

Interaction and Service Design Patterns in FLAME

Michael Boniface (*University of Southampton*) | Dirk Trossen (*InterDigital Europe*)

31/07/2019



Grant Agreement No.: 731677
Call: H2020-ICT-2016-2017
Topic: ICT-13-2016
Type of action: RIA

DISCLAIMER

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731677.

This document reflects only the authors' views and the Commission is not responsible for any use that may be made of the information it contains.



1 SERVICE DESIGN FOR INTERACTIVE SYSTEMS

The design of real-time interactive systems must consider how experience, performance and cost is influenced by the allocation of compute, storage and network resources necessary to process and delivery services and content. Here we outline the general challenges for service developers designing interactive systems and how the FLAME service delivery platform supports flexible management and control of highly distributed micro-services.

1.1 SERVICE DESIGN

The FLAME 5G service delivery platform offers the capability to flexibly place and connect services anywhere within the network from the far edge to the distant cloud. FLAME disrupts the prevalent mobile cloud architectures of the last 10 years by providing ways for mobile edge computing elements to be seamlessly used as part of the overall service execution environment. The combination of far edge, edge and distant clouds compute resources along with flexible and fast connectivity provides application developers with new deployment and scaling options, including entirely localised services that do not rely on distant public clouds.

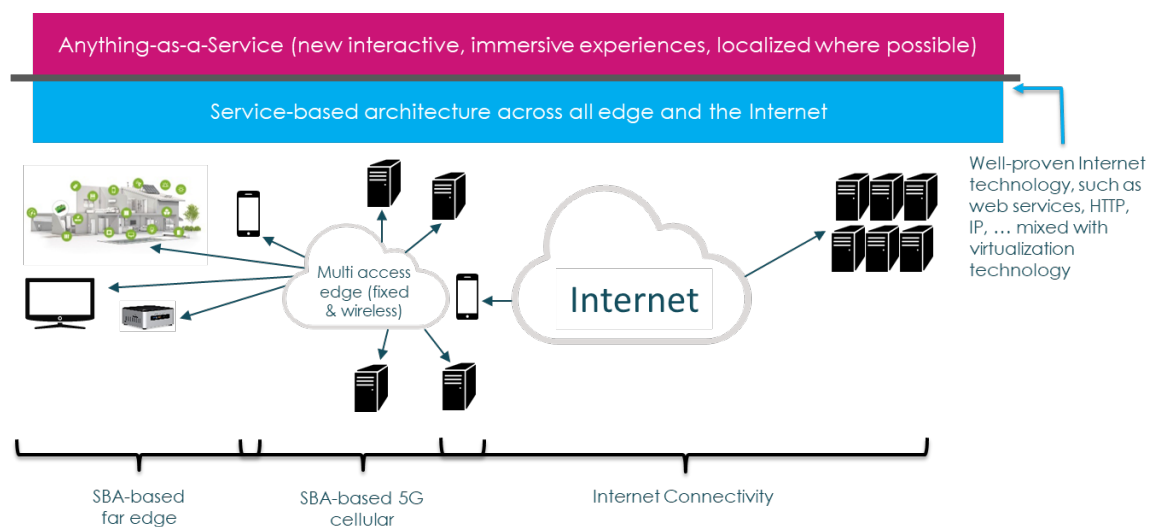


Figure 1: Evolution of service-based architectures from far edge to distant clouds

To benefit from this new distributed computing infrastructure a service developer must consider a wider variety of deployment options and how to manage deployment changes at runtime in response to demand. The computing infrastructure is now hierarchically scaled from small scale devices, to larger edge DCs, metro DCs and finally public clouds with seemingly infinity resources. Capacity constraints and costs of edge compute in comparison to the economies of scale achieved in large data centres are all factors to consider in service management decisions.

Now the key to success is to architect service function chains in ways that allow agility, resilience and scalability, taking into account the new opportunities opened up but also constraints imposed by this new environment. Service developers should strive to build services that are flexible and do not constraint business decisions. What this means is that a finer granularity in application/service decomposition is needed to allow for service functions to be intelligently

deployed on the most appropriate computing resource. To achieve this, functions within thick client applications on mobile devices and within monolithic services (typically deployed in the distant cloud) need to be refactored to create collections of micro-services. This allows for applications to offload to the edge or for services running in distant clouds to be placed closer to the end user.

