Deliverable D8.11
Service Provider Architecture Pilot Follow-up

Abstract
This amendment describes the outcomes of the Service Provider Architecture pilot. The first part of the document focuses on the implementation of the first phase of the pilot: porting the E-Line service to the Service Provider. The second part of the document provides the results of the pilot in terms of performance testing, functionality and acceptance testing and user feedback. The final section concludes the findings of the pilot.
# Table of Contents

Executive Summary .......................... 1

1 Introduction .............................. 2

2 Phase 1: Porting CCS to SPA: The E-Line Project .......................... 4
   2.1 E-Line Implementation in SPA .......................... 4
      2.1.1 OpenNSA Overview .......................... 5
      2.1.2 Additional QoS Parameters .......................... 6
      2.1.3 Customer-Facing and Resource-Facing Services Specification .......................... 6
      2.1.4 Service Inventory Implementation .......................... 9
      2.1.5 Resource Inventory Implementation .......................... 11
      2.1.6 Interfacing with OpenNSA .......................... 14
   2.2 Other Preparations .......................... 15
   2.3 Phase 1 Lessons Learned .......................... 16

3 Phase 2: User Experience and Feedback .......................... 18
   3.1 SPA Integration and Usability Testing Results .......................... 18
   3.2 Performance Testing Results .......................... 19
      3.2.1 API Delays .......................... 20
      3.2.2 SSP Loading Measurements .......................... 22
      3.2.3 jMeter Performance Testing Results .......................... 24
   3.3 OSS/BSS Metrics .......................... 28
   3.4 User Feedback Analysis .......................... 29
   3.5 Phase 2 Lessons Learned .......................... 30

4 Overall Pilot Results Analysis and Future Development Remarks .......................... 32
   4.1 Scope of Effort .......................... 32
   4.2 Issues with Information Synchronisation .......................... 33
   4.3 Migrating Services to SPA .......................... 34
   4.4 Adoption in the Community .......................... 35
   4.5 Mixed Environments .......................... 35

5 Conclusion .............................. 36

Appendix A .............................. 38
   A.1 Service Specifications json Example .......................... 38
      A.1.1 OpenNSA P2P circuit RFS .......................... 42
Table of Figures

Figure 2.1: An example network domain with STPs and SDPs (P2P Circuit from STP2 to STP10 – STP4-STP6 | STP8+STP9) 5
Figure 2.2: E-Line CFS parameters as provided via the SSP 7
Figure 2.3: OpenNSA P2P circuit RFS parameters as provided via the SSP 7
Figure 2.4: OTRS:ITSM Service catalogue listing 8
Figure 2.5: SLA and configuration item references in the service catalogue 8
Figure 2.6: GÉANT E-Line CFS specification in the service catalogue 8
Figure 2.7: CFS and RFS relationship in service catalogue 9
Figure 2.8: Service inventory TMF SID compliant model 10
Figure 2.9: Resource inventory data model – short-version schema presenting main components 12
Figure 2.10: OpenNSA-based E-Line testbed 14
Figure 3.1: Comparison of delays in case of using the TMF API to get complete information about one item 21
Figure 3.2: Delay measurements for information retrieval when using OTRS 22
Figure 3.3: Self-Service Portal services page 23
Figure 3.4: SSP services page loading measurements for the case of 4 offered services, 15 orders, and 7 service instances 23
Figure 3.5: SSP services page load time dependency on number of orders 24
Figure 3.6: Minimum and average delay for all actions in the Self-Service Portal (seconds) 26
Figure 3.7: User opinion on SPA functionalities implementation 29
Figure 3.8: User opinion on the importance of adding new SPA functionalities 30
Table of Tables

Table 3.1: Test scenarios 19
Table 3.2: Average delay per TMF compliant API GET call 20
Table 3.3: jMeter performance testing results 25
Table 3.4: Actions implemented using Activiti business processes - performance results 27
Table 3.5: Duration of user input tasks that are part of the implemented business processes 27
Table 3.6: Basic OSS/BSS metrics for the SPA Pilot 28
Executive Summary

It is important to differentiate Service Provider Architecture (SPA) from a typical network service development and implementation. SPA attempts to provide a business/operations support architecture that encompasses all processes involved across the network services lifecycle. This represents a new approach in the GÉANT community that defies the traditional siloed service development design and has implications that are not easy to grasp or understand.

SPA not only provides a higher level of process automation, it also attempts to unify all of an NREN’s services under one umbrella, which represents a significant shift in approach.

Further to the work carried out as part of deliverable D8.5 Service Provider Architecture Pilot v1.0 [D8.5], this document presents results from a recent pilot of the SPA, as well as the benefits and challenges involved in the process of developing and implementing GÉANT’s next generation OSS/BSS solutions. The experience gathered during this process has helped identify the weak links in the (SPA) architecture, the importance of handling performance issues and providing high levels of reliability to users.

In order to obtain the high flexibility of service-oriented design based on microservices, significant effort needs to be invested in design, development, testing and fine tuning SPA, both in terms of manpower and diversity of skills required.

Migration of services to the SPA environment also needs to be carefully handled in order to correctly map these services to the SPA environment. In addition to the standard steps of service specification and inventory definition, one important part of this transition must ensure the existing processes and component features fit the new service.

The SPA Pilot running the E-Line service has managed to provide necessary functionality in a way that is easy to use and follow. No major problems or degradations in performance have been reported, and most of the requested improvements were focused on additional features in the GUI.

In response to user requests, the remaining development time will focus on implementation of service assurance (monitoring and troubleshooting) and feasibility, eduGAIN integration and SSP design and improvements. The main goal of the task will be to provide a more complete showcase solution that can help the community decide on the direction of future efforts implementing OSS/BSS components using standardised APIs.

The decision to implement SPA has the potential to fast-forward network disaggregation. The development of an OSS/BSS solution that interacts with the underlying network components via APIs significantly helps the process of creating a highly automated and configurable network environment that supports fast service lifecycles and promptly responds to infrastructure changes. Thus, the team believes that the work done in this task can help the GÉANT community to implement the long-term vision of dynamic networks.
1 Introduction

Developed by the Joint Research Activity 2, Task 2 (JRA2 T2) team as part of the GN4-2 project, Service Provider Architecture (SPA) is based on a service-oriented architecture, leveraging Representational State Transfer (REST)-based microservices and event-driven responses orchestrated using a business process engine and standardised service bus messages. The main goal of SPA is to define flexible, lightweight Operations and Business Support Systems (OSS/BSS) that will offer a user-oriented approach to the full service lifecycle.

The SPA distributed solution is implemented using a set of components (self-service portal, service catalogue, order management, service and logical resource inventory, and activation and configuration), reachable via corresponding TMF Open APIs and service-fulfilment-related business processes that enable activating complex actions such as service ordering, change and termination.

Deliverable D8.5 Service Provider Architecture (SPA) Pilot v1.0 [D8.5] describes the high-level blueprint of the proposed GÉANT SPA, the next generation OSS/BSS for use by the GÉANT community, and the plan to undertake a pilot of SPA to identify weak links in the architecture, possible performance or scalability issues, or problems with data storing and synchronisation across the layers.

This document follows-up the work started in D8.5 and describes the implementation and outcomes of the two phases of the SPA pilot:

1. Migrating the existing Consolidated Connection Service (CCS) software suite to function within the SPA environment. This is the so-called E-Line project, where the CCS service is offered under the term ‘E-Line’ in SPA, and each new E-Line order creates a new CCS circuit that is reflected as an E-Line service instance in the SPA information architecture.

2. Evaluation of the implementation based on the user experience of the initial alpha users. The evaluation encompasses performance measurements and usability testing. Specifically, the performance analysis focuses on responsiveness-related measurements, such as API delays and web page loading times, while the main goal of usability testing was assessing the informative and functional accuracy, consistency, correctness and overall efficiency of use. The user feedback includes additional remarks on the look and feel of the interface.

The main goal of Phase 1 is to document the steps necessary to port a typical connection-oriented service into the SPA environment and evaluate the scope of effort necessary to migrate such a service to SPA for future reference and planning.

Phase 2 goals include: testing the functionalities and usability provided by the business-process driven, self-service portal (SSP) by offering the E-Line service to a selected user group via the SSP. The user actions available include:
Introduction

- Browsing the SPA dynamic service catalogue and obtaining information about the available services published in the catalogue.
- Ordering a new E-Line service instance.
- Tracking the order completion in real-time, obtaining information about all past or current active E-Line services.
- Changing the E-Line service state (reserved, or active).
- Terminating an active service that is no longer needed.

The functionalities of the pilot have been checked using pre-defined test scenarios and user stories regarding E-Line from users who are knowledgeable about SPA. After initial testing and improvement, another set of users have been invited to use the SSP. This second user group has little or no knowledge of the SPA design and the E-Line service. Their feedback has been used as necessary input to help increase the usability and widen acceptance of the SSP. Another aim of the pilot was to analyse the overall performance of the system, which has been done by measuring a number of OSS/BSS metrics, as well as system performance metrics, such as loading time and delay.

This document represents the outcomes of deploying and running the initial version of SPA, which will be used as input for the design and development of future versions. The rest of this document focuses on the steps performed and results gathered from Phase 1 and 2 of the pilot, providing combined conclusions and recommendations for future SPA development.
2 Phase 1: Porting CCS to SPA: The E-Line Project

Prior to Phase 1 of the SPA Pilot, the implemented features of SPA included the following components:

- Self-Service Portal (SSP) – based on Django CMS that represents the information stored in the internal SPA components and triggers business processes on-demand.
- Service Catalogue – based on the OTRS:ITSM module that stores the information about the offered services, their relationship and specification.
- Customer Relationship Management (CRM) – based on the open source SuiteCRM, which stores essential customer information (divided into personal and organisation information).
- Order management – based on an open source ticket request system (OTRS) that stores the information about each user’s order.
- Service inventory – an in-house developed database management system, using PostgreSQL DB, that stores all service instances and their relationship to used resources.
- Logical resource inventory – in-house developed inventory, using PostgreSQL DB, that stores information on the logical resources.
- Business processes – developed and run in open source workflow engine, Activiti, providing the orchestration of all components in order to implement the actions of service fulfilment in a TMF e-TOM compliant manner [eTOM].

