Future Service Oriented Networks

www.fusion-project.eu

Deliverable D5.1

Use Case Requirements, and Initial Design

Public report, Version 1.1, 22 January 2014

Authors

UCL David Griffin, Miguel Rio

ALUB Frederik Vandeputte, Luc Vermoesen

TPSA Dariusz Bursztynowski

SPINOR Michael Franke, Folker Schamel

IMINDS Pieter Simoens

Reviewers Dariusz Bursztynowski, Folker Schamel, Frederik Vandeputte

Abstract This deliverable first describes the initial requirements with respect to the use cases that we will use for evaluating FUSION, followed by a description and initial modelling of these use cases, as well as the testbeds we will use for either the test scenarios or the demonstrators. We also provide an initial overview of specific test scenarios we plan to build for evaluating subcomponents of the FUSION architecture and which can subsequently serve as input towards the demonstrator. Finally, the design of an initial demonstrator setup is described, which focuses on building an initial environment and bringing together initial software components representing various use cases.

Keywords FUSION, use cases, requirements, demonstrator, testbeds, test scenarios

Project funded by the European Union under the
Information and Communication Technologies FP7 Cooperation Programme
Grant Agreement number 318205

Revision history

Date	Editor	Status	Version	Changes
16/09/2013	Frederik Vandeputte	Initial Version	0.1	Initial ToC
28/10/2013	Frederik Vandeputte	Initial Version	0.3	Revised ToC with partial input
02/12/2013	Frederik Vandeputte	Initial Draft	0.4	First stable draft
18/12/2013	Frederik Vandeputte	Stable Draft	0.5	Ready for review
20/12/2013	Frederik Vandeputte	Stable Version	0.6	Review comments included
23/12/2013	Frederik Vandeputte	Final Version	1.0	Final version
22/01/2014	Frederik Vandeputte	Final Version	1.1	Minor improvements

GLOSSARY OF ACRONYMS

4G	Fourth generation of mobile phone mobile communication technology standards
API	Application Program Interface
BGP	Border Gateway Protocol
BW	Bandwidth
C++	Object Oriented Programming Language
CPU	Central Processing Unit
Ctl	Control
DC	Data Center
DNS	Domain Name System
DRAM	Dynamic Random Access Memory
EC2	Amazon Elastic Compute Cloud
EPG	Electronic Program Guide
EZ	FUSION Execution Zone
FUSION	Future Service Oriented Networks
GB	Gigabytes
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTTP	HyperText Transport Protocol
IP	Internet Protocol
ISP	Internet Service Provider
MOSCOW	MUST, SHOULD, COULD, WON'T
MPLS	Multi Protocol Label Switching
NFS	Network File System
PaaS	Platform as a Service
PoC	Proof of Concept
QoS	Quality of Service
REST	Representational state transfer
RFB	Remote Frame Buffer
RGBA32	Red Green Blue Alpha 32 bit color space representation
RPC	Remote Procedure Call
Rsp	Response
RTP	Real-time Transport Protocol
RTSP	Real-time Streaming Protocol
RTT	Round-Trip Time
SMART	Specific, measurable, attainable, relevant and time-bound
SSD	Solid State Drive
SSH	Secure Shell

Tb/s	Terrabit per Second
TOSCA	OASIS Topology and Orchestration Specification for Cloud Applications
UI	User Interface
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNC	Virtual Network Computing
VoD	Video on Demand
VPN	Virtual Private Network
WAN	Wide Area Network
Wifi	WLAN products based on the IEEE 802.11 standards
WLAN	Wireless local area network
XML	Extensible Markup Language
YUV4MPEG2	Uncompressed frames of YCbCr video formatted as YCbCr 4:2:0

EXECUTIVE SUMMARY

This document is a public deliverable of the “Future Service-Oriented Networks” (FUSION) FP7 project. This deliverable focuses on the use case requirements, use case modelling as well as initial test cases and initial demonstrator design, which will be developed within the scope of the project for evaluating the FUSION architecture.

The scope of this deliverable is twofold. First, it captures the functional and non-functional requirements of the range of use cases we envision to be deployed with the FUSION architecture. Second, it describes our initial plans for evaluating several key aspects of the FUSION architecture, as well as our plans and methodology for building an initial demonstrator.

In FUSION, we target demanding interactive services that need to be deployed, managed and accessed in a flexible manner. Due to the low-latency low-jitter interaction between the client application and service either for making a service request or for interacting with the service, the location and distribution of these services across a distributed set of heterogeneous data centres is of key importance.

This deliverable first describes the key functional and non-functional requirements of these use cases and services with respect to the FUSION architecture. Then we discuss a selected number of these use cases in more detail, followed by an overview of what key FUSION aspects each of these use case is high-lighting.

The second part of this deliverable then discusses the test environments and test scenarios that will be used in the second year of the project for evaluating the functionality of FUSION with respect to the selected use cases. We first describe the initial set of testbeds on which the FUSION demonstrators or specific test scenarios will be deployed. Then we describe an initial list of specific test cases we want to evaluate during the project.

The final part of the deliverable describes an initial demonstrator design. For building the demonstrators, we have decided to adopt an agile design and development approach. This enables us to start evaluating several key functions of the FUSION architecture early on and provide continuous feedback regarding bottlenecks during the design and development of the architecture and demonstrators. It also allows us to start aligning interfaces and start integrating key functions as soon as possible. As the FUSION architecture comprises both an orchestration, networking, execution and service layer, we believe this approach will result in an efficient and scalable design.

TABLE OF CONTENTS

1. SCOPE OF THIS DELIVERABLE

This deliverable focuses on describing the key use case requirements, modelling a selected amount of use cases as well as describing an initial set of test cases as well as an initial demonstrator design, which we will develop within the scope of the project for evaluating the FUSION architecture.

We first discuss the key identified requirements related to the type of services and use cases we want to support with the FUSION architecture. These requirements relate to the demanding and interactive nature of these use cases as well as the inherent dynamicity and flexibility regarding deployment and service selection.

The use cases have been chosen based on the list of functional and non-functional requirements each of these use cases exhibit and the relevance to be used as primary candidates for deployment and exploitation in case of a real FUSION deployment.

In the second part of this deliverable, we then focus on the initial test environments we have selected for evaluating several aspects of the FUSION architecture and for building the demonstrators. We also already provided an initial design and methodology for building a FUSION demonstrator, using an agile development process, starting with a very simple demonstrator, which will then be further expanded for building the final integrated demonstrator.

In summary, this deliverable captures the key requirements as well as key use cases we identified in the first year of the project, as well as an initial design and methodology for evaluating the architecture and implementation of all key FUSION layers, each of which will be worked out in more detail in the second year of the project.

2. USE CASE REQUIREMENTS

In this section, we provide an overview of the key requirements of a selected number of use cases we want to evaluate with the FUSION architecture, which has been described in deliverable D2.1 and which has been worked out in more detail in deliverables D3.1 and D4.1. We will exploit these use cases for evaluating and high-lighting the core values of the FUSION architecture with respect to these key features.

2.1 Service interaction and longevity

In FUSION, we are targeting different types of demanding services, ranging from short-lived request-response type of services for which a response is expected with very low delay, towards highly interactive streaming services, which typically consist of long-lived sessions for which the initial setup and response time is not critical, but for which the operational response time needs to be very fast.

2.1.1 Request-response services

In case of request-response services, the client application making a service request is expecting a fast response. As a result, these sessions are typically short-lived, where subsequent service requests result in new FUSION sessions for which FUSION could select different service instances based on availability or networking characteristics. An example service we will work out in detail is an image tagging service, where the client application provides an image, and expects an augmented image or metadata information as fast as possible. In this case, FUSION orchestration needs to make sure that there are enough service instances available up-front across an orchestration domain to handle all incoming requests. The FUSION routing domain on the other hand needs to quickly and efficiently route all requests to appropriate instances based on various metrics.

2.1.2 Streaming services

In case of regular streaming services, a client application or FUSION service connects to such streaming service and the selected service instance will stream the content (e.g., the video content) to the connected application for the duration of the session. In this case, it is not always so important to select and connect to an instance as fast as possible compared to selecting an optimal instance. Due to the lack of interactivity, the round-trip time between the streaming service and the requesting client application is also less important. Instead, the overall network bandwidth is likely to be more important. Consequently, the FUSION routing should mainly take this aspect into account. The FUSION orchestration on the other hand needs to make sure that enough streaming service instances are deployed in the appropriate locations to reduce the streaming bandwidth in the network. For the FUSION use cases and demonstrator, these streaming services will mainly be used as helper services in more complex service graphs for providing input streams towards for example an interactive Electronic Programming Guide (EPG) service.

2.1.3 Interactive streaming services

Interactive streaming services are services with typically long-lived sessions during which the client application can interact in real-time with such a service. The service itself is streaming dynamic content (e.g., a video stream, game state updates, etc.) towards the client application with very low latency. For these services, the round-trip time or lagging time during the service session between the client application and the interactive service is very important. This is illustrated in Figure 1.

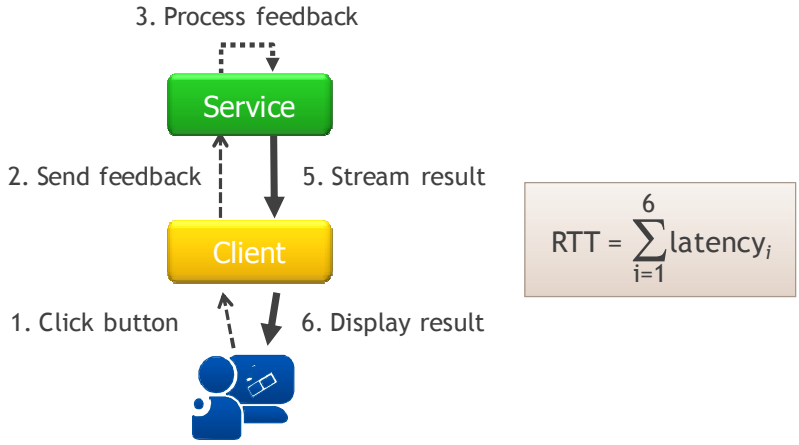

Figure 1: Contributing factors of total feedback roundtrip time

The total round-trip time in between the client sending some input and the client receiving the resulting reaction should be as small as possible for highly interactive services to reduce lagging time. On the other hand, when initiating a new session (by issuing a service request), it may be acceptable that finding an optimal service instance may take some time. For some services, it may be acceptable to only deploy a new instance on demand in order to provide the best quality of service and to avoid costly inaccurate or suboptimal pre-deployments. As a result, these services stress mainly the routing layer of FUSION for finding an instance with minimal roundtrip time and possibly reduced networking bandwidth requirements, as well as the orchestration layer and execution layer for efficiently deploying new instances on demand.

2.1.4 Requirements

Below an overview of the service requirements with respect to FUSION.

Req no.	Description	Level[1]
UUI-1	Support for short-lived request-response services.	M
UUI-2	Support for long-lived streaming services.	M
UUI-3	Support for long-lived interactive streaming services.	M
UUI-4	Provide low-latency service responses	M
UUI-5	Provide low-latency low-jitter feedback loops and interactivity	M

2.2 Multi-user aspects

In FUSION, we target services that can be serving both individual clients as well as multiple clients in parallel. The first set of services provide personalized sessions towards a single client, whereas in the second set of use cases, multiple clients can make use of a shared service, or can even collaborate and interact with each other using the same session. An example of a single-user personalized service is an EPG service, where each user has its own personalized session, which is completely isolated from the other users. An example of a shared service is a shared dashboard or a surveillance

[1] The abbreviations indicate the level of importance and follow the MoSCoW method. http://en.wikipedia.org/wiki/MoSCoW_Method

mosaic, where multiple users can have a common view. In this case, the individual users not always have to be aware that other users are involved in the same session. Finally, examples of collaborative services include a multi-player game, or a video conference service.

2.2.1 Personalized services

In case of personalized services, each service session is completely independent from the other, at least, at the level of what FUSION needs to manage and take into account. As a result, FUSION routing does not have to take into account the location of each session, and FUSION is completely free of choosing appropriate instances for handling a new service request. How stateful application services are handled is discussed in Section 2.3. FUSION service routers are responsible for finding an appropriate instance for each new service request. When no appropriate instance is available, the FUSION orchestration domain needs to be triggered to deploy new instances on demand. The orchestration domain has two main roles, namely (i) to properly anticipate service requests coming from particular regions of the network and appropriately pre-deploy instances based on predicted demand, and (ii) to efficiently and quickly deploy new instances on demand when needed.

2.2.2 Shared services

In case of shared services, two or more clients or service instances need to be able to connect to the same service session and therefore need to be routed and connected to the same service instance session. The simplest example is illustrated in Figure 2.

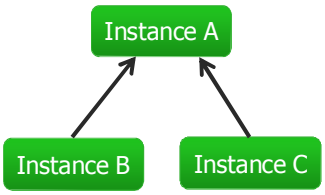

Figure 2: Trivial example of service sharing

In this scenario, service instance B and C (which can also be application clients) both need to be able to specify somehow that they are both interested in sharing the same service instance A. In case the services A, B and C are all part of a static service graph, it is the responsibility of FUSION orchestration to properly connect service instances B and C to A. In case B and C are in fact client applications or services that are dynamically connecting to A, B and C need to be able to identify a specific instance of A. This means that they both need to be provided first with an identifier or locator for A. In the current FUSION architecture, we assume this will occur outside of FUSION orchestration and routing. In the future however, we may study possible ways to enhance this, for example by means of introducing FUSION service instance IDs apart from service locators.

In this model, B and C cannot directly use FUSION routing for finding the best instance of A. Nevertheless, FUSION orchestration and routing still can and will play a key role in:

- Finding the optimal instance A, possibly given the location and properties of B and C are known before A needs to be selected (for example by D);
- Finding optimal locations for deploying A, B and/or C on demand.

2.2.3 Collaborative services

Collaborative services face the same problems as shared resources, with the extra constraint that both services B and C can have a direct impact on the behaviour of A, and thus on what B and C will perceive. In case of a multi-game server for example, this means that A needs to be selected in such a way that B and C will experience a similar roundtrip time, resulting in even tighter requirements regarding placement or selection.

