EXPERIMEDIA

D2.1.6

Second Blueprint Architecture for Social and Networked Media Testbeds

v1.1: 2013-08-16

David Salama (ATOS), Michael Boniface (IT Innovation), Simon Crowle (IT Innovation), Stephen C. Phillips (IT Innovation), Nicholas Vretos (CERTH), Kleopatra Konstanteli (NTUA), Thanos Voulodimos (NTUA), Stefan Prettenhofer (Infonova), Sandra Murg (JRS), Peter Ljungstrand (Interactive)

This second blueprint architecture for social and networked media testbeds provides the foundation for the EXPERIMEDIA facility for baseline component development during the expansion phase (Year 2) and for experiments conducted using the baseline (Year 3). The document builds on the first blue print architecture D2.1.3. The purpose of the architecture is described along with requirement considerations. A high-level description of the baseline architecture is provided and how each component is integrated within experiments for both instrumentation/observation and also orchestration of information flows. The capabilities of each specific component are described including those supporting FMI content lifecycles and the Experiment Content Component supporting overall experiment management.

www.experimedia.eu

Project acronym	EXPERIMEDIA
Full title	Experiments in live social and networked media experiences
Grant agreement number	287966
Funding scheme	Large-scale Integrating Project (IP)
Work programme topic	Objective ICT-2011.1.6 Future Internet Research and Experimentation (FIRE)
Project start date	2011-10-01
Project duration	36 months
Activity 2	Construction
Workpackage 2.1	Architecture Blueprint
Deliverable lead organisation	IT Innovation
Authors	David Salama (ATOS), Michael Boniface (IT Innovation), Simon Crowle (IT Innovation), Stephen C. Phillips (IT Innovation), Nicholas Vretos (CERTH), Kleopatra Konstanteli (NTUA), Thanos Voulodimos (NTUA), Stefan Prettenhofer (Infonova), Sandra Murg (JRS), Peter Ljungstrand (Interactive)
Reviewers	Sandra Murg (JRS)
Version	1.1
Status	Final
Dissemination level	PU: Public
Due date	PM18 (2013-03-31)
Delivery date	v1.0: 2013-08-01; v1.1: 2013-08-16

Version	Changes
1.0	Initial release
1.1	Added information on SLAs, security and website monitoring; refined the detail of the provenance discussion; added more PCC sub-components

Table of Contents

List of Figures

1. Executive Summary

This document is deliverable D2.1.6 "Second Blueprint Architecture" of the EXPERIMEDIA project 287966 describing an architecture for social and networked media test-beds. The document is the second iteration of the architecture superseding deliverable D2.1.3 "First Blueprint Architecture" published on 11 April 2012. It provides a framework for further enhancements to the baseline components during the second year, the expansion phase, of the project and during the period of adaptation to the second open call experiments.

The architecture builds on the first architecture's framework, describing additional composition patterns to create richer inter-component data flows, and also lays out the variety of deployment options available to experimenters along with a strategy for assessing the security and privacy implications of those deployment choices.

A substantial part of the document is dedicated to proposed extensions to the experiment content component: the configuration management, the monitoring sub-system and visualisation requirements. Of these topics, the extensions to the monitoring system to add in provenance data are most significant. For the driving and first open call experiments, the project defined a method for reporting metrics which could represent quality of service and quality of experience. By adding a structure way to keep track of user interactions and the causes of the metric reports, the second architecture will provide additional insight into this crucial data.

Finally, the document provides an updated view of the available baseline components to help the experimenter understand what is available, how they may be deployed and what communication patterns already exist.

2. Introduction

2.1. Purpose

The purpose of this document is to provide facility developers and experimenters with a description of generic Future Media Internet capabilities and technologies offered by EXPERIMEDIA and how such technologies can be integrated and used in social and networked media experiments at the facility.

The purpose of architecture is to provide an abstract description of the structure and behaviour of a system, and the desired impact the system is required to have on its environment. Architecture describes the system scope, what outputs a system produces (in response to inputs), the processes for delivering the outputs, and the resources necessary both in terms of people and other assets.

Architecture is fundamentally communication mechanism and a way to help everyone understand a system. A significant challenge in comprehending a system is that most are complex. A primary goal is to deal with complexity through abstraction and decomposition techniques in a way that considers design principles such as of encapsulation, high cohesion, and loose coupling. Many methodologies have emerged in recent years to support the process of architecture definition. The evolution of methods is driven by both advances in technologies and the types of systems under construction. Our objective is to intelligently select techniques that are most useful for the specific architectural characteristics and challenges faced by EXPERIMEDIA rather than to adopt a single methodology universally.

The primary audience are those responsible for developing implementation technologies, integrating and interconnecting related systems, and operating all or parts of the EXPERIMEDIA systems. It also serves as an introduction of the EXPERIMEDIA architecture to new experimenters.

2.2. Scope

The document describes architecture for EXPERIMEDIA facility for the Expansion Phase (Year 2) whose implementation will provide the foundation for experiments conducted in the final year of the project. The first architecture provided a high-level view of components within the EXPERIMEDIA facility with some suggestions on how such components can be integrated. The primary focus for the first version of the EXPERIMEDIA architecture was instrumentation and observation of communities and components. The ability to collect data from multiple heterogeneous platforms was seen as the essential element for experiments requiring the observation of individuals and communities, and how to explore the relationship between quality of service (QoS) and quality of experience (QoE). The second architecture extends these concepts but also focuses in more detail on how various components can be composed to orchestrate information flows and how such information flows can increase quality of experience. Building on the conceptualisation in the first architecture various types of content components are considered including (social, audio-visual, pervasive and 3D) and test-bed management services supporting the experiment lifecycle. This document builds previous

deliverables: D2.1.4 "Second EXPERIMEDIA methodology"[1], D2.1.5 "Second Scenarios and Requirements"[2] and D3.1.5 "Second Infrastructure and Software Assets Inventory".

EXPERIMEDIA needs to describe the capabilities expected within a Future Media Internet (FMI) architecture and not just the EXPERIMEDIA facility or a specific experiment. As such the descriptions needs to consider the generic Architecture model for a FMI experimental facilities such as those being offered by EXPERIMEDIA's venues: Schladming, CAR and FHW. As part of the work to produce the Architecture Blueprint we need to reach a consensus on what capabilities are within an FMI system. By providing a capability map for the FMI with baseline components providing basic implementations we offer the possibility for experimenters to understand how to integrate their technology within the EXPERIMEDIA ecosystem and to support multiple implementations of the same capability if necessary. For example, one experiment may want to focus on P2P content delivery whilst another may focus on augmented reality applications. What they need to is to understand where their experimental components fit into the overall FMI architecture and what generic baseline components from EXPERIMEDIA are available to integrate with to provide the additional capabilities they require.

2.3. Architectural Considerations

The specific characteristics of EXPERIMEDIA that must be considered throughout the architectural design are included in the following list.

- **Evolving Requirements**: we are describing architecture but cannot know all requirements in advance. We can describe the general capabilities for an FMI architecture and what it means to operate a facility supporting such systems. However, new requirements will emerge from experiments using the facility that cannot be envisaged now.
- **Integration and Adaptation:** each experiment will develop and operate a FMI system that consists of EXPERIMEDIA baseline technology components, EXPERIMEDIA infrastructure components and experimental components. Architecture must be developed in a way that ensures loose coupling between and efficient integration of components in a way that creates a system of systems. Standardised interfaces should be adopted where possible to reduce need for specific adaptations.
- **Experimentation:** experiments typically require components with high degrees of instrumentation and control to attain insight into the behaviour of systems, their relationship with users and to ensure validity by reducing the influence of extraneous factors and providing repeatability.
- **Security and Privacy:** experiments must be legally compliant in accordance with data protection legislation and security and privacy therefore must be considered a critical attribute of component and systemic capabilities. Security and privacy must be by design rather than an add-on.
- **Technology Baseline:** EXPERIMEDIA is not architecting a system from scratch but from a set of technologies supporting different capabilities within the Future Media

[1] http://www.scribd.com/doc/137302530
[2] http://www.scribd.com/doc/129728380

Internet, and targeting known infrastructure environments. The architectural process needs to combine top down analysis of desired capabilities alongside a bottom up assessment of how each baseline technology and infrastructure supports them. Through this process overlaps, gaps and integration points can be determined which can inform future development tasks

- **Constraints:** Each component delivers a capability but also has technological and operational constraints on use. For example, technically a component may only support specific protocols or in operation may be only available at certain times and with limited resources. This is especially relevant for infrastructure components at each venue that are operated, sometimes by 3rd party companies, for "other" purposes (i.e. EXPERIMEDIA does not have exclusive access).
- **Time Limitations:** the system lifecycle is organised into iterative and incremental activities, with each iteration expected to add functionality. The first iteration is the most challenging considering the novelty of the process, the levels of domain knowledge and maturity of collaborative relationships. The scope of the architecture and capability descriptions is likely to far exceed what can be delivered during the first iteration with significant need to prioritise critical components and integrations between them
- **Viewpoints:** architecture can be described from multiple perspectives; we need to consider how the architecture is presented to different stakeholders.
- **Moving to Market:** the architecture must be designed so that the systems required for experimentation (specifically the experiment content component) can be removed without breaking the end-user experience.

3. High Level Architecture

3.1. Technology Enablers

Technology enablers are software or service components whose capability allows users to achieve added value through use, either by design (i.e. the purpose is known in advance) or more frequently by openness (i.e. the purpose is opportunistically established by the user). Technology enablers are a key part of future innovation in programmes such as FIRE[3] and the FI-PPP[4]. Technology enablers of the FMI must address the needs novel applications and services allow them to exploit a range of social, audio/visual, pervasive content and 3D content. Each class of content has distinct characteristics, content lifecycles (authoring, management and delivery) and platforms to support them. Developing a new platform supporting all content types is unrealistic and the approach must focus on developing open interfaces to existing platforms that allow for greater levels of interaction between information and control flows.

EXPERIMEDIA defines a component model that focuses on different content aspects within the FMI with implementation technologies supporting the lifecycle of the specific content. Tools and services are provided that support the mixing of different content types in the delivery of user experience where the content lifecycles could be implemented within separate systems. Figure 1 shows the EXPERIMEDIA component model: the social content component (SCC), audio-visual content component (AVCC), pervasive content component (PCC) and 3D content component (3DCC) to which we add an experiment content component (ECC) supporting all data and processes related to the setup, execution, monitoring, analysis and security of experiments. A key element of the components is that they are designed on the principle of openness and transparency in terms of observability, configuration and security policy. Each component includes a structural (i.e. entities) and behaviour model (i.e. QoE, QoS and quality of community or QoC), and is instrumented to allow deep measurements. The disclosure of such information is essential for understanding the interplay between different system components, along with the observation of behaviours in larger composed Internet ecosystems including communities. A configuration interface is provided that supports set up and runtime adaptation of some QoS parameters. A security model is provided that describes authentication, access control and how personal data is processed. The latter element is necessary to assure compliance with ethical experimentation and associated data protection legislation. The capabilities of each component are described in more detail in the following sections.

[3] http://cordis.europa.eu/fp7/ict/fire/
[4] http://www.fi-ppp.eu/

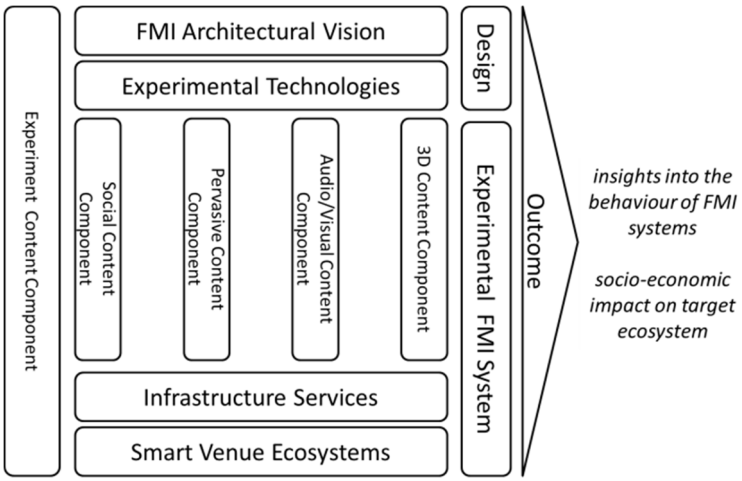


Figure 1: Technology enablers of an FMI testing facility

3.2. Fundamental Composition Patterns

Composition Patterns are standard ways that technology enablers can be used together to investigate new forms of social interaction and experience. We define a set of important patterns to help experimenters understand how the components of the facility can best support their experimental objectives and to provide stimulus for new ideas.

3.2.1. Instrumentation and Observation

The 1st pattern "Instrumentation and Observation" focuses on instrumentation of technology enablers. This was the focus on the implementation in the first year. Each component is described in terms of QoS, QoE and QoC metrics associated with their specific content domains (social, audio-visual, pervasive, and 3D) and is required to generate measurements of these metrics during the runtime. Additional infrastructure metrics regarding infrastructure performance are generated by hosting components such as the CloudManager (e.g. compute, storage and networking). All metrics assist experimenters in understanding the behaviour of the system in terms of both technical performance and user experience. For example, the audio-visual content component (AVCC) generates metrics related to audio-visual (AV) streaming such as frame rates, frames dropped, video quality, etc. When combined with networking metrics (e.g. bandwidth, latency, etc) an experimenter can study the network characteristics necessary to deliver a certain QoS (e.g. 25 fps, HD with a 1/1000 frames dropped) to a group of consumers. This is standard, although not simple, and initiatives such as those undertaken in the ITU QoE study areas (e.g. ITU-R Rec. BT.500-11) provide a methodology for the subjective assessment of the video quality.

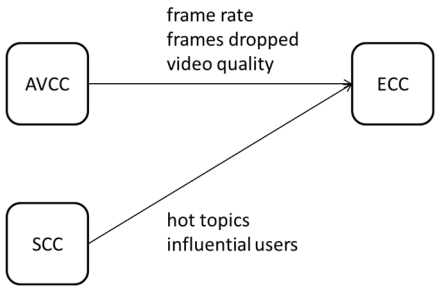


Figure 2: Correlating between discussion topics and delivered content

FMI technology enablers must focus more on how different content, aggregations of content and social interaction affect experience. With each Content Component generating metrics experimenters can begin to correlate human activities with monitoring data between components. For example, navigating to a certain location in virtual world may create a popular discussion in a social networking group. By identifying popular discussions and looking at which point in the story/presentation (e.g. seeking a specific time point a recorded video stream) when these occurred the experimenter can begin to understand why specific events cause specific outcomes in the target community, and if necessary initiate a deeper analysis (e.g. direct user evaluation) with the community on these target areas. Figure 2 illustrates data coming from both the AVCC and the SCC which can be correlated in the ECC to support this type of analysis. Changing the narrative after the production would be considered a "design" phase adaptation. However, increasingly we envisage adapting the narrative during the production based on emerging profiles and interests of social groups and how they react to the content being delivered. In this case rather than undertaking a post analysis of the metrics we could automatically annotate a video stream with metadata indicating points of interest/questions associated with the content. The Content Author could then adaptive the narrative based on discussions, questions, or votes for more information by reviewing an annotated stream timeline.

3.2.2. Mixed Information Flows

The second pattern "Mixed Information Flows" focuses on how content from each component can be orchestrated in information flows as part of a new experience. Examples include:

- annotating video streams (AVCC) with metadata from social networking trends (SCC);
- annotating video streams (AVCC) with metadata derived from sensors (PCC);
- adapting the narrative of a pervasive game (PCC-Creator) based on social networking trends;
- reconstructing people who are present in physically different locations (3DCC) in a single virtual location as 3D avatars.

An interesting element is how by mixing the content between different platforms influences the user experience and technical performance in each component. For example, does changing the narrative as a consequence of the social networking topic reduce the discussion on the social network because the focus of attention has changed?

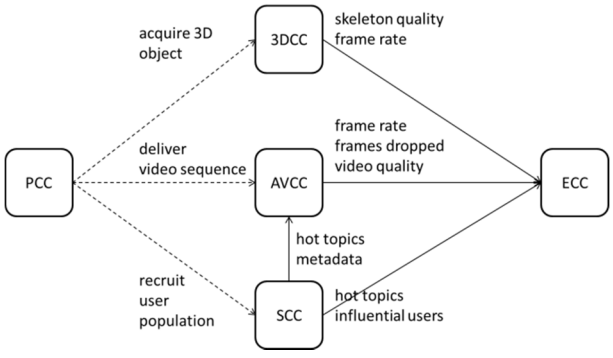


Figure 3: An integrated view of EXPERIMEDIA based on experiment composition patterns

It is critical that technology enablers are support added value composition patterns (see Figure 3). Here we show how all components can be used together in an FMI system. The PCC orchestrates the narrative (control flow is the dotted lines, data flow is the solid lines) for the gamification of activities. As such the PCC can initiate controlling actions such as recruiting user populations through information dissemination in social networks, delivering popular video sequences to specific communities and acquiring 3D representations of objects and people. With all components instrumented using a behavioural model resulting metrics generated are acquired by the ECC and available for real-time and post analysis.

4. Experiment Content Lifecycle Management

As EXPERIMEDIA is performing FMI *experiments* and not just creating FMI systems, the management of those experiments and their data is of utmost importance to the project. Even though the more complex component composition patterns discussed above are necessary, they still all involve the experiment content component. From the experiences of the driving and first open-call experiments various useful extensions and refinements to the ECC have been identified and this document therefore dedicates a significant portion to describing these advancements.

4.1. Experiment Content Component (ECC) Overview

Experiment content is produced and consumed by developers performing tests on FMI systems to understand and gain insight into structure, behaviour and performance. System configuration, system dependency graphs, input/out data sets, testing procedures and monitoring data all characterise experiment content.

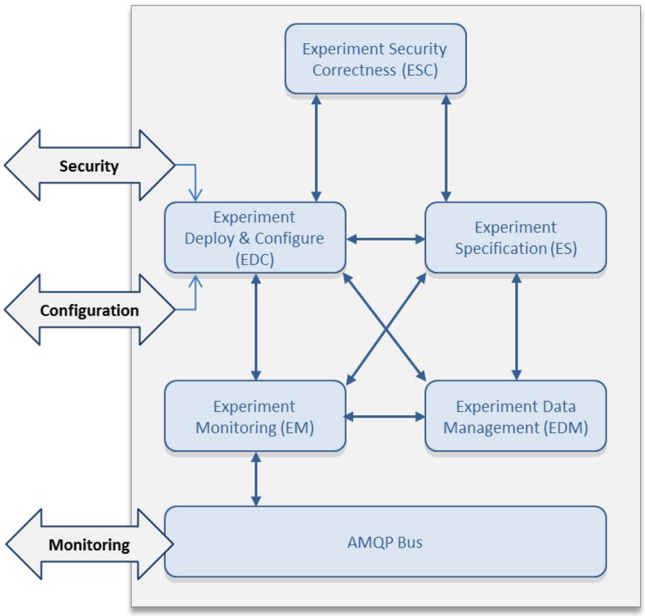


Figure 4: V1 ECC architecture

The ECC allows a developer to set up, execute and tear down tests on FMI systems deployed at different locations. The ECC monitors, derives experimental data from, and manages the system under test through integration with the ECC API. The ECC elicits QoS, QoE and QoC data from the other components and delivers it to the experimenters so they can analyse the behaviour of technical systems in relation to user experience. The ECC manages the delivery of monitoring metrics that are stored and available for both live and post/batch analytics. Monitoring clients are available for services, mobile clients and web applications thought an AMQP bus. A dashboard is provided leading developers through an experiment lifecycle that includes setup, live monitoring, analysis and tear down. The high level architecture for V1 is shown in Figure 4.

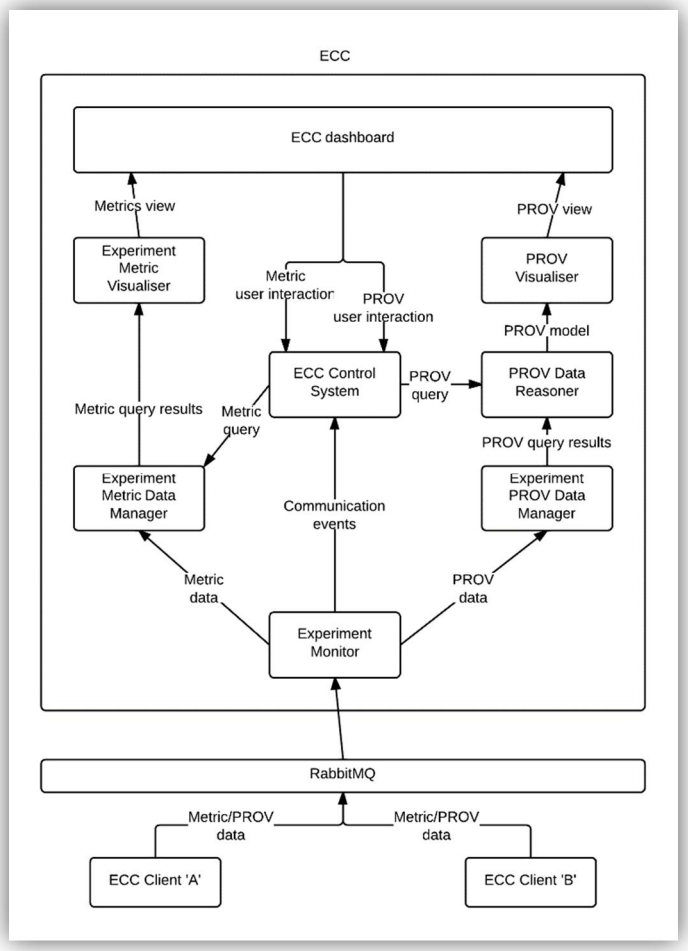


Figure 5: V2 ECC monitoring architecture

The V2 ECC architecture extends the state-of-the-art in experiment monitoring frameworks by providing a mechanism by which experimenters can investigate how system and user activities have led to changes in system performance or human experience (as observed by the ECC metric monitoring system). To this end, the experimental support provided by the ECC will be extended to include:

- An enhanced metric model meta-data to improve visualisation
- Time-line based navigation of metric data
- Metric data aggregation and descriptive statistical analysis
- A provenance based view on system and user activities

Key changes to the metric model will include:

- Extended support for Unit types to include:
  - Support for UCUM[5] unit specifications
  - Custom metric types (including Babylon based QoE types)

[5] http://unitsofmeasure.org/trac/

- Extended semantic information:
    - Measurement Group semantics (for example, the measurement sets in a particular group are all responses to a questionnaire)
    - Further entity meta-data, including URLs to online resources (photographs, videos and other representations of the entity being observed).

A further significant enhancement to the ECC architecture is the means by which system behaviours, events or human activities will be captured as an additional stream of provenance data. From an architectural point of view, the ECC has been extended to include a data provenance model, using the W3C PROV standard[6], which will be integrated with an updated ECC experiment metric framework.

In Figure 5, we see ECC clients now provide both metrics (indicating QoS/QoE characteristics) and provenance data that describes discrete activities enacted by agents on entities. Internally, the ECC captures these two data streams and stores them using the appropriate data managers. An experimenter interacting with the ECC via the dashboard then has the ability to visually navigate through the both sets along a common time-line such that interesting changes in metric data can be linked to a behavioural record of data activity associated with systems and people.

4.2. Metric Data Model

The ECC offers a metric modelling framework that offers support for a range of potential QoS, QoE and QoC measurements, see Figure 6. In this model, the objects of experimental observation (referred to as 'Entities') are loosely coupled with the agent (the ECC software client) making the observations. Entities themselves must contain one or more Attributes that are the subject of actual instrumentation and measurement activity. In version two of the metric model, Entities will optionally offer additional key-value pair meta-data to the experimenter (such as URIs to online content).

