Enabling micro/internal services with USP

BBF Wiki

Exported on 03/04/2020

Table of Contents

1 Contributors

This pages describes a proposal from prplFoundation, supported by the following members.

- ADTRAN
- British Telecom
- Deutsche Telekom
- Domos
- Intel
- IOPSYS
- Kaon
- Orange
- Sagemcom
- SAM
- SoftAtHome
- Verizon
- Vodafone

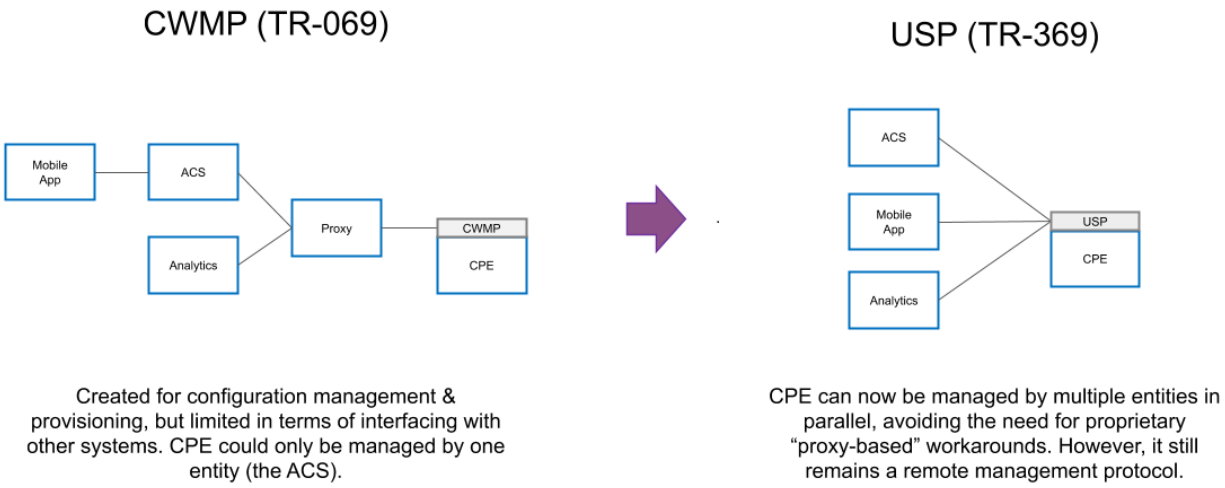
2 Context

The CPE WAN Management Protocol (CWMP/TR-069) offered a standard mechanism for service providers to manage their devices remotely, ranging from basic device identification, firmware management or configuration of individual services, but mostly focused on provisioning and activation.

Having a well-established protocol, with a rich set of data-models, it rapidly started to be adopted for other purposes, including analytics, telemetry, troubleshooting and even in some cases Mobile Applications. However, despite the increasing number of use-cases, the solution was not scalable as all communication has to be managed by a central unit the ACS, or alternatively proxied by an intermediate entity often provided by proprietary means.

In order to cope with this problem, USP/TR-369 (the natural successor of CWMP/TR-069) hosts the ability for the CPE to be remotely managed by multiple controllers, whilst also introducing other important optimisation and security mechanisms, such as ACLs. All these additions, resulted into the creation of a solid foundation, named the “User Services Platform”, which enables third-party software companies to build services on top of CPEs managed by ISPs.

Popular use-cases include Wi-Fi Cloud Controllers, enhanced device type recognition capabilities, Smart-Home, Parental Controls and Security related propositions, i.e. a set of services, which typically require significant computing power, seldom available on constrained devices such as the CPE.



Created for configuration management & provisioning, but limited in terms of interfacing with other systems. CPE could only be managed by one entity (the ACS).

CPE can now be managed by multiple entities in parallel, avoiding the need for proprietary “proxy-based” workarounds. However, it still remains a remote management protocol.