Micro-service architectures are not new. Typically, micro-services are deployed independently, persist their own data/state, communicate through well-defined APIs, and may have entirely different technology stacks. There are many benefits of micro-services including increased agility, resilience, scalability, for highly distributed systems. However, these benefits come with their own challenges in terms of increased system complexity, testing, and de-centralised governance.

FLAME's distinct lifecycle management and control features deliver benefits to a set of micro-service patterns specifically designed to support interaction between users through the content they produce and consume. FLAME explicitly focuses on the relationship between interaction design patterns and service design patterns to allow for a closer alignment and orchestration of socially distributed services and a socially distributed virtual infrastructure.

1.2 PROGRAMMABLE MANAGEMENT AND CONTROL

FLAME's platform management and control features are summarised in the white paper "Enabling 5G in FLAME" [5G_PAPER] and in detail within the FLAME Architecture [D3.10].

Service developers describe a service function chain, infrastructure resources and control policies using a TOSCA resource specification template. The policies control the lifecycle state of service function endpoints (SFE) placed within a distributed collection of data centres. The state machine for the lifecycle of service function endpoints is shown in **Errore. L'origine riferimento non è stata trovata.** including placed (Image deployed on cluster), booted (SFE booted on cluster) and connected (SFE connected to network)

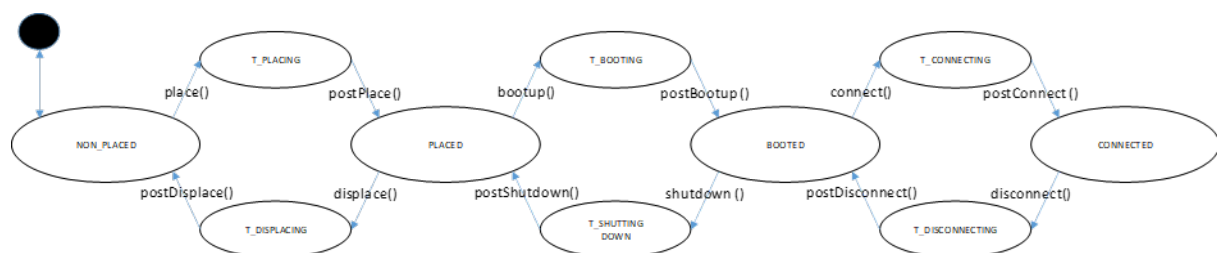
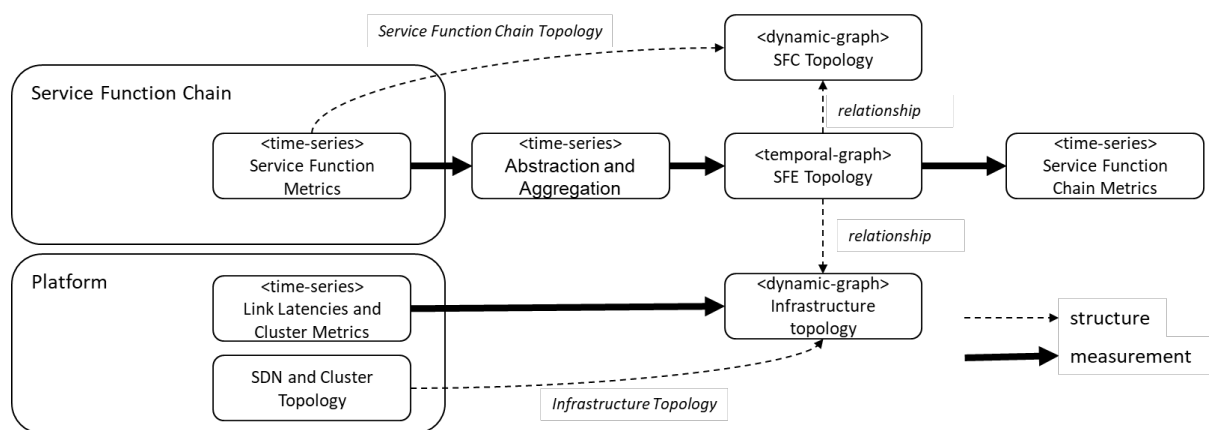


Figure 2: FLAME- Spanning from Distant to the Far Edge Cloud

The state of SFEs are changed at runtime using FLAME's using cross layer management and control which offers monitoring, measurement and analysis of service function chains considering both temporal and dynamic topological characteristics of system elements contributing to performance, as outlined in Figure 3. Service developers describe monitoring and alerts using a TOSCA alert specification. This creates network-aware service function chains that allow orchestration processes to understand how services respond to changes in workload and resourcing, and how such changes can be used to design, adapt and trial policies.