An example scenario for a multi-user game scenario is depicted in Figure 3.

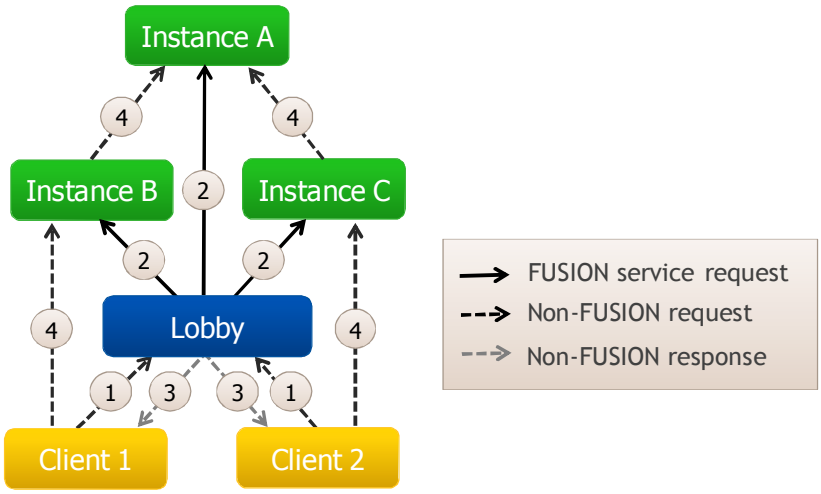

Figure 3: Connecting to the same instance via an external component.

1) In this scenario, both clients first connect to an external lobby service. This lobby service can be a FUSION service, but could just as well also be an external service that is not managed by FUSION, like for example a classical web service. The clients know how to locate this lobby service and announce their presence to the lobby service using an application-specific communication protocol.

2) Upon the arrival of all clients, the lobby service issues a FUSION service request. This can be done either via the service routing or via the domain orchestration. In both cases, the lobby will receive the locators of instances A, B and C.

3) These locators are returned to the respective clients, which then can directly connect to their corresponding instances B or C. Note that instances A, B and C should be chosen with respect to the location and characteristics of both clients. For example, if client 1 is residing in one country, and client 2 in another country, the locations of instances A, B and C should be optimized accordingly, based on the service requirements.

4) In the last step, both clients connect directly to the corresponding instances B and C, which in their turn need to connect to the same instance A, using the locator of A. The latter may have been passed to B and C either in step 2, as one of the deployment or service selection parameters, or in step 3 and even in step 4 via the clients.

Note that the example above only high-lights one conceivable scenario, and alternative scenarios are possible.

2.2.4 Requirements

Below an overview of the multi-user requirements with respect to FUSION.

Req no.	Description	Level
UMU-1	Personalized services	M
UMU-2	Automated distributed deployment of personalized services	M
UMU-3	Automated service routing to personalized services	M
UMU-4	Shared services	M
UMU-5	Automated distributed deployment of shared services	M

UMU-6	Automated service routing to shared services	C
UMU-7	Collaborative services	M
UMU-8	Automated distributed deployment of collaborative services	M
UMU-9	Automated service routing to collaborative services	C

2.3 Statefulness of service instances

Within a single service session, application services are assumed to be stateful. This means that for the data plane, the client needs to be able to communicate with the same service instance for the duration of the FUSION session. As FUSION routing only targets the control plane, this means clients need to be able to directly address the service instance after service selection. This can be done for example by passing back the locator (e.g., the IP address) of the selected FUSION service instance as a result of the service request.

FUSION instances may however also want to maintain state across multiple service sessions coming from multiple service requests from the same or even different clients. For example, a user may want to resume its game session, even when the user switched from a game console to a tablet. For other service types however, this is not necessary. We will elaborate on both cases and discuss the role of FUSION orchestration and routing.

2.3.1 Stateless service instances

For stateless service instances, the situation is trivial, as no state needs to be preserved across multiple sessions, neither by the application service, nor by FUSION. An example is a simple EPG service that always starts from the home page each time you reconnect to that service. This allows ultimate flexibility to FUSION for deploying and selecting an instance to handle the session, and does not require any session state to be maintained by the application services themselves.

2.3.2 Stateful service instances

In case of stateful instances, the session state should be preserved across multiple service requests. In general, this can be achieved in a number of ways:

- A first approach is for the client to bypass FUSION routing when making a new session initiation request and directly contact the same service instance that was responsible for handling the previous session. This approach obviously has a number of drawbacks. First, as FUSION is bypassed for subsequent requests, it cannot assist in finding the location of the best instance for the client, which may have changed as for example the location of the client or the network availability may have changed since the last FUSION session. There may also not be enough session slots available for the corresponding service instance to properly handle the (unexpected) service request. Secondly, the instance or the session data may have expired in the mean time, rendering this option useless in case the previous session ended already a long time ago. Third, when another application client is used, the location of the service instance still needs to be transferred somehow to the new application client.
- A second approach is to allow service requests to be done on FUSION service instance IDs or even FUSION service session IDs. This would still allow for some flexibility regarding the migration of service instances or sessions to other execution zones. However, this drastically increases the overall complexity and state that needs to be maintained by the service routers, with only limited return in value. For example, this approach still requires FUSION services to keep track of the old sessions and perhaps the corresponding allocated resources for a particular amount of time, which places additional burden onto the FUSION application services.

- A third approach is to leave it up to the application services to save the session state in an external database that can be accessed by any service instance. This way, FUSION can freely choose the optimal instance that then first downloads the session state from the known database or repository (which itself may be a FUSION service or an external service). Depending on the service type and the size of the session state, FUSION could even take the location of that remote repository into account via standard FUSION service selection requirements. The disadvantage of this approach is that the FUSION application services themselves are responsible for implementing their own service state management and corresponding repositories or database management.
- A fourth approach is for FUSION orchestration to provide additional support for saving and restoring session state in a decentralized manner. This still allows for the flexibility of freely being able to choose service instances, but reduces the burden for application services to implement their own session state preservation and repositories. We envision this additional support as one of the possible future FUSION orchestration extension services that FUSION services can use. These extension services obviously would be treated as regular FUSION services with special storage and networking requirements.

2.3.3 Requirements

Below an overview of the service state management requirements with respect to FUSION.

Req no.	Description	Level
UST-1	Service routing to stateless application services	M
UST-2	Service routing to session/transaction stateful services	M
UST-3	Service routing to dialog/application stateful services	C

2.4 Service session slots

In FUSION, we propose to make the available number of parallel sessions a particular service instance can handle more explicit. FUSION application services will typically be able to handle a particular amount of service requests in parallel. This significantly reduces the overhead related to service deployment and enables resource sharing and other optimizations at various levels. By making the number of available session slots explicitly visible, we envision to significantly reduce the amount of information that needs to be distributed and managed in the FUSION routing domain. FUSION application services can implement the concept of session slots via an internal service factory.

Req no.	Description	Level
USSS-1	Service instances supporting multiple parallel sessions	M
USSS-2	Service routing to available sessions rather than specific instances	S

2.5 Service selection

A key functional requirement for FUSION is being able to automatically select the optimal service instance for a particular service request. Different types of services may have different requirements regarding service selection. Some services may completely rely on the FUSION routing domain for automatically selecting the optimal instance. For other more complex services, like for example collaborating services, this is not possible. For these services, it may be up to the orchestration domain and/or the application itself for selecting an appropriate service instance, or for deciding to deploy a new instance in an appropriate location on demand.

2.5.1 Network-driven service selection

For simple single-user services with few critical parameters, the optimal service selection can be done automatically by the FUSION routing domain. For example, a client requests FUSION to find to an EPG instance, and the FUSION service routers automatically locate and/or connect the client to an optimal instance.

2.5.2 Orchestration-driven service selection

In other cases, it is not possible for the FUSION routing to automatically select the optimal service instance. For example, in case of a multi-user game or video conference, multiple users may want to explicitly join the same game or conference session. To optimize service selection for these use cases, another service should be used, e.g., a game lobby services, as depicted earlier in Figure 3 on page 13. This lobby service can trigger the FUSION orchestrator (for example when all users are present to start the game) to start selecting and/or deploying the composite service, taking into account the location of the individual users and the service requirements. The orchestrator then needs to return all necessary service locators back to the requesting lobby service, which then are returned to the relevant clients. Note that in this case, the service selection does not need to be optimized for the requesting client, but for the client applications that will be part of the multi-user session.

2.5.3 Application-driven service selection

For complex composite services with complex interaction patterns, FUSION is not responsible to coordinate the full choreography of message exchanges in between the various service components involved in the composite service function. For these services, it is up to the application to drive the service instance selection as well as the individual routing of messages.

Req no.	Description	Level
USS-1	Network-driven service selection	M
USS-2	Orchestration-driven service selection	M
USS-3	Application-driven service selection	S

2.6 Service scaling and placement

Similar to service selection, different service types may require different service scaling, placement and deployment strategies, and may demand or rely on different FUSION components for efficiently deploying new service instances at different moments in time (i.e., pre-provisioned or on-demand).

2.6.1 Network-driven service scaling and placement

In case of network-driven placement and deployment, the trigger for deploying a new service instance is coming from the FUSION routing domain, for example as part of a (series of) service request(s) for services for which no appropriate instances have been deployed yet. This can be because of an unexpected increase in service requests, or for services that typically require a more on demand approach.

2.6.2 Orchestration-driven service scaling and placement

Next to the on demand network-driven placement and deployment, the domain orchestration layer is responsible for pre-provisioning and pre-deploying service instances at particular locations based on predicted service demands, deployment policies and strategies coming from the service provider when registering a service. For example, in case of the image tagging or EPG services, the service provider may want to ensure a minimum amount of available service instances or session slots at specific moments in time, and expects from the FUSION orchestration domain to automatically scale up or down the amount of instances accordingly.

2.6.3 Zone-driven service scaling and placement

Inside an execution zone, the zone manager is responsible for deploying and placing the service instances inside the underlying data centre. This will typically involve a different set of placement algorithms compared to network or orchestration layer placement. We also envision execution zones to autonomously be able to scale up and down the number of active instances, for example to match the promised amount of available session slots per service type.

2.6.4 Application-driven service scaling and placement

Applications themselves may also trigger FUSION to place and deploy new service instances in an orchestration domain. For example, in case of a multi-player game or video conference, the request to set up a new session comes from another application making an explicit request to deploy a new instance in an orchestration domain.

2.6.5 Evaluator services

In FUSION, we envision using evaluator services for efficiently placing and deploying new service instances onto particular execution environments, without FUSION having to be fully aware or capable for understanding all detailed functional or non-functional requirements the services may have. With these evaluator services, some of the evaluation can be offloaded and handled in a customized fashion by the services themselves, which then provides scoring metrics that are fed back into FUSION for making the final selection.

Req no.	Description	Level
USP-1	Network-driven service scaling and placement	M
USP-2	Orchestration-driven service scaling and placement	M
USP-3	Zone-driven service scaling and placement	C
USP-4	Application-driven service scaling and placement	M
USP-5	Support for evaluator services for placement	M

2.7 Service composition

Next to atomic FUSION services, we also want to be able to describe, deploy and manage composite services, which consists of a connected graph of service atoms. Each of the atoms can be both autonomous services or merely subcomponents that are part of an end-to-end service delivered to the end users. As discussed in following sections, these graphs can be either static or dynamic in nature, and may be deployed in a distributed manner across multiple execution zones. Most of the use cases we will elaborate are in fact composite services.

Req no.	Description	Level
USC-1	Atomic services	M
USC-2	Composite service graphs	M

2.8 Dynamic service instance graphs

We want to be able to support services with service instance graphs ranging from completely static service instance graphs to fully dynamic service instance graphs, both at specification as well as execution time. This range of dynamicity has a significant impact on the overall orchestration, execution as well as routing of these services.

2.8.1 Fully static service instance graphs

In the case of completely static service instance graphs, the entire service graph is known beforehand during registration. For these types of composite services, the FUSION orchestration domain knows the complete service graph, dependencies and perhaps previous monitoring and quality information upfront. This complete picture allows for more optimal service placement strategies. It also simplifies optimal service routing as both the complete graph as well as all service atoms are known beforehand. An example is the combination of object recognition service and a database service working together in a pre-defined manner.

2.8.2 Fully dynamic service instance graphs

In case of a fully dynamic service instance graph, FUSION only has a direct view of the individual service atoms and at best only indirectly of the entire service graph. The service instance graph is constructed dynamically and it is typically up to the responsibility of the service itself (e.g., through a special coordination service atom) to keep track of the complete service instance graph. Individual service atom instances will typically have their own lifecycle management, and thus may be deployed at different moments in time. As FUSION orchestration does not have information on the complete service instance graph, it cannot easily optimize the deployment or composition of the atoms with respect to each other, and at best only can optimize the point-to-point connections between each connected service atom pair in the graph. In case the service instance graph contains a special orchestration service to manage the dynamic service instance graph, this service could however communicate to FUSION orchestration for passing information concerning the dynamic service instance graph, and it may request specific deployments based on partial service graphs rather than point-to-point connections. An example is a dashboard service containing multiple dynamic elements implemented by other services like news feeds or video streams, composed dynamically by the user.

2.8.3 Dynamic graph of static service instance subgraphs

This category captures the range of service graphs that fall in between fully static and fully dynamic service instance graphs. Basically, it consists of a number of service instance graphs that are dynamically connected to each other, forming a larger dynamic service instance graph. Each individual subgraph can be deployed and managed as a static graph, which can be connected and disconnected on demand. Example use cases include a dashboard or EPG, onto which the output from other service instances or service instance graphs can be added or removed dynamically, as well as a multi-player game or video conference, in which users can be added or removed, either statically when the session starts, either dynamically during the game or video conference session. In the former case, when the session starts, the orchestrator can still assist in optimally placing the dynamically constructed graph, whereas in the latter case, FUSION can only do a best-effort approach, although it could assist in migrating components of the service graph when the placement becomes suboptimal.