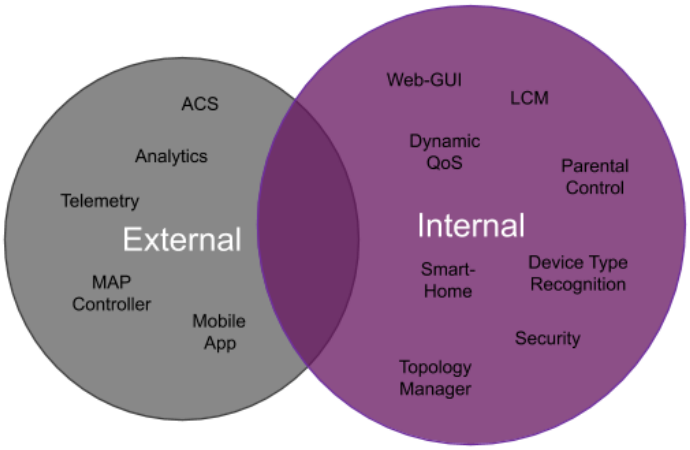
3 Problem Statement

Nevertheless, even though USP enables applications to control the CPE remotely and offload demanding tasks to external devices (e.g. Cloud/Edge server), many of these services typically also require certain operations to be performed locally (by the CPE itself). Some examples include steering Wi-Fi clients to different access-points, deep packet inspection and manipulation of traffic hardware acceleration. This resembles a significant subset of operations, which are usually modelled as proprietary APIs (i.e. seldom available through standard means), may require near-real time computing, and are usually not protected by any sort of access-controls.

In fact, as-of-today this can be perceived as one of the biggest challenges service providers and application developers face whilst collaborating with each other in an attempt to integrate services on different CPE models, software stacks and hardware manufacturers. A problem, that rapidly sparked the interest in containerized technologies, such as Linux Containers (LXC), as well as other initiatives such as the HL-API (aimed towards internal service development) hosted by prplFoundation, a different problem space addressed by USP (external service development).

Major differences consist on the nature on how these two distinct kind of services interface with the CPE. Remote USP services take advantage of TR-181's data-centric approach. Requests are wrapped in MTPs such as MQTT, WebSockets and resembles changes to parameters, which can be grouped into transactions that lock the system until completion. This enables messages to be transported securely over the Internet, but introduce a significant overhead that makes it complex for the device to cope with race/concurrent requests. Internal services on the other hand follow a distinct approach. Requests are mostly comprised of atomic operations, such as invoking commands and publishing/listening events (asynchronous model), which results into a very lightweight and resource efficient model, properties important for latency sensitive services.

In spite of the different mechanisms, internal and external services share an increasing common scope, the divergences lie on how they interface with the system. Clear examples, include the user interfaces (MobileApp and Web-GUI), which offer similar functionality.



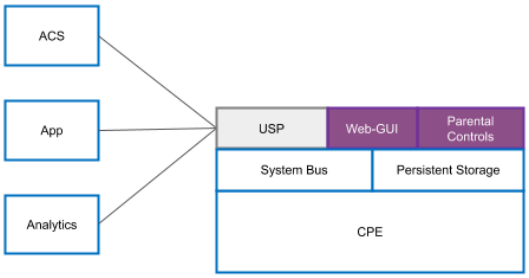
Significant scope overlap (e.g. Mobile App vs. Web-GUI).

The former may leverage on a standard remote management protocol such as USP, whereas the latter is likely to interface with vendor-specific components such as the system bus (e.g. ubus on OpenWrt, ccsp/rbus on RDK) or the local persistent storage manager (e.g. uci on OpenWrt, psm on RDK). As a consequence, application developers are forced to fork their implementations to match the target platforms, as opposed to focusing on the development of new features.

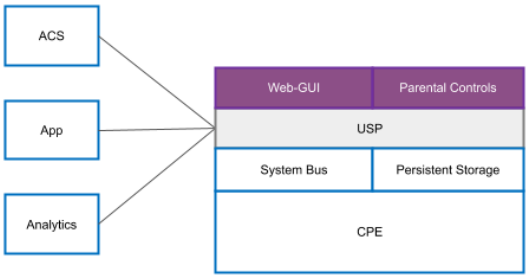
4 Proposal

In aid of this, prplFoundation on behalf of its members considers that there is a good opportunity for USP to extend its current scope beyond basic remote management, and tackle both external and internal service development.

A successful expansion, would enable application developers to focus on the creation of new portable services (as opposed to re-integrating), whilst CPE manufacturers could benefit from a simpler architectures (by removing unnecessary redundant abstraction layers), ultimately resulting into an overall better time-to-market.



Portable external services, but non-portable internal services.