With the increasing geographical distribution of data centres including those at the edge or even far edge, policies now need to consider how to “Scale Geographically” allowing a service to scale in response to geographic location. This is the focus of FLAME and the motivation for SFE lifecycle states (PLACED, BOOTED, CONNECTED) controlled by policies and alert specifications.



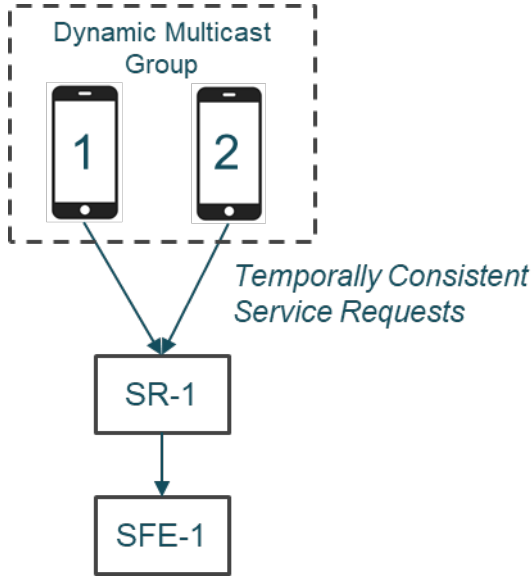
Page 5 of 13



2 SERVICE DESIGN PATTERNS

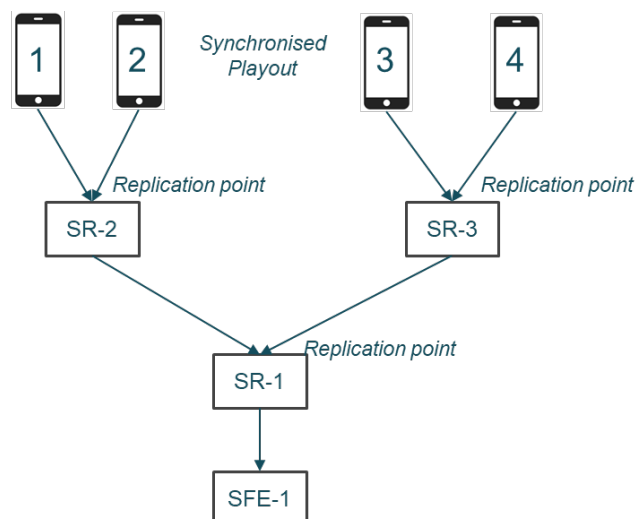
In this section we describe a series of service design patterns highlighting common interactions between service functions within a service function chain. The following list is not exhaustive but representative in that it outlines useful patterns that specifically benefit from the features of the FLAME platform.

2.1 OPPORTUNISTIC MULTICAST

Name	Opportunistics Multicast
Pattern	 <p>The diagram illustrates the Opportunistic Multicast pattern. At the top, a dashed box labeled 'Dynamic Multicast Group' contains two mobile devices, labeled '1' and '2'. Arrows from both devices point down to a box labeled 'SR-1'. A label 'Temporally Consistent Service Requests' points to these arrows. From 'SR-1', an arrow points down to a box labeled 'SFE-1'.</p>
Description	<p>Content is opportunistically delivered from a server to independently operating clients through multicast groups that are determined dynamically by requests for the same content at the same time. By tracking temporal consistency of service requests for HTTP-based segments within the network content is delivered in a multicast fashion, entirely based on interaction with content streams, for example, starting at a specific point or pressing pause. Such multicast delivery does not require explicit support, e.g., through join/leave operations, by the client but is done transparently by the FLAME platform. This is in contrast to existing IP multicast solution where such delivery relation is static over a longer period of time, with explicit join and leave operations performed at the receiving clients.</p> <p>In the above example client 1 and 2 are considered to be in the same multicast group if they request the same HTTP segment from SFE-1 at the same time, for example starting to watch the same video together on different devices. If client 1 and 2 pull at the same time the content is multicast at SR-1. If the clients are attached to different SRs, the multicast delivery will be achieved only in the</p>