Each component was wrapped with the corresponding TMF Open API [TOA], using Apache Camel connectors to translate the native component API to TMF Open API and vice versa. The business processes ‘request-to-answer’, ‘order-to-fulfilment’, ‘request-to-change’, ‘termination-to-confirmation’ have been implemented in a service-agnostic manner. For more detailed information about the implemented APIs and business processes please refer to the D8.5 SPA pilot v1.0 description document [D8.5].

As previously mentioned, the Consolidated Connection Service (CCS), developed by JRA2-Task 1 (T1), has been identified as the initial service to be migrated to SPA. The effort of migrating CCS to SPA is referred to as the E-Line project, where the customer service provided is to be similar to the Ethernet Private Line (EPL) as defined by the Metro Ethernet Forum (MEF) [EPL].

2.1 E-Line Implementation in SPA

The implementation of Phase 1 started with an in-depth analysis of the E-Line service definition, as provided by JRA2-T1 [CCS-SD]. According to the service definition documentation, the E-Line service aims to provide fully transparent Ethernet circuits over the GÉANT network, using specific software
tools for automated provisioning. Its key characteristics are automated provisioning provided via an API and a resource reservation capability.

The main components that need to be configured and/or specifically designed in order to implement the E-Line service include: the service catalogue that stores the service specification, the service and resources inventory that stores the service instances and the representation of the network infrastructure resources, and the activation and configuration module, a service-specific module which, in this case, interfaces with the OpenNSA aggregator.

In order to compile a TMF-compliant service specification of the E-Line service definition, the T2 team also analysed the available documentation on the NSI protocol [NSI2] and its OpenNSA implementation [ONSA], together with the MEF EPL [EPL] standard.

2.1.1 OpenNSA Overview

The goal of OpenNSA is to enable the fast, dynamic establishment of circuits in a multi-domain environment, in a potentially hierarchical fashion. OpenNSA is currently under development by JRA2 T1, and thus some features that are defined in the NSI documentation are only partially implemented.

The physical entities related to provisioning E-Line using OpenNSA are:

- The Network Service Agents (NSA), running on each network element that supports the creation of E-Line circuits, and
- The GÉANT NSI aggregator that acts as the orchestrator for the whole domain.

The aggregator provides the NSI interface (the API) which other systems can use to request or query circuits. For SPA, this API is the point of interaction with the underlying E-Line network infrastructure represented as a set of NSAs.

![Figure 2.1: An example network domain with STPs and SDPs (P2P Circuit from STP2 to STP10 – STP4-STP6 | STP8+STP9)](image)

Single network domain with a given network ID
There are two types of transmission resources from an OpenNSA point of view: Service Termination Point (STP) and Service Demarcation Point (SDP). An STP can be an uni- or bi-directional port/interface on a given device. In the case when the STP is representing a part of the physical port (logical interface) on the device, a label is used to further identify the STP (ex. vlan 500). Each STP has its own local ID. Also, each physical port has a unique port name. SDPs are formed of two STPs belonging to neighbouring domains and represent their direct connection.

Each device in the network that can be used to set up E-Line P2P circuits runs a local OpenNSA agent. A P2P circuit is to be set up within the network domain starting from one STP on the domain edge and ending on the other STP on the domain edge. This circuit may be composed of a number of segments that traverse the inner network domain devices, where each port of the device can also be represented as STP.

A P2P circuit can be reserved in advance with given start and end time (or default values ‘now’ for start and ‘forever’ for end). The circuit can also have a requested bandwidth-capacity associated with it. In this case, the circuit will be reserved only if the required bandwidth is available across all segments that make up this circuit.

### 2.1.2 Additional QoS Parameters

Although the current E-line implementation is a best-effort service, in order to prepare the SPA environment for the final version of E-Line that supports QoS, as well as to be able to provide necessary QoS information to the performance monitoring and verification architecture designed and developed by JRA2 T4, such additional parameters have now been added to E-Line.

In collaboration with JRA2 T4, additional parameters have been added to the latest version of E-Line but are not in use at the moment: availability, max delay, frame loss, jitter. These parameters also correspond to the EPL service definition in MEF [EPL].

### 2.1.3 Customer-Facing and Resource-Facing Services Specification

Based on the analysis of the OpenNSA implementation and the MEF EPL definition, in coordination with JRA2 T1 and T4, the corresponding E-Line customer-facing and P2P circuit resource-facing services were defined as follows:

- **GÉANT E-Line Customer Facing Service (CFS) specification** – storing all parameter definitions that characterise an E-Line circuit independent of the technology used to implement it. These include parameters such as: circuit ID, name, description, start and end time, directionality, and quality of service parameters, see Figure 2.2.
Phase 1: Porting CCS to SPA: The E-Line Project

Deliverable D8.11 Service Provider Architecture Pilot Service Follow-up Document ID: GN4-2-18-42FE3D

Figure 2.2: E-Line CFS parameters as provided via the SSP

- OpenNSA P2P circuit Resource Facing Service (RFS) specification – representing all parameters that are OpenNSA specific such as STPs and VLANs, see Figure 2.3.

Figure 2.3: OpenNSA P2P circuit RFS parameters as provided via the SSP

Both the CFS and the RFS specifications have been defined in a manner that is TMF-compliant, according to the guidelines provided in [TMFSS] and [DDS]. Please refer to Appendix A for the complete service specification for the CFS and the RFS, provided in json format.

Both the CFS and the RFS are defined as separate services in the service catalogue implemented using the ITSM module of OTRS, where GÉANT E-Line CFS is defined as the End User Service and the OpenNSA P2P circuit RFS is a back-end service (see Figure 2.4).
Phase 1: Porting CCS to SPA: The E-Line Project

<table>
<thead>
<tr>
<th>STATE</th>
<th>SERVICE</th>
<th>COMMENT</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GEANT E-Line</td>
<td>End User Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GEANT E-Line: OpenNSA p2p circuit</td>
<td>Back End</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.4: OTRS:ITSM Service catalogue listing

For both the CFS and the RFS, a corresponding configuration item is created in the OTRS ConfDB using the “alternative to” relationship. These configuration items store the complete service specification.

At the moment, there is no separate SLA component and the CCS service is running under best-effort QoS, a null SLA has been associated with the CFS, see Figure 2.5.

Figure 2.5: SLA and configuration item references in the service catalogue

Figure 2.6 shows a sample of the information stored for the GÉANT E-Line CFS. Some fields, such as OTRS: ITSM, are native fields, while the rest, such as “Related Party”, are custom fields, added to obtain a TMF-compliant service catalogue that stores all of the necessary information.

Figure 2.6: GÉANT E-Line CFS specification in the service catalogue
Phase 1: Porting CCS to SPA: The E-Line Project

Within the service catalogue the two services are connected using a “Depends on” relationship, which demonstrates that in order to complete service fulfilment for the CFS, one needs to implement the corresponding RFS, as shown in Figure 2.7.

2.1.4 Service Inventory Implementation

It has been necessary to map service-specific information with corresponding attributes and add this to the service inventory.

The service inventory PostgreSQL database has been designed from scratch, based on the TMF published service inventory API data model [SIAPI]. In order to use the service inventory to store information about E-Line service instances, a set of rules was defined to ensure consistent information storage and use.

The main representation of the service inventory data model is provided in Figure 2.8. For a full description of all tables and relationships presented, please refer to the TMF Service Overview document [TMFSO]. In the following text, a condensed representation is given of the mapping of E-Line to the SID data model, as defined by JRA2 T2.
For each new E-Line CFS, a new service instance is created in the inventory and all related additional parameters to the service instance are stored using the ServiceCharacteristicValue table. Here, characteristics of interest are described using the ServiceSpecCharacteristic and ServiceSpecCharacteristicValue in the CFS service specification. In addition, a new RFS service instance is created and these two instances are connected using a corresponding relationship. The RFS additional parameters are also stored using the ServiceCharacteristicValue table.
The RFS service is set up by configuring a number of devices and the RFServiceConfiguredUsing relationship points to these LogicalDevices that enable the service. These are the switches or routers chosen to provide the full path between the starting and ending STPs.

The service may depend on multiple intermediate STPs, which are described using the ServiceLRDependency relationship that points to the corresponding TerminationPoint instances.

The start and end STPs for the circuit are the ServiceAccessPoints for the service related to the service via the ServiceAccessedVia relationship. Based on the start and end STPs, two TrailTerminationPoints are created. The complete circuit from the start STP to the end STP is then represented as a Trail that starts and ends with a TrailTerminationPoint. The trail consists of a number of Connections where each of these Connections represents one pair of STPs (one pair of ConnectionTerminationPoints). In this way, the service inventory can provide a general view of the circuit and a detailed view of the circuit components.

For an example CFS and RFS service instance that follows the mapping please see Appendix B.1.

### 2.1.5 Resource Inventory Implementation

#### 2.1.5.1 Resource Inventory Design

Similar to the service inventory, the resource inventory data model used is based on the data model defined for the TMF resource inventory API [RIAPI]. Upon building the generic logical resource inventory, the next step for the T2 team was to map the logical topology information obtained from the OpenNSA agent configuration files into the available attributes in the inventory, as shown in Figure 2.9.
Within the resource inventory, all switches and routers are represented as logical devices that are a type of logical resource. By attaching a logical resource role to the logical device, one can differentiate the type of logical device (role = “switch” or role = “router”). In addition, all OpenNSA capable devices and corresponding interfaces and ports have the role of “E-Line” attached in order to mark the resources that can potentially be used when creating new E-Line service instances. The information about whether the device is available and operational is provided in the LogicalResource attributes. The Logical Device attributes also provide information on the OS used on the device. All OpenNSA capable devices in one NSI domain are gathered together in a Network that holds the NSI domain name.
Each device can have one or more device interfaces which are, in turn, related to a physical port. The physical port that the device interface references is the physical port, where the STPs related to the device interface reside. The Name attribute provides the name of the port, while typeOfPort provides the technology of the port (e.g. Ethernet).