[6] http://www.w3.org/TR/prov-overview/

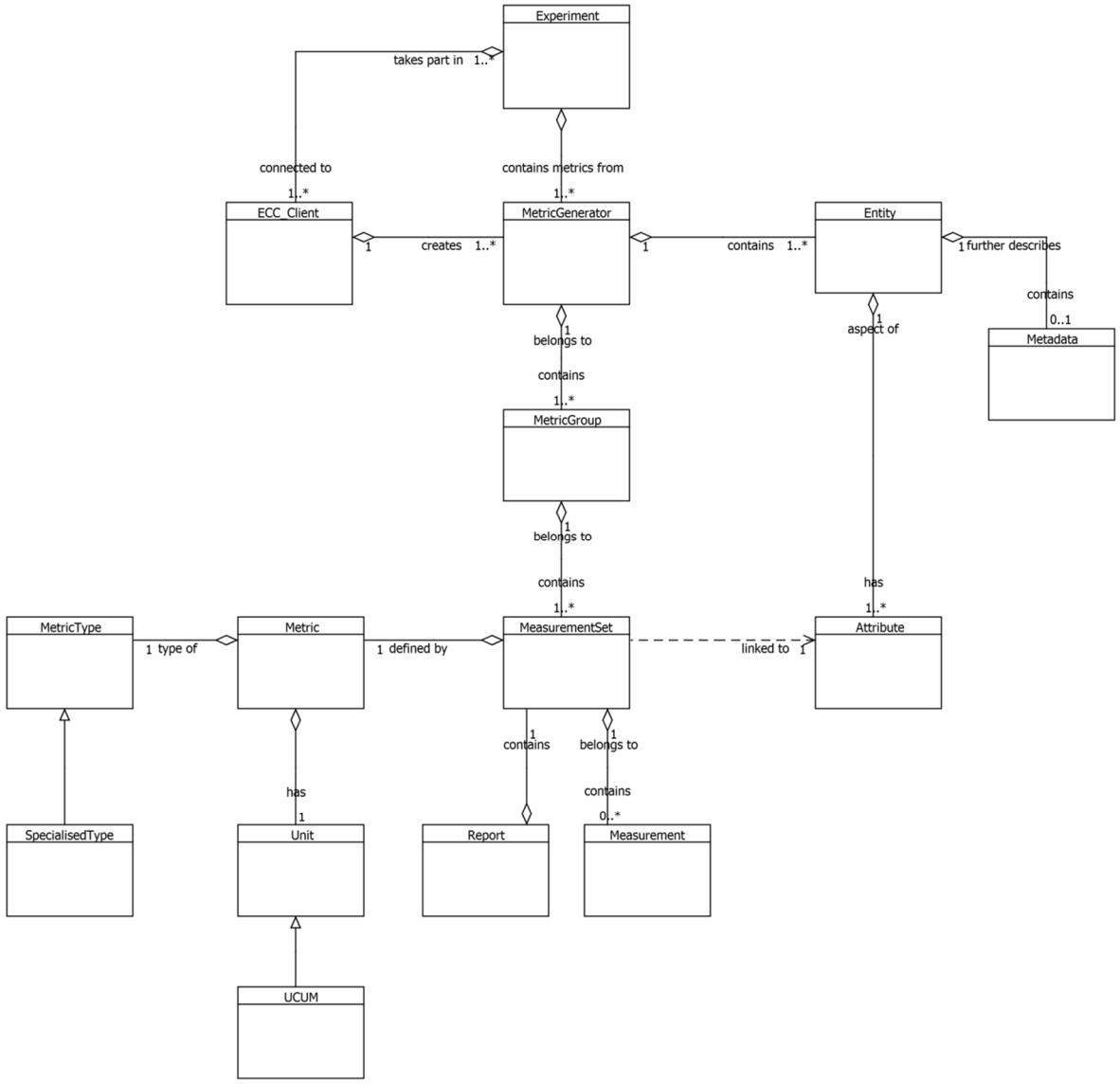


Figure 6: Updated ECC metric model

In this model, Entities themselves must contain one or more Attributes that are the subject of actual instrumentation and measurement activity. Measurement data itself is logically structured within Metric Generators (typically used to represent metrics linked to a particular sub-system or user). Further organisation is offered through the grouping of sets of measurements using one or more named Metric Groups. A Measurement Set contains zero or more measurements that are specific to a particular attribute; Metric Groups may contain one or more Measurement Sets. The semantics of each Measurement Set is defined by its Metric, which in turn has a Metric Type and Unit of measure. In version 2 of the metric model, specialisations of Metric Type and Unit will be provided to improve formalisation and enhance visualisation.

This metric model is explored a little further in the following simple example in which an ECC client (called 'SocialAuth ECC client') observes a Facebook event and sends metric data to the ECC dashboard, see the figure below.

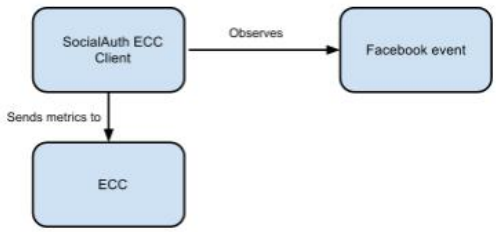


Figure 7: Simple example of observing a Facebook event

This very basic relationship need to be developed further however, since a) entities (in this case the 'Facebook event') will have certain attributes that are of interest to the experimenter and the b) some organisation of the structure of the metric data associated with the entity must also be specified. To see how this is arranged, consider Figure 8..

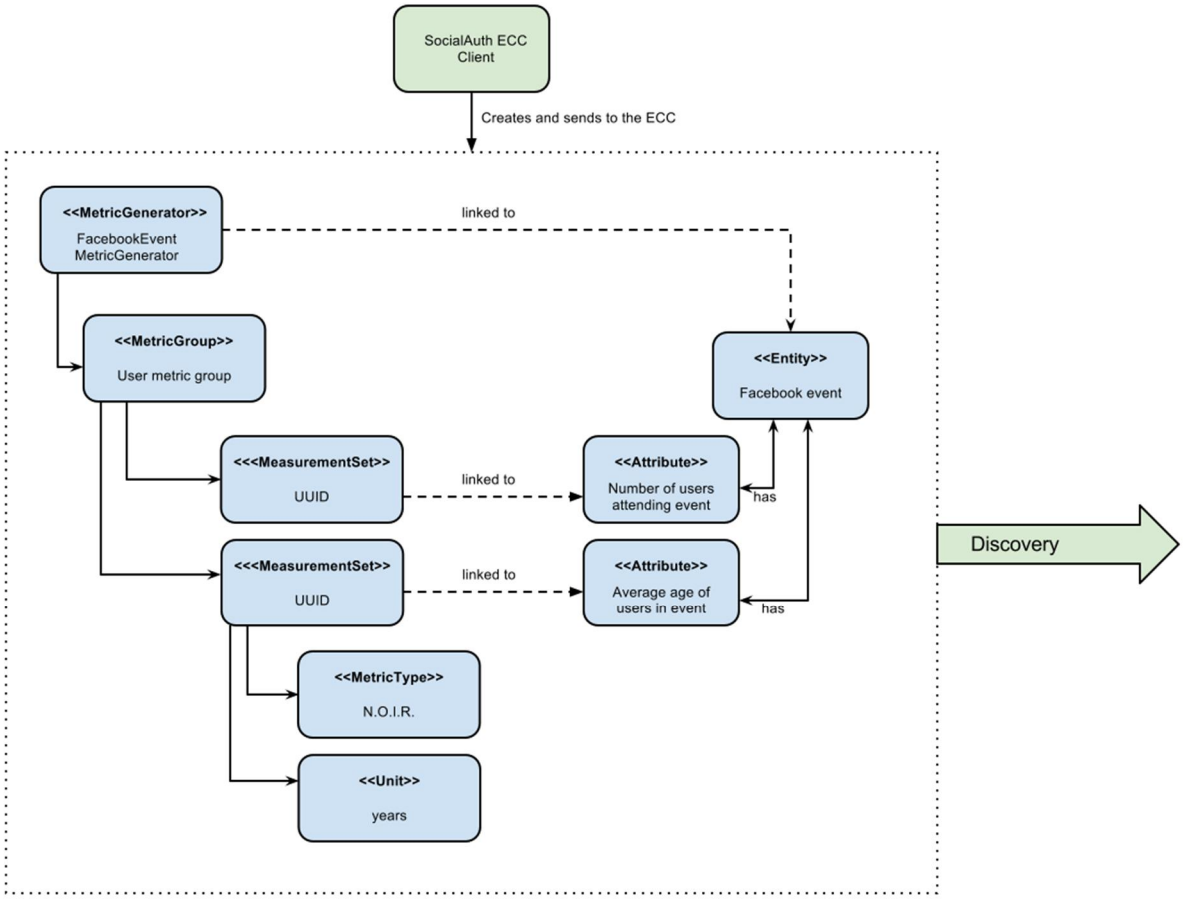


Figure 8: Monitoring a Facebook Event

In this example, we have added two attribute instances to the entity, representing aspects of the Facebook event we have an interest in observing (i) the number of users attending the event and (ii) the average age of users in the event (see Figure 8). We can consider the data management structures that support the collection of data representing these two attributes from either a 'top-down' perspective (starting from Metric Generators) or from a 'bottom-up' view point, starting with a data collection type (the Measurement Set type) that is mapped directly to an attribute of interest. For this example, we will take the latter approach and start by directly linking data sets to an attribute.

The Measurement Set type holds a set of measurements that specifically relate to an attribute and in addition has associated with it a metric meta-data indicating its Metric Type (nominal; ordinal; interval or ratio) and its Unit of measure. In the diagram above, we see two instances of Measurement Sets (each uniquely identified by a UUID value) which are mapped directly to the attributes of interest.

Moving up the data hierarchy, the next level of logical organisation is the Metric Group – a container used to perform one level of partitioning for collections of measurements that relate (for example, "online user" metrics). Metric Groups themselves are collected together by the top level data organisation, the Metric Generator. As previously indicated, the Metric Generator represents a higher, system-level component that generate metrics, for example it may be useful to differentiate server and client based metric generators. An additional mapping, similar to that used to link measurement data sets to attributes is specified linking metric generators to entities under observation since it is likely that individual systems will be deployed to observe different entity types. ECC client software must send their specification of the metrics they are going to provide the ECC in this way, during the Discovery phase. In this way, the experimenter has a means by which to understand which clients are performing what kind of measurements, and what they relate to within the experimental venue.

An exploration is being done for possible alignment of the ECC metric data model with the METRIC ABE in the telecommunications industry standard Information Framework, Release 13.5, currently under development in the SDO TM Forum. Information Framework elements, formerly known as the SID (Shared Information and Data Model) are also typically republished by the ITU, as in M.3190 for a previous release. Access to the current model under development is restricted to TM Forum member organisations, including Infonova and University of Southampton.

4.3. Bootstrapping the ECC

A note on terminology: commonly in EXPERIMEDIA we have used the work "experiment" to refer to one of the driving or open call experiments, meaning all activities associated with that work-package. However, in the following we will use the word "project" to mean one of the experiments funded by EXPERIMEDIA, then we can say that a project will run many "experiments" where an experiment involves (potentially) provisioning services, recruiting participants, getting monitoring clients connected, collecting data, tearing down the connections and then analysing the data.

A single instance of the ECC supports multiple (concurrent) experiments running in a single project. ECC's are not shared between projects. Communication between the ECC and its clients is done through RabbbitMQ and an installation of RabbitMQ can be used for multiple projects (see Figure 9).

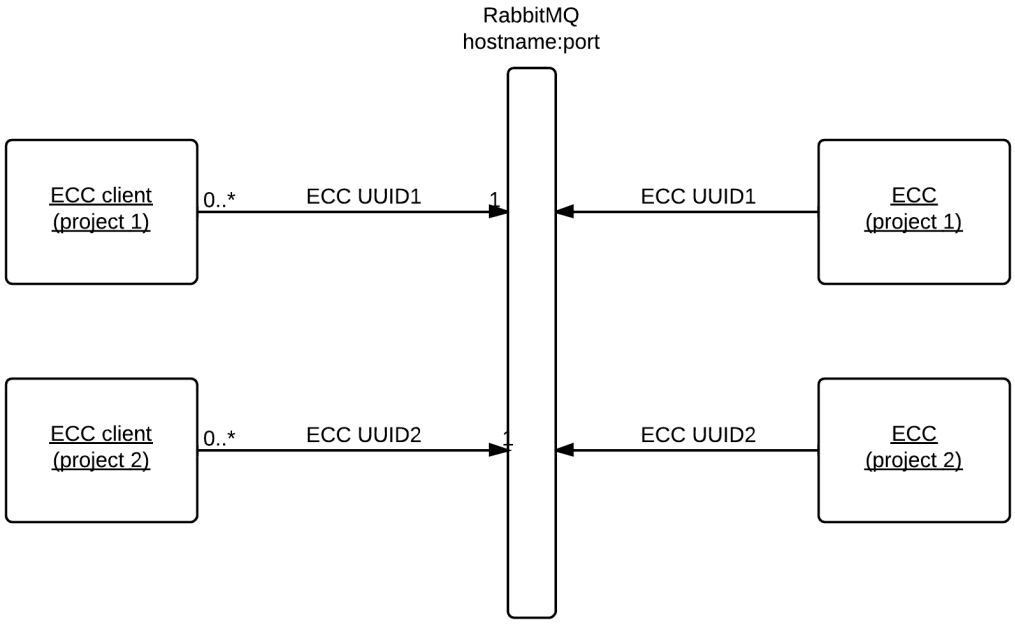


Figure 9: An illustration of the multiplicities of ECC clients, RabbitMQ and ECC (dashboard) instances

Each ECC is identified by a universally unique identifier (UUID)[7] which is assigned to the ECC through configuration at deployment time. When an instance of the ECC starts, it connects to the RabbitMQ bus and creates an exchange identified by its UUID. For an ECC client to connect to the ECC dashboard, it must know the hostname and port of the RabbitMQ bus and also the relevant ECC UUID.

In V1, software is created with, at best the RabbitMQ hostname and ECC UUID in a configuration file, and at worst, the data baked in to the software itself. Some ECC clients assume that RabbitMQ and the ECC are dedicated to a single Project and therefore clients use the default dummy ECC UUID of "00000000-0000-0000-0000-000000000000". This approach does not work if multiple ECC instances are using the same RabbitMQ. In addition, distributing configuration information to large numbers of ECC clients is inefficient and the use of UUID's makes identifiers difficult for humans to read.

In V2, configuration data is published to a Configuration Registry at a well-known URL (e.g. http://config.experimedia.eu). ECC clients download the configuration data at start up and use this information to connect to the ECC. Two options for the configuration server have been identified:

- WebDAV[8] (for instance using an Apache HTTPD server[9])
- Zookeeper instance[10] as used in Hadoop[11].

The primary use case is where an ECC service is manually deployed on behalf of a project (as this is an occasional need). The ECC service is configured with the name of the project and on

[7] A universally unique identifier (UUID) is an identifier standard used in software construction
[8] http://www.webdav.org/specs/rfc2518.html
[9] http://httpd.apache.org/
[10] http://zookeeper.apache.org/
[11] http://hadoop.apache.org/

start-up it stores its ECC UUID and RabbitMQ hostname and port in the Configuration Registry under the project name. ECC clients deployed for the project know their project's name and the address of the configuration registry. Using this information they retrieve the ECC configuration from the registry and connect. The pattern can be repeated for other services deployed for a project such as the SAD or AVCC.

A secondary use case is using the Configuration Registry for configuration of applications or services developed by the projects themselves (as opposed to baseline components). For instance, a mobile client deployed on many devices for a project that was investigating different interface types could use the configuration registry to look up which interface to display during a particular experiment run.

We define the following hierarchical naming structure:

```
/<baseline component>
    /<sub-component>
        /<project>
            /Document containing data describing the instance of the sub-component
             pertaining to project <project>
        /default
            /Document containing default configuration data for the sub-component
/project
    /<project>
        /Document(s) or subfolders containing data required specifically for
         project <project> unrelated to the baseline components
```

For instance:

```
/ECC
    /RabbitMQ
        /BLUE
            /Document containing the RabbitMQ hostname:port for project BLUE
        /default
            /Document containing the Atos RabbitMQ hostname:port
    /dashboard
        /BLUE
            /Document containing the ECC UUID for project BLUE
/SCC
    /SAD
        /BLUE
            /Document containing SAD hostname:port for project BLUE
/project
    /BLUE
        /Document containing BLUE-specific configuration data
```

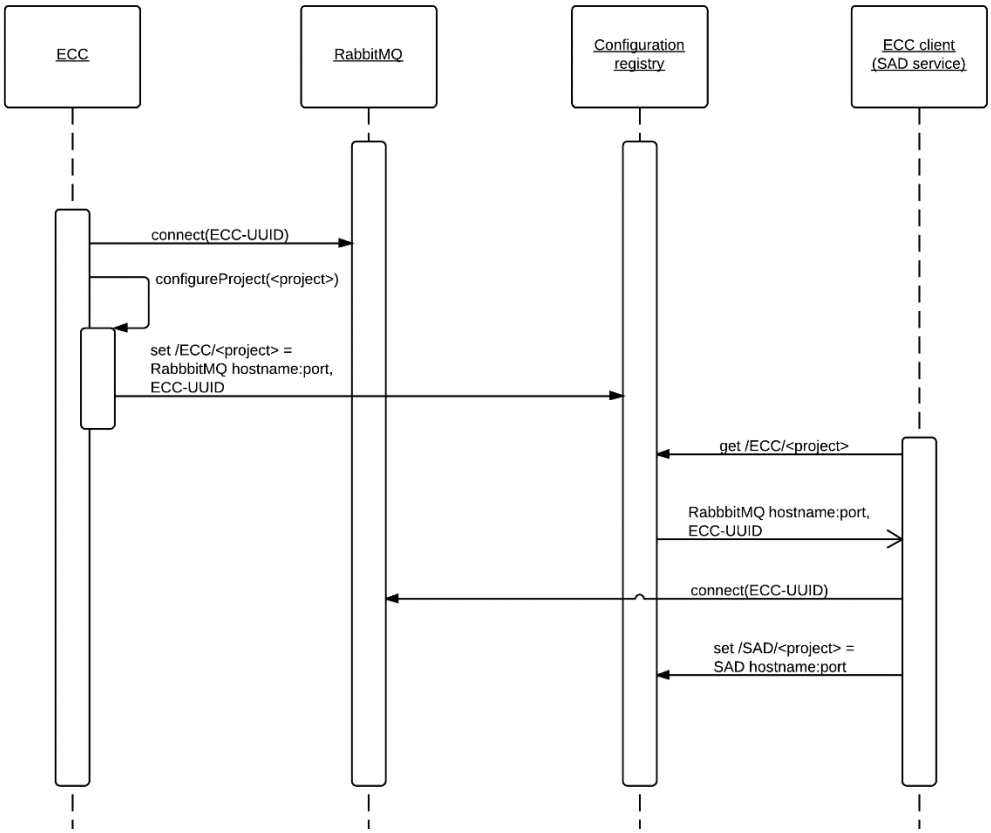


Figure 10: Sequence diagram illustrating the storing and retrieval of ECC connection data and storing of SAD service location

By arranging the data as described above, access control policies can be implemented to control which systems are able to read or write the configuration data. For instance, it could be configured such that the ECC dashboard software was authorised to write in to the "/ECC/dashboard" space and that BLUE project software was permitted to read from the "/ECC/dashboard/BLUE" space. The need for such policies depends on the sensitivity of the data being written and read which may vary across components and projects. Access control could be implemented using usernames and passwords or using some sort of web-key[12].

4.4. Naming

Naming is concerned with the rules for choosing identifiers to denote applications, software, data, people and things. Naming must consider the scope and relative uniqueness of identifiers. Naming syntax and conventions are especially important in distributed systems where things are interacting with, shared with or being observed by multiple system components often developed independently. For example:

- Two different sensors measuring the "speed" attribute of a person.
- Linking comments from Facebook and Twitter to user accounts of the same person
- Correlating Quality of Experience from Babylon mobile application with Quality of Skeleton from the 3DCC for a particular athlete.

[12] http://waterken.sourceforge.net/web-key/

There are then two basic ways to deal with the problem:

- **Convention**: agreement on the names to describe entities prior to execution. There are situations where prior agreement cannot be done because either the entities are not known at the start or existing naming conventions are already established.
- **Resolution**: once entities exist within the system, techniques such as feature extraction can be used to establish equivalence.

Where possible it is important to establish convention because feature extraction algorithms can be complex and depending on the availability of features offer varying levels of robustness. There are various cases where a consistent naming scheme is needed in EXPERIMEDIA.

Monitoring data are observations about the system under test collected during an experiment. Entities are things of interest (e.g. services, data or people interacting with the system) and Attributes are behaviours/characteristics associated with an Entity. For example:

- an entity could be a Person and attributes could be Running Speed, Opinion or Preference
- an entity could be a Service and attributes could Response time, Storage Capacity, Uptime

Entity attributes are reported to the ECC by ECC Clients. An experiment can have multiple ECC Clients reporting on a set of Entities. Two different ECC Clients could measure the same Entity and assign different UUIDs. It is important to know the ECC Client where the data has come from, but it is also important to know that the data from both sources refers to the same entity and/or attribute.

The full ECC metric model is shown previously in Figure 6. Table 1 shows the identifiers in the ECC metric model. Each object in the model is identified by a UUID and some objects have additional metadata that can provide human readable names (e.g. experimentId, entity Id). The generic nature of the ECC allows experimenters to dynamically define entities of interest to them and does not currently impose naming conventions. Although this offers a flexible approach it does not encourage best practice. If an experiment was completely in charge of what was reported to the ECC then the naming convention would only matter to the experimenter and how they configure their software. However, an experimenter will use baseline components (which report to the ECC), other 3rd party services (e.g. Facebook) and their own software. Identifiers chosen by the experimenter must be consistent where possible with the identifiers assigned in other contexts.

	UUID (uuid)	name (String)	description (String)	entityID (String)	experimentID (String)
Client	x	x			
Experiment	x	x	x		x
MetricGenerator	x	x	x		
Entity	x	x	x	x	
Attribute	x	x	x		
MetricGroup	x	x	x		
MeasurementSet	x				
Measurement	x				
Report	x				
Metric	x				
Unit		x			

Table 1: Identifiers in the ECC metric model

Here we define the principles for naming entities and attributes reported to the ECC.

- Entity identifiers are assigned when an entity is born by another entity responsible for creating them.
- Entity identifiers should be unique enough so that they do not clash in a context of use (e.g. within a set of experiments).
- Entity identifiers should be structured according to URIs where possible.
- Entity identifiers based on URIs can be dereferenceable but this is not mandatory.

Figure 11 shows an example of how identities are assigned to entities for the SCC baseline component. The figure shows the actor responsible for assigning the identifier, the entity and the identifier itself. The figure also shows type annotations (in italics) which must also be associated with identities. In this case when the SAD Service is deployed it is uniquely identified by a URL where the service is hosted. The SAD Service includes a set of software plugins responsible for social analytics. Each plugin is identified by a URI prefixed with the SAD Service URL. The plugin URI does not need to be dereferenceable. The SAD plugin accesses and analyses entities from Facebook and Twitter. Here the Facebook Event is identified by a URL assigned by the Facebook Service and the Facebook Account is identified by a URL assigned by the Account Owner.

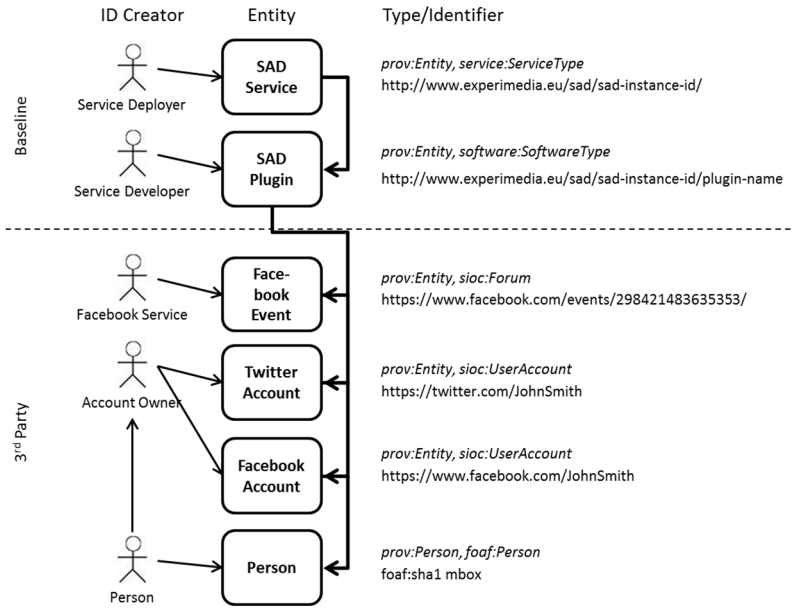


Figure 11: Assigning Identity for a baseline component and 3rd party service

The Person entity is a special case. In fact physical objects such as people are not created by the system under test. They are objects that exist in the real world already and become known to system either through configuration or through observations during monitoring. People do not have a unique identifier but are identified by a set of characteristics such as name, address, email and account ids. People can be known in advance to a system through a user account registration or may appear through interactions.

