., Wang, Z., et al. (2010). Direct your personal coverage of large athletic events. Ieee Multimedia, 17(4).

[Schopman, 2010] Schopman, B., Brickly, D., Aroyo, L., Van Aart, C., Buser, V., Siebes, R., Nixon, L., et al., et al. (2010). NoTube: making the Web part of personalised TV. Scenario, 1-8. Retrieved from http://journal.webscience.org/354/

[Stuckenschmidt,2010] Heiner Stuckenschmidt, Stefano Ceri, Emanuele Della Valle, and Frank van Harmelen. Towards expressive stream reasoning. In Proceedings of the Dagstuhl Seminar on Semantic Aspects of Sensor Networks, 2010.

[Ursu, 2008] Ursu, M. F., Kegel, I., Williams, D., Thomas, M., Mayer, H., Zsombori, V., Tuomola, M. L., et al. (2008). ShapeShifting TV: interactive screen media narratives. Multimedia Systems, 14(2), 115-132. Springer. Retrieved from http://dx.doi.org/10.1007/s00530-008-0119-z

9 Glossary

Terms used within the FascinatE project, sorted alphabetically.

AV	Audiovisual
ID	Identity
NSL	Narrative Structure Language
OOI	Object of interest
PSE	Production Scripting Engine
ROI	Region of interest
SE	Scripting Engine
SLMS	Semantic Layer Management System
XML	Extensible Markup Language, http://www.w3.org/XML/

Partner Acronyms

ALU	Alcatel-Lucent Bell NV, BE
ARI	Arnold & Richter Cine Technik GMBH & Co Betriebs KG, DE
BBC	British Broadcasting Corporation
DTO	Technicolor, DE
HHI	Heinrich Hertz Institut, Fraunhofer Gesellschaft zur Förderung der Angewandten Forschung e.V., DE
JRS	JOANNEUM RESEARCH Forschungsgesellschaft mbH, AT
SES	Softeco Sismat S.P.A., IT
TII	The Interactive Institute, SE
TNO	Nederlandse Organisatie voor Toegapast Natuurwetenschappelijk Onderzoek – TNO, NL
UOS	The University of Salford, UK
UPC	Universitat Politecnica de Catalunya, ES