An instance of ConnectionTerminationPoint is a representation of a single STP, where the endPointLabel attribute is used to represent the label information, if any, making it a fully qualified STP. The vendorTPname attribute is used to store the information about the local ID. If there is no label information, then the STP represents the full physical port and there can be no other STPs related to this port. Also, in the case of full port assignment, the mappingMode attribute will be set to full rate mapping, while in case of subdivision, mappingMode will be set to mapped at lower than maximum rate. Direction and typeOfTP attributes in TerminationPoint describe the capabilities for the direction of the STP (one way or bidirectional) and the nature of the STP (source, sink or both). SupportedLayerRate attribute can be used to provide information about the bandwidth of the STP. ctpState is used to describe the current state of the STP: unknown if it is not in use; sink, source or bidirectional when it is used (active or reserved) for a circuit.

An instance of Connection is a relationship between two connection termination points and represents a segment that connects two STPs, i.e. SDP. In order to distinguish between an SDP that is a logical construction and a real established circuit, the parameter operationalState is used:

- If operationalState = 0 (status not available) the connection is an SDP.
- If operationalState = 1 (enabled and in service) the connection is a part of an active circuit.
- If operationalState = 2 (enabled but not in service) the connection is a part of a reserved circuit.

All SDPs have a role “SDP” connected to them. The isUniDirectional attribute of Pipe provides information about the direction of the SDP. OperatingLayerRate provides information about the rate on which the connection is established.

The description of all used elements from the SID model can be found in the TMF SID document on Logical and Compound Resource [RIDD].

### 2.1.5.2 OpenNSA Testbed

For the purposes of the SPA pilot, JRA2 T1 CCS developed an OpenNSA virtual production network environment, presented in Figure 2.10.
Phase 1: Porting CCS to SPA: The E-Line Project

Figure 2.10: OpenNSA-based E-Line testbed

The per-device information gathered from the OpenNSA topology files has been used to populate the resource inventory with all the information necessary to represent the CCS network infrastructure. For these purposes, an automated tool has been created that reads in the OpenNSA topology files and translates them into a set of SQL expressions that effectively push the information into the inventory and create the corresponding relationships between the tables.

For an example representation of the devices and connection points in the resource inventory, please refer to Appendix B.2.

2.1.6 Interfacing with OpenNSA

The interface of E-Line with the OpenNSA aggregator is done via the SPA activation and configuration module. This service-specific module has been implemented as an in-house developed OpenNSA
requester agent that is wrapped with the TMF Activation and Configuration API [ACAPI] in Apache Camel.

The Activation and Configuration API receives a request for service reservation, activation, deactivation or termination, along with all the required service parameters in the form of a fully specified service specification. It then provides a translated request to the requester agent, which, in turn, passes it to the OpenNSA aggregator.

The SPA activation and configuration module is the most important since it provides an automated way of configuring, or programming, the underlying network infrastructure and thus implementing the E-Line service-related actions. Many problems need to be overcome while implementing this interface, including dealing with timeouts in connections, receipt of reservation confirmation before actual reservation takes place, and circuits hanging.

### 2.2 Other Preparations

In addition to the previously described technical steps needed to port the E-Line service to SPA, preparations have also been made during Phase 1, including:

- SPA pilot introductory home page on the self-service portal (SSP).
- Development of a user guide for the SSP [SSPUG].
- Special E-Line service user guide [ELSUG].
- Overall SPA versioning for tracking the changes made during the pilot.
- Pilot integration and usability testing plan definition.
- User feedback survey definition.
- New user sign-up procedure definition.

An anonymous online survey has been created to capture user feedback from Phase 2, together with a two-step procedure for adding new test users, where the newly interested user fills out a form providing essential login data and an email, which creates a corresponding John/Jane Doe customer in the CRM, linked to this user. Once the new user is linked to the CRM, the user is notified via email that the account is active and the portal is ready to be used. No real customer data is been stored in the CRM to protect the privacy of the test users. Each test user can deactivate his/her created account at any time by sending an email to the system admin. This anonymizing of records is only done during the testing phases. Once the service has been launch then the CRM will function as a user database.

As a last step of Phase 2 preparations, a GTS testbed has been created in which the test SPA environment has been cloned, and the SSP has been given a hostname and a SSL certificate to provide server authentication and a secure communication channel. With this, the SPA Pilot v1.0 was made ready for use at [https://ssp.spa.GÉANT.org](https://ssp.spa.GÉANT.org).
2.3 Phase 1 Lessons Learned

The outcome of SPA Pilot Phase 1 is an implementation of SPA that supports the creation, activation, and termination of E-Line service instances, accompanied by a clear understanding of the general and service specific parts of the OSS/BSS solution and an analysis of the effort necessary to port/add a connectivity service to SPA.

The steps outlined earlier in this section described the process of porting a service to SPA. However, it is important to bear in mind that the E-Line service is a fairly standard, connection-oriented service that is relatively straightforward to migrate to SPA. However, this requires a highly skilled team of people that have an in-depth understanding of the service that is being migrated as well as an in-depth understanding of the way TMF represents services, their relationships and the underlying resources involved. In case of a more complex service, such as a service composed of several subservice components, the migration process will increase in complexity. Also, non-typical services, such as a trust and identity type of service, or a VM service, will require considerably more effort in the future since the provided TMF data models and examples do not cover these types of services.

The process involving the service specification in the service catalogue is independent for each new service, and it can be successfully implemented as long as the service can be specified using the dynamic data specification [DDS]. However, if this specification does not meet the requirements of the service to be ported, then the specification schema needs to be extended with additional attributes and relationships, which will create a ripple effect in changes needed in the business processes implementation.

It should be noted that although the JRA2 T2 team did not need to change the implementation of the previously defined generic business processes to migrate E-Line to SPA, this may not be the case for other more complex, or non-standard services. As a result, great care must be taken during the migration process to ensure that any business processes implemented will serve a new service, as expected.

Regarding the service and resource inventory design and implementation, the current inventories implemented according the TMF SID model should be able to support any type of connectivity service that might be migrated in the future. Depending on the details required, additional tables may be needed to store additional data, such as routing protocols. For this step, great care must be taken in maintaining consistency in mapping the service information to the table attributes. It is essential that the same mapping is used for different services, so that the data access and interpretation remain coherent across services. However, it must be noted that the current implementation of the logical resource inventory does not include the ability to store information about software, NFV, software-defined networking (SDN) or other similar components, and it thus must be extended in order to support services that use these types of components.

In addition to the scope of effort needed for migrating services, and the complexity of the process, there were other observations made by the JRA2 T2 team that offer valuable insight into the effort needed to manage, maintain and upgrade SPA in the future.

For example, during Phase 1, the OTRS component was upgraded to a new version. The new version did not involve any changes in the OTRS native API, however, major changes were made to the internal
workings regarding ticket attributes, which led to the need to make changes in the order management API wrapper in Camel. This experience suggests the importance of thoroughly testing and evaluating new versions of the underlying SPA components before they are introduced to the system, since they might require software engineering skills in addition to the traditional system admin support required for the upgrades.

The initial integration testing of E-Line identified several performance issues. For example, as the number of test orders grew, the delay experienced in the SSP when trying to retrieve the customer-related orders increased. Changes were needed to add advanced filtering to the order API so that the filtering is done on the order management component side instead of on the side of the SSP. The original ordering API, as given by TMF, does not include this type of filtering, however, it needed to be added to make the system responsive to user actions. It is expected that as the use of the pilot progresses, other examples of the need for advanced filtering will appear. Another example is exception handling in the Camel wrappers, which, if not done properly stalls all API-related processes and causes time outs in other components. All Camel wrappers have now been refactored in order to minimise the incurred delay and make sure that proper exception handling is implemented.

Once fully integrated, working within the SPA environment is not an easy task for a team of developers. Every change in an interface that impacts two (or usually more) components must be thoroughly tested, and a team effort is required for debugging and fixing. If the test environment is not fully stable, and the VMs that host the components restart due to power failure or go down because of a hardware failure, the task becomes very slow and tedious.

Since SPA system components are so intricately interconnected, they require a very stable, fault tolerant and reliable production environment. In addition, because of this interdependency, the system will fail even if just one component fails, meaning that the system must be implemented in a redundant fashion to increase its overall reliability.

Another task for the next version of the SPA list is also related to the distributed nature of the SPA solution. In order to gauge why the system is not performing at a particular moment, continuous, real-time monitoring must be implemented for each component and each API involved, so that when failure occurs the fault can be easily localised.

From a performance perspective, it was also concluded that the SPA deployed in the GTS testbed environment has shown higher delay and slow responsiveness to the user requests due to the additional latency incurred from the distributed geographical location of the VMs that host the different SPA components. This delay is experienced due to the need to make 15-20 API calls to different components from a single business process. As a result, in order to obtain maximum performance from the SPA, it is imperative that the system is implemented in a local environment where the services are as close to each other as possible to minimise any communication delay. In addition, during Phase 1, the implementation of the business processes has been refactored to ensure the number of API calls is minimised, where possible, using caching or other methods.

Code refactoring has also been done based on the integration and usability testing outcomes, as well as performance measurements undertaken as a part of Phase 2. These results and effects on the stability and performances of the SPA implementation are discussed in the next section.
3  Phase 2: User Experience and Feedback

In Phase 2, the E-Line service was offered for use via the SPA self-service portal to a selected subset of alpha users with the goal to analyse the appropriateness and effectiveness of the offered SPA features. Based on the pilot testing plan, Phase 2 was divided into three parts:

- First wave of users: Internal task users that performed a thorough integration and performance testing.
- Second wave of users: E-Line knowledgeable users (selected team members from JRA2 T1 and JRA2 T4).
- Third wave of users: Novice users of E-Line from different NRENs, GN4-2 Service Activity 2 (SA2), and GÉANT.

The main goal of Phase 2 is to analyse whether the implementation of the SPA provides the necessary functionality for successful provisioning of the E-Line service and related self-service actions.

3.1  SPA Integration and Usability Testing Results

The developed integration and usability test plan includes specific real-life scenarios that were purposefully devised to stress the flexibility and robustness of SPA, including problems such as:

- Service requests in a multitenant environment.
- Dealing with incorrect and stale input.
- Correct serialisation of simultaneous requests to avoid deadlock or conflicts.
- Automated use of new information added to or changed in the inventory.
- Correct, real-time feedback in the self-service portal.
- Error message handling and propagation.