We propose to describe the Person entity using the FOAF[13] ontology. Using FOAF a person can be described using various attributes. The following example comes from "An Introduction to FOAF".[14]

```
<foaf:Person>
   <foaf:name>Peter Parker</foaf:name>
   <foaf:gender>Male</foaf:gender>
   <foaf:title>Mr</foaf:title>
   <foaf:givenname>Peter</foaf:givenname>
   <foaf:family_name>Parker</foaf:family_name>
   <foaf:mbox_sha1sum>cf2f4bd069302febd8d7c26d803f63fa7f20bd82</foaf:mbox_sha1sum>
   <foaf:homepage rdf:resource="http://www.peterparker.com"/>
   <foaf:weblog rdf:resource="http://www.peterparker.com/blog/"/>
</foaf:Person>
```

Email address is an important attribute for identifying people on the web. FOAF defines a <foaf:mbox> property:

```
   <foaf:mbox rdf:resource="mailto:peter.parker@dailybugle.com"/>
```

FOAF does not an assign a URI to the resource called Peter Parker, i.e. there is no rdf:about attribute on the foaf:Person resource:

[13] http://www.foaf-project.org
[14] http://www.xml.com/pub/a/2004/02/04/foaf.html

```
<foaf:Person rdf:about="..uri to identify peter..."/>
```

That's because there is still some debate around both the social and technical implications of assigning URIs to people. Which URI identifies you? Who assigns these URIs? What problems are associated with having multiple URIs (assigned by different people) for the same person? Side-stepping this potential minefield, FOAF borrows the concept of an "inverse functional property" (IFP) from OWL, the Web Ontology Language. An inverse functional property is simply a property whose value uniquely identifies a resource.

The FOAF schema defines several inverse functional properties, including foaf:mbox, foaf:mbox_sha1sum, and foaf:homepage. An application harvesting FOAF data can, on encountering two resources that have the same values for an inverse functional property, safely merge the description of each and the relations of which they are part. This process, often referred to as "smushing", must be carried out when aggregating FOAF data to ensure that data about different resources is correctly merged.

The ECC will use a prioritised list of foaf:Person properties (email, homepage, userAccountId, etc) to identify people depending upon what information is available about that Person.

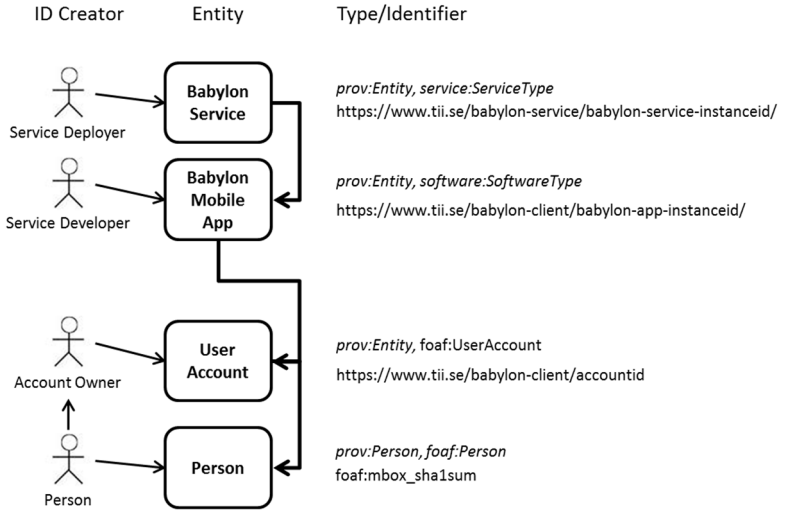


Figure 12: Baseline component and known user account

Figure 12 shows an example of a baseline component and a known user reporting quality of experience using the Babylon Mobile application. This is a general pattern for identifying users from existing user accounts, in this case an account associated with an EXPERIMEDIA baseline component. The Babylon service is assigned a URL when it is deployed by a Service Deployer. The instanceId of the Babylon Mobile App running on the mobile device will be determined by the Service Developer. This URI does not have to be dereferenceable. The User Account is assigned by the Account Owner when they register and the Person who is the account owner is assigned an identity based on an email address.

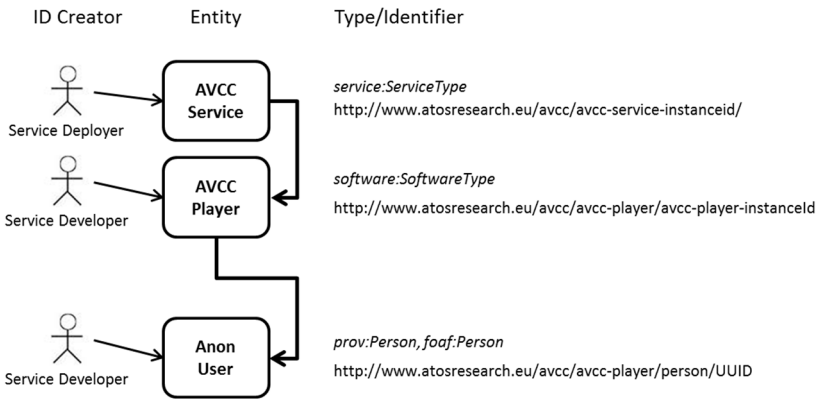


Figure 13: Baseline component and unknown user identifier

Figure 13 shows the situation where a component knows there's a user interacting with a system but does not have an identifier. In this case a User is viewing a video through the AVCC Player in a web browser. The AVCC player knows a Person is interacting with the video but has no specific identifiable attributes about that person. Of course other tracking information such as location and time could be used later to identify that the interaction was caused by a specific individual but at the time of interaction this is not known. What's important is that an Anon User is recorded in the system responsible for the interactions with the AVCC player. Any identifier could be used and in this case we just assign a URI where the personID is a UUID.

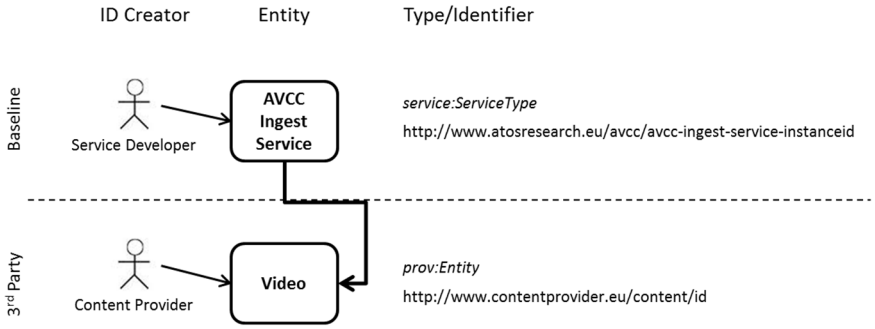


Figure 14: Baseline component and video data asset

Figure 14 shows the situation a Content Provider is ingesting a video into the AVCC. In line with the principles about it is the responsibility of the Content Provider to assign an identifier to the video.

Finally, it's important that we not only define consisting naming for specific entities but also entity and attribute types. Globally shared identifiers of attributes are useful in displaying data. If an attribute was identified as a certain type then it could be plotted accordingly in the dashboard. The ECC will use the following scheme:

- **Entities**: Entities will be described by URIs pointing to concepts in semantic models. All entities will have type prov:Entity. Additional concepts can be annotated by baseline or experimenter components through the ECC Client API, if no concept is provided the type will default to the prov:Entity. Entities can subclass multiple concepts.

- **Attributes**: For most attributes the Unified Code for Units of Measure (UCUM) will be used to describe attributes. For attributes that are not covered by UCUM, for example geo-locations, colours, emotions, etc, alternative well known descriptors will be used.

4.5. Experiment monitoring process

Metric generating software clients that engage with an ECC based experiment go through a process that may include up to six distinct phases. The initial two phases: 'connection' and 'discovery' are mandatory; the remaining parts of the process are optional. A high level representation of the interactions between metric generating clients and the ECC for each of the phases is shown in Figure 15.

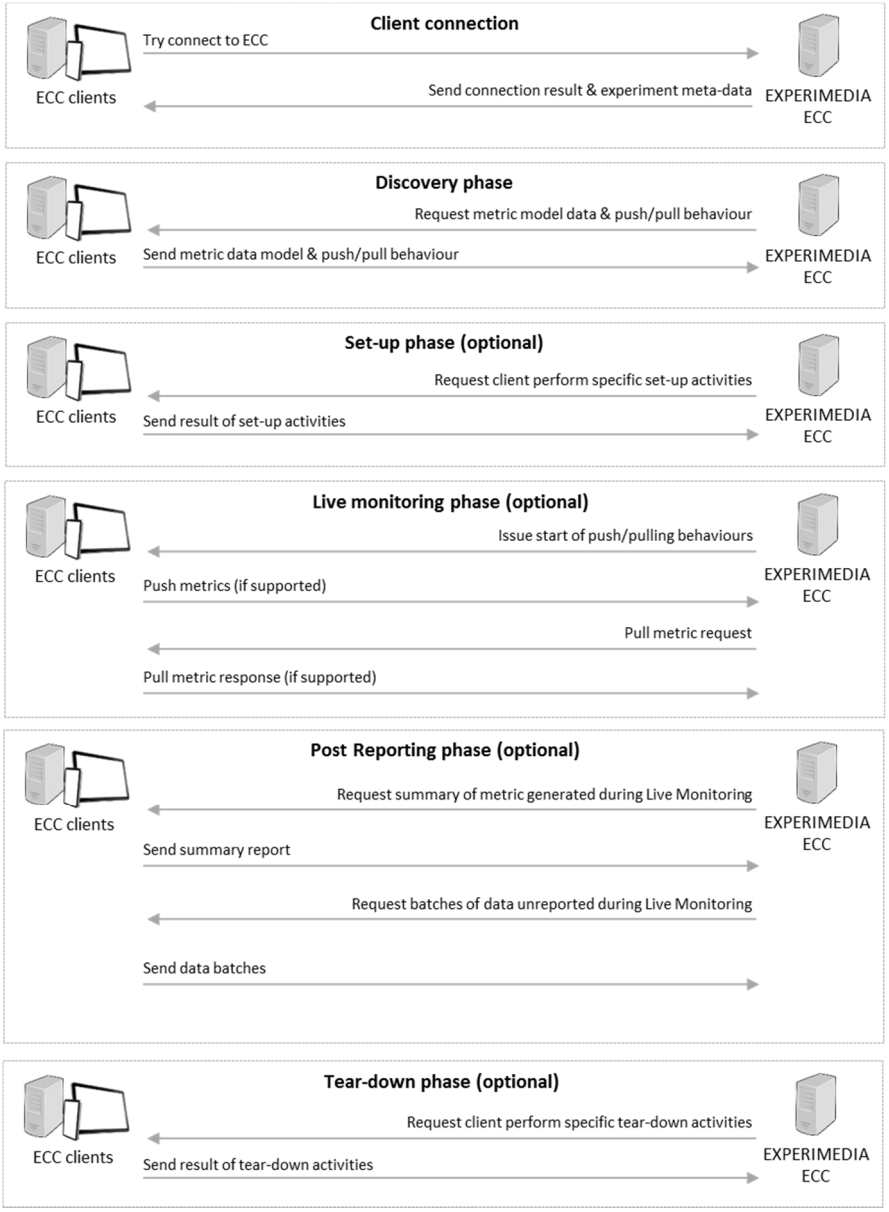


Figure 15: Experimental Monitoring Process

Whilst an ECC client developer will need to be aware of the experiment monitoring process described above, many of the interactions between the ECC and their instrumented software are

handled at a low level by the ECC client writer's API. In the following sections, each of the phases depicted above are described in more detail with an outline of client side behaviour.

Once a client has reported their capabilities and metric descriptions, it may enter a **Set-up Phase** (if it supports it). Here, the ECC requires the client to progressively set up the metric generators they have available for use. Clients supporting this phase respond with the result of each set-up attempt.

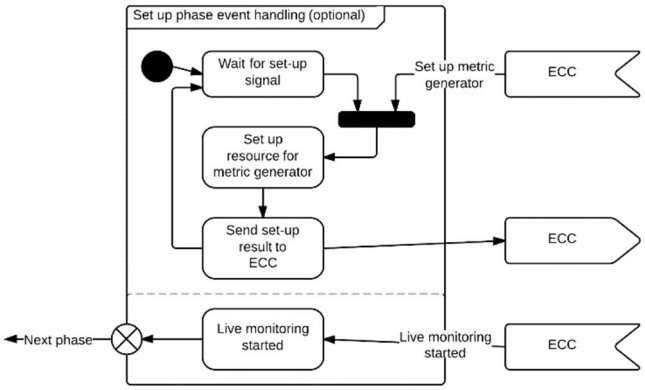


Figure 16: State Model for Setup Phase

The ***Live Monitoring Phase*** is the main part of the experimental process in which the ECC gathers metrics from all connected clients. Clients will have specified whether they support the pushing or pulling (or both) of metric data by the ECC. In the former case, clients are able to push any metric of their choosing on an ad-hoc basis (they should always wait for an acknowledgement from the ECC after each push, however). Alternatively, clients may be pulled for a specific measurement (identified in their specific metric model) by the ECC; a pull request is sent to the client on a periodic basis – it is the client's responsibility to return the appropriate measure. This phase continues indefinitely until the experimenter concludes that sufficient measurements have been taken.

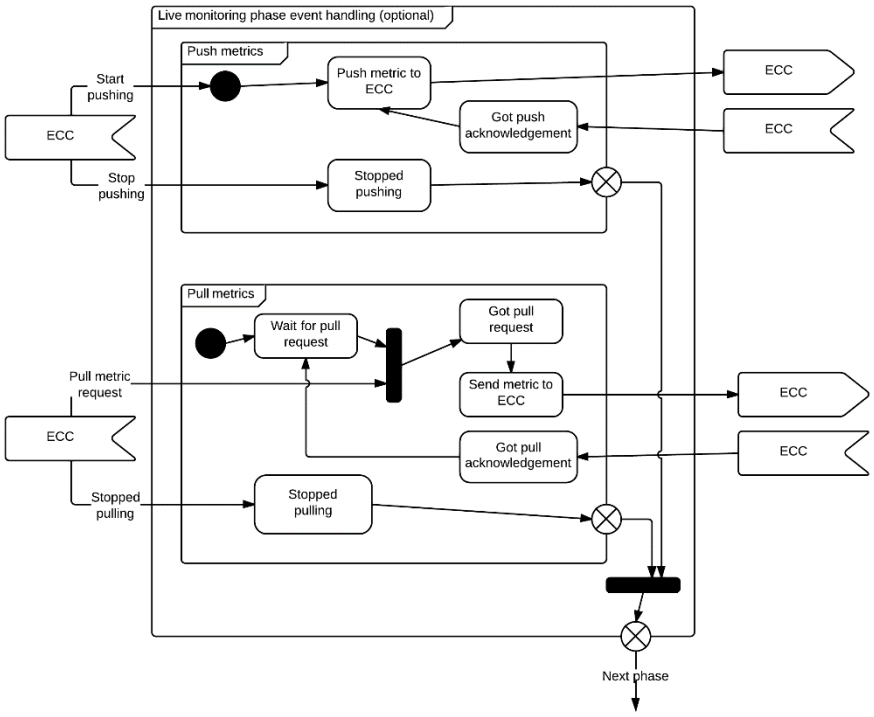


Figure 17: State Model for Live Monitoring Phase

After the live monitoring phase has completed, the ECC will contact the appropriate clients to begin the ***Post Reporting Phase***. The purpose of this phase is to allow the ECC to retrieve metric data that was not possible to collect during the Live Monitoring phase. For example, some clients may generate data too quickly or have a network connection that is too slow for all of their data to be transferred to the ECC in time. During this phase, clients will be requested to first provide a summary of all the data they have collected during the Live Monitoring phase, and then be asked to send metric 'data batches' that will allow the ECC to complete its centrally stored data set for that client.

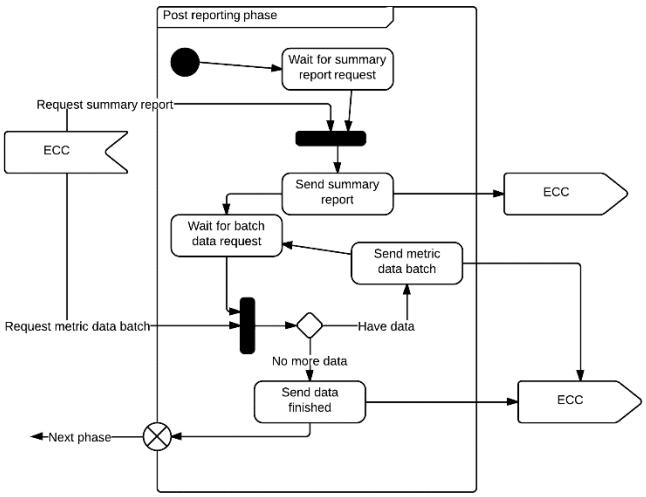


Figure 18: State Model for Post Reporting Phase

Finally, some clients may be able to report on their Tear-Down process for some or all of their metric generators. In some cases, it will be useful for the experimenter to know whether the tear-down process has succeeded or not. For example, the experimenter will need to know whether or not users (represented by the connected client) have been successfully de-briefed on the completion of an experiment.

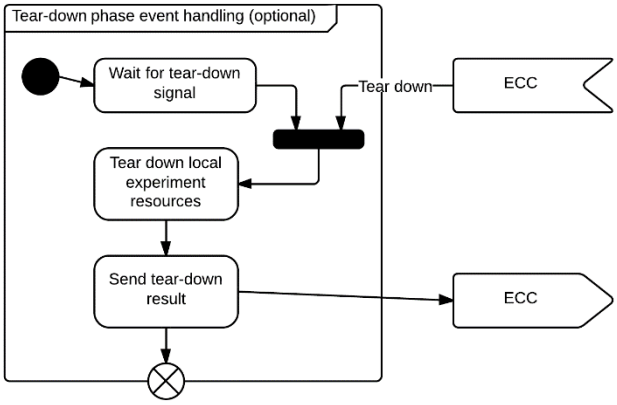


Figure 19: State Model for Tear-Down Phase

4.5.1. Monitoring Sources

Before an EXPERIMEDIA project can execute an experiment, a set of FMI technologies and services taken from the baseline components or project specific technologies must be selected for instrumentation. The metrics provided by each instrumented component will reflect the observational requirements of the experiment design and are specified using the ECC metric data model (see Section 4.2). An API supporting this data model and communication with the ECC using AMQP is available for the following technology platforms:

- Java (common JRE)
- Java on Android
- C#
- C++
- Ruby

While the low-level implementation details for each platform of course vary, the general architectural pattern for the client ECC architecture is common:

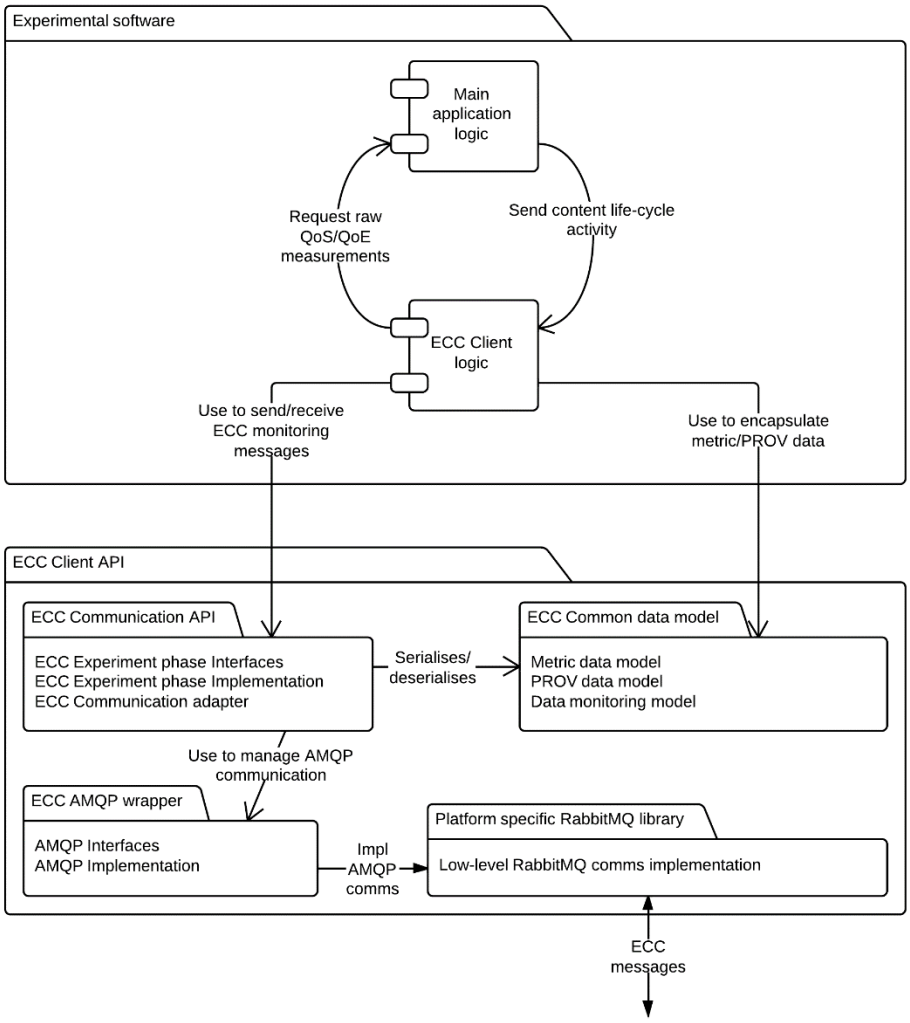


Figure 20: Inter-dependences for instrumented software

Figure 20 shows a high level overview of the inter-dependencies between an instrumented piece of experimental software and the ECC client API. Above, the 'ECC client logic component' adopts a controller role: responding to requests for QoS/QoE data from the ECC and also sending on interesting content life-cycle based provenance data, as it occurs. For convenience, a number of helper classes exist to help the client writer:

- An adapter class that simplifies communication with the ECC, offering simple communication methods and alerts to ECC monitoring requests
- A data helper class to quickly create instances of the ECC common data model

A selection of source code based examples of ECC clients can be found in the ECC software distribution within the 'samples' folder.

4.5.1.1. Monitoring Web Sites

In two of the projects funded by the first open call, DigitalSchladming and MediaConnect, experiment participants used web interfaces for some or all of the experiment. There is a wealth

of data that is commonly gathered by websites using technologies such as Google Analytics[15] or Piwik[16]:

- Interaction data:
  - which pages are visited;
  - which links are clicked on;
  - how long is spent on a page;
  - the duration of a visit;
  - which pages are viewed first and last.
- User data:
  - demographics (language and location);
  - technology they are using (browser, OS, network);
  - what type of device they are using (desktop, model of phone or tablet, screen resolution);
  - whether they are a new or returning visitor.
- The reason the user arrived at the site:
  - what site they were browsing previously;
  - what search terms they used.
- The performance of the website:
  - the speed of page-loads.

This data is gathered using JavaScript code that is loaded along with the web page and sent to a server for storage and analysis. The systems make use of long-lived cookies stored in the user's web browser that uniquely identify them as they browse from page to page.[17]

Google Analytics (GA) is the best known web analytics system and provides an impressive web analysis dashboard to filter and navigate the data. Use of GA requires a Google account to be set up by the owner of the website and a code associated with that account to be inserted into each page to be monitored along with the necessary JavaScript. A useful feature of GA for experimenters interested in their system's user interface is automated A/B testing where different groups of users are shown different interfaces and the data analysed accordingly. Experimenters choosing to use Google Analytics for monitoring web sites must carefully consider (with the support of the Ethics Advisory Board) the privacy implications of sending data to Google's servers. Google Analytics provides a data export function and the ECC should provide tools to make importing data from GA into the ECC dashboard easy.

Piwik is describes itself as the "leading open source web analytics platform". Rather than providing a hosted web-service like GA, Piwik provide their service as software for the

[15] http://www.google.com/analytics/
[16] http://piwik.org/
[17] The latest Google technology, Universal Analytics, can also work without cookies.

administrator to download and install on their own server. It is basically very similar to GA: gathering much the same data and providing a good web interface as well as a REST API and data export facilities. If sensitive data was being gathered and stored then it may be a better option than GA as the data goes directly to a server under the administrator's control. Again, the ECC should provide tools to make importing data from Piwik into the ECC dashboard easy.

Google Universal Analytics[18] (UA) has recently entered a public beta test. As well as the web page monitoring provided by GA and Piwik it provides an API for Android and iOS apps and a "measurement protocol" for other digital devices. The measurement protocol[19] provides a simple method to send data from other devices to the Google servers. In a similar way to the ECC dashboard it allows developers to measure how users interact with separate systems but gather the data in a central location. The measurement protocol is much more limited than the ECC metric model, allowing the reporting of up to 20 named string variables and 20 named integer variables (or 200 for a premium account). The measurement protocol may be useful in some experiments as a way of recording additional data, but this whole product is in beta and not all features are currently working.

4.5.2. Reporting of Self

At present within EXPERIMEDIA, the report of the self is most commonly provided by Babylon - which offers the ECC QoE in two different formats: 1) nominal emotion values [6 positive & 6 negative] and 2) a position on the colour wheel (x axis = affective response [negative-positive] and y axis = [arousal bored/excited]). Within the ECC metric model, an entity representing a user performing a specific role ('AR explorer', perhaps) would be defined with attributes reflecting 'emotion', 'affective response' and 'arousal'. Measurement sets uniquely generated by each individual user would be linked to the appropriate attribute of a shared entity instance; aggregated and averaged measurement data from each client regarding the same observations of the self would therefore provide the experimenter with an overall view of user experience in this context.