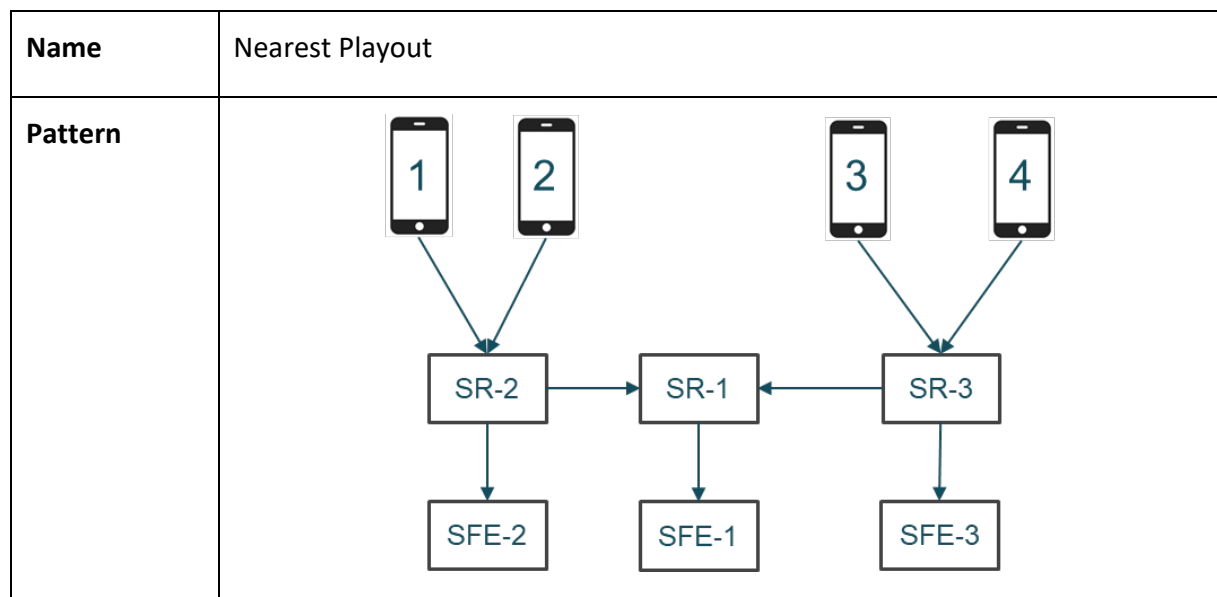
	joint delivery topology from the SFE-1 to the clients, while unicast delivery is used for the rest of the delivery.
Service Provider Benefit	<p>Network cost reduction due to reduced network usage as video is (partially) multicast in segments of the network</p> <p>Network cost reduction due to video being possibly front loaded to the edge</p> <p>Service cost reduction due to reduced server usage as http requests are suppressed when multicast occurs</p> <p>Scale of supported end users when using contented access link such as WiFi.</p>
Experience Benefit	<p>Reduction in start-up time for playout as segments are served from the potentially closer edge playout point.</p> <p>Improved quality of experience because of using less network resources due to multicast delivery, therefore potentially reducing overall contention in the network.</p>

2.2 SYNCHRONISED PLAYOUT

Name	Synchronised Playout
Pattern	 <p>The diagram illustrates the Synchronised Playout pattern. At the top, four mobile devices are shown, numbered 1, 2, 3, and 4. Arrows from devices 1 and 2 point to a box labeled 'SR-2'. An arrow from device 3 points to a box labeled 'SR-3'. An arrow from device 4 points to 'SR-3'. Arrows from 'SR-2' and 'SR-3' point to a central box labeled 'SR-1'. Below 'SR-1', an arrow points to a box labeled 'SFE-1'. The text 'Synchronised Playout' is written above the devices. The text 'Replication point' is written next to the arrows pointing to 'SR-2', 'SR-3', and 'SR-1'.</p>
Description	This pattern extends the scenario in 2.1 by streaming a single video (e.g. 4K 360-degree video) to multiple clients from one synchronised playout point avoiding the proliferation of unicast streams, using an HTTP-based chunk request approach such as MPEG DASH or HLS. While the unicast semantic is