Req no.	Description	Level
USG-1	Fully static service instance graph composition	M
USG-2	Fully dynamic service instance graph composition	M
USG-3	Step-wise service instance graph composition	M

2.9 Service distribution

Service distribution enables optimally distributing the individual service atoms of a more complex composite service across multiple execution zones to meet the overall application requirements, e.g. regarding network latency, execution requirements, etc. In other cases however, the inter-service interactions may require a local deployment in the same execution zone (and possibly even the same execution point) because of high-bandwidth and/or low-latency interconnections.

2.9.1 Local service deployment

The communication pattern in between particular services may require that services should ideally be placed onto the same execution environment to minimize communication and computation overhead. There is a trade-off between placing two services next to each other and directly streaming raw data from one service to the other, versus deploying these services in remote locations and inserting computational intensive transcode operations to reduce network bandwidth at the expense of computational overhead and extra delay. In FUSION, we will evaluate this trade-off in a number of scenarios and evaluate the effectiveness of optimized inter-service communication channels in case of local deployments.

2.9.2 Distributed service deployment

In other cases, a distributed deployment may be necessary in order to meet the application requirements regarding network latency, compute and storage capabilities and/or efficient data access. In case of multi-user services for example (multi-user games or an advanced personalized video conferencing system), some of the shared services (e.g., the game server) should best run in a more centralized location, whereas the more personalized services should likely be placed closer to the corresponding end users. We will explore the impact of distributed deployment for such use cases in more detail.

Req no.	Description	Level
USDD-1	Local service deployment inside the same execution point	S
USDD-2	Local service deployment inside the same execution zone	M
USDD-3	Distributed service deployment across execution zones	M
USDD-4	Distributed service deployment across FUSION domains	C
USDD-5	Mixed distribution service deployment	M

2.10 Inter-service communication

Due to the inherent nature of the service types we want to deploy and evaluate in FUSION, inter-service communication will often be very critical, be it in between services that are deployed locally on the same execution environment or distributed across different execution zones.

2.10.1 Low-latency

In case of a distributed deployment, low latency will be of key importance for particular sets of related services that heavily communicate with each other.

2.10.2 Late binding

Depending on the placement of heavily communicating services, the optimal communication channel and interconnect may be different. In case two services are running in the same environment or on the same machine, special communication channels, like for example shared memory, could be used for efficiently streaming raw high-bandwidth data in between services. In case these services are deployed on different locations or in different environments and regular networking infrastructure has to be used, additional functionality like transcode and encryption functionality needs to be inserted, impacting both the delay as well as computation requirements. We will evaluate the impact and consequences for both scenarios.

Req no.	Description	Level
UIC-1	Low-latency inter-service communication	M
UIC-2	Late binding support	S

2.11 Platform requirements

In FUSION, we want to support the dynamic deployment and management of services that are demanding and interactive or real-time in nature. This poses a number of challenges and requirements towards the execution environment on which these services are to be deployed.

2.11.1 Fast service deployment

For particular services, it is not economically feasible to properly pre-deploy them across a wide range of heterogeneous and geographically distributed execution environments. In these cases, FUSION needs to be able to quickly deploy new instances on demand onto one or more of these execution environments. This requires the capability to quickly deploy new services on the fly and therefore poses a number of requirements with respect to the service encapsulation and

deployment models. In FUSION, we want to explore the possibilities of light-weight virtualization and deployment techniques for efficiently deploying such services.

2.11.2 Heterogeneous hardware

Due to the distributed nature of execution zones, the available hardware profiles across these zones may differ substantially, both in nature and size. Centralized data centres and clouds may typically contain virtually unlimited amounts of general purpose computing capabilities, whereas smaller execution nodes close to the access network may be much more specialized and limited in nature. Secondly, particular services like game renderers may require specialized hardware like GPUs for efficient processing. In FUSION, we want to be able to take both cases into account when placing and deploying services across and within execution zones.

Req no.	Description	Level
UPR-1	Fast service deployment	M
UPR-2	Support for heterogeneous hardware	M
UPR-3	Support for application-domain specific hardware acceleration	M

2.12 Smart objectives

FUSION has defined five SMART objectives which the use cases and evaluation scenarios will aim to meet through a combination of simulation and testbed-based experiments as detailed in the following.

2.12.1 Reduced startup-time

Reduce the start-up time for remotely executed service component instances to within the order of seconds compared to today's equivalent operation of instantiating a virtual machine in 10s to 100s of seconds.

The project aims to meet this objective through the use of lightweight containers for deploying service instances rather than creating a virtual machine for each new instance. This is applicable to pre-deployment of instances by the orchestrator as well as to instantiation on demand. Both cases will be included in the demonstrator and instantiation times will be measured.

2.12.2 Reduced network footprint

Reduce total network traffic footprint (bits/s x number of network links traversed) by 50% for applications remotely processing large bandwidth streams by optimising the placement of service processing nodes.

FUSION orchestration logic will use server placement optimisation algorithms to deploy instances in execution zones close to user demand thereby reducing the distance from user to server and the quantity of network resources that the traffic generated by the service will occupy. Having service instances running in execution zones close to the users is only part of the story as the service routing plane needs to be aware of the available instances and route requests to the closest execution zone where there are available resources to process the requests. As the iMinds and TPSA testbeds will have limited scope for wide-area network tests with large numbers of highly-distributed execution zones the evaluation will be augmented with simulations of larger-scale networks.

2.12.3 Reduced network latency

Reduce the network component of application latency by 50% for remotely processed services such as personalised video and networked games by optimising the placement of service processing nodes.

Intelligent server placement and service routing algorithms need to ensure service instances are geographically and topologically close to the users but they must also take into account the performance of the intervening network. Static models of network performance will be augmented by dynamically monitoring network latency and other network metrics. Tests to measure network latency will be undertaken in the project testbeds and will be complemented by large-scale evaluations through simulation.

2.12.4 Optimal service selection

The service resolution, selection and routing mechanisms should select service instances within no more than 200% of the optimal, in the worst cases, according to a combined metric that includes parameters such as RTT, throughput and service load.

The FUSION service routing plane should direct service requests to nearby execution zones in terms of network performance metrics - to reduce latency and increase throughput – but the selected execution zone should have sufficient capacity to fulfil the service invocation request. The FUSION service routing plane therefore needs to be aware of server load and make appropriate trade-offs between network and service level metrics when forwarding invocations. A first approach to achieving this is the proposal to inject session slot information into the routing plane. Tests to measure the effectiveness of the routing algorithms in selecting close to optimal network paths+service instances will be undertaken through simulations but the ability to route requests on both network and service metrics will be demonstrated in the project testbeds.

2.12.5 Low routing management overhead

The exchange of routing information between service-centric routers will not exceed 5% of the capacity of the interconnecting links.

Announcing the availability of a large number of services running in a large number of execution zones together with dynamically updated load and performance metrics can generate a significant amount of traffic unless aggregation and compact routing techniques are applied. The efficiency of the routing protocol overhead will be analysed through simulation to assess the trade-offs of overhead versus routing stretch and selection accuracy. Comprehensive evaluation of the effectiveness of the routing protocol versus the routing overhead is difficult to achieve in relatively small-scale testbeds so tests in the prototype will be limited to measuring the amount of routing and monitoring traffic generated in the project testbeds.

Req no.	Description	Level
USO-1	Demonstrate that the creation of a FUSION service instance in a light-weight container can be achieved in less than 10 seconds.	M
USO-2	Compare the traffic footprint of FUSION applications in different cases of server placement and service routing algorithms in the iMinds and TPSA testbeds, complemented by larger-scale simulations.	M
USO-3	Measure the network latency between user and execution zone under different server placement and selection/routing strategies in the iMinds and TPSA testbeds, complemented by larger-scale	M

	simulations. The performance of strategies with and without awareness of network performance metrics should be compared.	
USO-4	Demonstrate that the FUSION service routing and forwarding algorithms take both network metrics and server load information into account. Show the difference when requests are forwarded based on a) network metrics only, b) server-load only, c) network metrics and server load. Simulate larger-scale networks and calculate the performance of the selected paths/servers compared to the optimal.	M
USO-5	Measure the traffic generated by the routing protocol and monitoring systems in the FUSION testbeds. Evaluate routing overhead versus routing stretch and selection accuracy through simulation.	M

3. USE CASE MODELING & DESIGN

In Deliverable D2.1, we provided an overview and high-level description of possible use cases and applications that may benefit from FUSION. In this section, we will discuss in more detail a selected subset of these use cases, and we will high-light what their key requirements are with respect to the FUSION architecture. From a high-level point of view, the selected use cases can be placed into four major categories:

- Single-user personalized services: EPG, video tagging, single-user game
- Multi-user collaborative services: multi-user game, video conference
- Short-lived request/response services: image tagging
- Dynamic service composition: dashboard, EPG

In the following sections, we will discuss and model each use case in more detail. We selected these use cases because each of them showcases and evaluates one or more key functionalities and features of the FUSION architecture.

3.1 Dashboard and EPG

The media dashboard use case is an example of an advanced 2D or 3D user interface that is rendered in the network as a FUSION service and that represents a next-generation Electronic Program Guide (EPG). Next to the basic functions you typically expect from a media dashboard or EPG, like changing the channel, browsing through the TV guide, etc., it can also be a portal to other types of content, including personal videos and pictures, games, or other interactive FUSION applications.

Technically such a media consumption scenario may be implemented by a combination of multiple service software components. Of course, depending on the situation not all service components are necessary, or additional service components are used. An example media dashboard service is depicted in Figure 4.

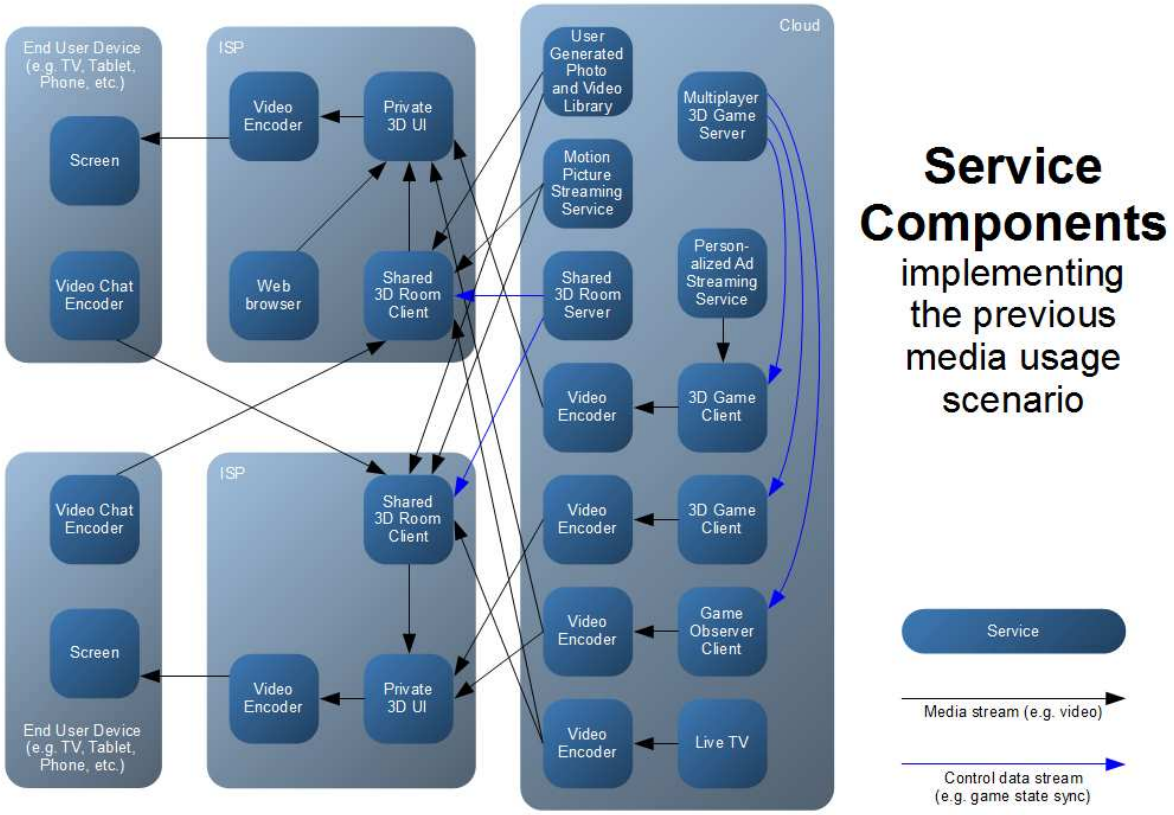

Figure 4: Example media dashboard service graph and service components

This diagram contains different types of possible service components:

- A private 3D UI contains the dashboard for one particular user, for example a 2D or 3D composition with per-user functionality. For example, the top user in the diagram may play a multi-user video game in this private 3D UI.
- In this particular scenario the private 3D UI is rendered not on the end-user device (for example because it is a mobile or thin TV device which is not powerful enough). Instead the 3D UI is rendered in a service software component that is running not on a cloud server, but on an ISP server that is closer to the user and therefore providing more responsiveness.
- Transferring the video stream from the private 3D UI service to the end-user device requires encoding and decoding the video stream, which may not be statically integrated into the particular software, but implemented as separate services. In the above sample diagram the video encoder services are demonstrated.
- A shared 3D room client is an example of a dashboard that is shared between multiple users. This provides for example an environment where two people can browse a photo library together, browse available VoD series episodes, have a shared image viewer and VoD selections.
- The execution environment hosts various service components for media consumption, e.g. VoD servers, image libraries or game servers, which transfer or stream their data to the dashboard service components.

A core aspect of the EPG use case is that it is a classic example of a low-latency interactive personalized streaming service, where both the personalization aspect as well as the low-latency interactivity are two crucial aspects. The former aspect may require dedicated hardware for either rendering or streaming the output in a cost-efficient manner to the end devices of the users, whereas the latter aspect impacts the placement and deployment of these services in the network.