All scenarios were defined in separate test scenario documents created during Phase 1. Table 3.1 provides a list of all defined test scenarios. Each test scenario was broken down into sub levels, and each sub level actions described in detail. The documents included all steps executed by the end user, the desired outcome of the scenario, and the obtained outcome for all scenarios where end users have observed a problem. Only minor issues were discovered, solved and retested during this phase.

<table>
<thead>
<tr>
<th>Affected components</th>
<th>Scenarios</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP</td>
<td>Login Page</td>
<td>Low</td>
</tr>
<tr>
<td>SSP/CRM</td>
<td>User Profile</td>
<td>Medium</td>
</tr>
<tr>
<td>SSP/OTRS:ITSM</td>
<td>Available Service Offers</td>
<td>Medium</td>
</tr>
<tr>
<td>ALL</td>
<td>Valid Fully Specified Order</td>
<td>High</td>
</tr>
<tr>
<td>ALL</td>
<td>Valid Not-Fully Specified Order</td>
<td>High</td>
</tr>
</tbody>
</table>
Phase 2: User Experience and Feedback

### Table 3.1: Test scenarios

<table>
<thead>
<tr>
<th>Affected components</th>
<th>Scenarios</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Failed Order (resources unavailable)</td>
<td>High</td>
</tr>
<tr>
<td>ALL</td>
<td>Invalid Order</td>
<td>High</td>
</tr>
<tr>
<td>SSP</td>
<td>Invalid Order Form</td>
<td>Medium</td>
</tr>
<tr>
<td>ALL</td>
<td>Simultaneous Orders</td>
<td>High</td>
</tr>
<tr>
<td>SSP</td>
<td>Show Only Open Orders</td>
<td>Low</td>
</tr>
<tr>
<td>SSP</td>
<td>Show Only Active Services</td>
<td>Low</td>
</tr>
<tr>
<td>ALL</td>
<td>Pause Active Service</td>
<td>High</td>
</tr>
<tr>
<td>ALL</td>
<td>Re-Activate Paused Service</td>
<td>High</td>
</tr>
<tr>
<td>ALL</td>
<td>Terminate Active Service</td>
<td>Medium</td>
</tr>
<tr>
<td>ALL</td>
<td>Terminate Paused Service</td>
<td>Medium</td>
</tr>
<tr>
<td>ALL</td>
<td>Auto Terminate at End Date</td>
<td>Medium</td>
</tr>
<tr>
<td>Inventory + ALL</td>
<td>Device Under Maintenance</td>
<td>High</td>
</tr>
</tbody>
</table>

A detailed business processes implementation code review was done, where the processes were inspected for conformance to the design specification, correctness and efficiency. All inter-process parameter passing was inspected, and optimisation was performed, where possible. As a result of the code review, a clear separation of the business and operational handling of the CFS and RFS instances was achieved, and an improvement in the time delay was gained by minimising the number of API calls involved in the process.

The main problems encountered during integration testing included some hardcoded fields in the processes and self-service portal, correct transition of ticket states during the ordering process (where some of the brief, transitional, states were removed in order to speed up execution), improvement in the representation of the detailed information about orders and services, change of service instance states in cases of delayed activation (first reserved, then activated at start time), no service instance creation in case of failed orders.

One issue that has been tagged for future improvements, since it requires more substantial efforts for its implementation, is the problem of error checking and error propagation. In the future versions of SPA, more attention needs to be given to providing more meaningful and descriptive error messages to the end user. The SSP needs to be extended to intelligently handle problems in communication with the rest of the components, as well as handling bad input from end-users. Also, the issue of error propagation from the underlying service to the end user needs to be extended so that the end user can better understand why the request failed.

### 3.2 Performance Testing Results

In addition to the integration and usability tests, additional performance-oriented tests were carried out to measure the responsiveness of the system and whether it can be improved.
Since SPA is a fully distributed solution that is based on the microservices approach (developing software in a modular way to ensure each component is self-contained and accessible via well-defined API), any complex action designed using a business process is implemented by calling a number of REST APIs. Thus, the overall responsiveness of the system is very much dependent on (reducing) the delays in getting the requested API responses. In accordance to this, the implementation of the TMF-compliant wrappers around the native APIs can have a tremendous impact on responsiveness for most of the components. In order to measure this delay and its impact, a set of tests were made where the delay incurred by the implementation of the Camel connectors is compared to the delay experienced when no connectors are necessary, and the case when the native API and/or GUI is used for the component.

The information about the implemented TMF APIs in SPA and their function is described in the preceding D8.5 SPA Pilot v1.0 deliverable [D8.5]. The detailed specification for each of these APIs can be found on the TMF Open APIs portal [TOA].

### 3.2.1 API Delays

For the API performance metering, the Postman tool for API testing was used [PMAN]. Direct API calls were made for all TMF-compliant APIs used in the pilot, Table 3.2 presents the obtained averaged values for the delay and the size of the response in number of items. In the cases when the response item is presented with its full json body, the term ‘full’ is given next to the number of items, while if the response item is represented with only a hypertext reference to its full specification, then ‘href’ is listed. It is obvious that the delay is dependent on the number of items in the response, where the highest measured delay is around 18s, occurring when requesting all tickets (1033 in the example) in the ordering queue in OTRS via the Camel serviceOrder wrapper.

<table>
<thead>
<tr>
<th>API call</th>
<th>delay in ms</th>
<th>Response size (no. of items)</th>
</tr>
</thead>
<tbody>
<tr>
<td>API/catalogManagement/serviceSpecification/</td>
<td>2554</td>
<td>6 full</td>
</tr>
<tr>
<td>API/catalogManagement/serviceSpecification/1</td>
<td>384</td>
<td>1 full</td>
</tr>
<tr>
<td>API/orderManagement/serviceOrder</td>
<td>17981</td>
<td>1033 full</td>
</tr>
<tr>
<td>API/orderManagement/serviceOrder/798</td>
<td>336</td>
<td>1 full</td>
</tr>
<tr>
<td>API/customerManagement/customer/</td>
<td>150</td>
<td>17 full</td>
</tr>
<tr>
<td>API/customerManagement/customer/1</td>
<td>84</td>
<td>1 full</td>
</tr>
<tr>
<td>API/v1/serviceInventoryManagement/service</td>
<td>73</td>
<td>62 href</td>
</tr>
<tr>
<td>API/v1/serviceInventoryManagement/service/60</td>
<td>73</td>
<td>1 full</td>
</tr>
<tr>
<td>API/v1/resourceInventoryManagement/logicalResource</td>
<td>108</td>
<td>95 href</td>
</tr>
<tr>
<td>API/v1/resourceInventoryManagement/connection/99</td>
<td>64</td>
<td>1 full</td>
</tr>
</tbody>
</table>

Table 3.2: Average delay per TMF compliant API GET call

In Figure 3.1, the comparison of the time necessary to obtain the full information about one item is presented for the different TMF compliant APIs. It must be noted that the service inventory and resource inventory API is implemented without using a Camel connector, because the inventory
component has been developed in-house from scratch, and its native API is the TMF-compliant API (which is why these two APIs have the minimum values for the delay). Compared to them, the customer API is performing very well, keeping in mind that the CRM component is implemented using the SuiteCRM software suite and that a Camel wrapper has been written to map the TMF API to the SuiteCRM API and vice-versa. However, this is not the case with OTRS. For both, orders and service specification, there is a large delay that is about five times greater compared to the inventory.

Figure 3.1: Comparison of delays in case of using the TMF API to get complete information about one item

In the case of service specification, this delay is due to the complexity of the action. Namely, once a service specification is retrieved in the wrapper, it then needs to be analysed to find its relationship to other services and resources, which need to be retrieved so that the corresponding TMF representation is created. In addition to this, the ITSM module used as the service catalogue is not as efficient as the core modules of OTRS. Also, OTRS does not have a native API for retrieving the service information. Instead, a generic API interface to OTRS needs to be used, and this process incurs additional delays.

Similar to this, the Order API is very slow due to the complex mapping process from the native ticket representation in OTRS to the TMF-compliant json service order representation. To investigate this problem in greater detail, another set of measurements was taken to compare the delay experienced with OTRS when using its web-based GUI and its generic API interface directly without a Camel wrapper.

In Figure 3.2 the delays measured when directly using the OTRS component are presented. When compared to the corresponding rows in Table 3.1, it can be concluded that the Camel wrapper adds almost zero additional delay for massive calls such as getting all tickets (the average per ticket delay equals to the OTRS API 1 ticket delay when getting the info of over 1000 tickets). However, since there is no native GET all tickets API in OTRS, it should be expected that as the number of tickets in the system rises, the performance of the order management component in the SPA are going to go down. To mitigate this problem, a regular “archiving” of old orders needs to be made by moving them into a separate queue. When comparing to the listing of tickets in the OTRS GUI, the delay presented by
OTRS is again much lower, but this is mainly because the list of all tickets presented does not include all ticket details, only the main headings, which makes the process about 3 times faster. This behaviour cannot be replicated in the SPA SSP, due to the inflexibility of the OTRS API. However, it must be noted for future improvement in case a new, more powerful API is developed for OTRS. On the other hand, the delay experience in the OTRS GUI when retrieving the information about one ticket is worse compared to the one ticket GET with the Camel wrapper, which is a positive effect for SPA. The same also applies to service specification.

![Figure 3.2: Delay measurements for information retrieval when using OTRS](image)

Due to these problems in performances with the ordering API, taking into account that this API is the most used, that tickets are created for each user action (create, change, terminate service) and that ticket status is amended several times during one business process, several optimisations have been implemented in order to make the API as fast as possible. One of the major optimisations is the advanced filter used for requesting the list of tickets to be presented to the end user in the SSP.

### 3.2.2 SSP Loading Measurements

The SSP responsiveness and loading times are critical factors that influence the end-user satisfaction of the self-service experience. In SPA, the SSP does not contain any special logic or data, and is heavily dependent on the API calls it makes to order components in order to get the necessary information for representation.