	<p>preserved, the delivery in the network is based on multicast replication on segments of the network. Synchronization here can happen randomly (e.g., the clients happen to request the same content at roughly the same time), or it can be enforced through a dedicated synchronization client.</p> <p>In the above example three SRs provide replication points for relaying segments. Synchronisation is achieved when the same HTTP-based segments are available for display at clients at the exact same playout time.</p>
Service Provider Benefit	<p>Network cost reduction due to reduced network usage as video is (partially) multicast in segments of the network</p> <p>Network cost reduction due to video being front loaded to the edge</p> <p>Service cost reduction due to reduced server usage as http request suppression when multicast occurs</p> <p>Scale of supported end users when using contented access link such as WiFi.</p>
Experience Benefit	<p>Reduction in start-up time for playout as segments are served from the potentially closer edge playout point.</p> <p>Improved quality of experience because of using less network resources due to multicast delivery, therefore potentially reducing overall contention in the network.</p>

2.3 NEAREST PLAYOUT



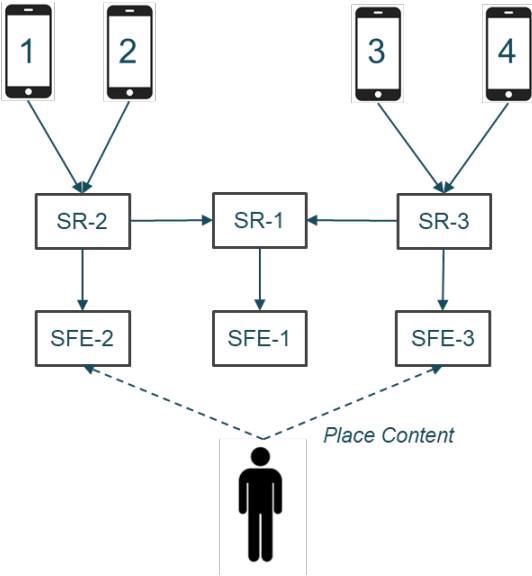
Description	<p>Serving requests from the closest service function endpoint where proximity is defined as the number of hops from the client to the endpoint</p> <p>The example shows three service function endpoints of the same type. Client 1 and 2 will be served by SFE-2 whilst client 3 & 4 will be served by SFE-3. No requests will be served by SFE-1</p>
Service Provider Benefits	<p>Network cost reduction due to lower network usage as service/content is front loaded to the edge</p> <p>Overall costs may increase if the compute and storage usage for placing SFE on edge DC's in a connected state exceeds the cost of reduced network usage</p>
Experience Benefits	<p>Reduction in start-up time for playout</p> <p>Reaction time to latency changes</p> <p>Latency stays comparable even when client moves, e.g., client 1 moves to SR3, now being served from SFE3 instead of SFE1</p> <p>Reachability of content from anywhere in the network</p>
Other	Assumption the closest SFE gives the best performance

2.4 PROXY CACHE PLAYOUT

Name	Proxy Cache Payout
Pattern	<pre> graph TD C1[1] --> SR2[SR-2] C2[2] --> SR2 C3[3] --> SR3[SR-3] C4[4] --> SR3 SR2 --> SR1[SR-1] SR3 --> SR1 SR2 --> SFE2P[SFE-2 Proxy] SR1 --> SFE1M[SFE-1 Master] SR3 --> SFE2P2[SFE-2 Proxy] </pre>
Description	Dynamically caching content close to the demand according to a specified caching policy.

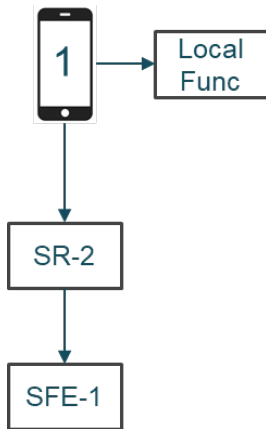
	The example shows a master storage service SFE-1 serving content to four clients. Client 1 and 2 are served by proxy service SFE-2 connected to SR-2 whilst client 3 and 4 are served by proxy service SFE-2 connected to SR-3. If client 1 accesses content the proxy service downloads the content from the master and caches it automatically. If client 2 then requests the same content it is served directly from the cache rather than the master.
Service Provider Benefits	<p>Network cost reduction due to lower network usage as service/content is automatically front loaded to the edge.</p> <p>No content preloading required as caches are dynamically updated depending on demand</p>
User Benefits	<p>Reduction in start-up time for repeated playout</p> <p>Reaction time to latency changes</p>