Depending on the dynamicity of the EPG service, the nature and location of the input streams for the EPG service may also impact the overall dynamicity and placement of the service as handled by the FUSION architecture. Being also a key service of many ISPs, the EPG use case also represents an excellent primary candidate for ISPs to start deploying in their network and data centre infrastructure using the FUSION architecture.

A core aspect of the dashboard use case is the feature that the service instance graph is not static, but has to change dynamically: a service can dynamically connect to new services at different times. For example, the user may use the dashboard to choose a video, and then the dashboard service uses FUSION to request that particular video service, and streams data from that video service instance into the existing dashboard service instance.

3.1.1 Client and Backend Interactions

In Figure 5, a sequence diagram is shown for a client connecting to a running EPG service instance. Solid arrow lines represent message-based communication, whereas dotted arrow lines represent streaming communication; similarly, black lines represent in-band FUSION communication, whereas blue lines represent out-of-band non-FUSION communication. As can be seen from the diagram, we modelled the video feeds to be streamed outside of FUSION, though this may only be required for legacy clients using legacy video streaming formats.

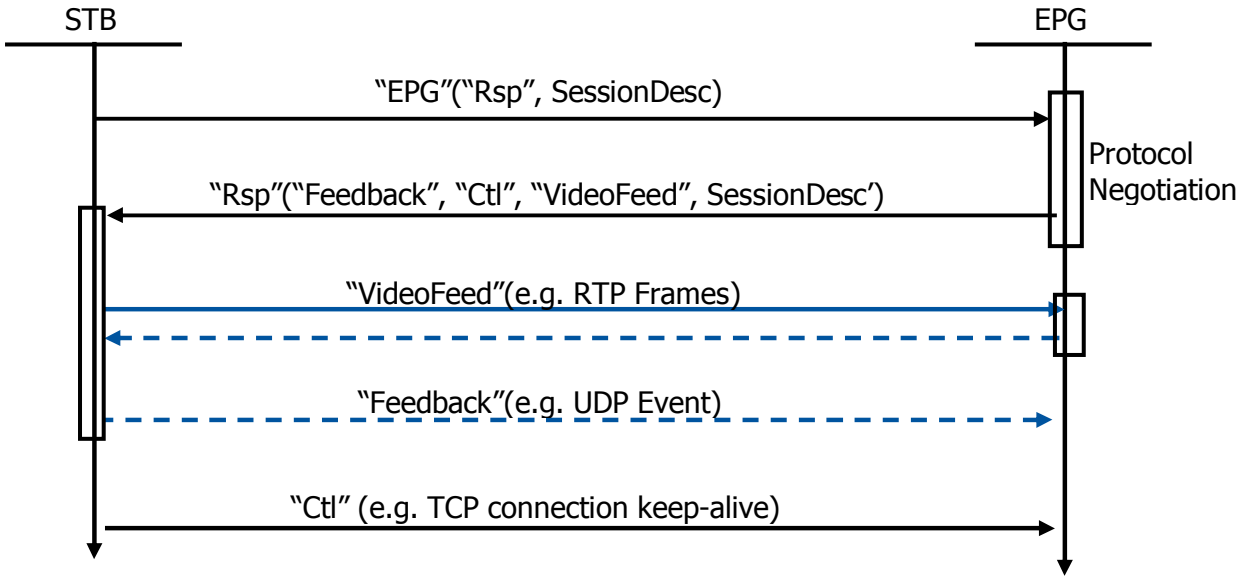

Figure 5: Client connecting to an EPG Service Instance

Basically, a client first issues a service request for a service with name "EPG", along with a number of FUSION and non-FUSION service request parameters, describing the context of the session. These parameters may include the response channel over which to send back the result, the resolution and quality, an identification of the end user, etc. As a result, the (by FUSION) selected service instance will create and prepare a temporary session to handle the service request, and send back a response to the client. This response may include how and where to contact the EPG session, including the video feed and feedback channel. There may also be a control channel that acts as a keep-alive so that the EPG service session can detect when the client disconnects (abruptly), after which it may decide to terminate the session.

The service instance itself will typically also communicate with a number of input sources that provide the main content for the EPG service. This may include both FUSION sources as well as external sources, and may consist of both static and interactive content. In the latter, the EPG service may have to forward the feedback coming from the client to the other service, so that the client

appears to be directly interacting with the other service (e.g., a game client). This is backend service diagram is depicted in Figure [ref].

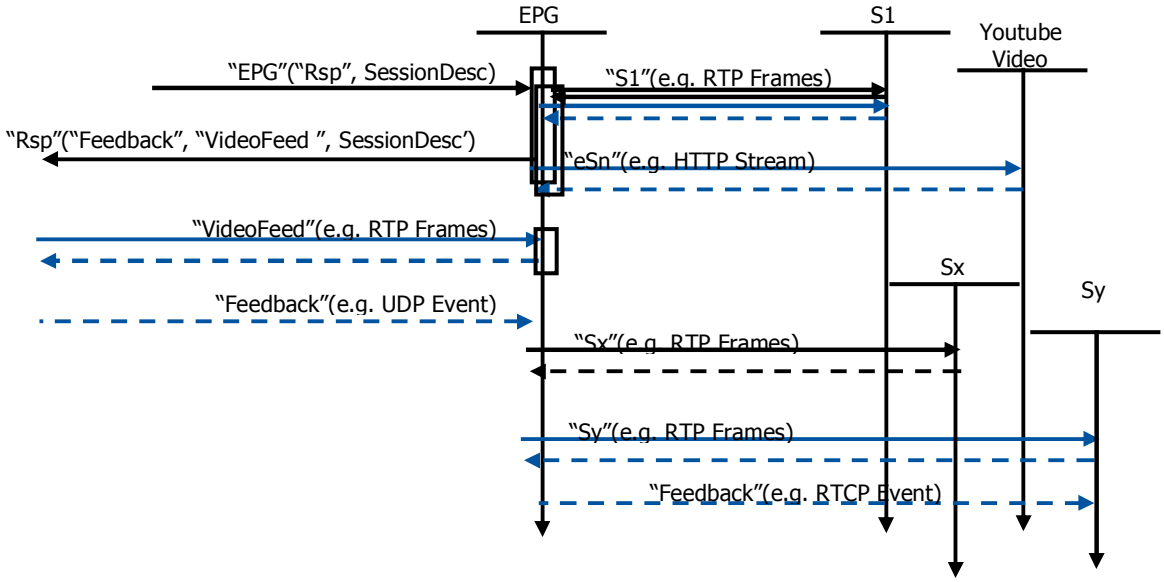

Figure 6: Backend Resources for EPG Service Instance

3.2 Real-world tagging

The tourist office of A Very Modern City rents head-up displays with integrated front-facing cameras to city visitors. Useful information is presented as a graphical overlay on the display. The service is tailored to the user's location and preferences. Going beyond elementary GPS-based location tracking to determine the position of the user, the content and the position on the display of the tags is calculated by analysing the video feed from the front-facing camera integrated in the head-up display. The presented tags may contain historical or practical information (e.g. expected arrival time of next bus), advertisements (e.g. to reserve a restaurant in the current street) and messages left by friends who have previously visited the same location (e.g. to discommend a restaurant).

The video feed from the front camera is continuously streamed to the back-end infrastructure, comprising multiple small execution zones managed by FUSION. The idea is presented in Figure 7.

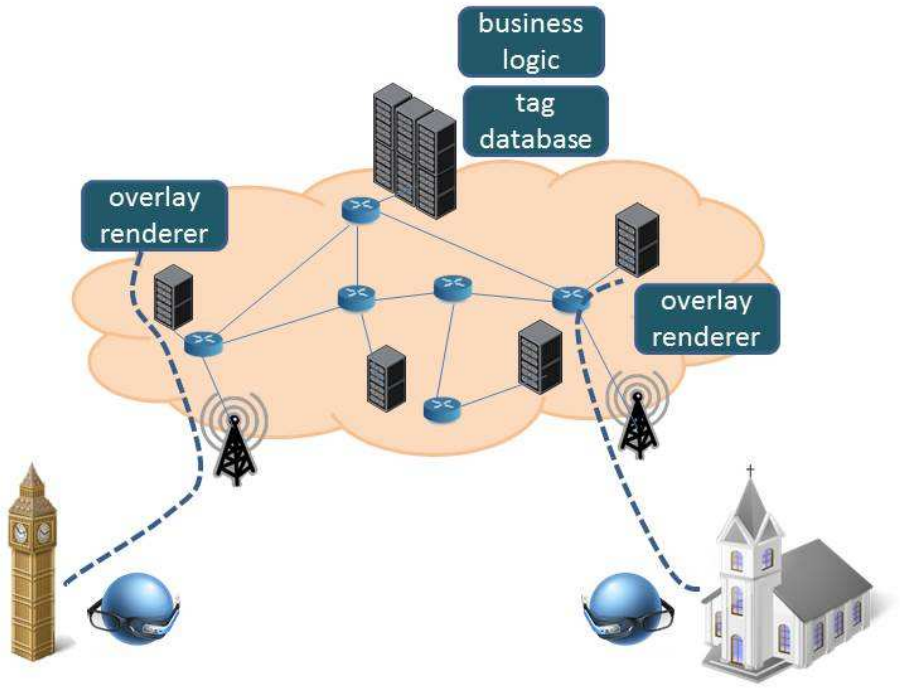

Figure 7: Video captured through head-mounted cameras are analyzed in the network. The resulting overlay with tags of recognized objects is transferred back to the client.

From the application perspective, low latency is key since the overlay must be correctly positioned according to the user's current gaze. From the system perspective, multiple replicas of the tourist guide service must be correctly distributed across the different zones to guarantee the application QoS constraints, taking into account the costs associated with allocating resources on the small-scale execution zones distributed in the access and aggregation network.

User mobility, demand variations and the use of different wireless access technologies (Wi-Fi, 4G) result in continuously varying context and network parameters that are monitored by the FUSION platform. The client component of the tourist guide, installed on the head-up display, is FUSION-aware and is always connected to the most optimal service replica.

We will demonstrate two scenarios, differentiated by the duration of the session between the client component and the overlay renderer.

3.2.1 Short-lived sessions

This scenario will demonstrate independent request-response patterns between a service instance and the FUSION service routing plane. A client will send a frame of the video along with his name-based request for the tagging service. FUSION routers will deliver this request to the most appropriate instance, based on the actual status of the network and server load. The response (position and size of the rectangle) will be returned via the FUSION service routing layers.

In the short-lived session scenario, the service specific data is piggybacked with the FUSION request for service instance selection. Possibly, a different instance is selected for each new request for the tagging service.

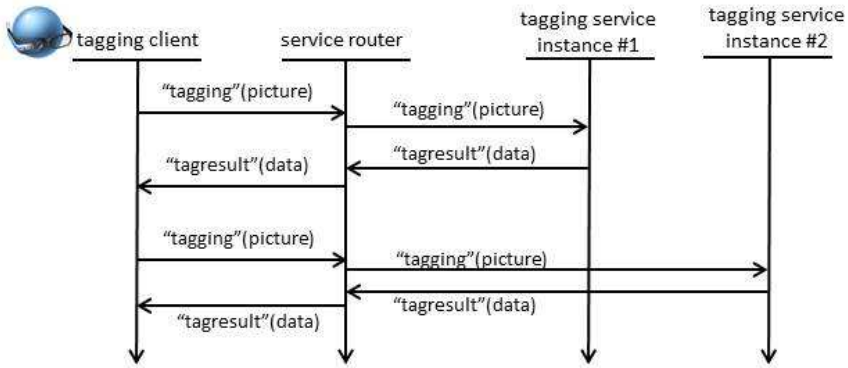

Figure 8: Each tagging request can be routed to a different instance.

3.2.2 Long-lived sessions

In this scenario, a session will be established between the client and the tagging service. The FUSION service routing plane is only used for instance selection and possibly session set-up. The service routers return to the client the locator of the selected instance. The client then sets up a session with the selected instance outside of the FUSION service routing plane.

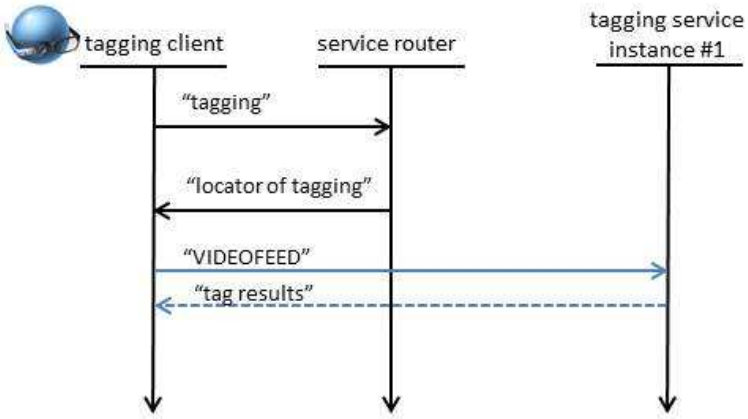

Figure 9: The FUSION service routers only select a service instance. Service-specific communication happens outside of the FUSION service routing plane.

3.3 Thin-Client 3D Game

In this section, we discuss the single-user game scenario.

3.3.1 Overview

In the 3d game use case a game service component is running for example in a centralized data centre or closer the edge. This use case is similar to today's cloud gaming: user input (keyboard, mouse, gamepad) is sent from a client device (e.g. PC or tablet) to a service instance in the network running the game. The service instance simulates the game, renders the resulting 3d image, and transfers the output as video stream back to the client device.

The intention of this use-case is to demonstrate the benefits of FUSION for cloud-gaming use cases. For example, a FUSION orchestrator can optimize the placement of a game service, e.g. at the edge closer to the user having lower response time instead of a data centre.

3.3.2 Resource usage optimization via the session slots mechanism

This use case will also demonstrate structural differences in the service implementation itself when using FUSION. In classical cloud gaming, one running instance of the game can usually handle only one game session, because typical games are usually designed as single-session applications for end-user devices, and ported as such onto cloud servers directly. In the FUSION prototype we plan to demonstrate the session slot mechanism: One instance of the game service can handle multiple independent game sessions for different users. This has various advantages compared to classical cloud gaming implementations.