[18] https://support.google.com/analytics/answer/2790010
[19] https://developers.google.com/analytics/devguides/collection/protocol/v1/

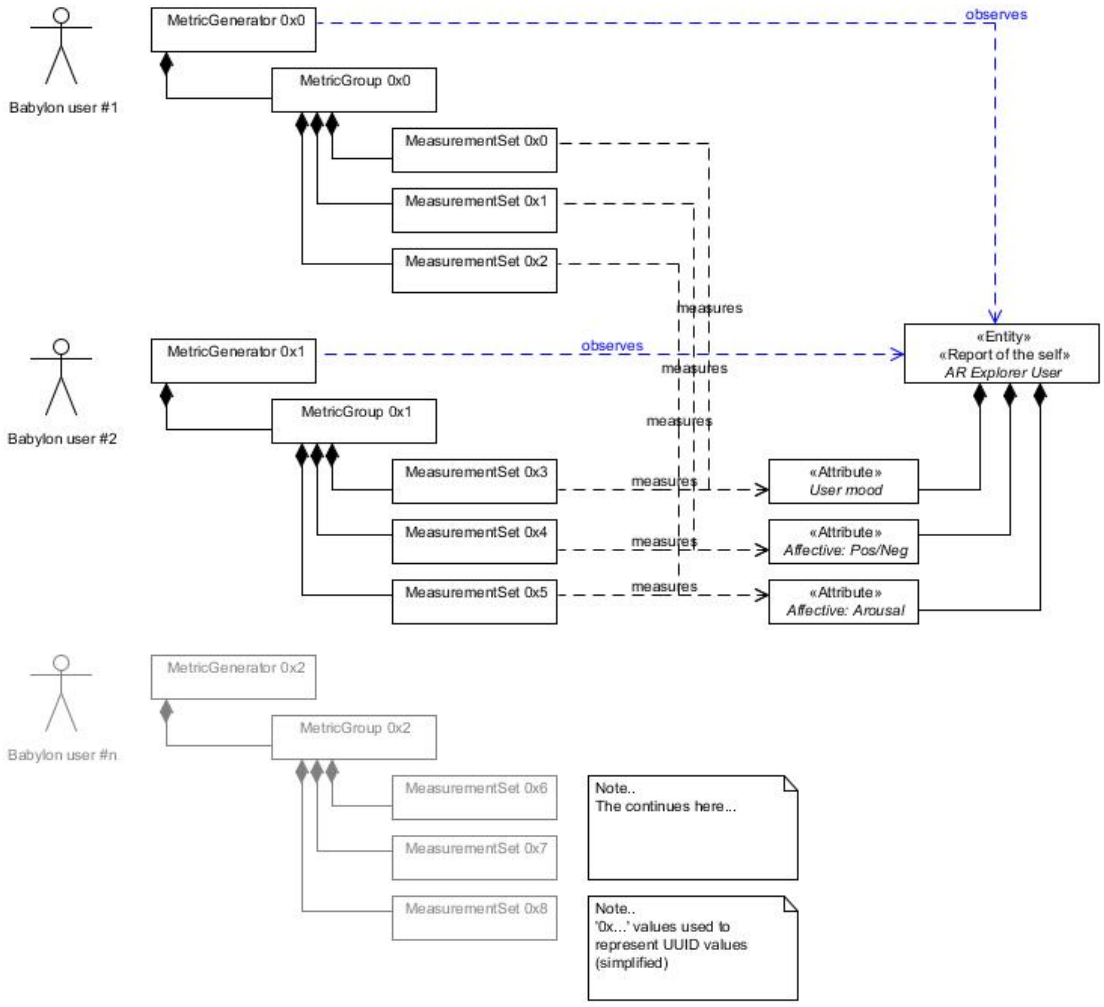


Figure 21: Reporting Self

4.5.3. Reporting perception of activity and usability qualities

Here, in both cases, the user will be reporting their attitudes/perceptions that relate specifically to a quality of an activity ('successful' or 'unsuccessful' training session, for example) or an interactive system that augments a particular activity (the AR client was 'responsive' or 'non responsive'). In many cases, a scale is used (Likert is usual) so that users can indicate their perception in degrees. Respondents are often required to indicate their attitude to the same aspect of an activity or system (again, in ECC parlance, this refers to an Attribute of an Entity) several times within a questionnaire (see Figure 22).

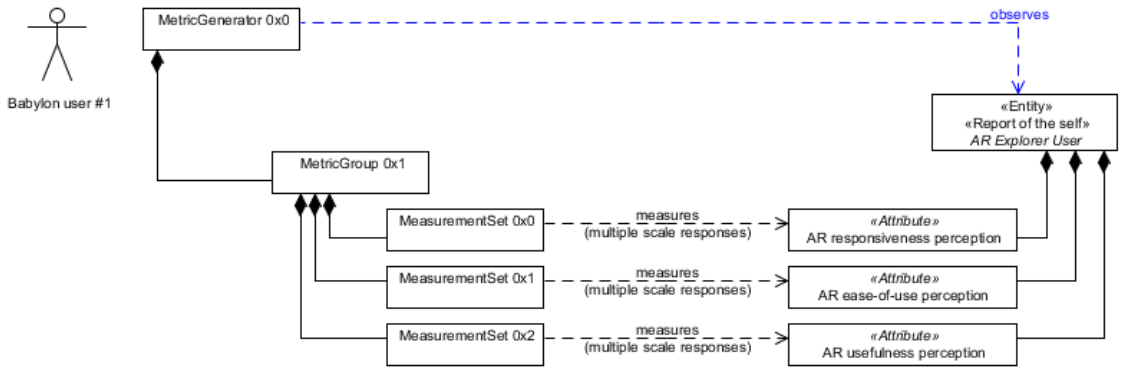


Figure 22: Measurement sets encapsulated in questionnaire metric group

From an ECC metric model point of view, each user's scaled responses (relating to a single attribute of an entity) would be would be held in a measurement set, which itself belongs to a metric group representing the complete questionnaire data set. Repeated samples from each user would then be averaged (noting responses that are either bi-polar or have a strong tendency towards the mean), giving an overall profile for a specific individual (uniquely represented by an ECC metric generator). Further aggregation of multiple users' attitudes towards a particular aspect of their experience makes it possible to see the overall attitude of a user population.

4.6. Provenance data model

So far we have considered the metric data that relates to QoS and QoE indicators of systems and users respectively. However, this data only provides a partial view of the overall behaviour of the experimental system – greater insight could be offered to the experimenter if he/she could further investigate the causes of the measurements that have been observed. Given such a view, the experiment can then frame the following types of question:

- What system or user driven content lifecycle events are associated with a specific set of metric observations?
- What are the differences in content lifecycle activities between related sets of metric observations?
- What effect did changes to digital content have on QoS and/or QoE?

Toward being able to answer this questions a data provenance modelling standard has been introduced into the EXPERIMEDIA experimental framework to support the traceability of interactions between systems and users. A substantial body of research in the area of data provenance is led by the W3C working group[20] on provenance; a high level characterisation of the properties of their provenance model is offered by the group as:

[20] http://www.w3.org/2011/prov/wiki/Main_Page

W3C PROV description (http://www.w3.org/TR/prov-primer/)

> *"The provenance of digital objects represents their origins. PROV is a specification to express provenance records, which contain descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object. Provenance can be used for many purposes, such as understanding how data was collected so it can be meaningfully used, determining ownership and rights over an object, making judgements about information to determine whether to trust it, verifying that the process and steps used to obtain a result complies with given requirements, and reproducing how something was generated."*

The key high-level concepts encoded in the W3C PROV ontology are *Entities*, *Activities* and *Agents* which are connected to one another using a number of common relationships, some of which are indicated in the W3C example seen in the figure below.

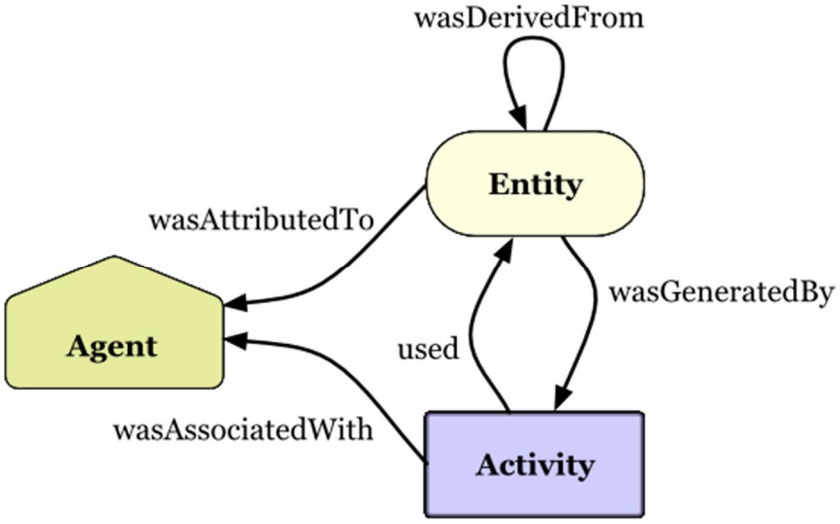


Figure 23: W3C PROV key concepts

Space in this document does not allow a full explanation of the W3C PROV model[21], however, in brief, according to the W3C ontology, an *Entity* may:

- Represent physical, digital or conceptual things
- Have relationships to other entities (such as a 'part-of' type relation)
- Have attributes that characterise an Entity (from different perspectives)

In Figure 23, we see a common relationship between an *Entity* and an *Activity* in that the former represents some action (generation) on the latter. Activities in this formalism represent:

- Dynamic actions/processes that affect change in the world
- The agency that affects change in the attributes of Entities

Finally, the *Agent* concept in the ontology provides:

[21] Interested readers should visit http://www.w3.org/TR/prov-primer/

- A representation of a thing (typically an *Entity*) that has taken on a role
- Whole or partial responsibility for an activity that has occurred

As activities that are driven by agents to generate or change entities occur over time, a historical record of these changes is built up in which new entities form relationships with older entities (such as the continued revision of a document's contents). Graphical timelines are often used to illustrate the chain of events and associated agent responsibilities; a simple of example from the W3C PROV group is presented below:

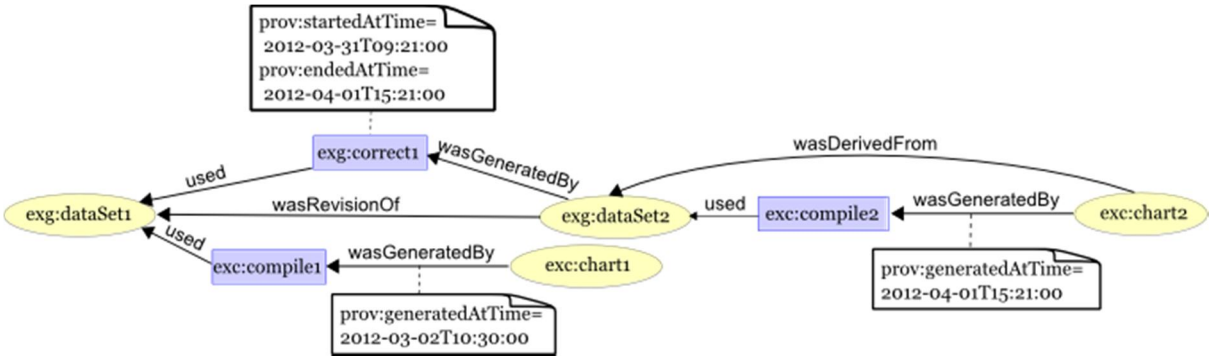


Figure 24: PROV timeline example

Above, we see how the entity 'dataSet1' is changed by an agent that corrects an attribute so that a new revision 'dataSet2' is created. We also see how two charts, which use data from the data sets are created and derived from the two different entity instances.

The V2 ECC architecture will take advantage of the overlap between Entities (and attributes) in from the existing metric model and the introduction of the W3C PROV ontology, using the ECC's common naming and ID management strategy to create the bridge between the two experimental data sets. It is anticipated that the integration of the EXPERIMEDIA metric model and the W3C provenance model will be initially scoped to cover a subset of the PROV-N[22] schema:

- Component 1: Entities, Activities, Generation, Usage, Start, End, Invalidation
- Component 3: Agent, Attribution, Association

Within the scope of the supported PROV data framework, additional semantic data may also be included to offer the experimenter further opportunities to examine behaviour within domain specific ontologies.

[22] See http://www.w3.org/TR/prov-n/#component1 for further information

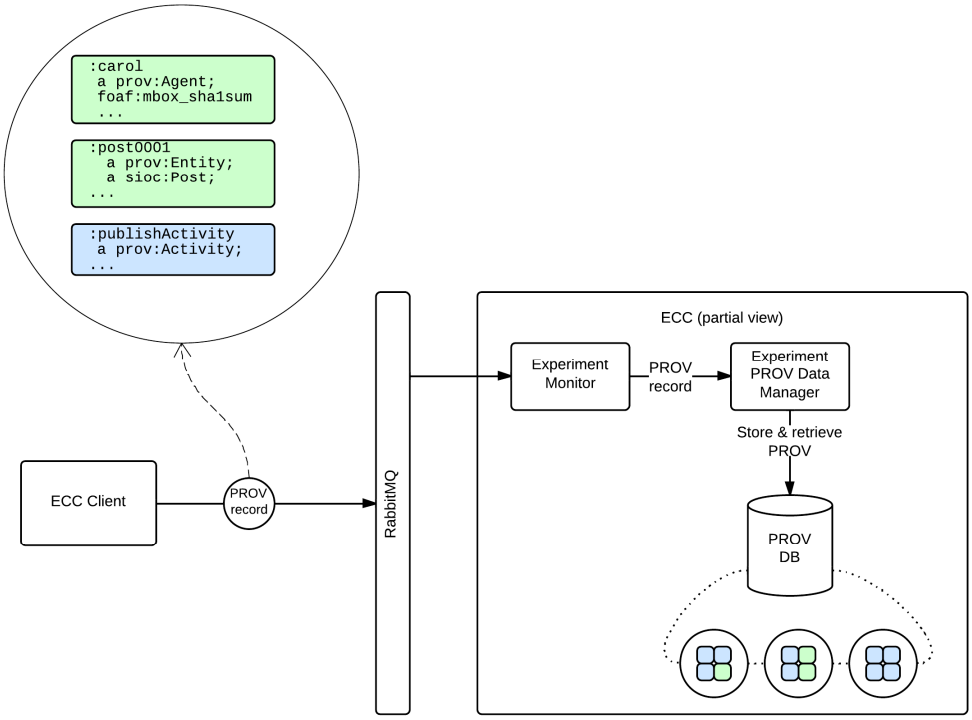


Figure 25: ECC client sends PROV data with mixed embedded ontologies

For example, where it is possible for an ECC monitoring client to do so, it could include additional semantics taken from specific ontological domains, such as SIOC[23] or FOAF[24]. In the figure above, we see a provenance record being sent to the ECC that embeds additional semantic FOAF related data about an Agent (Carol) and SIOC information about an Entity (an on-line post identified as 'post0001'). It is not expected that all provenance data sent to the ECC would carry domain specific information, but where it does, there is an opportunity to query and cross-reference data sets to learn something more about a specific context within the content lifecycle of an experiment.

4.7. Data Navigation, Analysis and Visualisation

4.7.1. Analysing QoS & QoE metric sets

It is useful to remind ourselves of some examples of QoS metrics from existing EXPERIMEDIA projects:

- Number of points-of-interest (POI) data (requests/hour)
- Average video stream rate (bytes/second)
- Average upload time (seconds/upload)

These metric types can be sampled continuously over time, easily lending themselves to temporal visualisation. After data collection is complete, QoS data can then be further processed using analytic approaches (there are many, of course) such as:

[23] See http://rdfs.org/sioc/spec/ for further information
[24] See http://xmlns.com/foaf/spec/ for further information

- Data filtering and segmenting (removing outliers, for example)
- Averaging repeated measurement sets taken over time
- Comparing distributions/variance between related measurement sets

Within just the QoS data context, we can start explore the performance relationships between technical components using correlation analysis such as the Pearson product-moment. Using such correlation techniques (assisted by scatter graph visualisations, typically) the experimenter can discover positive (or negative) relationships (such as an increase in rendering FPS might be positively correlated with an increase in rendering network traffic).

Again, let us return to some useful QoE examples already used within existing EXPERIMEDIA projects:

- Report of the self: Babylon emotion labels & colour wheel scales
- User interaction logging (discrete, nominal events)
- Perception of activity/system usability: questionnaires (Likert scales)

Unlike many QoS metrics, QoE type metric data is rarely available as a continuous sample stream. Users will interact or report emotions/perceptions/attitudes sporadically and in clusters. Depending on the experimental methodology, these data clusters are likely to appear in narrow temporal windows (perhaps before, and then after, as specific activity). QoE data can also be contrasted with QoE metric sets in that there will be many sources reporting on the same QoE focii (imagine 100 users, each independently taking part in activity 'X' and reporting on their level of interest in that activity).

A number of important data management activities face the experimenter once he or she has collected their metric data:

- Aggregating metric sets into a larger groups
- Pairing measurement sets for correlation analysis (and beyond)

The ECC metric model and underlying database has been designed so that it is possible to retrieve the appropriate data sets for these analytic exercises; subsequent developments in the ECC following version 2 of the EXPERIMEDIA architecture are planned to directly support these early analysis related activities.

4.7.1.1. Aggregating metric sets into larger groups

In both QoS and QoE cases, the experimenter may wish to aggregate measurement sets in order to draw some conclusions about the general behaviour of either:

- A single system or user[25] over a period of time (repeated measures pattern)
- Multiple systems/users' behaviours over the same time period
- Different groups of systems/users behaviours over a time period[26]

[25] Single user behaviour is only sometimes interesting however (in cases of 'outlying' data, for example).
[26] Comparing a control group with a conditioned group, for example

Aggregated metric sets are typically summarisations of some form of repeated observation; this could include averaged values; frequency analysis; other descriptive statistics (min; max; standard deviation etc).

4.7.1.2. Pairing metrics for correlation

Widely used forms of correlation analysis require the pairing of interval or ratio values from two data sets. As a rather contrived example, consider the pairing of a single value representing the average rendering speed (QoS, FPS) with a single value representing a user's perception (averaged) of system responsiveness (QoE, Likert scale -3 to +3). Let's imagine that we can match these values for 30 users who have taken part in the same experiment. Having generated these paired averages, we could plot these pairs on a scatter graph and apply a Pearson product-moment analysis. We might find there is a positive correlation between frame rate and the perception of system responsiveness.

4.7.2. Combining metric and provenance data

In the following sections, a more detailed description of the relationship between a provenance based view of the content lifecycle activity and the metrics based view of the experimental lifecycle within version 2.0 of the EXPERIMEDIA architecture is provided.

To illustrate the impact of how experimental metric and provenance data can be usefully combined, we will consider two paths along which an experimenter might travel to understand the meaning of the data he/she has collected:

- What causes can be attributed to set of QoS/QoE metric indicators?
- What is the result of system/user activities in terms of QoS/QoE indicators?

and a simple experimental context to understand the outcome. In the following sections, we explore the use of these concepts using an imagined experimental scenario; consider the following simple experimental deployment shown in Figure 26.

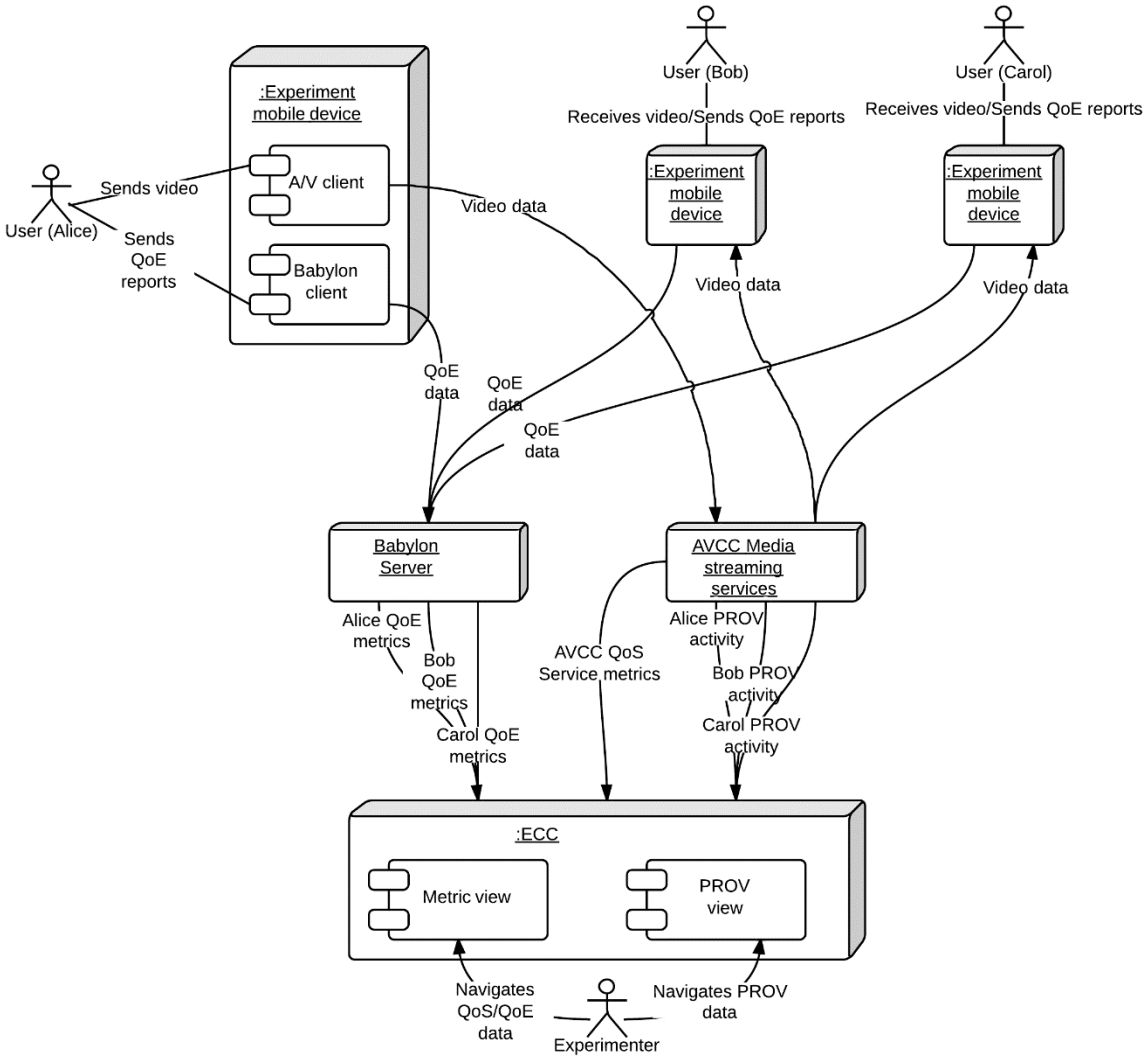


Figure 26: Experimental insight use case combining provenance and metrics

In this experiment we imagine a new video streaming service that offers a mobile user the ability to share a stream a video of her overlaid across a real-time 3D rendering of a virtual environment. Quality of service metric data reflecting CPU cycles dedicated to transcoding and streaming video streams are provided by the AVCC media streaming server to the ECC. Quality of experience metric data, reflecting each user's experience either as video stream providers or consumers is sent from each user to the ECC via the Babylon server.

During the experiment, Alice starts streaming live footage of herself combined with a 3D environment rendered by her mobile device to the AVCC media streaming service. Shortly after this has begun, Bob switches on his EXPERIMEDIA mobile application and logs in as a viewer of Alice's video stream. During mobile device usage, Bob subjectively reports on his quality of experience through the embedded Babylon client. A little while later Carol also connects to the AVCC media streaming service, starts viewing Alice, and also starts reporting subject QoE. After a few minutes, somebody starts sending some very negative QoE metrics via Babylon. What has happened and why?