2.5 CONTENT PLACEMENT

Name	Content Placement
Pattern	
Description	<p>Place content at a specific location within the network based on a prediction where demand for that content is likely to be needed.</p> <p>The example shows a service provider placing content at two service function endpoints (SFE-2 and SFE3) that are closer to clients than SFE-1</p>

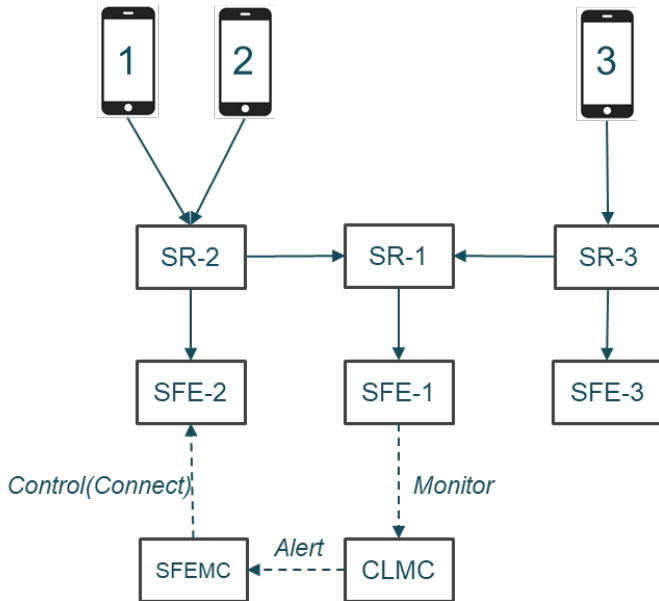
Service Provider Benefits	<p>Network cost reduction due to lower network usage as service/content is front loaded to the edge.</p> <p>Deployment of the content based on predicted demand</p> <p>Content placement costs can be decreased through a synchronized HTTP-based operation, resulting in multicast delivery of placed content</p>
User Benefits	Reduction in start-up time for playout
Other	Future FLAME platform version will provide the ability to redirect negative content requests, i.e., for content not being found locally, to other SFE instances for potentially positive result (partial content placement)

2.6 APPLICATION FUNCTION OFFLOADING

Name	Application Function Offloading
Pattern	 <pre> graph TD Device[1] --> LocalFunc[Local Func] Device --> SR2[SR-2] SR2 --> SFE1[SFE-1] </pre>
Description	<p>Offload local device-centric functions to network computing resources.</p> <p>The example shows a local function on a device (e.g. video processing) offloaded to SFE-1 running on a compute resource (i.e. edge or cloud DC).</p>
Service Provider Benefits	Assist end user with richer (in terms of functionality) SFE instance
User Benefits	Increased battery life as processing can be offloaded to another compute resource

	Opportunity to utilize better device capabilities within the network (e.g., transferring presentation to alternative displays)
--	--

2.7 SCALE GEOGRAPHICALLY

Name	Scale Geographically
Pattern	
Description	<p>Change the lifecycle state of a new service function endpoint dynamically within the network in response to changes in performance or usage.</p> <p>The example shows client 1 and 2 initially being served by SFE1 (as their closest instance), while an alert (e.g., based on measured latency at client 1 and/or 2) triggers the lifecycle state change at SFE2 to CONNECTED, now serving the content from a closer SFE instance to client 1 and 2.</p>
Service Provider Benefits	Distribute compute resources more equally across available edge compute cluster, serving more end users
User Benefits	Reduced latency by either serving from nearer SFE instance and/or avoiding overloaded original SFE instance

REFERENCES

- [5G_PAPER] "Enabling 5G with FLAME", 2019 <https://ict-flame.eu/wp-content/uploads/sites/3/2019/06/Enable-5G-with-FLAME-Whitepaper-v1.1.pdf>
- [D3.10] "D3.10: FLAME Platform Architecture and Infrastructure Specification V2", FLAME deliverable D3.10, 2018, available at <https://www.ict-flame.eu/download/d3-10-flame-platform-architecture-and-infrastructure-specification/?wpdmdl=1515&masterkey=5c790eda75dbb>