One advantage of the slot mechanism is better memory resource utilization: program code and game assets like 3D geometries or textures have to be loaded only once and can be shared by multiple sessions. Only the game state itself must be kept separately for each user. Technically this will be implemented by using a component based architecture (presented visually as nodes to the game developer), allowing to define factories for instantiating session specific component graphs, which access and use data (e.g. assets) from shared components. Based on an existing component architecture, FUSION implements a new factory mechanism for instantiating separate input channels and output render views, and connect them with instances of components for representing the individual game state for each user.

3.3.3 Component-based slot mechanism implementation

Technically this use-case will be built on an extended version of the commercial Shark 3D software of Spinor. This software is a commercial middleware and authoring editor for creating interactive real-time 3d applications like for example games or 3d user-interfaces. Spinor is working on modifying and enhancing this software to support the implementation of FUSION service components. Advantages to base the implementation on a commercial software are: simplifying the creation of non-trivial interactive service components, since the Shark 3D software provides many features out of the box which can be re-used, and allowing the creation of use-case prototypes which have industry-typical characteristics, since the software is currently being used in the industry.

3.3.4 Evaluator service

The gaming use-case is also an ideal sample of demonstrating the advantage of evaluator services, since such a gaming service has particular hardware demands that are not available on many typical cloud servers. This particular gaming use-cases require a GPU supporting at minimum Direct 3D feature level 10.1.

3.4 Thin-Client Multi-User 3D Game

Another use-case is a multi-user 3d game. In contrast to a classical multi-player gaming architecture of connecting multiple game clients running on client devices or on cloud servers to game servers, in this use-case we use one FUSION game server service to host the whole game for multiple users playing together.

This use-case is a sample where multiple users connect to the same service. Similar to the slot-based single-player gaming services, each user needs a separate instantiation of components, for example for rendering the user-specific personalized 3D output, because every player usually sees a different section of the virtual game world from a different perspective, and different state needs to be maintained per player, like for example health, location, items and scores. However, in contrast to the slot-based single-player gaming services, all players not only share the same assets, but share also the same 3D state. This will be implemented using existing components of the underlying software from Spinor combined with a new service-oriented FUSION-specific component structure.

Therefore technically, this use-case also provides a different case of how a service instance handles multiple users by combining component instances shared by all users with other component instances that are specific for each user.

3.5 Summary of use cases and their requirements

In the table below, an overview is shown of the selected use case scenarios and the list of requirements each of the use cases are stressing. The symbol "X" denotes that typical variants of the selected use case scenario contain the requirement, whereas "(X)" denotes that some but not necessarily all variants of the use case scenario exhibit the requirement.

	EPG	Image tagging	Video tagging	Single-user game	Multi-user game	Dashboard
Request-response service		X				
Streaming service			X			
Interactive service	X			X	X	X
Personalized service	X	X	X	X	X	X
Shared service			X			
Collaborative service					X	(X)
Stateless service	X	X	X	X	X	X
Stateful service	(X)			(X)	(X)	(X)
Service session slots	X	X	X	X	X	X
Network-driven selection	X	X	X	(X)		X
Orchestr.-driven selection	X			X	X	X
Application-driven selection					X	
Network-driven placement	X	X	X	X		X
Orchestr.-driven placement					X	X
Appl.-driven placement				X	X	X
Evaluator services	(X)			X	X	X
Composite service	X	X	X	X	X	X
Static service graph	X	X	X	X		
Semi-dynamic service graph	X				X	X
Dynamic service graph						X
Local deployment	X	X	X	X		
Distributed deployment		X	X		X	X
Low-latency communication	X	X	X	X	X	X
Late binding	X		X		X	X
Fast service deployment	X	X		(X)	(X)	X
Heterogeneous environment	X			X	X	X
Reduced startup-time	(X)		X		(X)	(X)
Reduced network footprint	X		X	X	X	
Reduced network latency	X	X	X	X	X	X
Optimal service selection	X	X	X	X	X	X
Low-overhead routing						

4. TESTBED ENVIRONMENTS

This section describes the main hardware and software environments and infrastructure that we intent to use for developing our prototypes and evaluating various key functions of the FUSION architecture.

4.1 IMinds Emulation Platform

4.1.1 Virtual Wall

The evaluations are performed on the iMinds Virtual Wall from its iLab.t Technology Centre (http://ilabt.iminds.be). The Virtual Wall facilities currently consist of 2 deployments (with 200 nodes and 100 nodes respectively). Servers have 4 to 6 cores per machine, and are connected with 4 or 6 gigabit Ethernet links to a non-blocking 1.5 Tb/s VLAN Ethernet switch. At this stage, the target in FUSION is to allow access to the newest deployment of the Virtual Wall of which all instances are reachable through the public Internet using the IPv6 protocol. The node specifications are listed in Table 1.

Table 1 - hardware configuration of the Virtual Wall

Regular nodes (100)	**GPU-equiped nodes (2)**
Dual INTEL XEON E5645 24 GB RAM 250 GB (single disk)	Dual INTEL XEON E5645 12 GB RAM 4x GeForce GTX 580 4x 1TB in RAID-5

The Virtual Wall nodes can be assigned different functionalities ranging from terminal, server and network node to impairment node. The nodes can be connected to test boxes for wireless terminals, generic test equipment, simulation nodes (for combined emulation and simulation) etc. The Virtual Wall features Full Automatic Install for fast context switching (e.g. 1 week experiments), as well as remote access.

Being an Emulab testbed at its core, the Virtual Wall offers the possibility to create any desired network topology and add any desired clients and servers. Emulab allows for repeatable, dedicated and confined experiments and is responsible for swapping in required operating system images, dynamically interconnecting the (virtual) nodes through VLANs on a network topology with emulated network links (with proper capacity, delay and packet loss characteristics).

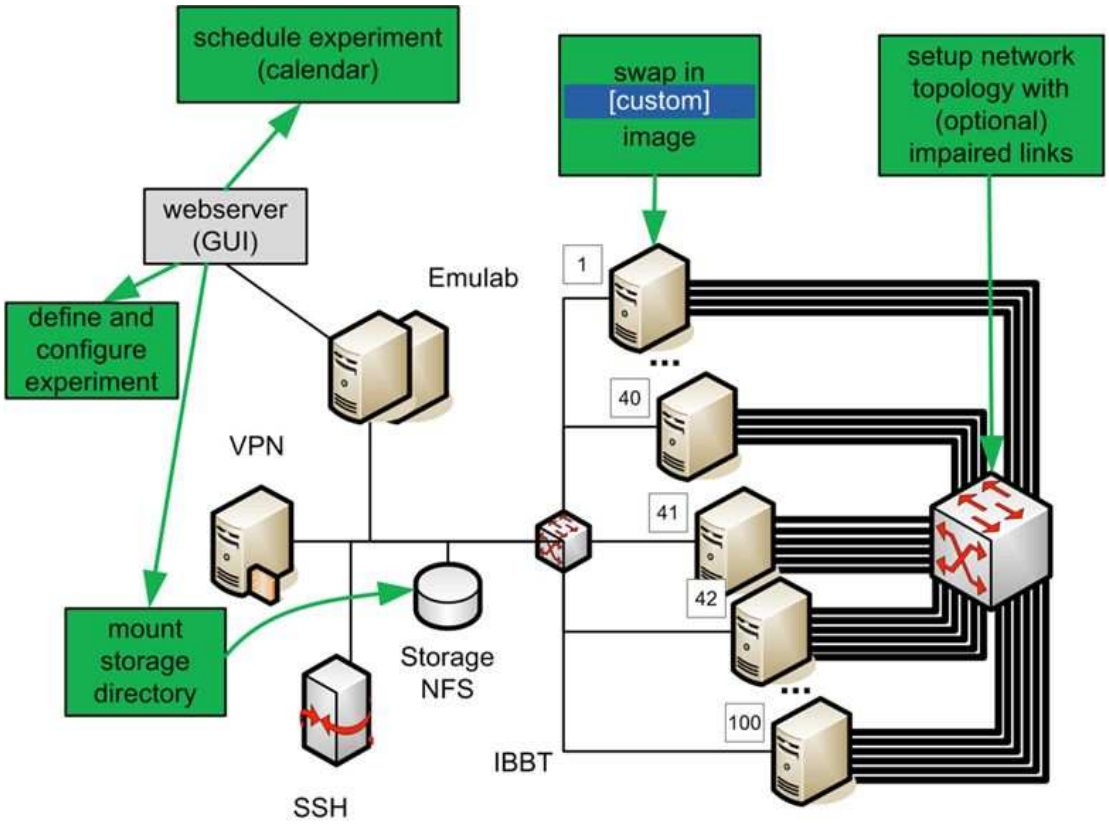

Figure 10 The Virtual Wall is controlled by Emulab management software for deployment of OS images, and configuration of network topology and network impairment.

The Virtual Wall also provides the necessary network monitoring tools, based on the Zabbix framework [ZAB13], to examine network traffic and consumption of memory, CPU, disk load and energy on the different nodes.

In addition to the network layer functionality, applications can be deployed on the Virtual Wall nodes controlled by the researchers through VM images. These instances can also be monitored (bandwidth, CPU and memory consumption, response times and availability over time) and can be invoked through request generators. These request generators can be tuned in a variety of different parameters (e.g. spatial request distribution, temporal request distribution, popularity of service instances) and assign the client locations through automated generation of scripts.

4.1.2 Validation tools

4.1.2.1 NS-2 experiment description

In Emulab, experiments are described using NS-2 scripts, for which advanced tutorials[2] are available. The deployment script describes all nodes and their interconnecting network links. For each node, we can describe desired functionalities (e.g. GPU required) and the operating system (e.g. Ubuntu, Windows).

By scripting the deployment, a wide range of hardware topologies can be evaluated. For example, we can study different zone architectures, or network topologies between zones. A sample script is shown in Figure 11.

[2] http://users.emulab.net/trac/emulab/wiki/AdvancedExample

```
#generated by Netbuild 1.03
set ns [new Simulator]
source tb_compat.tcl

set compute1 [$ns node]
tb-set-node-os $compute1 UBUNTU12-64-STD
$compute1 add-desire gpunode 1

set compute2 [$ns node]
tb-set-node-os $compute2 UBUNTU12-64-STD

set controller [$ns node]
tb-set-node-os $controller UBUNTU12-64-STD

set client [$ns node]
tb-set-node-os $client UBUNTU12-64-STD

set recognition [$ns node]
tb-set-node-os $recognition UBUNTU12-64-STD

set lan0 [$ns make-lan "$compute1 $compute2 $controller " 1000Mb 0ms]

tb-fix-node $controller n097-21a
tb-fix-node $compute2 n096-10a

set lan1 [$ns make-lan "$controller $recognition $client " 1000Mb 0ms]

tb-fix-node $recognition n096-06a
tb-fix-node $client n096-11a


$ns rtproto Static
$ns run
#netbuild-generated ns file ends.
```

Figure 11: Sample ns-2 configuration to deploy an experiment on the Virtual Wall.

4.1.2.2 Netbuild GUI

Besides an XML RPC[3] interface, a graphical user interface is available on the Virtual Wall platform as well. This allows for intuitive and rapid prototyping of different set-ups. An example is shown in Figure 12.

[3] https://www.emulab.net/xmlrpcapi.php3

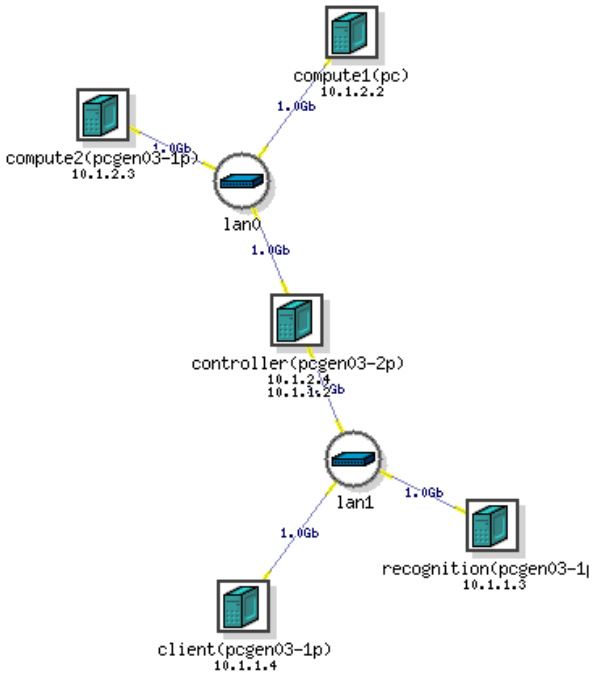

Figure 12: Rapid prototyping via the GUI

4.1.2.3 Monitoring

The Virtual Wall has several components that allow monitoring network and application performance. In some cases, some minor integration work with the Virtual Wall can be needed.

4.1.2.4 Monitoring of Hardware performance

To monitor the hardware performance of a server, a distributed monitoring architecture exists on the Virtual Wall. This monitoring architecture consists of several small monitoring probes that measure power consumption, memory usage, etc. and transmit it to a central server. Each monitoring probe contains a transactional database, to store the measured data in a cyclic manner, and a web interface, to provide visual information (e.g. through real-time graphs). The following metrics can be measured through the architecture:

- Memory usage
- CPU usage
- Transfer rate on the hard drive in terms of read and write operations
- Incoming and outgoing bit rate on a network interface
- Actual voltage
- Total power consumption
- Environmental temperature (e.g. temperature close to the device)

4.1.2.5 Monitoring of Network Performance

Two distinct types of network monitoring probes exist on the Virtual Wall. A Click Modular Router [KMCJ00] based monitoring probe, and a libpcap based monitoring probe. The first one can be used when integration with Click is advisable (e.g. when existing sessions need to be altered through shaping and policing). The latter is optimized for application specific monitoring situations (e.g. the monitoring of video quality).

Both monitoring probes monitor typical QoS parameters such as packet loss ratio, delay and jitter in real-time. Furthermore, the Click based monitoring probes contain different bandwidth measurement algorithms to monitor the used resources. Depending on the envisaged use case, monitoring can occur on session, subscriber and node level.