As it can be seen from Figure 3.3, the main actions page, the services page, provides the information about the user, all services in the service catalogue and their status (first column), all orders made by the user and their status (second column), and all service instances created and their status (third column). This information is gathered using TMF-compliant API calls to all other components (CRM, order management, service catalogue, service inventory). The received json information is then parsed and graphically represented. From a performance perspective this means that the time needed to gather and present this information is heavily dependent on the APIs’ response time.
In Figure 3.4, measurements for the loading times of the services page of the SSP are presented. These measurements were taken using an extension of the Chrome web browser, called Page Load Time, which uses the Web Timing API for precise measurement of the page load time. According to the obtained results, the slowest event for the portal is the request event, where the APIs to the respective components are made. Therefore, it is of great importance to minimise the request delay as much as possible.

**Figure 3.4: SSP services page loading measurements for the case of 4 offered services, 15 orders, and 7 service instances**
Since the service order API was the slowest, yet most-used API in the system, additional measurements were taken to analyse the impact of this API on the SSP services page loading time. In Figure 3.5, the SSP services page loading time is represented for cases when the test user has a different number of orders. It can be concluded that the load time is increasing almost linearly with the number of orders, with an increasing factor of around 0.27. This measurement can be used for future reference when deciding on the optimal maximum number of orders to be presented on the page.

### 3.2.3 jMeter Performance Testing Results

Another set of performance tests were made using a special performance testing automated tool, jMeter [JM]. The automated tool enables stress testing of the system, by simulating a given number of users that simultaneously use the SSP. The tools works in such a way that it records the actions of a test user when using the portal, and then replicates these actions a given number of times, while adding some random delay in between actions in order to simulate realistic usage by human end-users.

The automated testing was done using 10 simultaneous users, where each user goes through all possible actions provided by the SSP: creates a service, gets the details about the order and the service, pauses the service, re-activates the service, and finally, terminates the service. The results from the measurements are provided in Table 3.3.

<table>
<thead>
<tr>
<th>Events in section</th>
<th>Avg ms</th>
<th>Min ms</th>
<th>Std. Dev.</th>
<th>Throughput</th>
<th>Received KB/s</th>
<th>Sent KB/s</th>
<th>Avg. Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>22527</td>
<td>369</td>
<td>15080</td>
<td>0.15465</td>
<td>2.07</td>
<td>0.12</td>
<td>13675</td>
</tr>
<tr>
<td>/en/services/</td>
<td>40369</td>
<td>13301</td>
<td>10835</td>
<td>0.04441</td>
<td>7.81</td>
<td>0.03</td>
<td>185128</td>
</tr>
</tbody>
</table>
## Phase 2: User Experience and Feedback

<table>
<thead>
<tr>
<th>Events in section</th>
<th>Avg ms</th>
<th>Min ms</th>
<th>Std. Dev.</th>
<th>Throughput</th>
<th>Received KB/s</th>
<th>Sent KB/s</th>
<th>Avg. Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/en/user/login/</td>
<td>24862</td>
<td>55</td>
<td>15799</td>
<td>0.111295</td>
<td>1.32</td>
<td>0.14</td>
<td>12887</td>
</tr>
<tr>
<td>/ssp-services/get-order-list</td>
<td>38687</td>
<td>13506</td>
<td>13513</td>
<td>0.0386</td>
<td>4.42</td>
<td>0.04</td>
<td>118181</td>
</tr>
<tr>
<td>/ssp-services/get-service-list</td>
<td>32359</td>
<td>14436</td>
<td>9648</td>
<td>0.03744</td>
<td>0.87</td>
<td>0.03</td>
<td>23976</td>
</tr>
<tr>
<td>/ssp-services/handle-rfs</td>
<td>31835</td>
<td>4914</td>
<td>15640</td>
<td>0.05705</td>
<td>0.02</td>
<td>0.06</td>
<td>390</td>
</tr>
<tr>
<td>/ssp-services/service-activate</td>
<td>30028</td>
<td>12176</td>
<td>13456</td>
<td>0.03376</td>
<td>0.01</td>
<td>0.03</td>
<td>361</td>
</tr>
<tr>
<td>/ssp-services/service-offer</td>
<td>31268</td>
<td>2832</td>
<td>15658</td>
<td>0.06667</td>
<td>0.02</td>
<td>0.07</td>
<td>356</td>
</tr>
<tr>
<td>/ssp-services/service-stop</td>
<td>28723</td>
<td>13905</td>
<td>13355</td>
<td>0.01665</td>
<td>0.01</td>
<td>0.01</td>
<td>360</td>
</tr>
<tr>
<td>/ssp-services/service-terminate</td>
<td>38062</td>
<td>33541</td>
<td>4946</td>
<td>0.04194</td>
<td>0.02</td>
<td>0.04</td>
<td>369</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>30922</td>
<td>55</td>
<td>17162</td>
<td><strong>0.3514</strong></td>
<td><strong>17.96</strong></td>
<td><strong>0.32</strong></td>
<td><strong>52328</strong></td>
</tr>
</tbody>
</table>

*Table 3.3: JMeter performance testing results*
Figure 3.6: Minimum and average delay for all actions in the Self-Service Portal (seconds)

In Figure 3.6 the comparison of the minimum time spent in a different section (page) of the SSP compared to the average (human-like) time spent is presented. According to the obtained results, the longest minimum time is taken by the termination process and the related activities with the creation of termination order, confirmation of termination, sending termination request, receiving termination notification, and making corresponding amendments in the order management and inventory components. The rest of the actions take about 15 seconds, with an additional 15 seconds, on average.

The results show that there is room for improvement in the service termination process, and implementation of any procedure updates.

### 3.2.3.1 Activiti Business Process Performance Statistics Results

In the last performance analysis section, the T2 team analysed the performances of the implemented business processes by querying the performance statistics of the business process engine, Activiti. The chosen business process engine offers a special module on business analytics and statistics as a part of the admin GUI. However, this module is only available in the enterprise edition. Thus, for these purposes, the History REST API was used, which enables sending queries about the timing related to all deployed processes and tasks within processes for a given time.

<table>
<thead>
<tr>
<th>Actions</th>
<th>processes involved</th>
<th>no. instances</th>
<th>avg duration in s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Offers</td>
<td>serviceoffer:1:17</td>
<td>128</td>
<td>3.0</td>
</tr>
<tr>
<td>2 Order</td>
<td>serviceorder:1:52</td>
<td>99</td>
<td>20.3</td>
</tr>
<tr>
<td></td>
<td>handlerfsorders:1:60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>serviceprovision:1:56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Change</td>
<td>provisioningchange:1:13</td>
<td>81</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td>provisionresourcechange:1:5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Phase 2: User Experience and Feedback

<table>
<thead>
<tr>
<th>Action</th>
<th>Count</th>
<th>Duration (ms)</th>
<th>Work Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFS Parameter Input</td>
<td>128</td>
<td>78.23</td>
<td>16.52</td>
</tr>
<tr>
<td>serviceoffer:1:17</td>
<td>128</td>
<td>78.23</td>
<td>16.52</td>
</tr>
<tr>
<td>Provide Input Parameters</td>
<td>123</td>
<td>367.55</td>
<td>16.99</td>
</tr>
<tr>
<td>provisionservicechange:1:13</td>
<td>81</td>
<td>620.37</td>
<td>20.33</td>
</tr>
<tr>
<td>provision serviceterm:1:21</td>
<td>42</td>
<td>42.50</td>
<td>12.69</td>
</tr>
<tr>
<td>RFS Parameter Input</td>
<td>99</td>
<td>14056.74</td>
<td>18.33</td>
</tr>
<tr>
<td>handlerfso:1:60</td>
<td>99</td>
<td>14056.74</td>
<td>18.33</td>
</tr>
<tr>
<td>User Service Selection</td>
<td>128</td>
<td>112.80</td>
<td>16.74</td>
</tr>
<tr>
<td>serviceoffer:1:17</td>
<td>128</td>
<td>112.80</td>
<td>16.74</td>
</tr>
<tr>
<td>Grand Total</td>
<td>478</td>
<td>2921.81</td>
<td>17.05</td>
</tr>
</tbody>
</table>

Table 3.5: Duration of user input tasks that are part of the implemented business processes

As shown in Table 3.5, the process of parameter input for the service offer and handle RFS orders processes takes about 14 seconds, which means that the rest of the automated procedure of creating and tracking an order, creating and tracking a new service instance in the inventory, and activating and configuring the service via OpenNSA last around 5 to 6 seconds on average, which is very close to the performances of the change and termination actions. The passing of the user input parameters in the case of service change takes a bit more than 0.6 seconds, while in the case of service termination, this duration is a minimum 0.04 seconds, due to the small number of parameters needed (only service ID).
It must be noted that the Activiti API has proven to be a very powerful solution. Even the history API queries that gathered information for over 2000 process instances at a time were executing very fast. The main performance issues experienced with the business processes only occurred from the large number of API calls via Camel connectors. By implementing very careful refactoring and optimisation, one can expect high stability and performance from the business process engine.

### 3.3 OSS/BSS Metrics

Table 3.6 presents the final metrics calculated at the end of the pilot period that correspond to all activities during Phase 2. The presented metrics have been chosen from the TMF Metrics definition document given in [OBM](#).

The SPA pilot consisted of 20 users in total, representing three waves of users:

- Five users belonged to the task (JRA2 T2).
- Five were E-Line users from JRA2 T1 and JRA2 T4.
- Ten were representatives from GÉANT Operations activities SA1 and SA2 and NRENs.

Feedback was collected from 12 users, partially via an online feedback form and partially via VCs and email. During the pilot there were almost 500 attempts to create E-Line circuits, only 10% of which failed. On average, the circuit creation from start of order to ready for use was around 30 seconds, or 0.009 hours.

<table>
<thead>
<tr>
<th>TMF OSS/BSS metric</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Customer Requests, Per Customer</td>
<td>23.75</td>
</tr>
<tr>
<td># Hours Order Fulfilment Time, From Ordering, To Activation, Per Customer Order</td>
<td>00:00:33.38</td>
</tr>
<tr>
<td>Customer Satisfaction Surveys collected</td>
<td>12</td>
</tr>
<tr>
<td># Orders Attempted</td>
<td>475</td>
</tr>
<tr>
<td># Orders Successful</td>
<td>432</td>
</tr>
<tr>
<td># Orders Failed</td>
<td>43</td>
</tr>
</tbody>
</table>

**Table 3.6: Basic OSS/BSS metrics for the SPA Pilot**

The very low percentage of failed orders compared to the high stress of the system, with 475 circuits is a clear indicator of system stability, where the 10% of failed orders are due to unavailability of resources at the moment of request. There were no requests left hanging or failed due to internal problems of the system during the pilot.