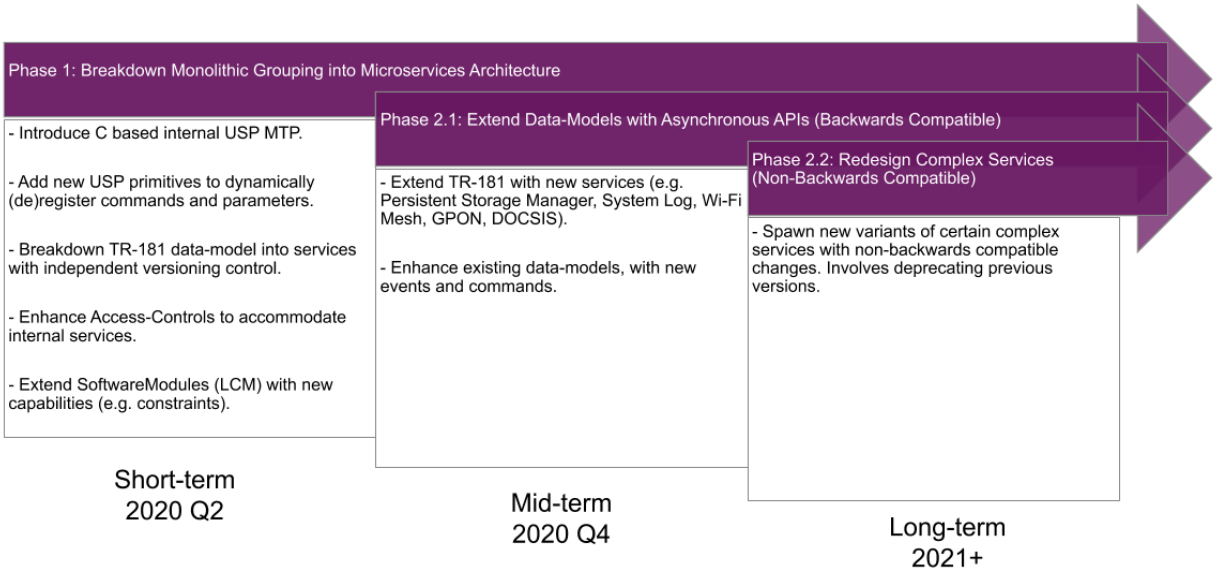


Portable external and internal services.

The following table, describes sample existing gaps in USP/TR-181, and includes proposals for addressing the missing scope.

	External (USP)	Internal	Proposal
Performance	Non-real time.	Latency sensitive. Current MTPs introduce too much overhead, making it challenging to implement real-time Wi-Fi steering logic, or handle deep traffic inspection.	Introduce an internal MTP (e.g. C interface), along with improved ACLs.
Role	Consumer. Only triggers operations, does not extend the scope of functionality.	Consumer / Provider. May also provide functionality.	Add new primitives to dynamically (de)register operations and parameters.
Scope	Typically a subset of internal APIs. Mostly configuration changes, and status queries.	Require extra-set of functionality. In addition to configuration changes, also includes triggering actions.	Extend TR-181 (e.g. Persistent Storage, Real-Time Logging, Advanced Packet Inspection, Container Management).
Paradigm	Data-centric synchronous transactions (e.g. “Get” and “Set”).	Typically atomic asynchronous APIs, heavily dependant on events, and functions calls.	Extend TR-181, with new events and commands. Consider spawning new reshaped versions of certain parts of the data-model.
Modularity	Typically perceives the device as a single-block of functionality implemented by a single entity, the Firmware owner.	Interfaces with different components comprised of independent life-cycles, and potentially implemented by different companies. May perform dependency checks based on component versions.	Breakdown TR-181 data-model, into smaller independent services. Assign independent versions per module.

The previously identified action-points encompasses a long-term vision. Encourage to phase activities into short, mid and long-term strategies.


Phase 1: Breakdown Monolithic Grouping into Microservices Architecture
- Introduce C based internal USP MTP.
- Add new USP primitives to dynamically (de)register commands and parameters.
- Breakdown TR-181 data-model into services with independent versioning control.
- Enhance Access-Controls to accommodate internal services.
- Extend SoftwareModules (LCM) with new capabilities (e.g. constraints).
Phase 2.1: Extend Data-Models with Asynchronous APIs (Backwards Compatible)
- Extend TR-181 with new services (e.g. Persistent Storage Manager, System Log, Wi-Fi Mesh, GPON, DOCSIS).
- Enhance existing data-models, with new events and commands.
Phase 2.2: Redesign Complex Services (Non-Backwards Compatible)
- Spawn new variants of certain complex services with non-backwards compatible changes. Involves deprecating previous versions.
Short-term
2020 Q2
Mid-term
2020 Q4
Long-term
2021+