Besides monitoring the actual network performance, the Click based monitoring probe can estimate packet loss, delay and jitter by monitoring intermediary in the network. This estimation can be interesting if user-experienced network performance must be measured in a network environment where the operator has no control on the end devices (e.g. a laptop in the home network). The carried out estimation has an accuracy between 95% and 100%.

4.2 TPSA evaluation platform

The evaluation platform to be used by TPSA in the FUSION demonstrator is shown schematically in Figure 6. In principle, this setup will enable modelling the components that represent users, the network and the data centre – all of them being relevant to the FUSION architecture.

The user and the core network sections will be based on a grid cluster containing 11 IBM BladeCenter servers (36 cores in total). The cluster operates in public IP address space and is fully accessible from outside, and thus can easily be interconnected with the facilities provided by other partners.

Apart from delivering services to FUSION-enabled clients, there will be a possibility to emulate background traffic generated using either artificial models available in software tools such as, e.g., D-ITG [DITG13] and Ostinato [OSTI13], or generated based on packet-level traces captured in a real network.

The core network, also hosted on the grid cluster infrastructure, is emulated using Vyatta [VYA13] software routers run under Ubuntu OS in VMware environment. Vyatta is an open source solution with the possibility to use dynamic routing, BGP, MPLS and also other features useful in WAN applications. Some of them that are relevant to FUSION experimentation are: measurements and monitoring for such parameters as bandwidth, injecting disruption in each node, and the ability for integration with network monitoring applications such as Nagios [NAG13].

The data centre is based on Cisco platform that currently includes the following main components:

- 4 x UCS-SP6-EV-B200 blade servers: CPU 8 x 2.00 GHz, RAM 512 GB, HDD: 4 x 600GB, SSD: 4 x 100GB, I/O Module: 2 x 8 Ext., 2 x 32 Int. 10Gb Ports
- 2 x Cisco ASR1002-X : 6 x GE, Dual P/S, 4GB DRAM
- 8 x Cisco Nexus 1000v switches, 2 x Cisco Nexus 5548 32 x 10GbE switches

In order to align with the majority of the experimentation scenarios, Cisco DC platform will be enabled to host an instance of OpenStack (apart from natively hosted VMware vSphere [VSPH13]). Being available to all partners of the consortium, this OpenStack instance will serve similar goals as those described in Section 4.4. However, its additional value will be to enable increased geographical span of FUSION services or their components thus providing a more realistic (distributed) environment for the evaluation of main FUSION concepts.

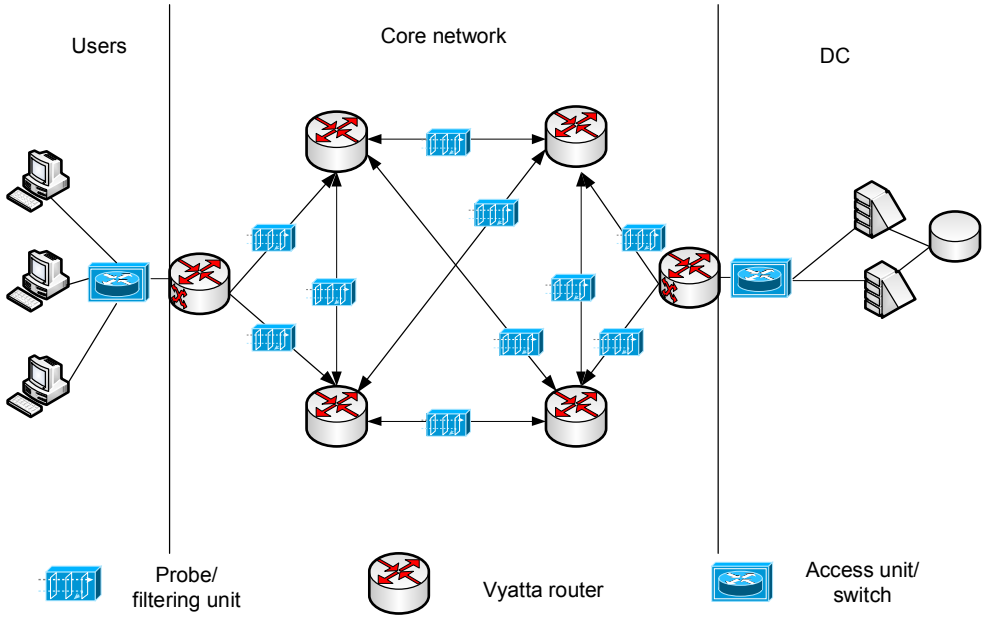

Figure 13 Schematic diagram of TPSA evaluation platform.

For the experimentation purposes, monitoring of the core network part will be based on Nagios. Actually, Nagios has already been integrated and used within the platform. We envisage that monitoring of cloud resources and service components running in OpenStack will be accomplished through the REST API of Ceilometer monitoring facility that has been developed within OpenStack project (Ceilometer closely resembles the Amazon Cloud Watch monitoring service).

4.3 Experiments in Public Clouds

The available partner testbeds are limited in geographical coverage which constrains the number and location of execution zones in the demonstrators. The project will also investigate the use of public clouds for wider-scale experiments. Virtual machines could be instantiated in resources in commercial services such Amazon EC2 or Rackspace [RACK13] which would allow tests to be conducted in and between widely distributed locations around the globe. One option would be to instantiate VMs on the EC2/Rackspace clouds in ten or more locations, each representing FUSION execution zones, and to make use of PlanetLab [PLAN13] to model users located at diverse locations. Such a test environment would enable experiments with FUSION server placement algorithms selecting between geographically distributed execution zones with more representative performance measurements being made. These options will be explored further during the second project year to design appropriate experiments that could benefit from running in such an environment.

4.4 OpenStack

As discussed in deliverable D3.1, we envision an overlay approach for implementing the key functionality of a FUSION execution zone on top of existing cloud or data centre management platforms. During the project, we plan to build upon and extend OpenStack in a number of ways:

- We will use an OpenStack environment for deploying many of our use-case driven application services;
- We will design and implement a prototype execution zone that can interface with the OpenStack APIs for automatically deploying FUSION services on top of an OpenStack environment;
- We will investigate how to extend OpenStack for supporting and enabling a number of key features:

- Efficiently deploying FUSION services using light-weight virtualization and deployment models;
- Exposing and exploiting heterogeneous hardware infrastructures and hardware accelerators;
- Advanced monitoring capabilities for enabling better QoS;
- Support for describing and orchestrating distributed service graphs across multiple execution zones.

Obviously, we will also keep track of the new features added by the OpenStack community related to these aspects listed above.

5. FUSION TEST CASES / SCENARIOS

This section provides an overview of specific test cases and scenarios that we envision to evaluate within the FUSION project. Each of these test cases cover a smaller aspect of the FUSION architecture and functionality, and can consist of a number of dedicated experiments, simulations, measurements, analyses, prototypes, PoCs, etc.

In this section, we will list an initial set of test cases and describe for each of these test cases a number of aspects:

- A high-level description of the test case
- The overall intent is of the test case
- An initial description of how we intent to implement and evaluate the test case
- How it may contribute to the FUSION architecture, design, objectives or demonstrator

We will categorize these test cases based on the key FUSION layer they operate in.

5.1 Orchestration layer scenarios

This section covers test cases related to registering, deploying and managing FUSION services across multiple execution zones in a FUSION domain.

5.1.1 Service registration

The first step for any service to be deployed and managed by FUSION is to register the service to a FUSION orchestration domain via a service manifest. After the registration procedure, the service should be immediately deployable. As a result of the service registration procedure, the orchestration domain could be immediately triggered in pre-deploying a number of instances based on provided service scaling policies.

In this project, we want to describe and evaluate a number of service registration scenarios in the demonstrator environment to evaluate the flexibility and agility of registering new services.

5.1.2 Evaluating the 4-step placement strategy

In Deliverable D3.1, we propose a four-step placement strategy, involving evaluator services, for evaluating the feasibility of a particular (set of) execution zone(s) for deploying new instances of an atomic or composite service.

We intend to implement and evaluate this placement strategy, both in the context of the final demonstrator as well as in a more dedicated setup.

5.1.3 Evaluating the effectiveness of evaluator services

In Deliverable D3.1, we claim the necessity of evaluator services for efficiently evaluating the feasibility of deploying new instances in particular execution zones, even in the case of complex service graphs or other related services.

We will evaluate the effectiveness and scalability of these evaluator services in the context of one or more service deployment scenarios that we will provide.

5.1.4 Composite service placement and deployment

In FUSION, we want to support placement and deployment of different types of composite services, either orchestration-driven, network-driven or application-driven. In the course of the project, we will evaluate these scenarios in a number of test scenarios as well as implement and evaluate particular scenarios in our demonstrator setups.

5.1.5 Service scaling

In FUSION, we will develop and evaluate service scaling algorithms for automatically scaling in and out the number of available service sessions based on predicted service load patterns. We will set up a series of dedicated experiments as well as a implement a limited scenario in the final demonstrator setup.

5.2 Networking layer scenarios

As regards the experimental testbed, networking layer scenarios will be focused on service routing optimisation for high-volume-fine-grained services. For scalability as well as practical reasons, static routing at the IP transport layer will be assumed. Optimisation will be achieved by dynamically mapping clients (requests) to execution zones under the assumption that service routing is controlled by the ISP and is based on up-to-date network monitoring information. This scenario will make use of topological diversity of execution zones that can host particular services. This diversity will be achieved via appropriate orchestration and then announced to the service routing layer. In general, two main options can be considered for the latter step, namely, announcing service availability through the service routing protocol (initiated from execution zones), and learning the location of execution zones by service forwarders through routing hints obtained by service forwarders from the orchestrator. It is expected that the properties of these two options will be studied in more detail through analytical and/or simulation means to indicate the final candidate(s) for development and use in the demonstrator.

5.3 Execution layer scenarios

This section covers test cases related to deploying and managing FUSION services in execution zones, as well as managing the execution zone services themselves.

5.3.1 Cross-platform service deployment

Due to the distributed nature of multiple execution zones within an orchestration domain, FUSION services should be able to be deployed automatically in heterogeneous environments, containing different hardware resources, data centre management platforms as well as the services having different platform requirements (e.g., Windows versus Linux environment). We will set up a number of specific test cases and evaluate the effectiveness regarding deployment efficiency and overhead.

5.3.2 Implementing a prototype EZ on top of OpenStack

As described in Deliverable D3.1, we plan to deploy a zone manager on top of an existing data centre management platform via a data centre specific abstraction layer. For the demonstrator, we will develop such abstraction layer on top of an OpenStack environment, using the OpenStack APIs as south-bound communication interfaces. This may also include extending or integrating particular components directly in OpenStack for extra support, for example regarding hardware acceleration, light-weight virtualization or monitoring.

5.3.3 Optimizing media applications in virtualized environments

Due to the inherently demanding nature of many of the selected use cases, efficiency and predictability are two key aspects. Within the scope of the project, we will evaluate the efficiency and predictability for media processing applications when deployed in virtualized and cloudified environments. These tests may consist of a series of both dedicated as well as integrated test setups.

5.3.4 Deploying a FUSION service using a light-weight containers

In FUSION, we want to be able to quickly deploy new instances on demand with small overhead both during deployment as well as execution time. One approach we want to evaluate is the use of light-weight virtualization and Linux containers for being able to quickly provision and deploy new

instances onto a new environment. This includes the overhead of fetching the software packages, preparing the environment, etc. We will set up a number of test cases and we will integrate this as well in the final demonstrator.

5.3.5 Enabling hardware acceleration for FUSION services

Many of the envisioned FUSION application services depend on specialized hardware for efficiently running the service. Examples include a 3D game service that heavily depends on a GPU, (interactive) streaming services that rely on scalable encoding or transcoding hardware for cost-efficient processing. We will evaluate the necessity, feasibility, and practical usage of hardware acceleration both from a theoretical/business point of view as well as integrate the usage of one or more hardware acceleration boards in the final demonstrator setup.

5.3.6 Late binding acceleration between co-located FUSION services

In FUSION, we want to be able to deploy new instances on demand with minimal communication overhead in case the services are co-located on the same executing platform. An approach we want to evaluate is the use of shared memory as a communication technology whereby the binding logic is implemented at application level or hosting platform level. We will set up a few test cases to demonstrate the alternatives and provide networking related metrics (BW, throughput, latency, etc.).

5.3.7 Monitoring of FUSION services and platforms

In FUSION, we will evaluate the necessity, feasibility and practical usage of hardware monitoring from a platform and application point of view as well as integrate the usage of one or more configurable monitor probes in the final demonstrator setup.

5.4 Service layer scenarios

This section covers test cases related to various service related aspects, such as service composition, monitoring, the session slots, service requests, etc.

5.4.1 Composite service description

In FUSION, we want to be able to support services with different types of service graphs (static or dynamic). We will evaluate the effectiveness of existing description languages like TOSCA for describing these service graphs, and we will also evaluate how this impact the overall handling of these types of services within the FUSION orchestration layer.

5.4.2 Session slots

We propose the concept of service session slots in FUSION for efficiently managing service instances and service sessions and efficiently routing individual service requests towards available instances. Within the scope of the project, we will implement the concept of service factories and session slots in most of our demonstrator services and exploit the concept at multiple layers throughput the demonstrator design. We also may perform additional theoretical studies regarding the overall flexibility and limitations of using session slots as key ingredient for efficient service deployment and service routing.

6. DEMONSTRATOR DESIGN

This section covers the motivation and initial plans for developing an integrated demonstrator, bringing together several core pieces of the FUSION architecture and key functions, applied to selected use case scenarios.

6.1 Goals and objectives

The overall goal of the demonstrator is to implement the FUSION core architecture and integrate several key FUSION functions in a demonstrator. This demonstrator will be used to validate the FUSION architecture for the key use cases, and show how the FUSION architecture can address their key requirements, including distributed deployment, automatic scaling, intelligent routing, efficient deployment on heterogeneous systems, etc. Vice versa, the demonstrator will also be used to validate a number of key principles early on, and to identify key bottlenecks and problems regarding some of the assumptions, thus providing valuable feedback early on when developing strategies and algorithms for efficient deployment of services and service request routing.