On average, around 23 new circuit requests were completed per customer, each taking about 30 seconds. Compared to the traditionally manual step-by-step procedure implemented for a point-to-point circuit ordered via the current GÉANT Partners Portal, which takes at least an hour, these results show that automation is not only possible, but it can be implemented efficiently and can deal with loads that are superseding the requirements.
3.4 User Feedback Analysis

Based on the feedback gathered via the online survey, where users were asked to score (1 to 5) the current SPA functionalities, it can be concluded that the SPA Pilot has been successfully implemented, see Figure 3.7.

![Figure 3.7: User opinion on SPA functionalities implementation](image)

When asked to provide feedback on their experience with each functionality, users reported minor problems with orders due to receiving failed order tickets, whereas the rest of the actions worked without problem. The main requirement from users was to include the reason why the order failed in the order details, so that the user can try to fix the problem if possible. Other comments focused on the labelling of end-points and requests for real-time checking of the input parameters during the ordering process. Some of the requests for changes in the SSP GUI are highly related to the definition of the E-Line service (end-point labelling, for example) and currently cannot be improved in the next version of the SPA, while the rest of the feedback will be addressed as soon as possible. The changes to be addresses include: intelligent date/time picker for start and end time of circuit, additional descriptive information about the input parameters, improved checks for the correct values of endpoint VLANs. Other important issues have been raised, such as dealing with order history, and providing the ability to search through old orders and all service instances.

When asked to give their opinion on what new features that should be added to SPA (lower score = higher importance in Figure 3.8), most of the users would like to see how SPA will integrate problem management and testing during the fulfilment process, closely followed by performance monitoring for active services. All users agree that adding additional services should be a medium priority for the time being.
Phase 2: User Experience and Feedback

Based on the general comments received, an important aspect for the SSP is how the service hierarchy and organisation can be visually presented to the end-user. The possibility of grouping services and creating a hierarchy of CFS services is already supported by the implemented service catalogue, but an intuitive and clear representation of this structure to the user is something that needs to be well designed in the SSP.

The users also requested integration with the eduGAIN service, which has been expected, and is already on the task roadmap for future versions. The integration however, will not solve the problem of user enrolment in the CRM, it will only check if the user is already part of the CRM or not. The process of enrolling the user and adding the credentials to the CRM needs to be well defined, and it is expected that this process will vary from one organisation to another.

Another point raised in the user feedback is the feasibility of the service. SPA supports two types of service feasibility checks: one based on business rules that define the customers that can order a service and another based on the resource availability when the user tries to activate the requested service. Currently for E-Line, the resource availability check is conducted by the OpenNSA agent and is therefore not part of SPA, while the business rules implementation that will define the type of service a user can order and/or view has been added to the new features list.

3.5 Phase 2 Lessons Learned

Reflecting on the activities in Phase 2, there are three clear requirements regarding the design and implementation of SPA: user perception, improved performance, and high reliability.

While the team focused on the internal workings of the SPA components and their interconnection, the main user feedback was concerned with the SSP GUI layout and its use. This raises an important question for the deployment of SPA, since the SSP component can be developed and tailored
differently by each partner that adopts the architecture. It is possible that this focus on the GUI is a consequence of the one-stop-shop implementation that hides all complexity from the users, and that the users who are not aware of the SPA design perceive the SSP as the only component to hold all of the implemented logic and data. Since the SSP layout and ease-of-use of its functions is important to end-users, improvements must be undertaken with great care. This also entails additional software development efforts when adopting the architecture. In response, it would be worthwhile to develop a modular template that would make the SSP development and tailoring easier.

From a performance point of view, the SSP could become a bottleneck because of the large number of API calls it makes to the rest of the components and the need for substantial processing in order to present the information in a user-friendly manner. There have been several performance-related issues that need to be investigated in detail, to ensure that the correct code refactoring is performed and the experienced delay is reduced. This fact, combined with the rest of the analysis, shows that there is a considerable requirement for efficient Camel connector implementation. Thorough performance testing is needed: (1) on a component level (to be done before choosing the component), (2) during the implementation of the Camel connectors, (3) during implementation of the business processes in Activiti, and finally (4) when integrating all components using the SSP. Fine tuning and high-quality implementation is needed for all components since each step adds to the overall loading time and user-delay. It has also been proven that all SPA components need to be deployed as close to each other as possible in order to minimise any communication lags.

Another major lesson learned relates to the reliability of the solution. Since SPA is a distributed solution where each action involves many components, it is crucial to have all components up and running 24/7 so that the system can work as a whole. This increases system complexity, maintenance and management compared to a single all-in-one solution. Each component must be developed in a redundant fashion so that a backup can come online if there is a problem with the primary instance. Also, it would be best to keep all components as separate VMs or containers in order to be able to perform live migrations whenever a hardware problem occurs. During Phase 2 there was a power outage in the cluster where the SPA Pilot solution was hosted, which resulted in service interruption for a considerable time period because the particular infrastructure was not maintained around the clock. Thus, it would be best to host all SPA components in a high-grade datacentre facility.

An important aspect that has been highlighted from work in Phase 2 is the need for appropriate user notification in case of errors and/or problems. According to feedback, users would like to have more real-time checking of their input when ordering a service in order to minimise the possibility of placing an invalid order. In addition, it is important that the failed orders contain a comprehensive, yet easy to understand, error message/notification so that a user can adjust input accordingly or understand why the requested service cannot be instantiated.
4 Overall Pilot Results Analysis and Future Development Remarks

It is important to differentiate SPA from a typical network service development and implementation. SPA attempts to provide a business/operations support architecture that encompasses all processes involved across the network services lifecycle. This represents a new approach in the GÉANT community that defies the traditional siloed service development design and has implications that are not easy to grasp or understand. This has led to a degree of confusion among users and stakeholders, who have had a difficult time differentiating between SPA as a system and E-Line as one service offered via the system. SPA not only provides a higher level of process automation, it also attempts to unify all of an NREN’s services under one umbrella, which is a significant shift compared to the current situation. It is therefore important to carefully introduce the concept and gauge its acceptance and implementation.

The pilot implementation, with E-Line as an initial example service, has proven the viability of the proposed SPA blueprint that defined the minimum set of components, processes and APIs for support of the basic service fulfilment lifecycle. However, it has also raised a number of issues that require further investigation and consideration. While the porting of E-Line has gone according to plan, and its mapping to the TMF data models and processes did not require any special customisation, it remains to be analysed how SPA can fit large-scale services or trust and identity-related services, which are considerably different compared to the traditional telecommunications network services, such as E-Line.

Based on the lessons learned during the SPA pilot implementation, further discussion of the major issues concerning the organisation-wide deployment of SPA as a unified OSS/BSS solution for all supported services is presented in the following sections.

4.1 Scope of Effort

The initial SPA pilot implementation has required a tremendous amount of effort, characterised by a steep learning curve for both designers and developers. A great amount of documentation required analysis and skills, such as development and implementation of the Activiti business process, needed to be acquired that were scarce in the network software development environment. The Task 2 team worked diligently to produce the pilot, and it was fortunate that there was no significant staff turnover among the members, as this would have caused a major setback.

Based on the experience so far, the scope of effort required for further development is quite high, not just in terms of necessary work effort (person months), but also in gathering the right set of people with the required skills. The development of each new component requires not only an understanding of the component, but also how it connects to the rest of SPA via the business processes in place that interact with the component using Camel connectors. Learning from Phase 1 suggested that the time taken to develop a new feature is an issue and that substantial time must be planned for testing and quality assurance. The current Task 2 team is quite small, with only one designer and two developers,
which is far from enough to produce a high-quality production solution. A much larger team is needed to address the issues in terms of performance, reliability, and usability discussed earlier.

Once a feature or component is developed and moved into production, corresponding effort is needed to support the operations of the item. Each component requires continuous maintenance from highly qualified staff. The issues experienced when upgrading a single component have shown that a ripple effect of other issues can occur if everything is not tested carefully beforehand. Thus, the flexibility and distributed nature of SPA comes at a price of increased requirements for maintenance and support.

The process of adding a new service, whether via new service development or existing service migration, also requires considerable effort and a highly skilled team. It is imperative that the team consists of people who are intimately acquainted with the service in question but are also very well-versed in TMF data modelling and processes to ensure a successful transition and mapping.

It is therefore evident that reaching a high-quality level of production is difficult to achieve and requires considerable time and effort, a large amount of which must be spent on testing and quality assurance. Harmonisation of the data model and the processes across vastly different services is just one example of the challenges. Since the community has adopted the “one-size doesn’t fit all” motto over the years, it is very difficult to expand SPA to incorporate all service types and their specific nuances. Another aspect that must be considered is the number of bespoke services offered to the project, which is considerable. All of these services must be thoroughly analysed in order to understand their bespoke aspects, and then these need to be incorporated in parallel to the standard processes in SPA.

A great deal of effort will also be needed to introduce high reliability and fault tolerance for the system as a whole. Reaching 5 nines (99.999% availability) with SPA will require the development and implementation of a reliability plan that will put into place many failover mechanisms and design a strong change management process that will control the component- and system-wide versioning.

4.2 Issues with Information Synchronisation

The SPA data model and all the implemented processes are based on the premise that the SPA data stored is the authoritative data. This requires the SPA resource and service inventory to store the authoritative data reflecting the current status of the network. Before any configuration (or other) change is made to the network, the information is stored in the service inventory, then pushed to the network, not the other way around, which is more typical of data management in the community today.

However, taking into account that SPA is another layer on top of any NMSs, EMSs and the actual hardware in question, information synchronisation is not a straightforward process. The information from SPA is pushed down through the layers, each representing and storing different views and data information in different formats and levels of granularity. If there is a problem in the communication with the NMS or EMS, there will be a problem when the system goes into an unstable state where the information across the layers is in conflict. There have already been experiences that tackle this issue using JunoSpace as an intermediary between the device and the service activation and configuration module, when the EMS is not responding to requests and the service is left hanging in a limbo state.
Another issue is using APIs to reach out to the hardware during the configuration process. There is much legacy hardware and software that is not reachable for automated configuration.