In this example, we imagine that the experimenter has a metric view composed of:

- Selected Babylon QoE metrics:
  - User emotional state (Ordinal values)
  - User physical location (Interval values)
- AVCC media streaming QoS metrics:
  - Stream count (Ratio values)
  - Average stream bitrate (Ratio values)

Complementary to this view of metric values changing over time (reflecting the *experiment lifecycle*), the experimenter would also expect to receive aspects of data provenance activity that is specifically focussed on the *content lifecycle*. Initially, Alice starts her mobile application and initialises a video stream of herself; a provenance model of this first activity is shown in Figure 27:

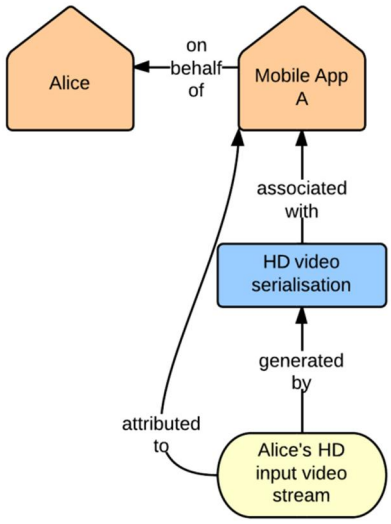


Figure 27: Provenance model for initialising the video stream

Above, we see "Alice" and "Mobile App A" as agents that are associated with serializing HD video data. This video stream (and related meta-data identifying Alice as a user of this service) is then connected to the AVCC streaming service, which starts de-serializing and caching the live HD video frames. These data related activities extend the provenance record further (See Figure 28).

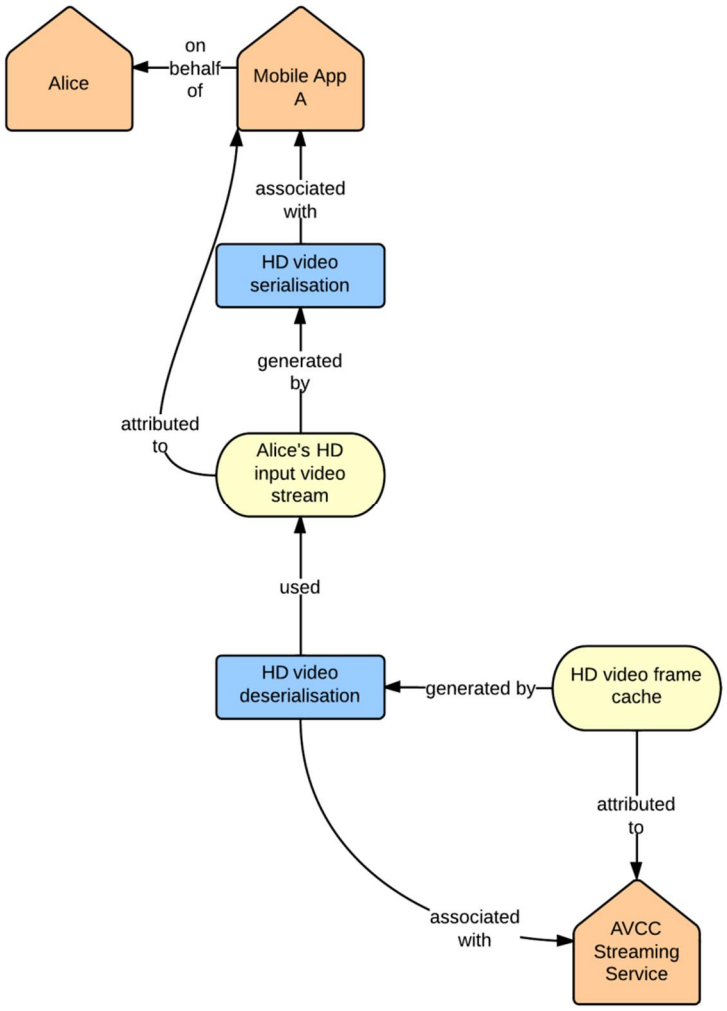


Figure 28: Extending the provenance record with AVCC streaming activities

In this addition to the provenance record, we see the AVCC streaming service agent enacting a video deserialization activity and generating an HD video frame cache for Alice's in-coming video data. Sometime after this, both Bob and Carol connect to the AVCC streaming service (using their respective mobile applications) to watch Alice. In the following provenance record, we will focus initially just on Bob's chain of activity. Bob's mobile application starts to pull an HD video stream from the AVCC service which is subsequently de-serialised so that he can view it (Figure 29).

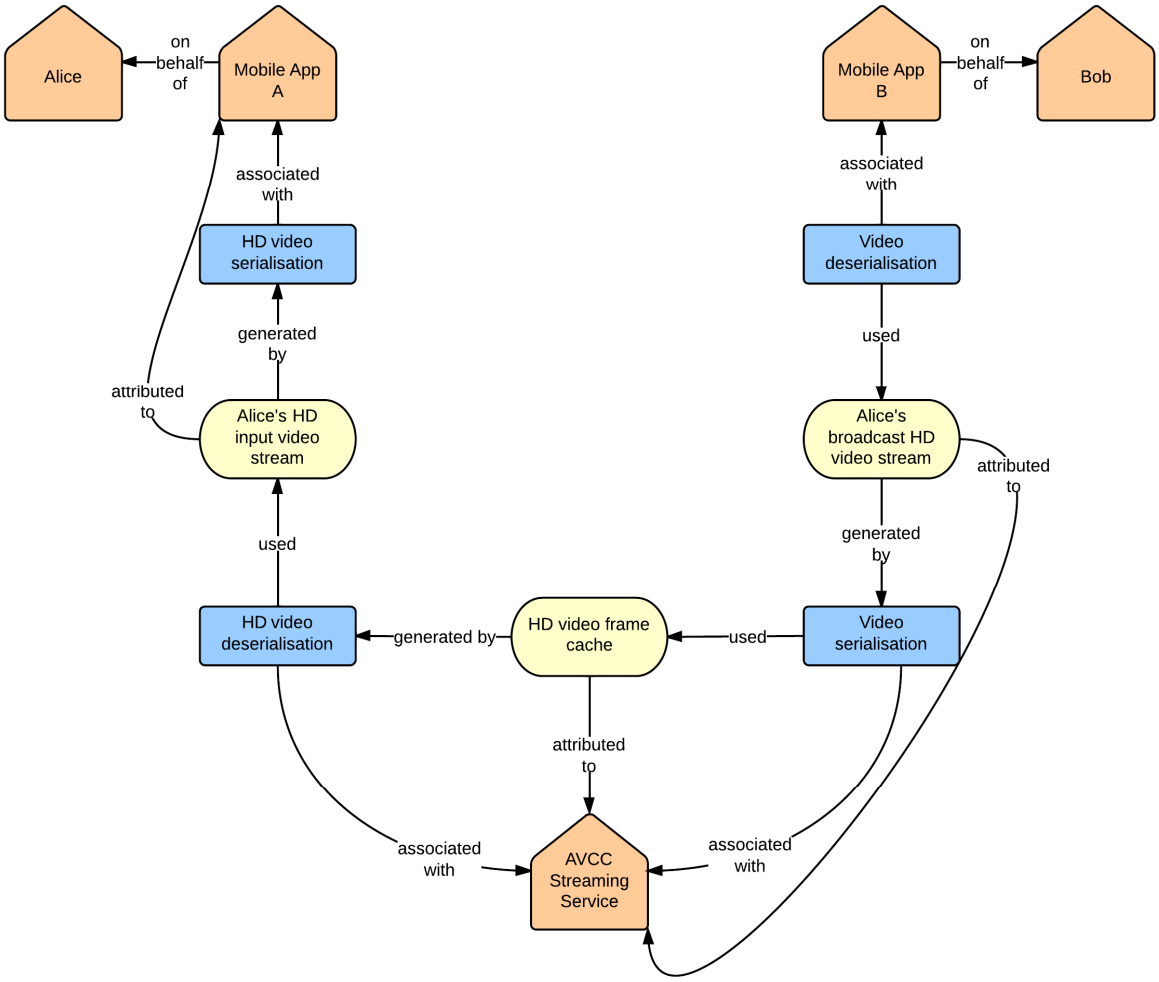


Figure 29: Adding video re-transmission to the provenance record

In doing so, the AVCC connects the contents of the HD video frame cache with a broadcast HD video stream, the content of which originates from Alice's input video stream. The completed PROV graph provides a connected content path with associated data processing between Alice's mobile device and Bob's – we would also expect to see a similar path related to Carol's later use. In the following sections, we examine how the complementary data collected from the experimental (metric) and content (provenance) lifecycles within the EXPERIMEDIA architecture assist the experimenter in gaining insight through experimentation.

4.7.3. Experimentation insight use-case: QoS & QoE indicators to system/user activity

In an attempt to understand what has happened during the course of the experiment described above, the experimenter might first look at the QoS/QoE indicators and notice that, after a period of time, the averaged emotion response reported by monitored users has moved from a broadly positive response to a more negative one (see Figure 30).

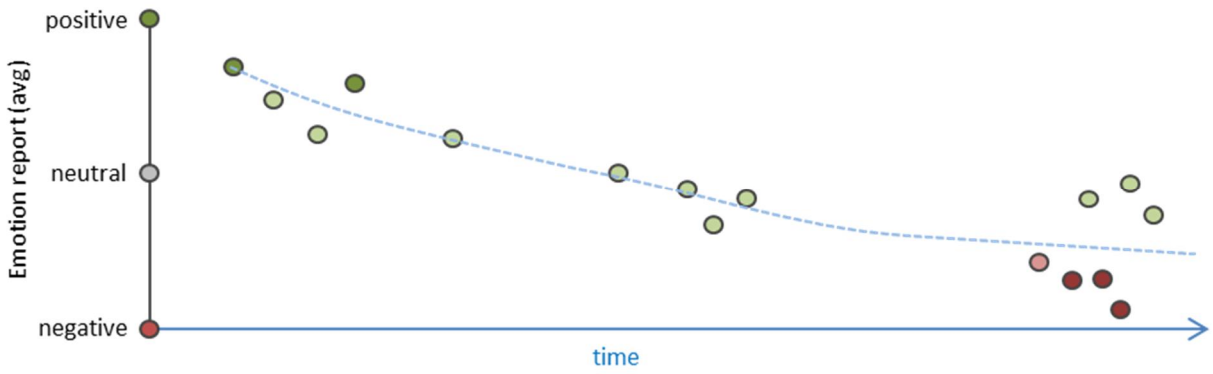


Figure 30: QoE self-report measurements (positive/negative scale)

Our experimenter decides to see if other metric data has a relationship with the averaged QoE samples and so investigates metric data for QoS stream count and average bitrate indicators.

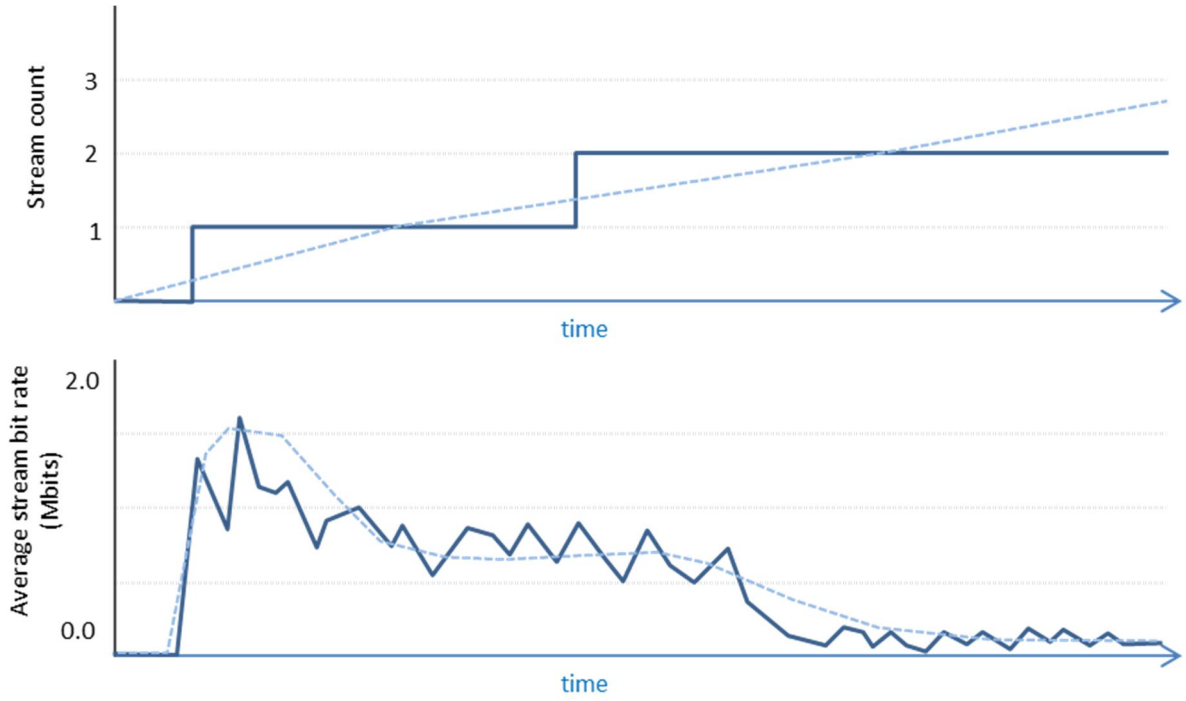


Figure 31: Metric timelines for the steam count and stream bit rate (average)

Examining the best fit curves for the averaged QoE and QoS metrics, the experimenter discovers a negative correlation between the emotion report and the stream count and a (weaker) positive correlation between the emotion report and the average stream bitrate. This seems to provide some initial clues as to what has happened, but the later QoE data is inconclusive as there seems to be a cluster of both negative and neutral reports over the period of time where average stream bit rate is low.

The experimenter decides she wants to see what happened in the content lifecycle to get a better understanding of system activity over this part of the experiment timeline. To do this, she starts by querying the metric data set to find the specific instances of the entities that reported QoE at the end of her temporal "data window":

Entity observation	Attribute	Entity URI	# of measurements
Video receiver	Emotion self-report	foaf:mbox_sha1sum(Bob)	4
Video receiver	Emotion self-report	foaf:mbox_sha1sum(Carol)	3

Here the experimenter finds two users, Bob and Carol, their related entity URIs (in this case, the "foaf:mbox_sha1sum" values that encode each user's unique e-mail box address rather than their actual names[27]) can be used to query the provenance records relating to the content life-cycle; this query is possible because the provenance agent representations of Bob and Carol carry the same URI used in the metric model's entity description.

Our experimenter first selects Bob and looks at just his QoE reports alongside his related provenance activity. This is illustrated in Figure 32: in this and subsequent figures the blue "activity" rectangles have been elongated to show their start and end times.

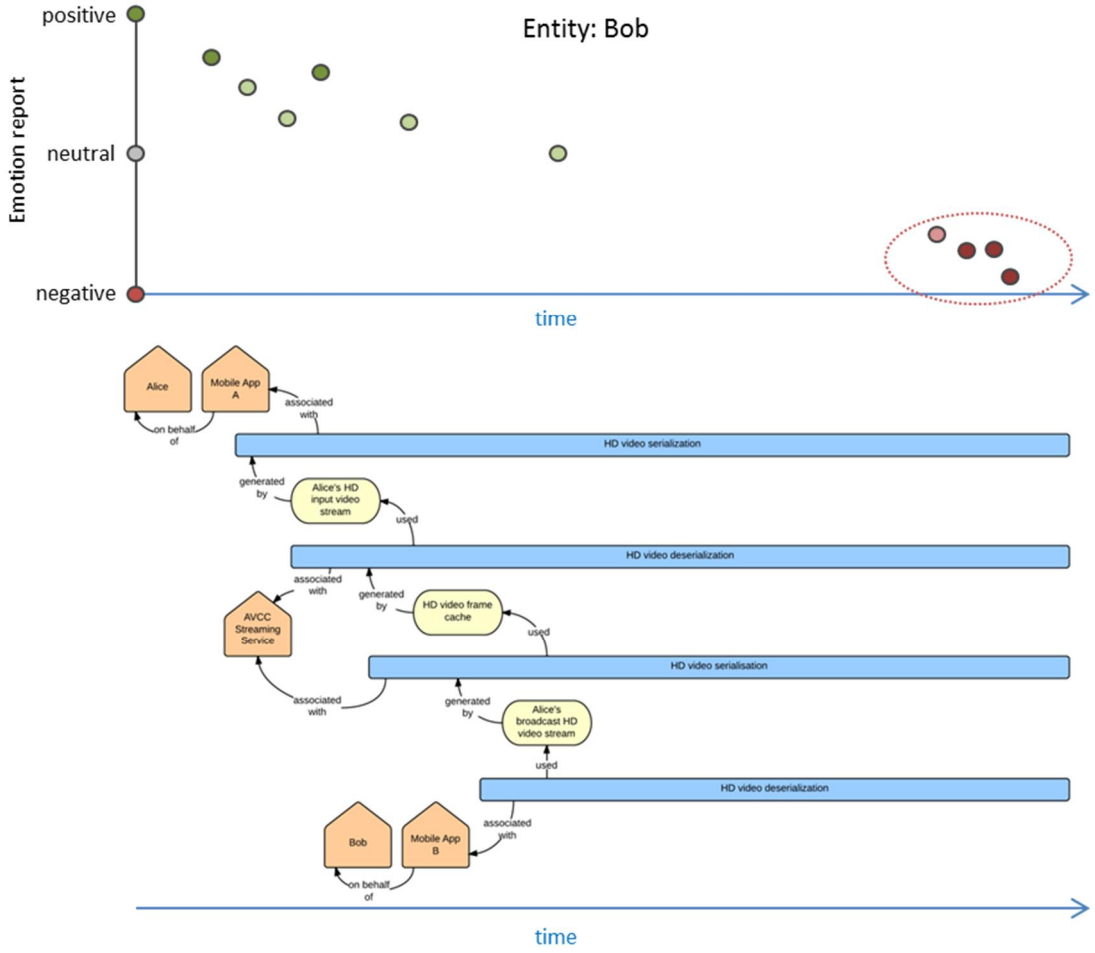


Figure 32: Bob's QoE reports set next to his PROV activity

The activity suggested by the provenance record does not seem to indicate any change between the initial comparatively good QoE reports and the subsequently poorer samples. Returning to

[27] See http://xmlns.com/foaf/spec/#term_mbox_sha1sum for further information.

her metric data view for both Bob and Carol, the experimenter tries to find further evidence of change by using more contextual QoE data, this time from the Babylon location data set (see Figure 33).

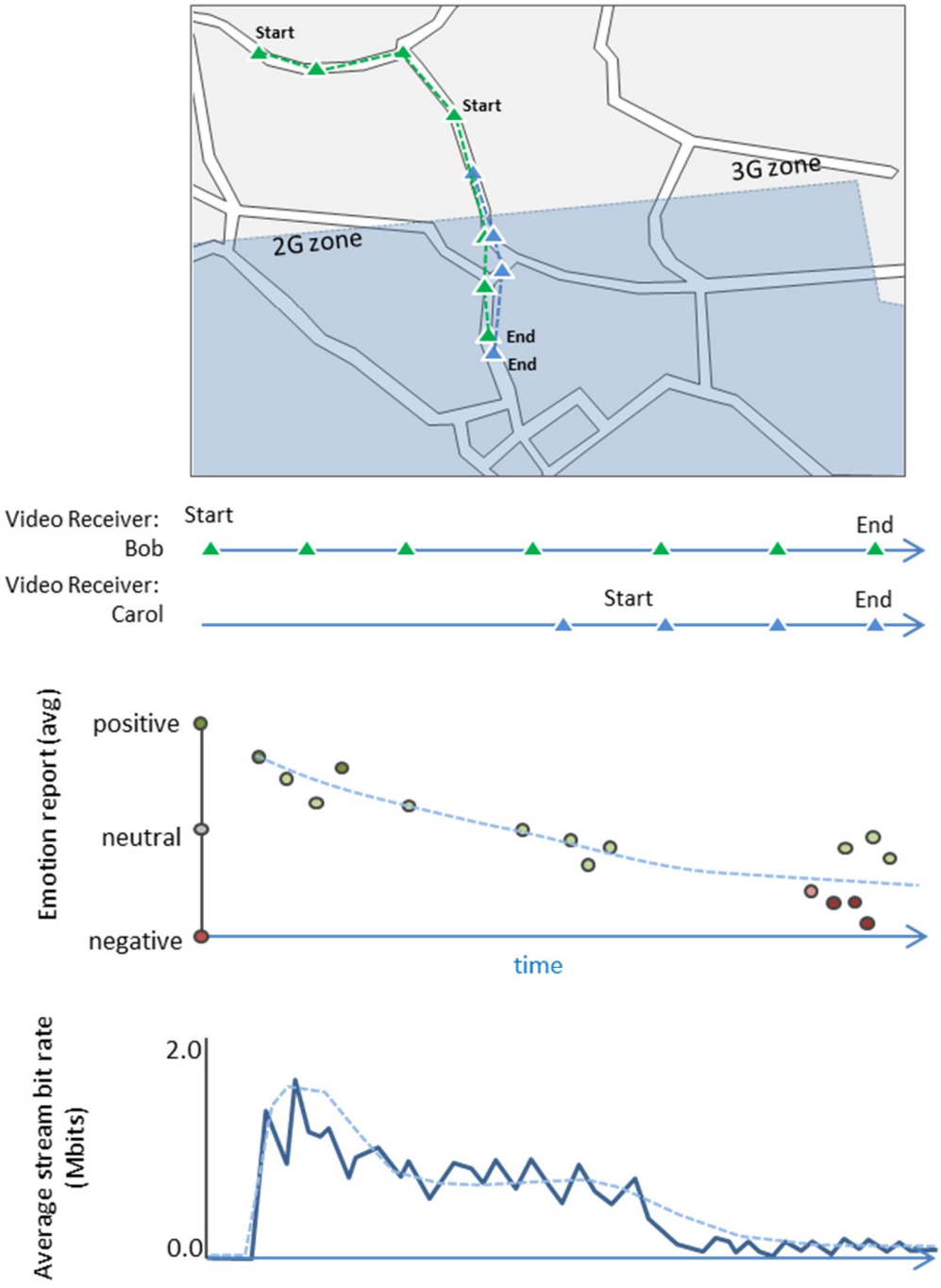


Figure 33: Bob and Carol's location data compared with QoE and QoS

At this stage it becomes clear that a probable reason for the overall drop in average bit stream bit might have something to do with the change in Bob and Carol's location which begins in a 3G mobile phone zone and then moves to a poorer 2G area. However, not all QoE samples toward the end of the metric record indicate a poor experience.

4.7.4. Experimentation insight use-case: System/user activity to QoS and QoE indicators

Our experimenter discovered some interesting overlaps between metric and provenance data in the previous section, but still cannot fully account for why QoS and QoE data for Bob and Carol changed in an unexpected direction. There was some indication that user location might have something to do with some of the observed changes, but the experimenter needs to find out more about the underlying system behaviour that ultimately lead to a different experience for the two users at the end of the experiment.

This time the experimenter retrieves provenance records for Bob and Carol (and all connected activities, entities and agents) to see if there are any differences in their comparative content lifecycles (see Figure 34).

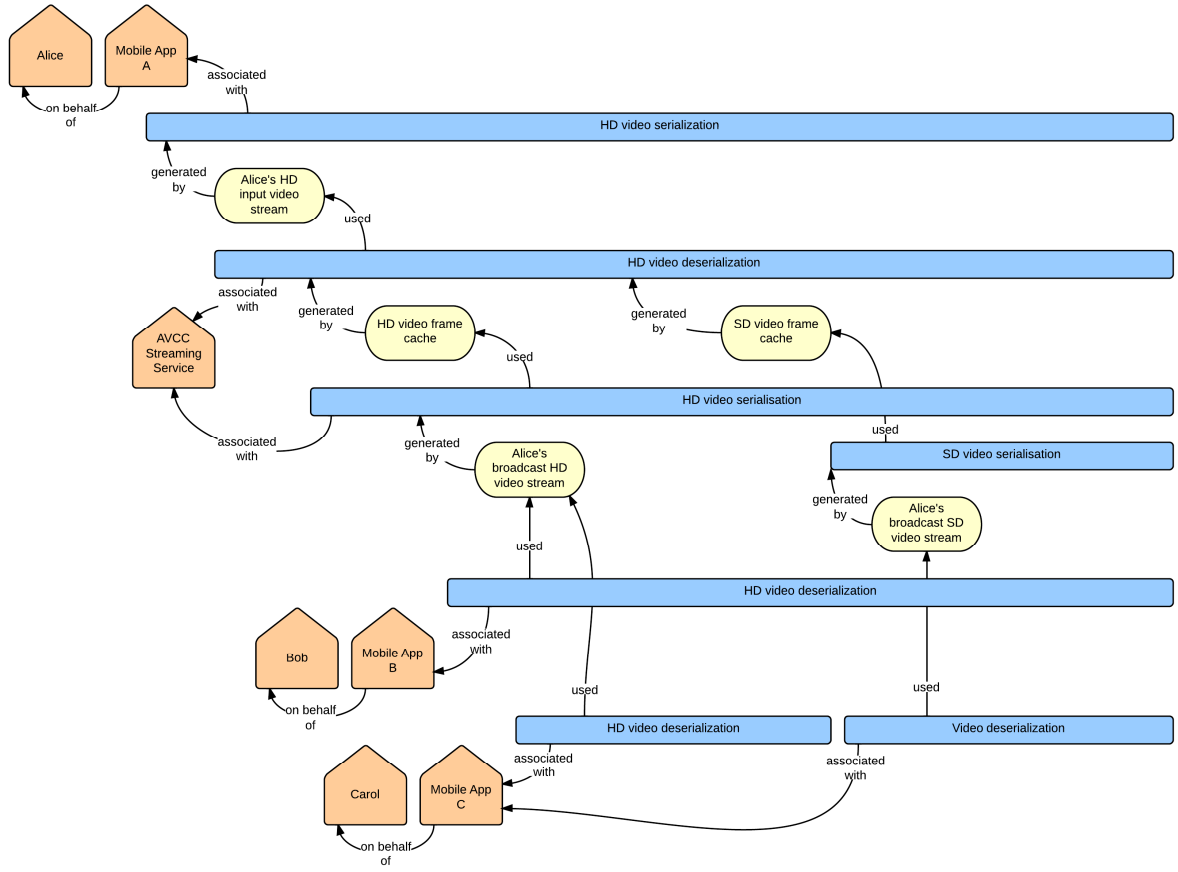


Figure 34: Provenance query for Bob and Carol (and related PROV records)

The provenance based timeline shows that the AVCC and Carol's mobile application interacted in a different way to Bob's over the latter part of the experiment. An examination of provenance record shows the AVCC created and started streaming Alice's video stream in SD resolution for Carol whilst attempting to maintain a HD broadcast for Bob. Comparing these provenance activities against Carol's QoE and the overall average bit stream rate seems to indicate the dynamic interactions between Carol's mobile application and the AVCC may be the reason why her quality of experience remained reasonable whilst Bob's did not (see Figure 37).