Consequently, the overall strategy we are taking with the demonstrator design is to start with a rather simple setup, focussing on first preparing an initial infrastructure as well as integrating a number of key service components, and add increasingly more functionality and dynamicity in an iterative agile fashion, rather than trying to design and build the full system at once from scratch. This agile development cycle will enables us to have a closer and quicker alignment between the key contributions from the various partners, and also allows us to more quickly find bottlenecks and problems in the overall design and implementation. On the downside, this iterative design process may also result in having to redesign portions of the solution in order to provide some new key feature or function. For example, to provide dynamic automated service deployment, some aspects of the basic infrastructure may have to be redesigned or reimplemented; to add service routing, the service components may need to be modified to support specific features and APIs, etc.

6.2 Initial demonstrator architecture and design

As mentioned before, the intention is to start from a simple initial design that only supports some basic features and later extend this in a structured manner. In this section, we provide details on related aspects, for example, how we envision the initial demonstrator architecture, what the key software and hardware components are that we plan to integrate, etc.

The overall goal of the initial demonstrator design is to bring together existing software components and services, agree on a number of common initial interfaces amongst the partners that provide these service components and deploy them on a common basic infrastructure. In this initial design, the services will be deployed manually in a specific environment; the services will be accessed via the standard Internet protocols (DNS, TCP/IP, etc.).

6.2.1 Basic infrastructure

In the first version of the demonstrator, a single FUSION zone will be emulated on the Virtual Wall environment of iMinds. The zone will be managed by OpenStack. Figure 14 demonstrates all components of the intended initial set-up.

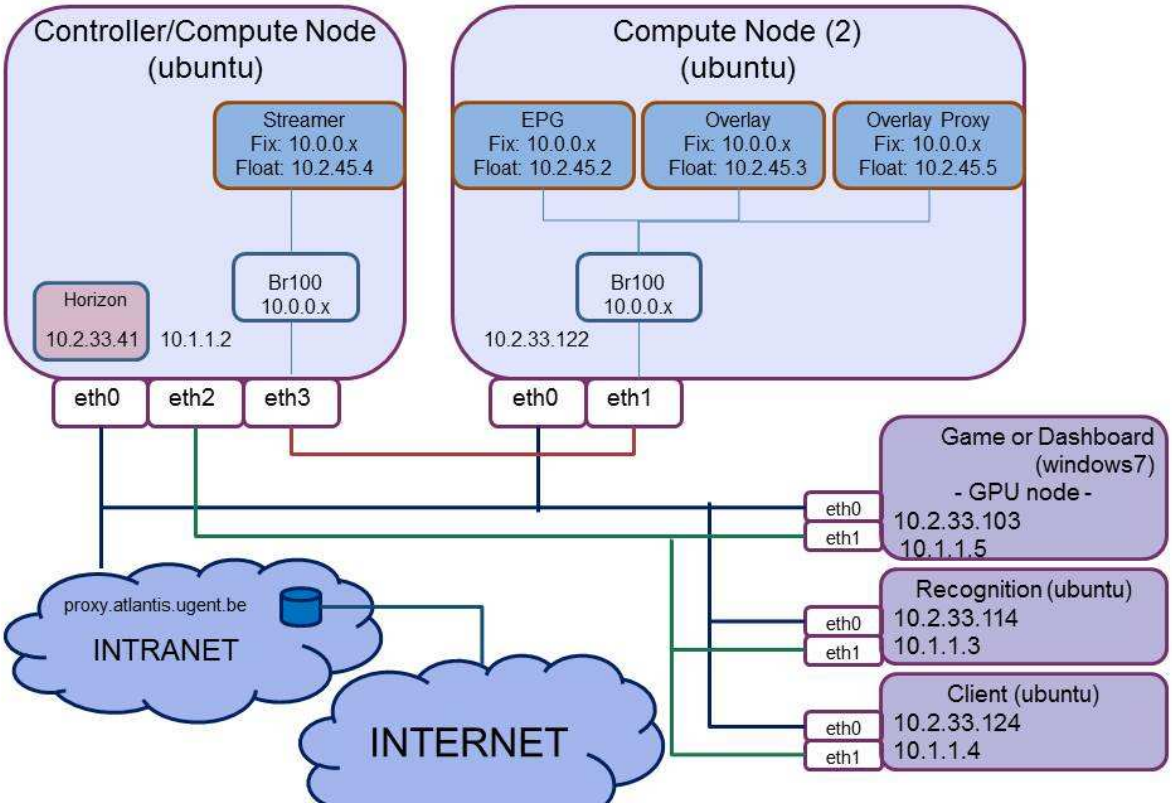

Figure 14: Deployment on the iMinds Virtual Wall

In the first version, all nodes will only be accessible over IPv4 after logging in to the intranet of iMinds via VPN. Public access via IPv6 will be added in a second version. The set-up emulates one Execution Zone in the FUSION architecture. The different nodes in the zone are managed at infrastructure level via OpenStack. At the platform level, we will study the integration with PaaS managers such as Docker and include Zone Orchestration functionality.

The Controller node hosts the OpenStack management software and also serves as Compute Node onto which Virtual Machines can be deployed. In Figure 14, VMs with the EPG (ALU), the Overlay Renderer (iMinds) and the Overlay Proxy (iMinds) have been deployed on the Compute Node, whereas two VMs (streamer (ALU) and game or dashboard (Spinor)) will be deployed on the node that is also the controller node. Owing to the lack of GPU pass-through support in OpenStack, Spinor's game and dashboard will not run on a Compute Node managed by OpenStack, but on an external node. The Recognition service (iMinds) is running on a 4th node. This emulates a 'distant' cloud. The Recognition Service will contain a database with objects to be recognized. Lastly, a client node will emulate a client service requesting a service from the FUSION service routing layer.

6.2.2 Service graph

The initial service graph is shown in Figure 15 below. We integrated a number of individual application services, each representing a different use case, into one combined service graph. This not only forces us to actively integrate the various services into a common infrastructure and use common communication APIs, it also serves as a first example of a non-trivial distributed service graph that we later can further extend and deploy in a distributed fashion across multiple execution zones. Obviously, it is also possible to define smaller service graphs that only contain a subset of services (e.g., only the EPG service with streamer, etc.).

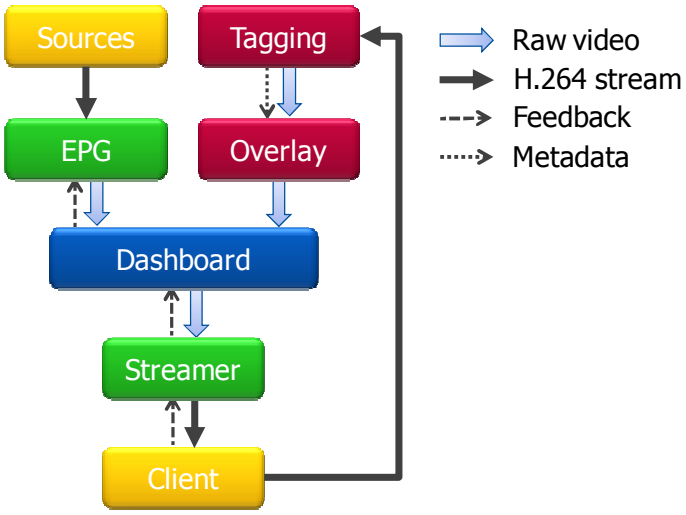

Figure 15: Schematic view of the integrated service graph

A central component in this service graph is the dashboard service, which has the role of aggregating a number of inputs coming from both external sources as well as other FUSION services, and building a fancy interactive 3D GUI around this. In this version, the output of the dashboard is a raw video stream, which is fed to a scalable real-time video encoder and streamer service. This streamer service takes in the raw video stream and encodes this in real-time as an H.264 video stream, which is then streamed towards a connected client. To have an interactive service, the client can also forward input events over a feedback channel towards the streamer, which will forward them towards the dashboard service. Depending on the interaction pattern, the service may decide to forward events or create its own events towards its connected services, like in case of the EPG service that is connected to the dashboard service. The last key service is the real-world tagging service, which consists of basically two key services, namely the tagging service and the overlaying service. The tagging service requires a live video input stream to perform the tagging, which in this case is also coming from the client application of the end user. We will elaborate on the function and implementation of the individual service instances and the communication protocols in more detail in the following sections.

It is important to note that in this service graph, the sources and the client (depicted in yellow boxes) are considered to be external to FUSION and thus are not running on top of FUSION infrastructure. Note also that in our demonstrator, each individual FUSION service component is deployed and running in its own environment, which is currently either a VM running on OpenStack or a native environment hosting the service.

6.2.3 Initial communication protocol

For the services to be able to interact, we agreed upon first versions of a few initial key communication protocols:

- Streaming raw video in between the service instances
- Feedback channel containing mouse and key input events for interactivity
- Metadata channel for exchanging metadata (e.g., the tagging locations)

We will discuss each of these protocols in more detail.

6.2.3.1 Raw video streaming protocol

For the initial demonstrator setup, we are planning to use raw RGBA32 video streaming to be able to quickly connect and stream raw video in between the service components of the various partners. Because of temporary nature of the protocol, we will leverage the YUV4MPEG2 format [YUV13], which we send over a raw TCP/IP connection. We will extend the format to also support RGBA32-based frames. In later iterations of the demonstrator, we will move towards more flexible and efficient communication protocols for exchanging video frames in between services (which may be collocated on the same physical machine and for which transcoding would result in too much overhead or latency issues).

6.2.3.2 Feedback protocol

For the feedback protocol, we will leverage portions of the RFB protocol specification [RT10], which is also used by VNC implementations for remote desktop sharing. We will only use the protocol messages for sending client input events (keyboard and mouse) towards the server application. We do not use the RFB protocol as primary protocol for raw video streaming, as this protocol is not suited for smooth raw video streaming, but is rather intended for incremental desktop GUI updates.

6.2.4 2D EPG service

For the first iteration of the demonstrator, we will leverage a basic 2D interactive EPG implementation that enables browsing through a number of dynamic or interactive video sources or static pictures, and that can easily be extended towards integrating the output from other FUSION services as well. This EPG service is developed in the Vampire framework [FV09], a media processing framework developed inside Bell Labs for quickly building media applications consisting of a number of reusable media components, each of which can be mapped onto a number of application threads. A snapshot of the basic EPG service is depicted in Figure 16.

Figure 16: Screenshot of a 2D EPG service prototype

For the initial setup, we will limit the EPG service to only consist of a fixed number of static input streams, together with a series of static images. The end user will be able to interact with the service either by pressing key strokes or mouse swipes to manually browse through the video sources. Alternatively, the user could also toggle the service to automatically scroll through all video sources. We will modify the basic implementation to support the basic raw video stream format as well as the RFB feedback protocol for handling all service communication as described above. We will also implement the service session slots concept into the initial implementation, allowing each EPG

service instance to handle a configurable amount of independent EPG sessions to be hosted in parallel. Using the underlying features of the Vampire framework, we will try to reuse all input video sources across all sessions using an internal Vampire multicast mechanism to significantly reduce the resource requirements. We will make the service instance fully parameterizable during instantiation, allowing to change the available parallel sessions, the output resolution, the endpoint port and the frame rate when a new instance is deployed. We will initially encapsulate this service into an OpenStack compliant virtual machine using an Ubuntu 12.04 64-bit server distribution as guest environment.

6.2.5 Video streamer service

The responsibility of the video streamer service is to transcode the raw video format coming from an internal FUSION service into an H.264 encoded video stream for streaming it towards external client applications. Vice versa, input events coming from the client application via a feedback channel will be forwarded to the FUSION service connected to the streamer service. The overall mechanism is shown in Figure 17.

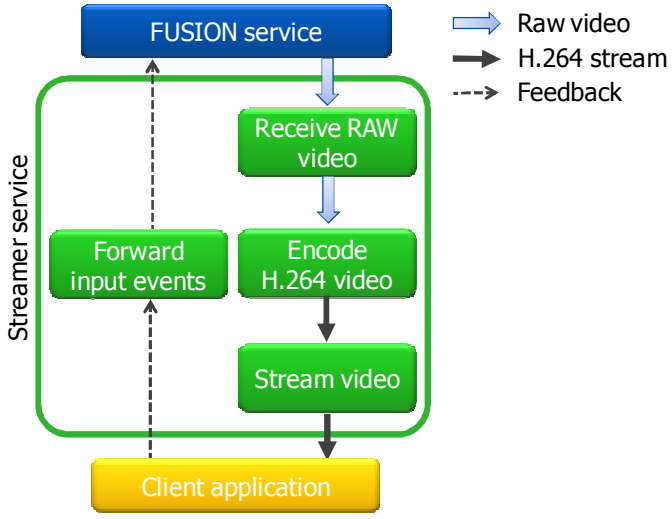

Figure 17: Schematic view of the streamer service

For implementing this service, we will use the same Vampire framework as for the 2D EPG service. In the first implementation, we use software-based video encoding using ffmpeg [FFM13] and/or X.264 [X13]. In the future, we envision integrating support for hardware accelerators for more efficiently encoding a large number of video streams in parallel, as real-time video encoding is very compute-intensive. We will also implement the service session mechanism, enabling a preconfigured maximum number of streaming sessions in parallel, each of which will automatically connect to a preconfigured FUSION service. We will also provide an interface that enables dynamically changing the default input service to which the streamer will automatically connect when a new streaming session is initiated.

We will make the streaming service instance fully parameterizable during instantiation as well as runtime, allowing to change the maximum number of parallel sessions, the available input FUSION service, the endpoint port, the video encoding format, frame rate and encoding quality when a new streaming instance or session is deployed. We will also wrap this service into an OpenStack compliant virtual machine using an Ubuntu 12.04 64-bit server distribution as guest environment.

6.2.6 Real-world tagging service

The tagging service will analyse video frames and detect known objects and/or faces. This service is realized by several components spread over two Execution Zones, as illustrated in Figure 18.