Thus, forcing the authority of the SPA inventory is not a straightforward procedure, and all repercussions and possible glitches may not be foreseen. It is very difficult to keep the real-time data in the SPA inventory up-to-date because of the long propagation across the layers.

A related issue is the completeness of the information stored in the SPA inventory. Namely, the NMS/EMS do not always supply the full information upon service configuration. For an example, when setting up a routing path, if Infinera DNA is used to setup a circuit, in order to learn the actual path the circuit will take, an additional software component needs to be developed in order to query Infinera DNA and discover the path. A similar problem occurs in the case of setting up E-Line when using the layered approach of SPA, NMS, JunoSpace, JunOS, RSVP, where the service is setup using the RSVP protocol and the resulting path needs to be queried and propagated to the SPA inventory after the circuit is set in place.

In essence, setting a declaration that SPA inventory is authority requires working layer by layer, redesigning and re-implementing other software to accept this authority and react accordingly. Such effort requires additional software development and more flexible access to the hardware to extract the necessary information.

It must be noted, however, that this effort will eventually provide a software solution that can make it easier to add/change/remove hardware to the system compared to the process in place today.

**4.3 Migrating Services to SPA**

The process of service migration, where services developed as silos are moved to a unified architecture such as SPA, is a substantial effort that must also be taken into account. The process is not a straightforward step-by-step procedure, and requires a detailed level of knowledge of the service and the service catalogue and inventory data models in SPA. Based on the lessons learned from the Phase 1 pilot implementation, it is very important to make a good and clear mapping of the service parameters and the service/inventory data fields, which will be consistent across services.

However, the more complex the service, the more adaptation and extensions may be required in the service specification description and the service/resource inventory. In addition, for any service to be added, the activation and configuration module for that service needs to be written from scratch.

It must be noted that adding a new service to SPA may require extensions and adjustments in other areas as well. Each service will require detailed investigation of whether or not the service fits into the developed business processes, or if there are any specifics that need to be taken into account and will cause changes in the processes implementation.

The service may require other modules to be extended as well. For example, the service may have a different payment model that needs to be incorporated or a specific feasibility check that needs to be translated into a new set of business rules.
4.4 Adoption in the Community

The challenges discussed so far raise practical issues around the potential for wide adoption of SPA by the NRENs. The system complexity and size is too big for small NRENs that do not have the staff to maintain it. In addition, the initial effort to set up and customise SPA is also something that must not be overlooked.

Alternatively, GÉANT as an organisation could benefit by implementing SPA. In this way, all core systems would be available via an integrated solution based on an internal, well-defined information architecture with feature-rich components and interfaces that will enable GÉANT to substantially increase the automation of the traditionally offered services. The separation of processes and hardware offered by SPA enables clear implementation of operational procedures that can decrease the time from service order to service use.

In addition, taking into account that the TMF data model is already NFV-ready, with special inventory data model and corresponding APIs that are focused on network function virtualisation [NFV], in addition to the support for describing other software-based resources and services, adopting SPA could bring GÉANT a significant step closer towards next-generation virtual network environments.

For the rest of the NRENs, it is expected that SPA adoption would happen on a case-by-case basis. Depending on the number of services and specifics of the inner organisation and procedures, substantial manpower (software engineers) will be needed to provide customisation and day-by-day support after the initial effort needed to adopt and align with the internal guidelines and best practices.

It is recommended that the adoption is implemented using two parallel tracks. Due to the far-reaching nature of SPA, a cold switch approach is not advisable. An incremental approach is needed, as the SPA environment can be deployed in a separate, self-contained environment that is able to run in parallel to the current operational environment. Then services can be carefully migrated one by one, where each migration will require further development of SPA (and possibly changes other systems such as NMSs and EMSs). In this way, until SPA is brought to the desired production level, the rest of the services can run in a parallel environment.

The current version of SPA needs to be further developed and expanded before the adoption process can start. SPA must first provide additional essential features such as service feasibility, monitoring and problem management, as requested by the pilot users.

4.5 Mixed Environments

When adopting SPA there are issues outside an organisation that also need to be taken into consideration. The use of SPA cannot be presented as a requirement to the GÉANT community so that a given service is adopted/supported by a partner or not. Even in the case when all of the community partners have adopted SPA, there will still be networks outside the community that use different OSS/BSS solutions and APIs.

The major question that arises is how SPA deals with multi-domain services that are deployed on top of SPA, especially in a mixed environment where some parties have adopted SPA and others have not.
In a mixed environment, the requesting partner must first check whether SPA is enabled, and then proceed accordingly. In the case of multi-domain services, the portion of the service that needs to be provided by the SPA-enabled domain is treated as a new service request inside SPA, but the request is made by a partner (the original multi-domain service provider) and not a typical customer. Even in this case, the creation of the new service needs to go through all service-ordering related steps. The main difference is that the partner will not interact with the SSP, and instead call the business processes APIs to request the service, and later receive information about the service. It is not advised to provide the partner with a direct access to the component APIs, since this will enable the bypass of the procedures implemented in the business processes and will lead to inconsistent data across the SPA component.

From a service development point of view this means that, when multi-domain services are developed, care must be taken whether the service is offered via SPA or not, and the service must be able to adapt to the settings (SPA or no SPA). This adds another layer of complexity when developing new software solutions. Guidelines need to be developed that will help manage this process successfully, so that the co-existence of SPA-enabled and SPA-free environments can cooperate.

Another issue is coordination of SPA updates, upgrades and related issues across the community. This is especially a problem if the updates include changes in APIs. For each multi-domain service, it will be necessary to define the minimum SPA version required, and make sure that the later versions of SPA fully support the service. It should be noted that the TMF specification, SID and APIs especially, have changed considerably since the beginning of this task, and are undergoing further change by the TMF. In order to remain TMF-compliant, each change must be reflected in the SPA, which requires substantial additional effort by the developers. This has already happened several times to the Task 2 team concerning the inventory and ordering APIs. Therefore, to avoid multiple implementation efforts of the same interface, it is our recommendation to implement only stable TMF specifications that are not expected to undergo significant change.

5 Conclusion

The SPA pilot results and analysis presented in this document have underpinned the benefits and challenges in the process of developing and implementing GÉANT’s next generation OSS/BSS solutions. The experience gathered during this process has helped identify the weak links in the architecture, the importance of handling performance issues and providing high reliability. It has been shown that in order to obtain the high flexibility of the service-oriented design based on microservices, significant effort needs to be invested in design, development, testing and fine tuning, both in terms of manpower and diversity of skills required.

The activities performed during Phase 1 of the pilot have shown that the process of migration of services to the SPA environment needs to be handled very carefully because all service details need to be correctly mapped to the SPA environment. In addition to the standard steps of service specification and inventory definition, one important part of this transition must ensure the existing processes and component features fit the new service. For example, the migration of a trust and identity service would require significant extension due to the limitation of the current inventory to capture the necessary service information.
Based on the feedback from the users, the SPA Pilot running the E-Line service has managed to provide the basic service fulfilment functionalities that include browsing services, ordering, reviewing, changing and terminating a service, in a way that is easy to use and follow. No major problems or degradations in performance have been reported, and most of the requested improvements were focused on additional features in the GUI. In response to the user requests, the remaining time in the project will focus on implementation of service assurance (monitoring and troubleshooting), service feasibility, eduGAIN integration, and SSP design and improvements. The main goal of the task will be to provide a more complete showcase solution that can help the community decide on the direction of future efforts implementing OSS/BSS components using standardised APIs, where complex actions that involve multiple components are automated using business processes.

The decision to implement SPA has the potential to fast-forward network disaggregation. The development of an OSS/BSS solution that interacts with the underlying network components via APIs significantly helps the process of creating a highly automated and configurable network environment that supports fast service lifecycles and promptly responds to infrastructure changes. Thus, the team believes that the work done in this task can help the GÉANT community to implement the long-term vision of dynamic networks.
Appendix A Service Specifications json Example