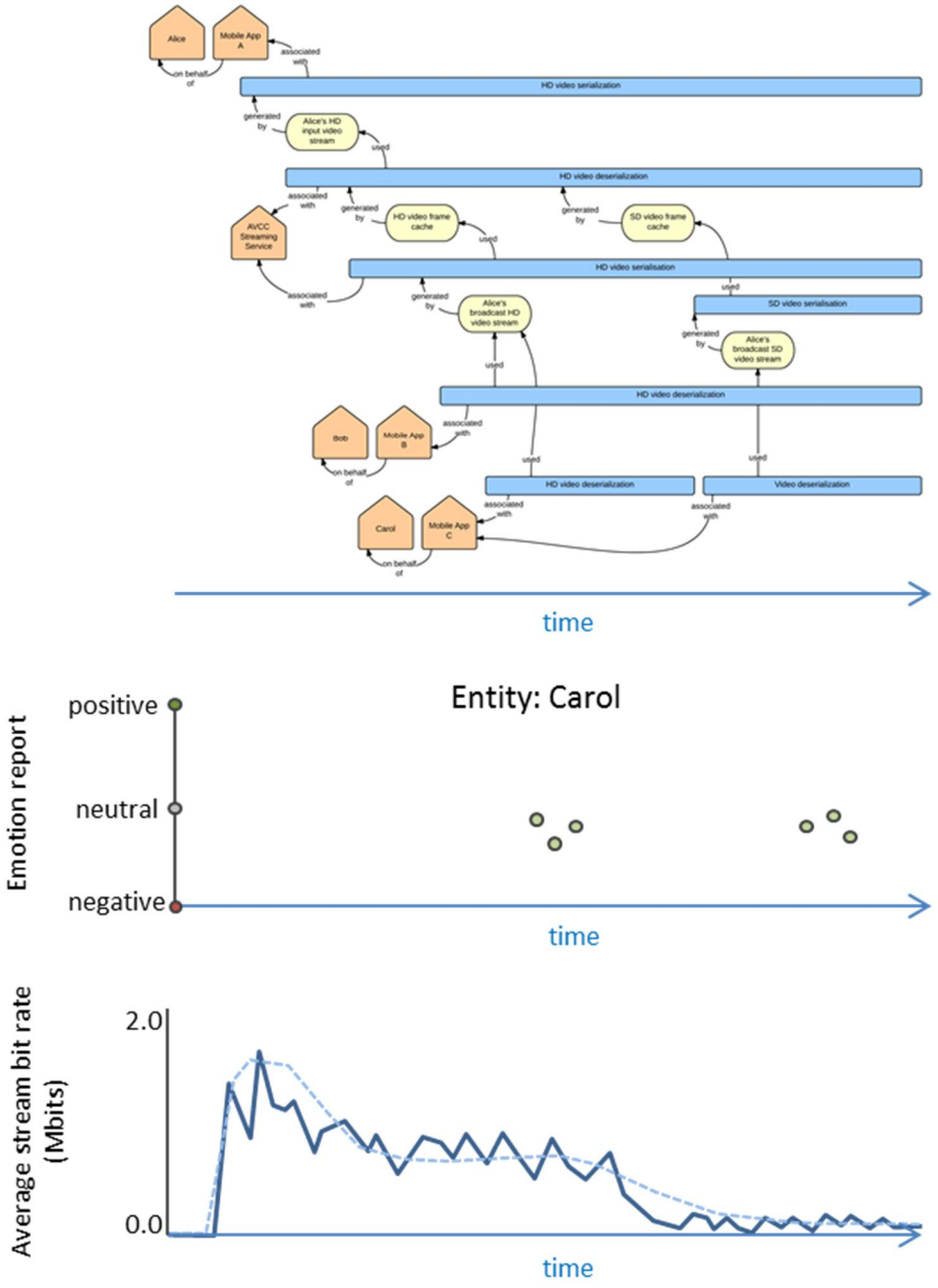


Figure 35: Timeline PROV data comparison against Carol's QoE and the AVCC QoS

This investigative process has shown how data derived from both metric and provenance records can be combined to better understand FMI technology behaviour, performance and contextual usage in a way that can lead to insights into the application of such technologies on users in real world scenarios.

4.8. Service Level Agreements

Service level agreements (SLAs) are primarily for setting the expectations of both parties of a bilateral agreement between a service provider and a consumer.

For the consumer the SLA defines what level of service they can expect and what redress they have if the service level falls below those expectations. This helps the consumer to compare

different service offers and understand the likelihood of service failure and whether contingency plans are required.

For the service provider the SLA puts a bound on the resources they need to commit to serving a particular consumer. It allows the service provider to plan the use of their resources efficiently: neither over- nor under-provisioning. A service provider will only put terms into an SLA about aspects of their offer that they can control. For instance, a cloud service provider will usually include a metric defining the "percentage up-time" for their service but will not include any guarantee or statement about the end-to-end latency of the network as the majority of the network path is out of their control.

In EXPERIMEDIA, some baseline components are offered as services by the core partners, such as the AVCC and Creator, but there is no central EXPERIMEDIA service provider. If EXPERIMEDIA used SLAs then an experimenter would need to agree SLAs with each service provider (see Figure 36) potentially using different SLA systems. Services and resources are instead offered to experimenters by the core partners on a "best effort" basis.

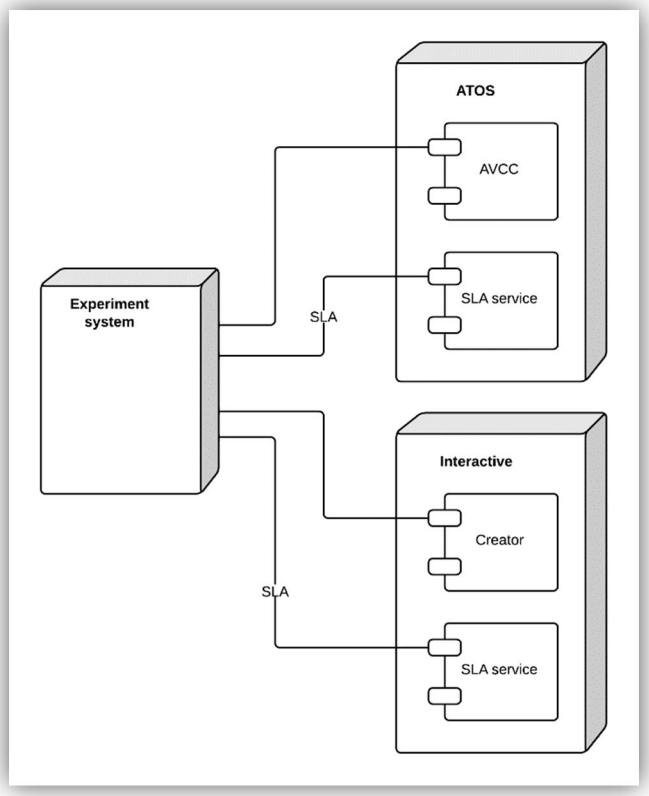


Figure 36: Illustration of bi-partite service level agreements

SLAs generally define measureable terms against which they can be judged. SLA systems therefore at a minimum must have some sort of monitoring and measurement infrastructure and ideally also include automatic control systems to adjust the resourcing for each SLA. EXPERIMEDIA uses the ECC to monitor experimental systems so that the experimenter can understand the end-users' quality of experience and relate that to the quality of service provided by the service components. The ECC can therefore service two purposes in relation to SLAs:

1) Providing information to the experimenter on the quality of service they received during an experiment so that they can understand what would be required if their system was deployed in a commercial environment.
2) If multiple EXPERIMEDIA services were deployed at a single partner site then the data already coming in to the ECC via the RabbitMQ bus could be used to compare against some SLA format such as that provided by SLA@SOI[28] and at a minimum raise alerts if service levels are not met.

We must remember though, that one of the architectural principles of EXPERIMEDIA is that the ECC can be removed from the system without breaking the end-user experience (see Section 2.3) so any adaptation of the ECC must be done with some care.

[28] http://sla-at-soi.eu/

5. FMI Content Lifecycle Management

5.1. Audio Visual Content Component (AVCC)

Audio visual content is primarily characterised by video and metadata that's streamed and consumed by applications (i.e. players). AV content is produced by professionals and users using content production, management and content distribution networks. The AVCC offers capabilities for all aspects of the content lifecycle (acquisition, production, transcoding, distribution, etc) and advanced capabilities for acquisition and synchronisation between cameras feeds, audio and metadata, including matching exact frames from different cameras.

5.1.1. Streaming

This subcomponent is in charge of content adaptation and delivering of all current industrial available streaming protocols. For live content this subcomponent also allows experimenters to transcode high quality content into different qualities adapting the content to the consumer networks capabilities. The streaming sub-component also allows the experimenter to record live content and inject real-time metadata which is multiplexed in the video stream and can be accessed by the player.

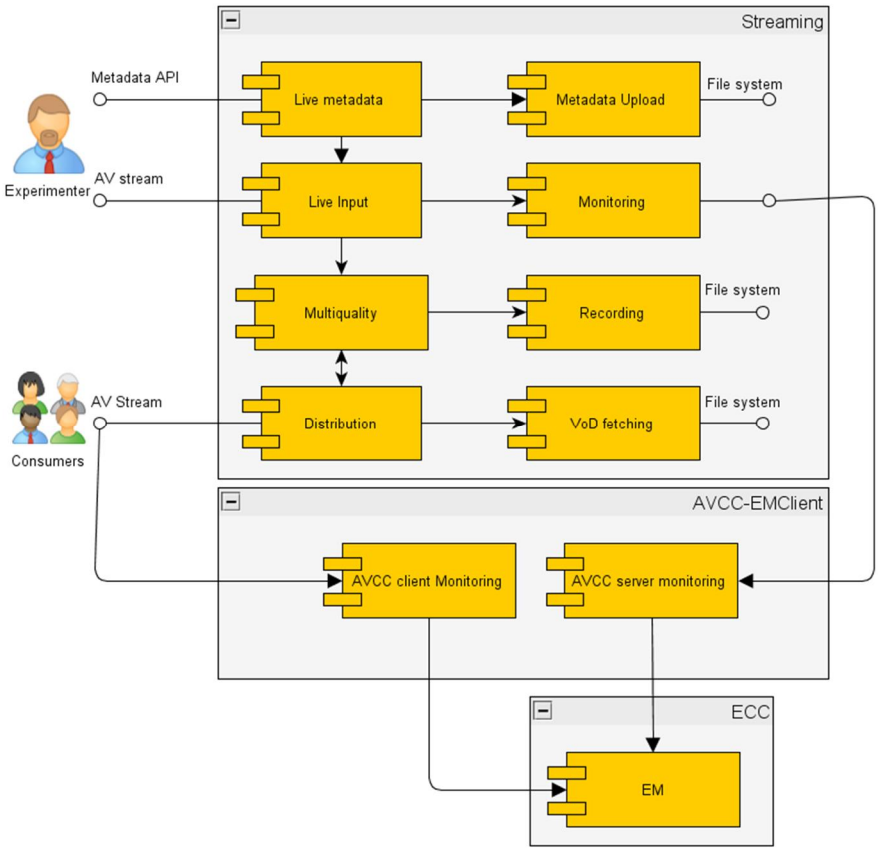


Figure 37: AVCC Stream deployed

This component is deployed at mediaserver1.experimedia.eu and in mediaserver2.experimedia.eu, however only mediaserver1.experimedia.eu has live transcoding capabilities.

Description of the internal subcomponents:

- **Live Input**: This module manages the reception of all live content including audio, video and metadata from the live metadata acquisition management. This module can receive as input video and audio from multiple formats, providing as output a decapsulated stream.
- **Live Metadata**: This module manages the reception of metadata and timestamps from the RHB Venue Information System. The metadata is converted into XML format and sent to the Metadata Upload module and the timestamps are sent to the Live Input module with a reference to the metadata so it can be synchronised with the video stream. The metadata format is defined by the Experimenter in such a way that depending on its content, the module behaves in different way.
- **Multi-quality**: The objective of this module is to perform transcoding into different bitrates in order to provide the same feed with different video qualities, so that the video player used by end users can change the quality depending on the network conditions and computer performance. In order to let the video player to change the quality smoothly and in a transparent way, the content is split into several chunks of a few seconds and aligned at the same frames. Additionally, the encoding is done in such way that complete GoPs are stored into each chunk.
- **Recording**: Records live video streams into mp4 files maintaining the original live metadata so it can be requested later as VoD files.
- **Media Distribution**: This module is in charge of the actual content delivery, which includes the continuous generation of a content manifest, final packaging of the content and transport protocols. It also produces all multiplexed media output of the main distribution. In case, Timeshift functionality is activated, the module continuously records live streams for immediate playback on deferred. It allows the user to have DVR experiences such as rewind, pause or fast forward.
- **VoD Fetching**: Retrieves recorded or previously uploaded VoD content under request of the edge distribution components. The content is accessed from the file system

5.1.2. VoD Ingest

The objective of this module is to support content adaptation and especially transcoding for VoD. The VoD ingest in integrated with the Stream platform so an experimenter can upload content and requesting multi-quality and the content will automatically be made available in HDS, HLS and Smooth streaming in pre-defined qualities.

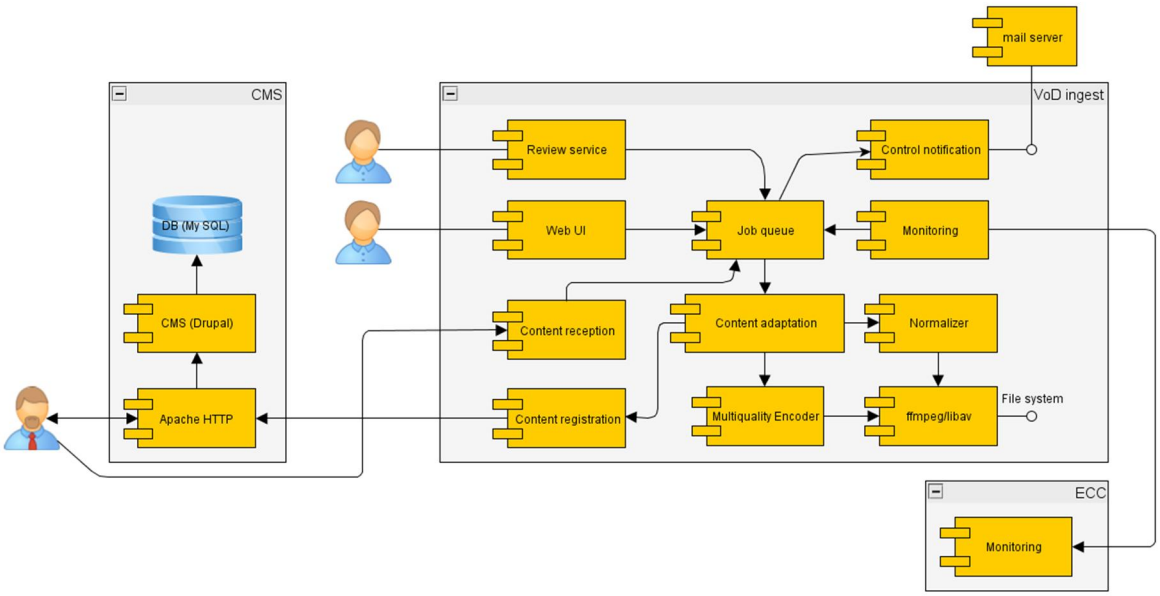


Figure 38: AVCC Ingest deployment

- **VoD Ingest – Review service**: This service allows a content reviewer to validate uploaded videos to approve or deny them. A link to this service is sent by email to the specified reviewer when content is uploaded.
- **VoD Ingest – Web UI**: This is the administration user interface to manually review the content queue and, if necessary, to manually trigger events.
- **VoD Ingest – Content reception**: This module receives the actual video in several POSTs request of fixed length packets. The module checks the MIME type, and generates a new UUID for the content and stores it into the server file system. After that, it creates a job into the Job Queue so it can be processed.
- **VoD Ingest – Content registration**: Once the content is processed, this module relays the information of the content together with the content URL to retrieve it to the CMS.
- **VoD Ingest – Job Queue**: This module keep control of the content flows, from the upload, control, adaptation and registration. It detects any problem and reacts / reports them properly.
- **VoD Ingest – Content adaptation**: This component, manage the process of adapting the content to the targeted profiles requested in terms of encoding and containers. It coordinates all the required transcoding which is delegated to the Multi-quality Encoder.
- **VoD Ingest – Multi-quality Encoder**: This module encodes the uploaded files to all qualities needed to support the multi-quality playing and the proper SMIL files for the content. The encoding is actually done by the underlying ffmpeg/libav libraries.
- **VoD Ingest – Normalizer**: Prepare the input video file to a mp4/h264 template using ffmpeg/libav so it will be used as a source by the Multi-quality Encoder.
- **VoD Ingest – ffmpeg / libav**: These open source libraries are used for encoding and decoding tasks.
- **VoD Ingest – Control Notification**: This module is responsible to generate the e-mail notifications to the content reviewers so they can approve or refuse the publication.

The VoD ingest is currently deployed in mediaserver2.experimedia.eu for several experiments. Each of the experiments has its own instance of the service with personalised configuration, following the name structure:

http://mediaserver2.experimedia.eu/ex[Experiment number]ingest/

Where [Experiment number] is the experiment number used in the DoW.

Under the experiment default path, the VoD ingest publish the administrative Web UI to monitor the pending tasks. Beside this Web UI, the VoD ingest has the following services:

Service Path	Type	Parameters	Type	Description
/UploadFile	POST	Token	text-field	Token expire in 3,5 h, only needed if token is active.
		title	text-field	String
		fulldescription	text-field	String
		event	text-field	Event relation in DB
		html5Mp4U	checkbox	Profile Selected
		html5WebM	checkbox	Profile Selected
		multi-quality	checkbox	Profile Selected
		uploadedfile	File	Binary file

The UploadFile service is the main service, which receives the source video file, basic description data and the content adaptation requests.

Service Path	Parameters	Type	Description
/avccinserter/avccupload	Token	text-field	Token expire in 3,5 h
	Title	text-field	String
	fulldescription	text-field	String
	event	text-field	Event relation in DB
	html5Mp4U	checkbox	Profile Selected
	html5WebM	checkbox	Profile Selected
	multi-quality	checkbox	Profile Selected
	uploadedfile	File	Binary file

Service Path	Type	Parameters	Type	Description
avccinserter/insert	POST/GET	Title	text-field	String

Service Path	Type	Parameters	Type	Description
		fulldescription	text-field	String
		event	text-field	Event relation in DB
		html5Mp4U	checkbox	Profile Selected
		html5WebM	checkbox	Profile Selected
		multi-quality	checkbox	Profile Selected
		uploadedfile	File	Binary file

It is expected that experimenters bring their own Content Management Services, however in order to facilitated the integration of the VoD ingest two Drupal Plugins and a presentation layout have been developed allowing Experimenters to use it as an example / reference, deploy the available Drupal or request a portal with the basic configuration.

The VoD ingest is personalised for each experiment, the main personalisation is achieved relaying in the main configuration file of each instance:

Name	Description	Default
media.rootPath	Path where the media is stored	/sample/dir
media.tempPath	Path to use for temporary files	/var/tmp/ingest
ext.thumbnailsUrl	External base URL for accessing thumbnails	http://example.org/dir/
ext.encodedUrl	External base URL for progressive downloaded media	http://example.org/dir/
ext.adaptativeUrl	External URL template for adaptive streaming	rtmp://example.org/dir/{ }.smil
cms.available	If there is a CMS to register content	true
cms.xmlResponse	Answer a XML page when a video is posted	false
cms.url	URL base for the CMS service	http://example.org
cms.login.enabled	CMS requires login	true
cms.login.user	User for logging into the CMS	mediaItemServer
cms.login.pass	Password for logging into the CMS	123456
cms.token.enabled	Require the usage of tokens for uploading videos	true
cms.token.sourceip	IP Address for the CMS	127.0.0.1
mail.enabled	Enable email validation	true
mail.from	Source mail address when sending emails	ingest@example.org
mail.destination	Destination mail address when sending notifications	validator@example.org
mail.smtp.host	Host for the SMTP server	localhost

Name	Description	Default
mail.smtp.port	Host for the SMTP server	25
mail.smtp.useSSL	SMTP requires SSL	false
mail.smtp.useAuth	SMTP requires authentication	false
mail.smtp.user	User for SMTP server	<empty>
mail.smtp.pass	Password for SMTP server	<empty>
media.dir.source	Source directory (where media is uploaded)	/source
media.dir.encoded	Encoded directory (where media is transcoded)	/encoded
media.dir.thumbnails	Thumbnails directory	/thumbnails
media.out.scripts	Output stream for running scripts	/dev/null
config.dir	Directory for webservice instance data (relative inside Tomcat)	data
config.script.deploy	Script executed during the deployment of the VoD ingest service	deploy.sh
cms.path.insert	Path to register content on the CMS	/mediacontentinserter/insert
cms.path.ok	Path of the Ok page	/videouploadok
cms.path.tokenerror	Path of the error page when an invalid token has been used	/videouploadtokenerror
cms.path.noprofile	Path of the error page when no profile has been selected	/noprofile
cms.token.timeout	Timeout for tokens	250

Further personalisation requires changes in other configuration files including details in the transcoding profiles needed for a specific experimenter upon direct request to Atos personnel.

5.2. Pervasive Content Component (PCC)

Pervasive content is produced by mobile users and sensors located in real-world environments. Human sensing (e.g. biomechanics, physiology, etc), human location tracking (indoors and outdoors), location-based content, real-world community interaction models, environment sensing, points of interest all characterise pervasive content.

The PCC offers capabilities that collectively gather data about a user's physical location, QoE, points of interest and interactions. Physical location is used in both the context of tracking a user's location and also as a means by which Augmented Reality (AR)-based content can be selected for delivery and user generated data can be mapped to a spatial location. A real-time orchestration platform is provided supporting the gamification of activities and allowing for adaptive narratives and content that's customised for different experiences. The platform allows professionals and users to co-create content, such as a locative game integrated with the

structure, narrative, and content of the event itself. Users attending the event can consume and produce content in real time using Smart mobile devices. The unfolding events, as experienced by users, can be adapted and orchestrated in real time. Users primarily participate locally at the event but can also contribute via the internet, and synchronized but distributed live events can be joined to provide a common experience. The platform allows access to content and services both before and after the event, thus supporting community building and operation. Metadata generated is published and can be used to annotate audio-visual stream so that other participants can search and retrieve for available content.

5.2.1. AR client

The augmented reality viewer was developed for Android based devices, the main concept was to show Points of Interests derived from the database provided by Infonova. The client uses either the coordinates from a localisation service such as GPS or signal triangulation (Wi-Fi, GSM) or gets a fixed set of coordinates from the developer. Both variants are valid and offer different purposes for various experiments.

The AR client uses a REST API offered by the point-of-interest (POI) service to get the relevant POIs which are cached on the device in a lightweight database. This offers the opportunity to get a list of all points in the region of interest and also works when the device is offline at the time of usage. The list can be re-retrieved and updated at any time.

The client offers a list of filters to determine which points are of interest to the user. Only the selected categories are then shown on the AR view component of the client. This helps to avoid clutter in the user interface. The points of interest are shown as customizable icons, which change position and size depending on the devices orientation with places further away smaller and higher up on the display. Each POI is a clickable item to display more information and to navigate to a detail view which shows all the information stored in the database, including the social network links (for the driving experiment this was the number of Facebook likes and check-ins, both the general value and the value of friends).