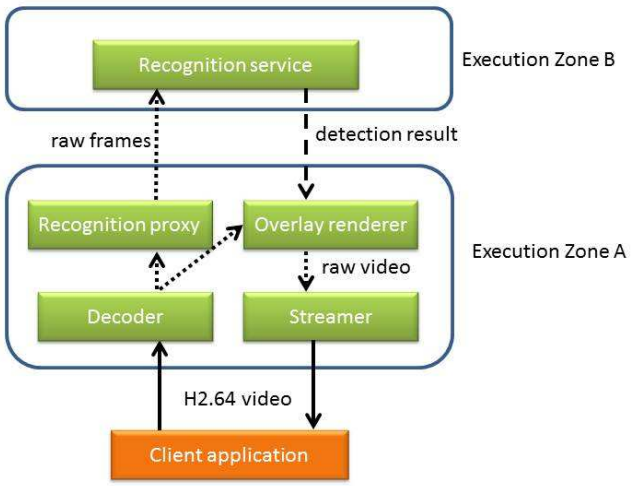

Figure 18: Recognition and tagging service components

The H.264 video feed from the client application is decoded to raw frames. These raw frames are forwarded to the Recognition Proxy and the Overlay Renderer. The Recognition Proxy forwards one out of every N frames to the Recognition Service running in another Execution Zone. The parameter N is dynamically adjusted to bandwidth and latency towards the Execution Zone, or by application-specific intelligence. For example, better recognition accuracy might be needed when this application is used for detection.

The recognition service returns the position and size of the rectangle spanning any detected object in the frame. The Overlay Renderer will draw these rectangles on the original raw videos. The rectangles are drawn on the same position for each new raw frame received from the decoder, until the Recognition Service provides updated input. Finally, the raw video is being decoded and streamed back to the client. The streamer component has been detailed in the previous section.

6.2.7 Dashboard service

This section covers the FUSION services built using the Shark 3D platform, which is currently being transformed for enabling FUSION service deployments.

6.2.7.1 Overview

The dashboard service will be built on the basis of the Shark 3D platform of Spinor, which is currently being transformed for enabling FUSION service deployments. Shark 3D based service components will offer connection requests, so that other software (e.g. other services or client applications) can connect to the dashboard service.

Technically such a particular service will be created visually using the Shark 3D authoring editor. In case advanced or special low-level features are required, Shark 3D offers Python and C++ APIs to implement custom functionality and integrate it both in the authoring editor and into the deployable run-time application.

6.2.8 Input and output channels

A main output channel is the render output. For example, the dashboard rendering is streamed to the client. Other output channels are also possible in principle transferring metadata, but usually not necessary.

A typical input channel contains input data from end-user input devices like mouse, keyboard, gamepad or other controllers. The user uses such input devices to interact with the 3D application. For example, it can be used to control the game or to interact with the dashboard.

6.2.9 Defining session slots

An extension of the Shark 3D platform we are creating for FUSION is the possibility to create a factory for session slots. In the editor such factories are called “producers”, see the editor screenshot below. This screenshot shows work-in-progress regarding the session slot implementation. Its main purpose is to define which functional components must be created for each session, for example a new unique 3D state and a rendering viewport which can stream the data to a TCP port. This is in contrast to shared data which must be created or loaded only once.

The following screenshot displays nodes of a very early stage prototype defining a factory for FUSION service session slots. Each of the nodes is implemented by one or more C++ components providing part of the overall functionality. The aggregation and configuration of these C++ components based on the visual arrangement as shown in the screenshot is implemented in Python.

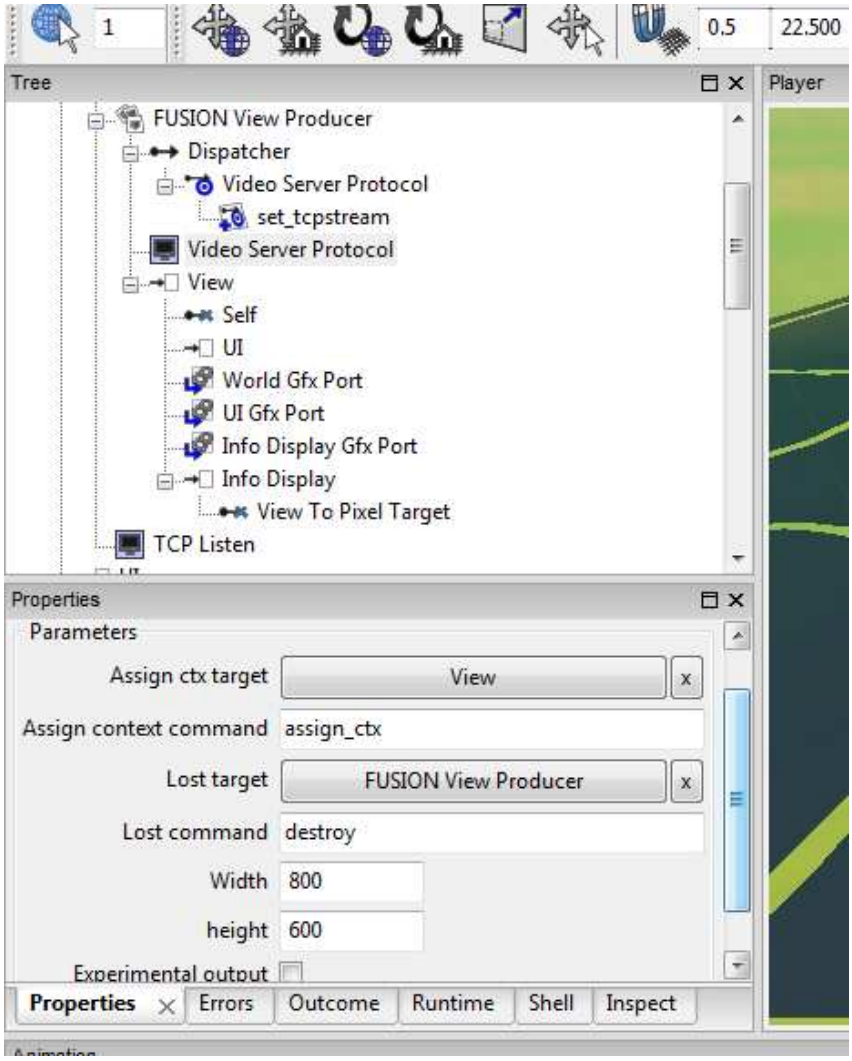

Figure 19: Screenshot of a dashboard service slot definition prototype

6.2.10 Game service

Similar to the dashboard service, also the game service will be based on the Shark 3D platform. A main output channel is the render output. For example, when using the Shark 3D service for implementing a game, the service streams the game render output to the client. Technically the game service is very similar to the dashboard service but with few differences, for example no need to have incoming video streams.

6.2.11 Client applications

We will develop a number of client applications for connecting and interacting with the FUSION demo services and for testing the communication protocols.

- First, we will develop a prototype application for testing the raw video format as well as the feedback channel for the input events. This application will serve two purposes. First, it will act as a reference implementation of both protocols, being able to both generate as well as receive raw video frames using the protocol. Second, we will also be able to use it for directly communicating and interacting with the FUSION services, without having to chain that service to the streamer service first.
- Second, we will also develop an initial client application that is able to decode and display an incoming encoded video stream coming from the streamer service, and also implements the feedback channel for interacting with the connected application.

6.2.12 Service routing and forwarding

A high-level scenario for this use case is depicted in Figure 20: Service routing/forwarding use case. Here, FUSION service routing configures service forwarding nodes taking into account the conditions (e.g., link utilisation or availability) in the IP network. Relevant metrics can be controlled in the experimental environment using Vyatta routers. Service requests are directed to different service instances located in different execution zones.

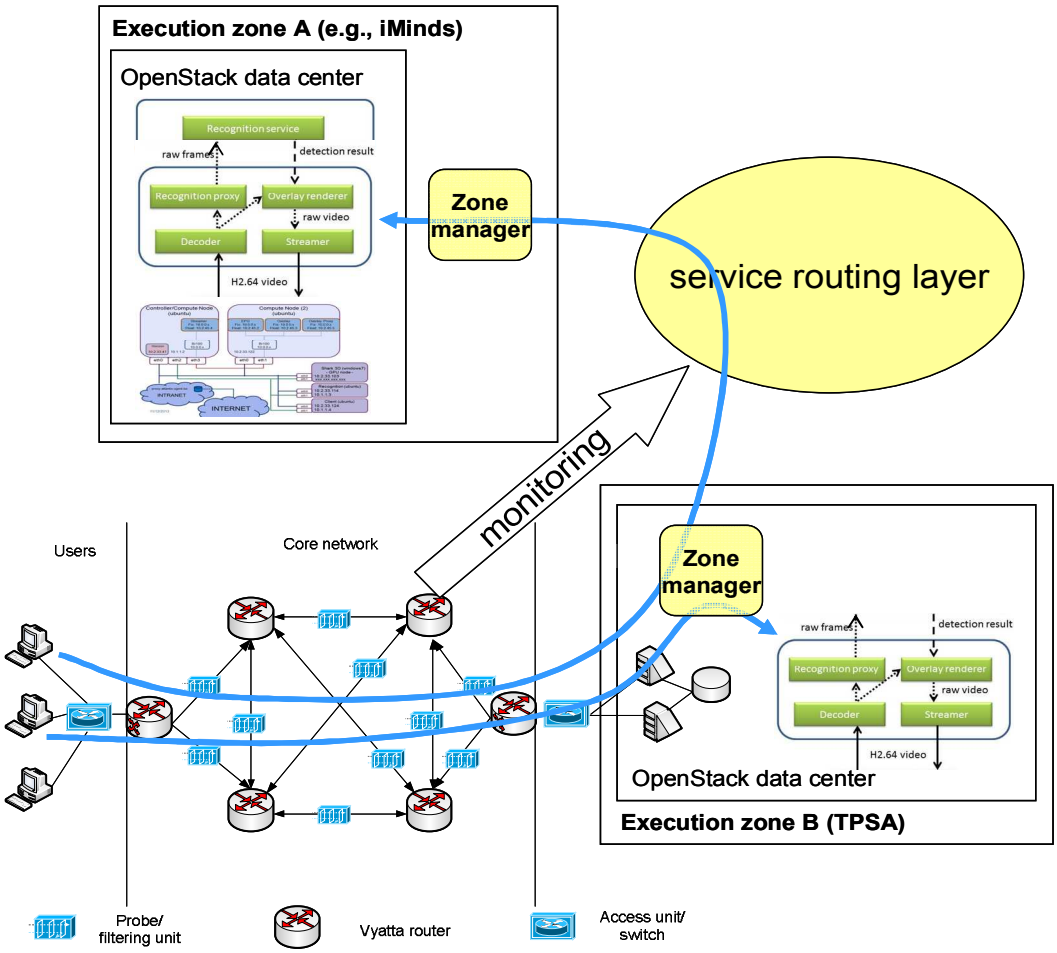

Figure 20: Service routing/forwarding use case.

The use case aims at demonstrating the capabilities of service routing function in FUSION architecture. One particular goal is to validate the idea of achieving service quality by dynamically mapping clients to execution zones under the assumption that the routing at the IP plane is static, the service routing is controlled by the ISP, and is based on up-to-date network monitoring information.

Execution zones in this scenario will be implemented using the infrastructure of both iMinds and TPSA. The service routing layer can be implemented in either the TPSA or iMinds testbed, but also PlanetLab will be considered.

7. CONCLUSIONS

This documents provides an initial description of the use case requirements as well as an initial description and modelling of a selected number of use cases that we will use for building a prototype and for evaluating the FUSION architecture. It also provides an overview of an initial set of test cases as well as a description of an initial demonstrator design that we are planning to build and that we will use as a starting point for implementing and evaluating key FUSION functionality in an agile manner.

In the first year of the project, we have been focussing on the use cases and particularly the requirements of these use cases and how FUSION should or must handle these requirements when deploying services in the FUSION architecture. We also opted for an agile prototyping and development cycle for building the final demonstrators, starting implementing and integrating new functionality in an iterative manner, as this will enable us to identify design or scalability bottlenecks early in the design process, allowing us to adapt the design or the algorithms where needed. As the FUSION architecture covers many domains and layers, we believe this approach is crucial for building an agile, scalable and flexible infrastructure for managing and accessing demanding interactive services in a distributed fashion.

In the second year of the project, we will continue working out the selected use cases in more detail. We will also start validating some of the requirements and designs by building targeted test cases. Third, we will make a plan for growing the initial demonstrator design into the final demonstrator design, and begin executing that plan, starting by integrating the various software components on the initial VirtualWall testbed environment and evaluating the key bottlenecks.

8. REFERENCES

[DITG13] Distributed Internet Traffic Generator, http://traffic.comics.unina.it/software/ITG/, 2013.

[FFM13] FFmpeg, http://www.ffmpeg.org/, 2013.

[FV09] Vandeputte, F., Vampire parallelization toolchain, IWT Vampire project, Deliverable D.B.4.3, 2009.

[KMCJ00] Kohler, E., Morris, R., Chen, B., Jannotti, J., & Kaashoek, M. F. (2000). The Click modular router. ACM Transactions on Computer Systems (TOCS), *18*(3), 263-297.

[NAG13] Nagios, http://www.nagios.org/, 2013.

[OSTI13] Ostinato, a packet/traffic generator and analyzer, http://code.google.com/p/ostinato/, 2013.

[PLAN13] PlanetLab, https://www.planet-lab.org/, 2013.

[RACK13] Rackspace, http://www.rackspace.com/, 2013.

[RT10] Richardson, T., The RFB Protocol, http://www.realvnc.com/docs/rfbproto.pdf, 2010.

[VSPH13] VMWare vSphere, http://www.vmware.com/products/vsphere/, 2013.

[VYA13] Vyatta, http://www.vyatta.org/, 2013.

[X13] X264, http://www.videolan.org/developers/x264.html, 2013.

[YUV13] YUV4MPEG2 file format, http://wiki.multimedia.cx/index.php?title=YUV4MPEG2, 2013.

[ZAB13] Zabbix, http://www.zabbix.org/, 2013.