5 Use-Cases

5.1 Sample 1 (Internal MTP)

Access to USP (for internal services) should be provided by an interface that introduces little overhead and/or latency, such as a C shared library accessible via Linux socket, as opposed to a network socket. It should include all primitives supported by USP.

```
# Session Handling
## Enables service to authenticate and establish a session.
int usp_connect(struct context* ctx,
                  struct credentials* auth
                  const char* socket);

## Closes session.
int usp_disconnect(struct context* ctx);

# Properties
## Retrieve parameter.
int usp_get(struct context* ctx,
            struct parameter* param);

## Modify a parameter.
int usp_set(struct context* ctx,
            struct parameter* param);

# Commands
## Invoke command.
int usp_operate(struct context*,
                const char* object,
                const char* command,
                struct data* arguments,
                struct data* response);

# Events
## Publish an event.
int usp_publish(struct context* ctx,
                const char* object,
                const char* event,
                const char* reason,
                const char data* body);

## Listen to event.
int usp_subscribe(struct context* ctx,
                  const char* object,
                  const char* event);
```

5.2 Sample 2 (New USP Primitives)

Adding or removing TR-181 parameters and commands is usually achieved by pushing new Firmware updates, which makes it hard to take advantage of dynamic life-cycle management capabilities as described in TR-157 SoftwareModules. It would be beneficial to include new USP primitives to enable services/applications to (de)register parameters and commands in a dynamic fashion.

```
# Parameters
## Register new parameter.
int usp_register(struct context* ctx,
                 struct parameter* param,
                 void* cb);

## Deregister parameter.
int usp_deregister(struct context* ctx,
                   struct parameter* param);

# Commands
## Register a new command.
int usp_register(struct context* ctx,
                 struct command* cmd
                 void* cb);

## Deregister a command.
int usp_deregister(struct context* ctx,
                   struct command* cmd);
```

5.3 Sample 3 (Breakdown TR-181 into Microservices)

Most service providers do not require TR-181 as a whole, but rather a subset of it. This is becomes easier to manage if data-models are broken-down into services/packages (profiles). In addition, it also enables services, to perform dependency checks and enable/disable certain features based on their availability. This means that it would also be equally important to assign a version number (based on semantic versioning) to each service accordingly.

DHCPv4 Client

```
Device.DHCPv4.Client.Version = "1.0.0"
Device.DHCPv4.Client.{i}.
Device.DHCPv4.Client.{i}.SentOption.{i}.
Device.DHCPv4.Client.{i}.ReqOption.{i}.
```

DHCPv4 Server

```
Device.DHCPv4.Server.Version = "1.0.2"
Device.DHCPv4.Server.Pool.{i}.
Device.DHCPv4.Server.Pool.{i}.StaticAddress.{i}.
Device.DHCPv4.Server.Pool.{i}.Option.{i}.
Device.DHCPv4.Server.Pool.{i}.Client.{i}.
Device.DHCPv4.Server.Pool.{i}.Client.{i}.IPv4Address.{i}.
Device.DHCPv4.Server.Pool.{i}.Client.{i}.Option.{i}.
```

5.4 Sample 4 (Enhanced ACLs)

USP introduces, ACLs and roles with authentication based on credentials and certificates. However, in addition to these, internal services may authenticate/identify themselves based on the running Linux user. This means that a new authentication model should be considered..

Furthermore, privileges of controllers, may depend not just on the role itself, but also on the "TrustZone" (e.g. "WAN", "LAN", "Internal"). Some agents may apply harder constraints for incoming WAN requests, whilst relax them for LAN.

5.5 Sample 5 (LCM - Extend SoftwareModules)

The "Device.SoftwareModules." data-models, enables services providers to deploy services independently from the Firmware release. However, it does not provide utilities for enforcing constraints. Proposal to consider new parameters, such as:

```
Device.SoftwareModules.ExecEnv.{i}.Contraints.CPUPriority
Device.SoftwareModules.ExecEnv.{i}.Contraints.CPUCap
Device.SoftwareModules.ExecEnv.{i}.Contraints.MemoryCap
Device.SoftwareModules.ExecEnv.{i}.Contraints.RAMCap
```