A slider on the AR view is another option to filter the points of interests. The slider determines the distance of shown POIs, e.g. show all POIs between 0 and 200 meters from the user's position, or show all POIs between 5 and 21 km, thus a user can decide whether they want to see places in walking distance or places for an excursion by car.

The client is complemented by the aforementioned detail view of the POIs, a list view including sorting options (distance, alphabetically, Facebook likes) and a map view, which uses the new Google Maps API to show markers of POIs in various zoom levels.

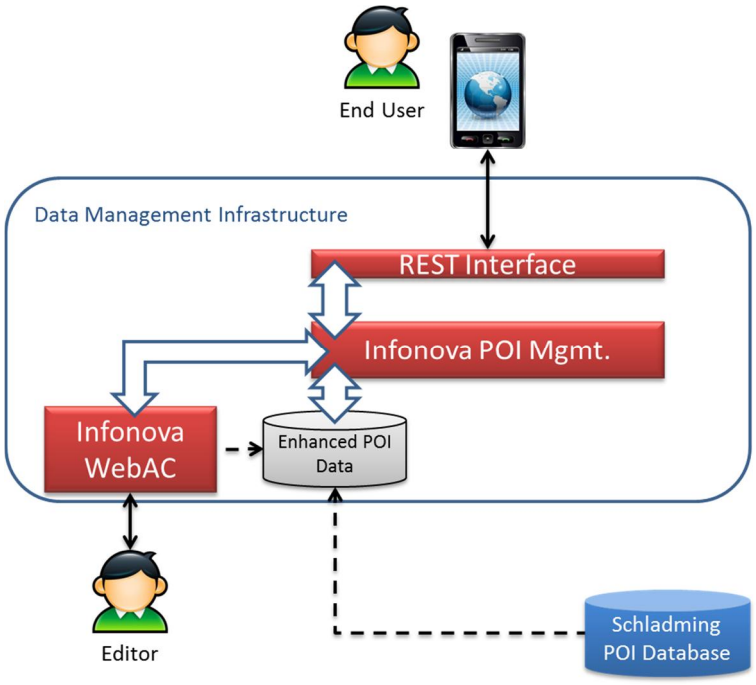
The AR client is available as a library for Android devices in the EXPERIMEDIA software repository. The AR component is a consuming service, producing user generated content could be done with the overlying device and sent with the SI sub-component for example. The data to be shown with the AR client is produced with the POI management service of Infonova by a content author or gathered from the social networks provided by the Social Content Component.

5.2.2. POI service

For the driving experiment in Schladming the tourism board information was integrated and used as a starting point to build the database. The POI database was updated and data from other sources was added to provide a fuller experience. In order to perform an automatic data import from the Schladming tourism database, a POI data import facility using CSV formatted files was implemented.

An Editor is able to create, modify and delete single POIs by using Infonova WebAC. The POI parameters used are: name (mandatory field), street, postcode, town, phoneNumber, website, email, shortDescription, description, latitude, longitude, externalId, facebookPageId and categoryId (can be used several times by using different category values).

The AR client (End User) uses a REST API offered by the Data Management Infrastructure. The client can get via API the relevant POIs which are cached on the device in a lightweight



database but can be updated if required.

Figure 39: Infonova Data Management Infrastructure (overview)

The POI service is offered as a hosted service by Infonova online[29] as a web-based GUI for experimenters (admins) and with a REST web-service for experiment apps.

5.2.3. Creator

Creator is a software platform for creating, setting up and running pervasive games and related location-based or otherwise context-aware services. The platform is quite scalable has been used for large-scale games with thousands of simultaneous players. When using Creator, the process is typically split into four distinct steps: game design, content creation, location adoption and

[29] https://isystem5.infonova.com:8181/experimedia/pois/

orchestration. The system platform is implemented as a web service and the content creation and orchestration application is accessed through a web browser. This approach makes it quite easy to integrate Creative-made games into other services and devices, as web technologies are ubiquitous. Creator supports a module system allowing connecting basically any kind of external service to it, e.g. web service or mobile clients (which either runs local native code on a mobile device, or is accessed via a mobile web browser), stationary or mobile sensors, etc. The Creator supports integration with a wide variety of hardware, software or custom objects, as is described in detail in D2.2.1.

The web interface to Creator is available for experimenters at http://creator.experimedia.eu/.

Figure 40: Creator user interface in authoring mode

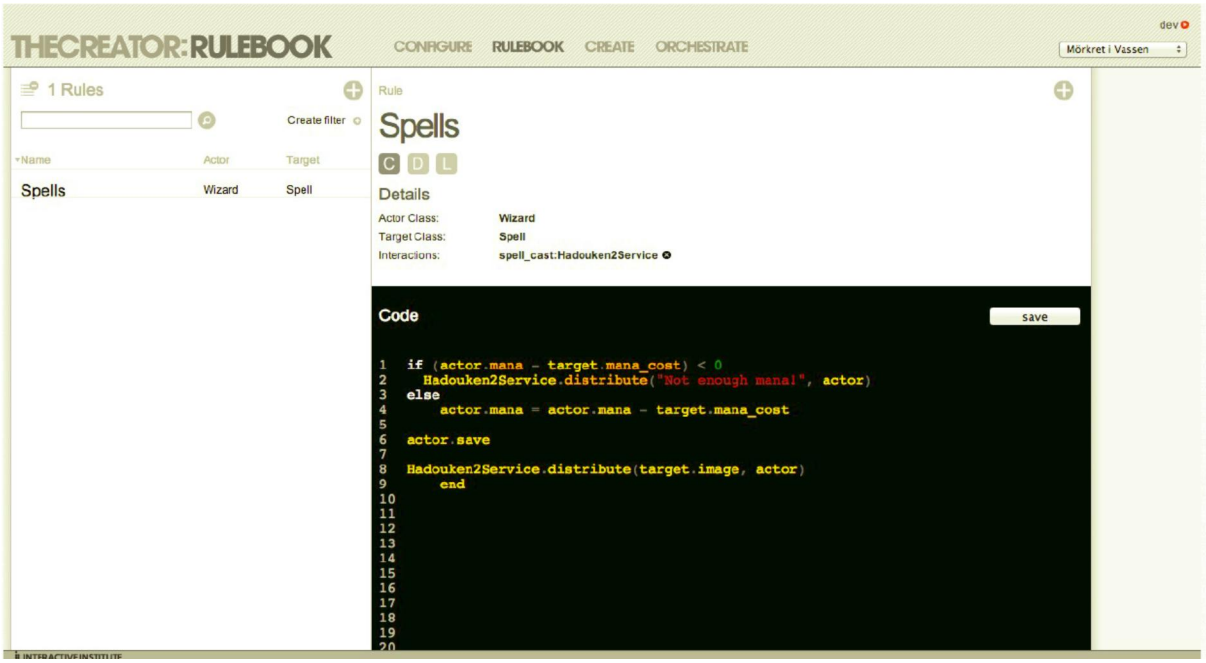


Figure 41: Creator user interface for rule creation and editing

Creator is essentially a rule engine with an editor which supports real-time construction as well as modification and orchestration of pervasive games. It functions as a server for the technical aspects of a pervasive experience and will only be indirectly available to the users through clients. The Creator builds on the REST architecture and clients communicate with it using HTTP calls. Instead of providing exact ways of communication, the experimenter can extend the system with services which can be built to serve specific needs. This might be receiving and sending text messages to mobile phones or control media feeds to clients. The experimenter can create rule scripts inside the Creator environment which determine the relationship between clients, services and users. A rule script might for instance cause a playback of a video on one client when receiving a text message from another. Furthermore the Creator allows experimenters to model users and game objects using an object oriented approach which helps in monitoring and orchestrating games.

The documentation available for the Creator covers rule engine, custom API extensions and examples on the language used for rules. While it is possible for an experimenter to use Creator without assistance, the system requires some basic knowledge to get started with. Interactive will assist experimenters in this process.

5.2.4. Babylon

Babylon is a tool that supports user-oriented evaluations of location-based services. Babylon makes it easy to evaluate the opinions of the users while they utilize the game or service, in contrast to focus groups or interviews which are typically carried out after an experiment is over. Thus it becomes possible to more easily find out what the users think and experience while using the location-based service and how that user experience might change over time.

Babylon is a tool for capturing quality of experience (QoE) and location data from end users. It captures this data by sampling a self-assessment of emotional state (or some other relevant measure for the experiment in question) from end users using mobile devices such as

smartphones or tablets. The users quickly tap on a graphical user interface with several orthogonal axes, such as happy-sad, engaged-bored, active-passive, etc. The type of data captured is primarily QoE-related in the form of (semi)real-time, self-assessments that can be repeated at regular intervals if needed by the experiment. In addition, user id, location data (if available) and timestamp are collected.

Figure 42: Babylon GUI on an iPhone

Babylon has a service and clients. The service is deployed for the experimenter at Interactive (or can be downloaded and deployed by the experimenter) and the client software is deployed on mobile devices (iOS or Android) either as a stand-alone application or integrated into other software. Babylon clients send data directly to the Babylon server. The Babylon server stores this information and also passes the readings on to the ECC. Babylon clients would be used by experiment participants and data can be viewed in the ECC by the experimenter. In addition, the web user interface of Babylon allows experimenters to analyse and reflect over the feedback provided by the clients using a timeline interface.

Sample Babylon source code is available for both Android and iOS platforms. Babylon can either be run as a standalone app, or it could be integrated into another one that is used in the main experiment.

5.2.5. Tracker

Tracker is a system for tracking and analysing movement across great distances or large time spans, with a graphical visualization interface with location data overlaid on a map. For instance, the system has been used to track real-time locations of runners at a marathon event.

Tracker consists of a server component which integrates with hardware or software conforming to a simple API protocol. At this time, the system works with the integrated GPS tracking devices GSAT TR-151and GSAT TR-131 (which continuously send GPS data via a GSM data connection). There is also an iOS application which can run on iPhone devices, which provides the same kind of data to the server. All integrations have been tested in production deployments.

Tracker uses GPS, GSM triangulation and Wi-Fi-positioning (depending on available hardware) in order to continuously provide location updates on a per-user (or per-device) level. Users are authenticated on a per-device level.

Administrators can track location in real-time using a web-based system compatible with all major, modern browsers and smartphones. The system provides monitoring functionality for administrators and supports simultaneous users from multiple physical locations.

The Tracker web interface can be accessed at http://keyhole.sykewarrior.com/.

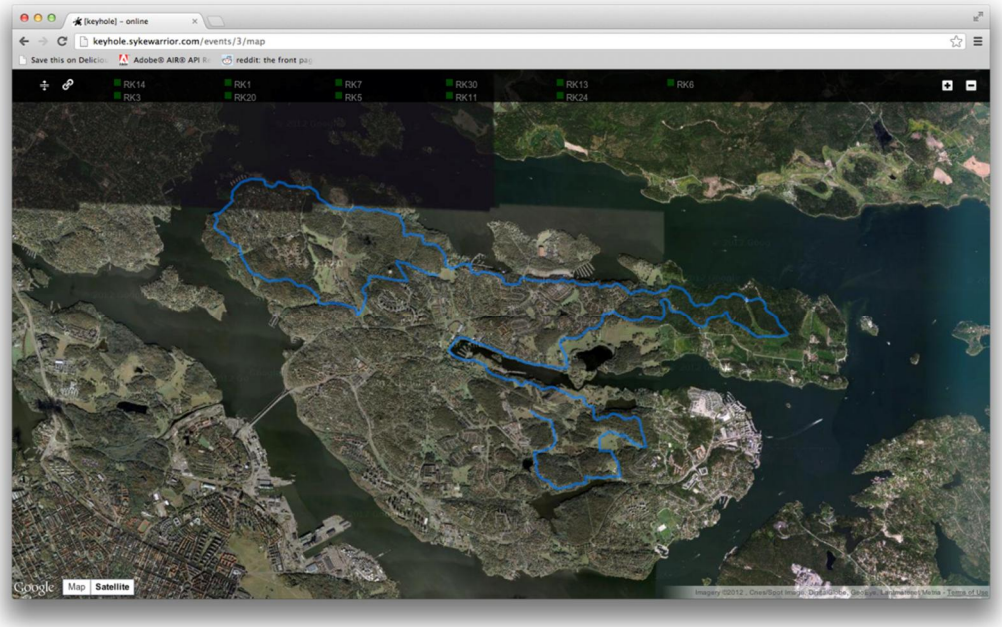


Figure 43: Example Tracker GUI with overlaid tracks from Stockholm Marathon

5.2.6. Ping!

Ping! is a kind of audio-only augmented reality system. It runs on mobile clients such as mobile phones, and combines location and orientation tracking with spatial audio sounds, usually presented to the user via headphones. Sometimes it is not desirable to expect a mobile user to look at a small screen while performing some other task. In particular this can involve tasks where there might be safety risks, such as while walking and crossing roads in a city with heavy traffic, downhill skiing, mountain biking, etc. It might also be useful in other tasks where safety is not a major concern, but where the experimenter wants the user to have their visual attention on something else but the screen of a smartphone.

Ping! is currently running on iOS only. The system has been tested with a tourist guide app, where the user could point their mobile phone in different directions to query for nearby restaurants. The user would hear different 'pings' in the earphones, with restaurant type encoded as sound pitch, and the distance from the current location encoded as volume. By pointing the phone in different directions, the user could instantly get a rough idea of in which directions there might be interesting restaurants while looking at the real surroundings and not the screen, and then look down at the screen for details. The app could then guide the user to the chosen restaurant using audio cues, thus freeing the user from having to look at the screen while walking in the city. The Ping! system has also been tested in a downhill skiing context. Audio-based information about the last ski run such as maximum speed and length of the slope was presented to the skier while going up the lift. Another possibility was to quickly locate the rough

whereabouts of friends by 'scanning' with the phone in different directions, similarly to the restaurant guide.

Figure 44: Example user interface of a POI locator for tourists using Ping!

Ping! is implemented as client software for iOS devices. The system can be adapted to fit the needs of experimenters, but it currently requires assistance from the developers, both in terms of software integration and sound design.

5.3. Social Content Component (SCC)

Social content is characterised by user generated content produced by and consumed within online communities. Photos, videos, comments and opinion is disseminated by individuals to related friends using social networking platforms. The SCC offers the capability to access social content, explore a social graphs, extract general social knowledge (e.g. sentiment and controversy) and media specific QoS/QoE for adaptive, efficient and personalised delivery of experiences. Using an open social API, experiments can navigate a range of social networking platforms. The virtualisation of social network APIs is important as although the predominant network is Facebook, other online platforms are used by target participant communities. A pluggable social analytics dashboard is offered allowing different algorithms to be incorporated with default algorithms provided to detect individual and group preferences based on attitudes, selections and beliefs. The dominant attitudes, beliefs and communications ways for social groups (rather than individuals) can be used to optimise streamed, delivered or even transmitted media content. In addition, the detection of the proximity of consumers to content, similar behaviours and searching for popular UGC can potentially improve media delivery, enhance live streams, or augment information that is aligned with preferences of consumers.

5.3.1. Social Integrator

In order for an application that is part of an experiment to interact with SNs it has to use the interface that they expose. Each social network has a different logic, i.e. it is meant to support different sets of social activity and offer a different set of activities that a developer could integrate in its application. The interfaces and the technologies that are offered to the developers working with them are quite diverse as well. Also, the fact that the social networks are rapidly

and continuously evolving, often results in changes in their APIs and/or technologies they are using with no backwards compatibility, making the maintenance of the applications that build on them a non-trivial task.

The Social Integrator has been developed: a set of Java libraries that provide to the developers of the experiments an easy mechanism to build social-aware applications that access multiple social networks. One of the primary design attributes of the Social Integrator is SN transparency, i.e. providing the same API regardless of the social network that is used in the background so that the development of applications that support multiple social media becomes easier, much faster and easier to maintain since most of the required changes are pushed down to the Social Integrator, without having the need to change any code on the application level.

Basically, the Social Integrator offers a Java API that enables user authentication and sharing content through different SNs in a common way, while hiding all the intricacies that the different API's used by the social media impose. There are two versions of the Social Integrator API: the Social Integrator Android API for implementing applications that run on Android devices, and the Social Integrator Core API for building non-Android Java applications (see Figure 45). For achieving authentication transparency, the Social Integrator Core and Android APIs build on top the functionality of the SocialAuth Core and Android Java Libraries respectively that provide a common authentication mechanism for a number of different social network providers. The Social Integrator extends this functionality by adding various methods that provide support for posting and retrieving various sorts of content such as direct messages, comments, questions, photos, videos, etc.

The offered methods have been designed in a generic manner where possible. For example, posting a photo, which is an action supported by most social networks, is implemented by a common method whereby the developer only needs to specify the targeted social network while the implementation differences remain hidden from the developer. At the point of writing, these methods cover most of the functionality offered by Facebook, Twitter and Instagram. It should be noted that the SocialAuth did not provide authentication support for Instagram and therefore it has been developed from scratch and successfully contributed back to the SocialAuth community.

Under the EXPERIMEDIA framework, the Social Integrator has been used to develop two applications, an Android one and a web-based one, which were inspired and used in the FHW driving experiment: the visitors' mobile application and the expert's web application. These two applications serve as a basis for the development of social-aware applications in the context of other experiments.

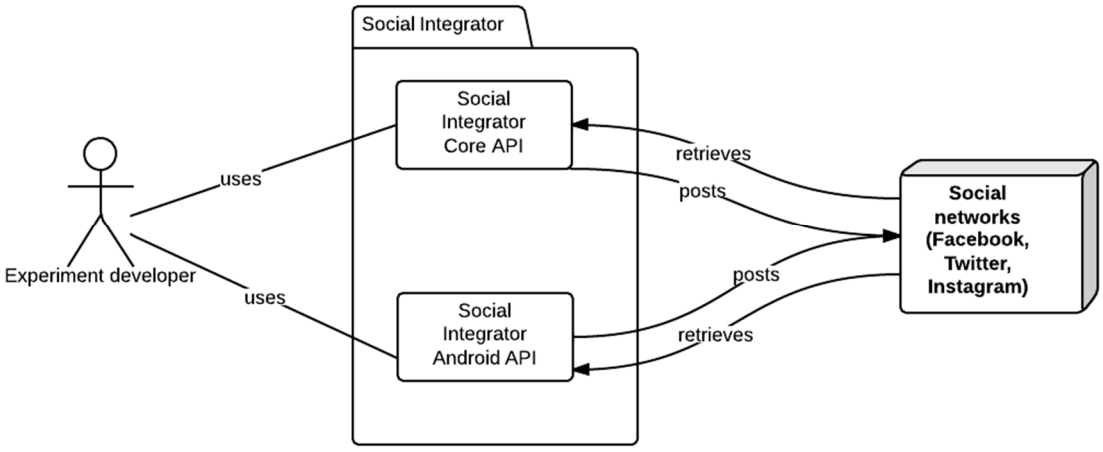


Figure 45: Social Integrator deployment

5.3.2. Social Monitor

The Social Monitor is responsible for collecting data from the SNs that are being exchanged among the participants during the experiments. These data are used to calculate social network related Quality of Experience (QoE) metrics about the overall participants' engagement in the social activity that are of interest to the experimenter. Thus, the metrics that are monitored may vary depending on the nature of the experiment.

The collected SN data involve significant benefits for the experimenters, supplying them with live, valuable, comprehensive and accurate feedback which cannot be collected otherwise and which can significantly help them improve the offered experience of the end users. For example, this monitoring data can help the experimenter understand whether the audience (and what part of the audience such as an age group) liked the new experimental system that is offered, by retrieving data such as the number of attendees, their average age, and the average number of comments/questions per attendee. However, more specific to the experiment metrics can be collected in the context of each experiment. For example, in the case of the FHW driving experiment, the aim was to collect information about the way the audience perceived different parts of the movie that was presented to them. Various photos, each one representing a different part of the movie, were hosted in the SN event. Each photo became a monitoring entity and several attributes were attached to it, such as number of likes, comments, questions and answers per photo, as well as the top comment, question and answer per photo.

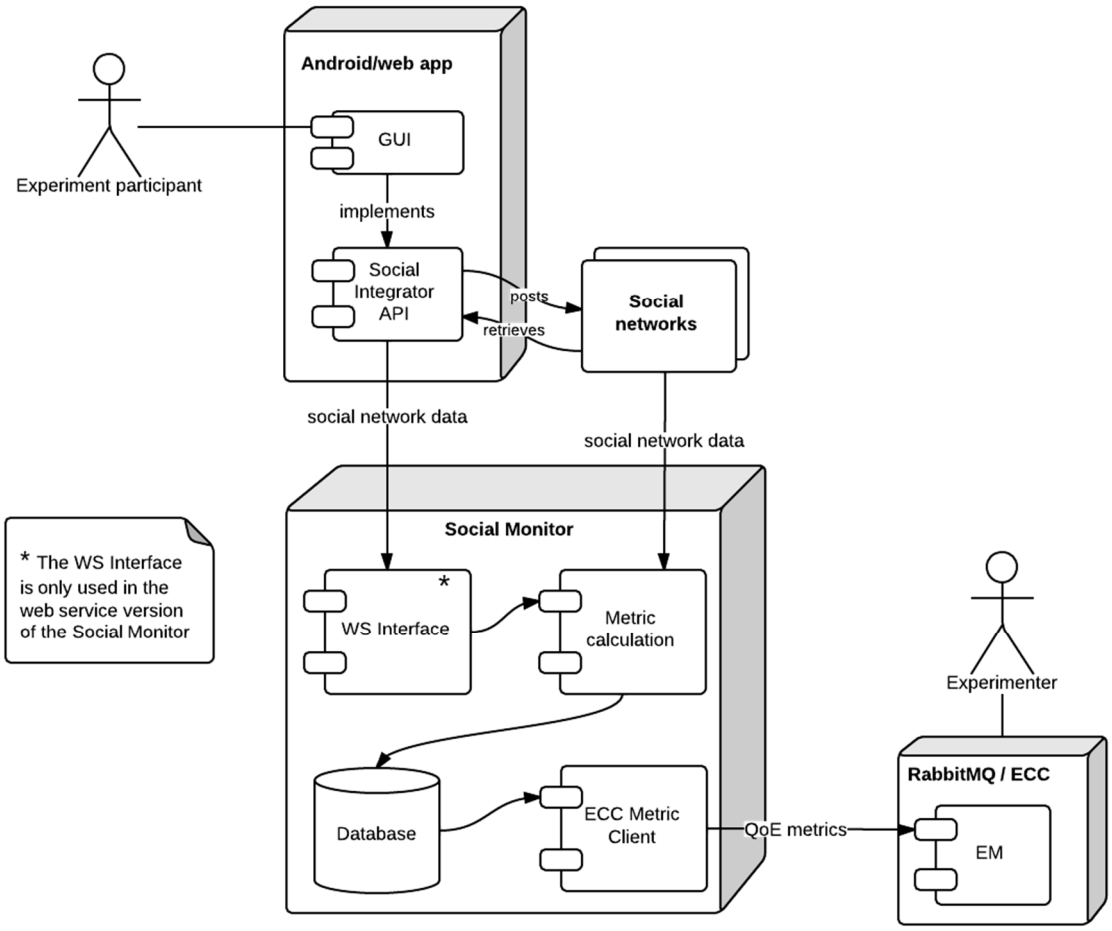


Figure 46: Social Monitor deployment

There are two versions of the Social Monitor:

- *Java standalone program:* This version of the Social Monitor is a stand-alone Java application. It is meant to be used in experiments whereby the participants are exchanging data over specific targeted social activity in the social media. In order to access this activity, credentials of SNs' accounts with sufficient permissions and access are required for authorization purposes. No specific deployment requirements are needed: this version can be deployed on any machine running Java v6 or later that has access to the Internet.
- *Web service version:* This is a web service version of the Social Monitor that is meant to be used in experiments whereby the social data of interest are not posted within a specific social activity but are posted by the end users on their own personal social accounts. To this end, contrary to the Java standalone version as discussed above, this service version is designed to receive data directly from the end-users applications (and not directly by the social networks). On the end-users side, the applications that are being used are using specific client code which is offered as part of the Social Integrator API in order to communicate the information of interest to the Social Monitor service.

In the background, the Social Monitor (both the standalone Java application & web service version) acts as a client to the ECC, i.e. the calculated metrics are being communicated via RabbitMQ to the ECC EM, as demonstrated in Figure 46.

5.3.3. Social Analytics Dashboard

The Social Analytics Dashboard, or SAD, is a web service for collecting data from social networks, analysing it and presenting both the raw data and the analysis to other services via RSS or a REST API or directly via a web interface. An overview is provided in Figure 47 below.

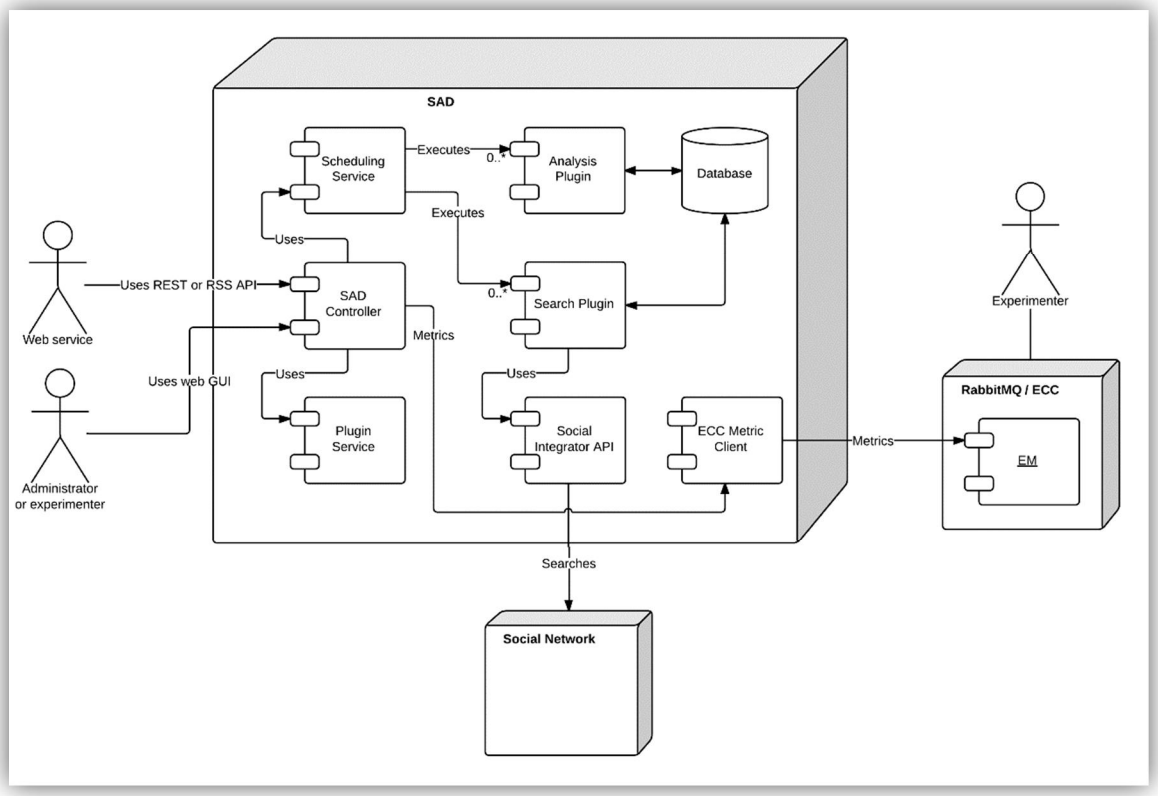


Figure 47: Social Analytics Dashboard (SAD) deployment

The SAD employs a plugin architecture and provides plugins for searching Facebook, searching Twitter and analysing the search results in a variety of ways including sentiment, hot topics, geographic location and influence. The search plugins make use of the Social Integrator API (see above). The plugin architecture facilitates the easy addition of new features such as new social network searches or additional analysis tools. The main SAD service acts as a job scheduler, executing the plugins according to a configurable schedule.

Data collected by the search plugins or generated by the analysis plugins is added to a local database. In the current release, this is a PostgreSQL database but the next release will use the NoSQL database MongoDB for better performance and flexibility.

An instance of the SAD is deployed for a particular experiment and administrative control over the service can be given to the experimenter. The SAD is deployed in a standard web service container such as Tomcat.

Data generated by the SAD can be consumed by other services via a REST API or a customisable RSS feed. The SAD also communicates with the ECC: reporting metrics about the service itself such as the number of plugin executions. Work is underway to enable arbitrary metrics to be reported from the plugins to the ECC via the SAD service. This will enable applications such as the Social Monitor (see above) to be integrated as a plugin in the SAD.

The SAD provides a web interface to the experimenter or administrator to configure the plugin execution schedule and parameters of the plugins. This interface is shown in Figure 48. The administration interface also displays the status of scheduled and previously executed plugins (see Figure 49).

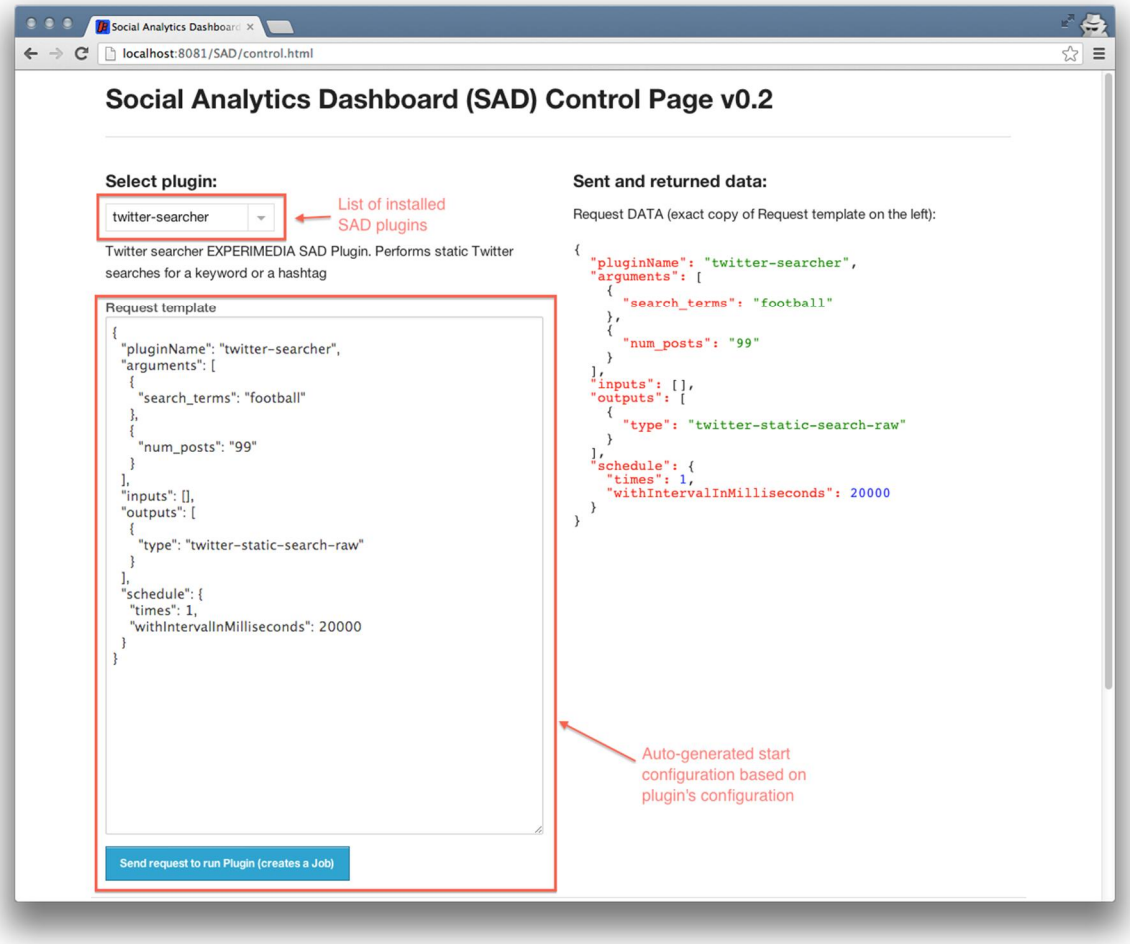


Figure 48: The Social Analytics Dashboard control page

Job ID	Plugin name and data/details links	Execs	Status	Created	Last run
5	twitter-searcher [raw data] [visualized data] [details]	1	finished	2013-07-18 16:09:57	2013-07-18 16:10:00
4	twitter-searcher [raw data] [visualized data] [details]	1	finished	2013-07-18 16:09:24	2013-07-18 16:09:29
3	basic-sns-stats [raw data] [visualized data] [details]	1	finished	2013-07-18 12:29:14	2013-07-18 12:29:16
2	facebook-collector [raw data] [visualized data] [details]	1	finished	2013-07-18 12:28:52	2013-07-18 12:28:57
1	twitter-searcher [raw data] [visualized data] [details]	1	finished	2013-07-18 12:28:39	2013-07-18 12:28:43

Figure 49: List of SAD jobs on the administration page

5.4. 3D Content Component (3DCC)

3DCC is the main component for 3D information acquisition, enhancement and manipulation. It is comprised of 8 basic sub components that interoperate to provide useful information that can be used from the experimenter. The 3DCC functionalities can be divided into three major categories that provide different levels of interaction with the hardware (i.e., the Kinects). An overview of these subcomponents is provided right below:

Low-Level Functionalities

Depth Acquisition: This provides the experimenter with the raw depth information of a scene. It is the middle layer between the Kinect device and the experimenter. Simple and easy to use functions provide the experimenter with per pixel information about the depth of the scene.

Skeleton Acquisition: In the case where humans are involved in a scene, 3DCC can provide robust skeleton extraction for up to 15 joints. Moreover, skeleton tracking can be performed for human motion analysis.

RGB Acquisition: Images coming from the Kinect, along with their registration to depth pixel transformation, can be provided to the experimenter. This is important, since texturing of a post produced 3D model can be made possible through this information.

Mid-Level Functionalities

Depth Enhancement: Since raw depth data is noisy we provide several filtering algorithm to smooth and de-noise the raw information so that more accurate depth measurement can be made possible.

Skeleton Enhancement: Jerky (noisy) skeleton joints are detected and tracked and therefore corrected through a sophisticated tailored filtering framework to provide a more realistic skeleton.

Biomechanical Measurements: The 3DCC can provide several biomechanical measurements that are inferred from both depth and skeleton information. The most important being: angles

between bones, human joints and calibrated objects' velocities, human body parts surface areas and calibrated objects' surface area. These measurements can be used by the experimenters in a multitude of ways to infer high level information that suits their needs.

High-Level Functionalities

Avatar Creation: The 3DCC provides an avatar authoring tool so that experimenters can create their own avatars that can be easily integrated into a virtual world. Other than a simple database of several features that can create artificial avatars, the 3DCC avatar creation tool can provide custom authoring capabilities that provide functionalities such as avatar personalization (so that the user's facial image can appear on the avatar).

Avatar Motion: The 3DCC can also interactively move the avatar using a Kinect alone. This functionality can be used from the experimenter in a multitude of ways to animate his avatar and interact into a virtual world.

3DCC is partitioned in two libraries and one application. One library, written in C#, is where all functionalities concerning low and midlevel functions are implemented such as the Acquisition modules, the Enhancement modules, the Skeleton Motion analysis modules and the Biomechanical analysis modules. A second library, written in C++, is where the high level (Avatar motion modules) functions are implemented. Finally a web application that provides means to create avatars from the scratch that can be used along with 3DCC.

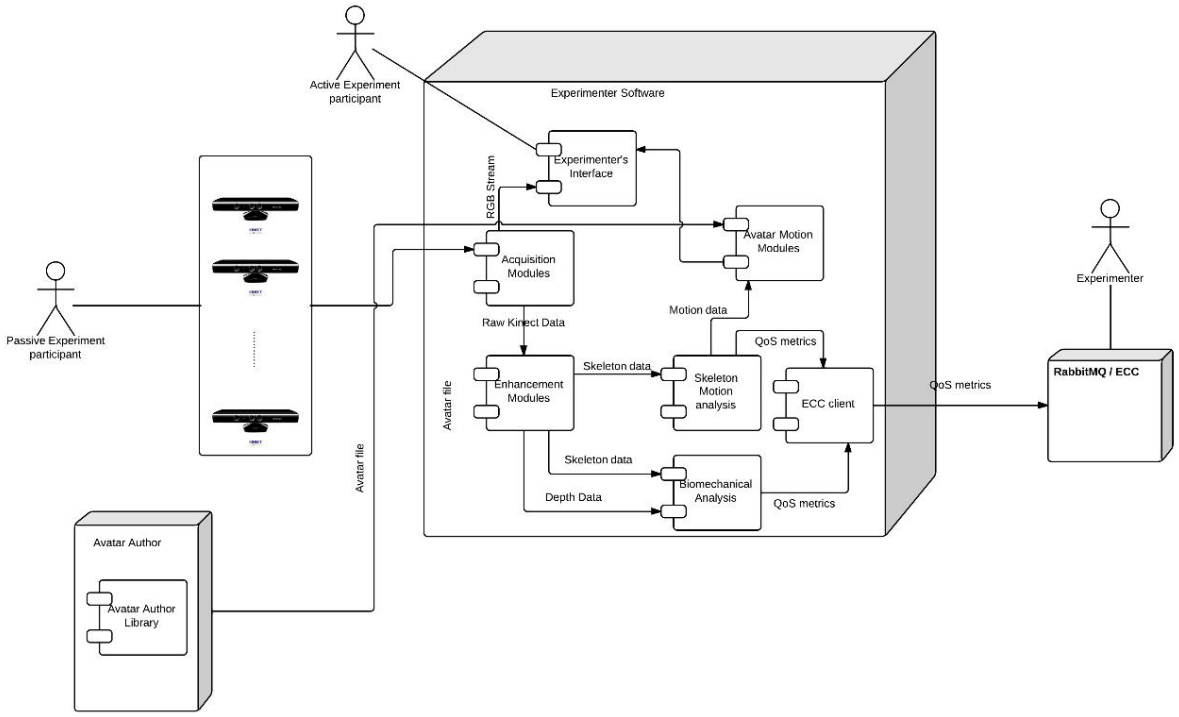


Figure 50: 3DCC deployment

3DCC can be used whenever an experiment needs to track and/or gather real-time 3D information. To do so, a computer connected to Kinects needs to run the experimenter's software where all functionalities from 3DCC are imported from the two previously mentioned libraries. Moreover, it can be used for smart real-time rendering of human-like motion avatars.

The library can be deployed at any software the experimenter is developing and therefore all before mentioned tools integrate with the experimenter's software. 3DCC can deliver depth, RGB and skeleton data as files (in this case they are stored from 3DCC) and as a stream that can be captured from the experimenter's software. For the high level functionality the avatar is stored in a file (the experimenter's software can then ingest it through appropriate functions provided) and the skeleton motion data are streamed to the experimenter's software. Finally, 3DCC can also deliver QoS measurements that can be fed to the ECC through an ECC client. These QoS measurements are depth quality, skeleton quality, biomechanical measurements quality and frame rates of depth and skeleton acquisition from the Kinects in frames per second (fps).

3DCC information can be streamed to the experiment participant through an appropriated GUI and the experimenter can use the ECC to monitor the QoS measurements along the lifespan of the experiment.

6. Deployment Constraints

Experimenters have to design, develop and deploy an experimental system that consists of both baseline components from the EXPERIMEDIA Facility and technologies the experimenter is developing.

Figure 51 describes the relationship between concepts related to deployment and shows how partners develop components but also offer hosting sites. It also shows how 3rd party hosters (e.g. Amazon) and service providers (e.g. Facebook) fit into the landscape. Each of the concepts is described in more detail within Table 2.

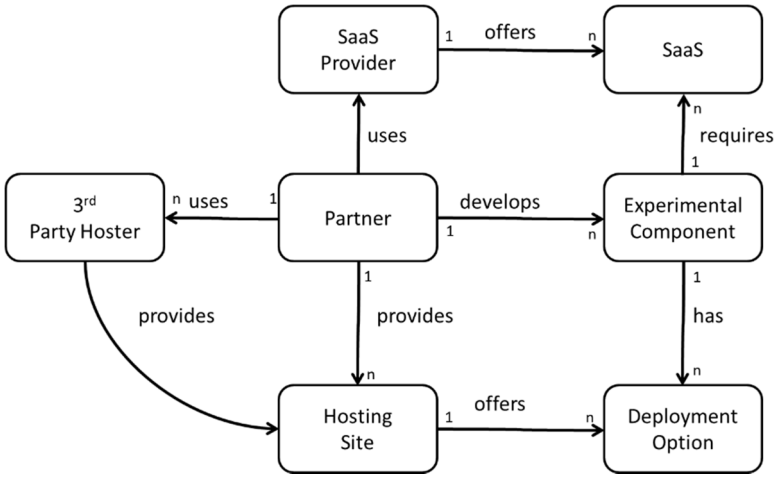


Figure 51: Conceptual Model of Deployment Options for Components

Table 2: Deployment concept descriptions

Concept	Description	Example
Experimental Component	An experiment component from A2 or A4 that needs to run as part of an experiment. Could include both software and sensors. Experimental Components require 1 or more hosting options.	SAD, POI Service, ECC, AR Client
Hosting Option	A container for experimental components to run in ranging from a real physical space to a virtualised container	Hosted Service, Service Container, Virtualisation, Physical machine, Physical Rack, Physical Location
Hosting Site	A physical location where experimental components can be deployed	Partner site, 3rd Party service provider
Partner	An EXPERIMEDIA project partner	ATOS, IT Innovation
3rd Party Hoster	A hosting provider	Amazon, Rightscale
SaaS Provider	A software as a service provider	Facebook, Twitter

Concept	Description	Example
SaaS	The service offered by a a SaaS Provider	Facebook, Twitter

Each component in the system has deployment constraints that limit how and where a component can be deployed. For example, a cloud deployable service could be packaged for execution on Amazon or CAR's private cloud, where a mobile client library may be constraint to a mobile device with a specific operating system. There are many hosting options ranging from choices about the physical location of the server machine right up to the manner of deployment of the application itself. An interesting deployment case is sensors and cameras which often require human experts to deploy and configure them. This is the case for the 3DCC where multiple Kinect cameras are used.

Table 3: Hosting options

Hosting Option	Description	Example
Hosted Service	Software as a Service hosted by an partner or 3rd party that is configured and maintained by an EXPERIMEDIA partner	POI Service hosted at Infonova
Service Container	A environment to host services offering a set of high level common management functions (e.g. security, monitoring, etc)	R6, JBoss, Tomcat
Virtualisation	A environment to host VMimages on virtual machines	VMWare
Physical machine	A physical machine running a dedicated operating system	Machine running Linux OS
Physical Rack	A dedicated place to install physical machines	Machine room at CAR
Physical Location	A dedicated place to install other hardware (e.g. cameras and sensors)	Tholos Theatre at FHW, Taekwondo room at CAR

Table 4 shows the deployment options for the baseline component services. During the 1st year many of these components were deployed as hosted services for experimenters. The benefit of this approach is that experimenters do not have to learn about how to operate the components and can focus on the objectives of their experiment. The ECC and SAD were initially provided as software distributions that could be flexibly deployed by experimenters in different containers rather than offered as hosted services. For some experimenters the process of installing the ECC was challenging and time consuming, reducing the efficiency of the experiment and usability of the ECC software. In V2 EXPERIMEDIA will be hosting all services for experimenters where possible. This decision moves EXPERIMEDIA towards a more centralised view of facility services and operating models that consider not only distributing software but also maintaining services that participate in experiments. An instance of the ECC is being offered to

experimenters as a hosted service by Infonova. Deployment is currently (July 2013) in the final stages.

Table 4: Baseline component services - deployment options

Hosting Option	ECC (all)	AVCC (all)	Creator	POI Service	Babylon Service	Social Monitor	Social Analytics Dashboard	3DCC (all)
Hosted Service	X	X	X	X	X	X	X	
Service Container	X					X	X	
Virtualisation	X					X	X	
Physical machine	X					X	X	
Physical Rack								X
Physical Location								X

6.1. Security and Privacy

As stated in the architectural considerations (Section 2.3): "experiments must be legally compliant in accordance with data protection legislation and security and privacy therefore must be considered a critical attribute of component and systemic capabilities. Security and privacy must be by design rather than an add-on."

Essentially we are addressing information security issues. A useful definition of "information security" is provided in the United States legal code[30]:

> The term "information security" means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide—
>
> - integrity, which means guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity;
> - confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information; and
> - availability, which means ensuring timely and reliable access to and use of information.

Clearly, all these aspects of information security are applicable to EXPERIMEDIA and in particular the "confidentiality" is of primary import.

In architectural terms, we must ensure that EXPERIMEDIA baseline software can be operated to provide integrity, confidentiality and availability. This encompasses both the design and testing of the software and the manner in which it is deployed. The experiments in the first open

[30] http://www.law.cornell.edu/uscode/text/44/3542

call avoided the issue of storing personally identifiable data (and indeed, such data should not be stored unless necessary) but we need to be able to support experiments where such sensitive data must be stored.

6.1.1. Service Hosting

As described above, there are various deployment options for the different baseline components. The ECC dashboard is a good example as it could be deployed in many ways:

- it could be deployed for the experimenter by a core partner (such as Infonova) or by the experimenter themselves;
- it could be deployed on hardware operated by the core partner/experimenter or on leased hardware at another site operated by another company (e.g. a cloud provider).

According to discussions with the Ethics Advisory Board, it is the responsibility of the owner of the service to protect the data in the service. So for example, if Infonova deploy the service then it is their responsibility. It is their responsibility regardless of where the service is deployed, so to continue the example, if Infonova deploy the ECC dashboard on a machine at a hosting provider then it is still Infonova's responsibility. This implies that hosting providers (if used) should be chosen carefully: the data-centre where the host is located should (at least) implement ISO 27001:2005[31] and should also be based in Europe for the best legal protection.

ISO 27001 defines a model for establishing, implementing, operating, monitoring, reviewing, maintaining and improving an Information Security Management System. It adopts the Plan – Do – Check – Act (PDCA) model of continuous improvement. The standard covers physical security as well as other aspects such as network security. Although a data centre specifying that it is ISO 27001 certified is a good thing, it is important to understand which controls of the standard have been implemented and which have not.

6.1.2. Risk Based Approach

The ISO 27005:2011[32] standard defines a risk-based approach for managing information system security aligned with the continuous Plan – Do – Check – Act methodology of ISO 27001. Threats must be identified, analysed and evaluated and a risk treatment chosen. Possible treatments are:

- risk modification: apply a control to reduce the risk;
- risk retention: accept the risk with no further action;
- risk avoidance: completely change the plan so that the risk cannot materialise;
- risk sharing: sub-contract another party to deal with the risk or insure against it.

Threats can come from three sources: they can be deliberate, accidental or environmental (natural). For instance, a hacker breaking into a system and stealing data is deliberate, an

[31] ISO/IEC 27001:2005, Information technology - Security techniques - Information security management systems – Requirements: http://www.iso.org/iso/catalogue_detail?csnumber=42103
[32] ISO/IEC 27005:2011, Information technology - Security techniques - Information security risk management: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=56742

employee mistakenly copying sensitive data to a public folder is accidental and environmental threats are generally larger-scale disturbances such as floods and earthquakes.

Assets must be identified and their value assessed. By considering each asset, the likelihood of occurrence of a threat and the ease of exploitation of the threat it is possible to rank the risks and therefore understand which ones need most attention. When considering the threats to an asset, particular attention should be paid to human threat sources and the possible motivation of different types of people.

For example, what are the assets and threats to the configuration registry described in Section 4.3? The assets include the RabbitMQ hostname and port for a project's ECC dashboard. A threat is someone who is not supposed to know the data reading it. The value of that asset in part depends on the risks to the RabbitMQ service so we must look at that in turn. Given knowledge of the RabbitMQ service, a malicious user could execute a denial of service attack by flooding the server with requests, but what would be their motivation? The likelihood seems low. This suggests that the value of the asset in the configuration registry is also low and controls on the identified threat may not be necessary.

By following this process for other baseline systems and assets we can make considered judgements about what controls to apply where.

7. Conclusion

This document has described the Second Blueprint Architecture for social and networked media testbeds. The architecture builds on the First Blueprint Architecture by extending the component models for experiment and FMI content lifecycle management. The architecture describes two composition patterns for use of baseline components within experimentation: Instrumentation and Observation, Mixed Information Flows. Each of the baseline components are described in terms of capabilities, architecture and deployment model.

The architecture addresses the need to improve Experiment Lifecycle Management through extensions to the ECC, as the central element responsible for conducting experiments at the EXPERIMEDIA facility. V2 ECC architecture extends the state-of-the-art in experiment monitoring frameworks by providing a mechanism by which experimenters can investigate how system and user activities have led to changes in system performance or human experience (as observed by the ECC metric monitoring system). To this end, the experimental support provided by the ECC will include an enhanced metric model meta-data to improve visualisation, time-line based navigation of metric data, metric data aggregation and descriptive statistical analysis and a provenance based view on system and user activities.

Each of FMI Lifecycle components are described covering audio-visual (AVCC), pervasive (PCC), social (SCC) and 3D (3DCC) content. Each of these components will be extended to support the new instrumentation and observation model defined by the ECC.