A.1 E-Line CFS

{
  "ServiceSpecification": {
    "ID": "some id - 111",
    "Name": "GEANT P2P",
    "Version": "1.0",
    "LastUpdate": "2018-01-17T10:30:00+01:00",
    "LifeCycleStatus": "Draft",
    "Description": "P2P circuit CFS specification",
    "Category": "CFS",
    "ValidFor": {
      "StartDateDateTime": "2018-01-17T10:30:00+01:00",
      "EndDateDateTime": "2019-01-17T10:30:00+01:00"
    },
    "relatedParty": [
      {
        "role": "owner",
        "id": "1",
        "href": "http://10.10.10.1:1111/partyManagement/organisation/1"
      }
    ],
    "serviceSpecRelationship": [
      {
        "type": "dependency",
        "ID": "222",
        "ValidFor": {
          "StartDateDateTime": "2018-01-17T10:30:00+01:00",
          "EndDateDateTime": "2019-01-17T10:30:00+01:00"
        }
      }
    ],
    "ServiceSpecCharacteristic": [
      {
        "ID": "resID",
        "Name": "Reservation ID",
      }
    ]
  }
}
"Description": "ID of the reservation provided by the customer",
"ValueType": "KeyValue",
"Configurable": "true",
"MinCardinality": "1",
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [{
"ValueType": "String",
"Default": "false",
"ValidFor": {
"StartDateDateTime": "2018-01-17T10:30:00+01:00"
}
}]
},
{
"ID": "resNameID",
"Name": "Reservation name",
"Description": "descriptive name of the reservation",
"ValueType": "KeyValue",
"Configurable": "true",
"MinCardinality": "0",
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [{
"ValueType": "String",
"Default": "true",
"ValidFor": {
"StartDateDateTime": "2018-01-17T10:30:00+01:00"
}
}]
},
{
"ID": "startTimeID",
"Name": "Start Time",
"Description": "Start time of reservation",
"ValueType": "KeyValue",
"Configurable": "true",
"MinCardinality": "0",
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [{
"ValueType": "DateTime",
"Default": "true",
"ValidFor": {
"StartDateDateTime": "2018-01-17T10:30:00+01:00"
}
}]
},
{
"ID": "endTimeID",
"Name": "End Time",
"Description": "End time of reservation",
"ValueType": "KeyValue",
"Configurable": "true",
"MinCardinality": "0",
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [{
"ValueType": "DateTime",
"Default": "true",
"ValidFor": {
"StartDateDateTime": "2018-01-17T10:30:00+01:00"
}
}]
}
```json
"ValidFor": {
    "StartDateTime": "2018-01-17T10:30:00+01:00"
}
}

{ "ID": "CapacityID", "Name": "Capacity", "Description": "capacity of the circuit in bps", "ValueType": "Range", "Configurable": "true", "MinCardinality": "1", "MaxCardinality": "1", "ServiceSpecCharacteristicValue": [{ "ValueType": "Integer", "Default": "true", "Value": "0", "UnitOfMeasure": "bps", "ValueFrom": "0", "ValueTo": "100000000000", "ValidFor": {
    "StartDateTime": "2018-01-17T10:30:00+01:00"
    }
    }]
},

{ "ID": "delayID", "Name": "Max delay", "Description": "Max packet delay", "ValueType": "Range", "Configurable": "true", "MinCardinality": "1", "MaxCardinality": "1", "ServiceSpecCharacteristicValue": [{ "ValueType": "Float", "Default": "true", "Value": "100", "UnitOfMeasure": "ms", "ValueFrom": "0", "ValueTo": "100000", "ValidFor": {
    "StartDateTime": "2018-01-17T10:30:00+01:00"
    }
    }]
},

{ "ID": "FrameLossID", "Name": "Max frame loss rate", "Description": "Max frame loss rate in % of packets sent", "ValueType": "Range", "Configurable": "true", "MinCardinality": "1", "MaxCardinality": "1", "ServiceSpecCharacteristicValue": [{ "ValueType": "Float", "Default": "true", "Value": "0", "UnitOfMeasure": "", "ValueFrom": "0", "ValueTo": "1", "ValidFor": {
    "StartDateTime": "2018-01-17T10:30:00+01:00"
    }
    }]
}
"Value": "100",
"UnitOfMeasure": ": %",
"ValueFrom": "0",
"ValueTo": "100",
"ValidFor": {
  "StartDateDateTime": "2018-01-17T10:30:00+01:00"
}
},
{
  "ID": "JitterID",
  "Name": "Jitter",
  "Description": "Jitter of the circuit in ms",
  "ValueType": "Range",
  "Configurable": "true",
  "MinCardinality": "1",
  "MaxCardinality": "1",
  "ServiceSpecCharacteristicValue": [{
    "ValueType": "Integer",
    "Default": "true",
    "Value": "10000",
    "UnitOfMeasure": "ms",
    "ValueFrom": "0",
    "ValueTo": "10000",
    "ValidFor": {
      "StartDateDateTime": "2018-01-17T10:30:00+01:00"
    }
  }]
},
{
  "ID": "AvailabilityID",
  "Name": "Availability",
  "Description": "Availability of the circuit in %",
  "ValueType": "Range",
  "Configurable": "true",
  "MinCardinality": "1",
  "MaxCardinality": "1",
  "ServiceSpecCharacteristicValue": [{
    "ValueType": "Float",
    "Default": "true",
    "Value": "0",
    "UnitOfMeasure": ": %",
    "ValueFrom": "0",
    "ValueTo": "100",
    "ValidFor": {
      "StartDateDateTime": "2018-01-17T10:30:00+01:00"
    }
  }]
},
{
  "ID": "DirectionalityID",
  "Name": "Directionality",
  "Description": "whether the circuit requested is to be \unidirectional\" or \bidirectional\"
, "ValueType": "Enumerated",
  "Configurable": "true",
  "MinCardinality": "1",
,}
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [
  {
    "ValueType": "String",
    "Default": "true",
    "Value": "Bidirectional",
    "ValidFor": {
      "StartDateTime": "2017-01-18T10:30:00+01:00"
    }
  },
  {
    "ValueType": "String",
    "Default": "false",
    "Value": "Unidirectional",
    "ValidFor": {
      "StartDateTime": "2017-01-18T10:30:00+01:00"
    }
  }
]
}

"ID": "SymmetricID",
"Name": "Symmetric path",
"Description": "should the reverse path be the same?",
"ValueType": "KeyValue",
"Configurable": "true",
"MinCardinality": "0",
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [{
  "ValueType": "Boolean",
  "Default": "true",
  "Value": "true",
  "ValidFor": {
    "StartDateTime": "2018-01-18T10:30:00+01:00"
  }
}]

A.1.1 OpenNSA P2P circuit RFS

"ServiceSpecification": {
  "ID": "some id - 222",
  "Name": "OpenNSA p2p circuit",
  "Version": "1.0",
  "LastUpdate": "2018-01-18T10:30:00+01:00",
  "LifeCycleStatus": "Draft",
  "Description": "OpenNSA P2P circuit RFS specification as per CS-NSI docs",
  "Category": "RFS",
  "ValidFor": {
"StartDateTime": "2018-01-17T10:30:00+01:00",
"EndDateTime": "2019-01-17T10:30:00+01:00"
},
"relatedParty": [ 
  
},
"requiredServiceSpecification":[
  
},
"ServiceSpecCharacteristic": [ 
  
],
"ID": "111",
"href":
"Name": "GEANT P2P",
"ValidFor": { 
  "StartDateTime": "2018-01-17T10:30:00+01:00",
  "EndDateTime": "2019-01-17T10:30:00+01:00"
},
"ID": "SourceSTPID",
"Name": "Source STP",
"Description": "id of the source STP for the circuit",
"ValueType": "KeyValue",
"Configurable": "true",
"MinCardinality": "1",
"MaxCardinality": "1",
"ServiceSpecCharacteristicValue": [{
  "ValueType": "String",
  "Default": "false",
  "ValidFor": { 
  "StartDateTime": "2018-01-17T10:30:00+01:00" } 
}]
},
{ 
  "ID": "DestSTPID",
  "Name": "Destination STP",
  "Description": "id of the dest. STP for the circuit",
  "ValueType": "KeyValue",
  "Configurable": "true",
  "MinCardinality": "1",
  "MaxCardinality": "1",
  "ServiceSpecCharacteristicValue": [{
    "ValueType": "String",
    "Default": "false",
    "ValidFor": { 
    "StartDateTime": "2018-01-17T10:30:00+01:00" } 
  }]
}
{ "ID": "sVLANID", "Name": "Source VLAN", "Description": "source VLAN id in range (0-4095)", "ValueType": "Range", "Configurable": "true", "MinCardinality": "1", "MaxCardinality": "1", "ServiceSpecCharacteristicValue": [{ "ValueType": "Integer", "Default": "false", "ValueFrom": "0", "ValueTo": "4095", "ValidFor": { "StartDateTime": "2018-01-17T10:30:00+01:00" } } ], },
{ "ID": "dVLANID", "Name": "Destination VLAN", "Description": "destination VLAN id in range (0-4095)", "ValueType": "Range", "Configurable": "true", "MinCardinality": "1", "MaxCardinality": "1", "ServiceSpecCharacteristicValue": [{ "ValueType": "Integer", "Default": "false", "ValueFrom": "0", "ValueTo": "4095", "ValidFor": { "StartDateTime": "2018-01-17T10:30:00+01:00" } } ] }
}
Appendix B  Inventory json Examples

B.1  Service Instance Examples

B.1.1  CFS Instance Example

```json
{
    "category": "pilot",
    "description": "circuit 12",
    "status": "active",
    "type": "CFS",
    "hasStarted": true,
    "href": "http://10.10.1.1111/api/v1/serviceInventoryManagement/service/60",
    "id": "60",
    "isServiceEnabled": true,
    "isStatful": true,
    "name": "circuit 12",
    "startMode": "auto",
    "orderDate": "2018-04-03T08:40:19.971Z",
    "startDate": "2018-04-03T08:39:46.000Z",
    "endDate": "2018-04-03T09:39:46.000Z",
    "serviceSpecification": {
        "id": "1",
        "name": "GÉANT E-Line"
    },
    "relatedParty": [
        {
            "href": "http://10.10.1.1111/API/customerManagement/customerAccount/3",
            "id": "3",
            "name": "GÉANT",
            "role": "owner"
        },
        {
            "href": "http://10.10.1.1111/API/customerManagement/customer/4",
            "id": "4",
            "name": "Sonja Filiposka",
            "role": "requester"
        }
    ],
    "place": [],
    "serviceOrder": [
        {
            "id": "772",
```
"href": "http://10.10.10.1:1111/API/orderManagement/serviceOrder/772"
}
,"serviceRelationship": [],
"serviceCharacteristic": [
{
"name": "Symmetric path",
"value": "True"
},
{
"name": "Start Time",
"value": "2018-04-03T10:39:46+02:00"
},
{
"name": "Capacity",
"value": "250000000"
},
{
"name": "Directionality",
"value": "Bidirectional"
},
{
"name": "Reservation name",
"value": "circuit 12"
},
{
"name": "Jitter",
"value": "10000"
},
{
"name": "Service Type",
"value": "GÉANT E-line"
},
{
"name": "Max delay",
"value": "100.0"
},
{
"name": "End Time",
"value": "2018-04-03T11:39:46+02:00"
},
{
"name": "Reservation ID",
"value": "303"
},
{
"name": "Max frame loss rate",
"value": "100.0"
},
{
"name": "Availability",
"value": "0.0"
}
B.1.2 RFS Instance Example

{
    "category": "pilot",
    "description": "circuit 12",
    "status": "active",
    "type": "RFS",
    "hasStarted": true,
    "href": "http://10.10.1:1111/api/v1/serviceInventoryManagement/service/61",
    "id": "61",
    "isServiceEnabled": true,
    "isStatful": true,
    "name": "circuit 12",
    "startMode": "auto",
    "orderDate": "2018-04-03T08:40:19.971Z",
    "startDate": "2018-04-03T08:39:46.000Z",
    "endDate": "2018-04-03T09:39:46.000Z",
    "serviceSpecification": {
        "id": "2",
        "name": "OpenNSA p2p circuit"
    },
    "relatedParty": [          
        "href": "http://10.10.1:1111/API/customerManagement/customerAccount/3",
        "id": "3",
        "name": "GÉANT",
        "role": "owner"
    ],
    "id": "4",
    "name": "Sonja Filiposka",
    "role": "requester"
}

"place": [],
"serviceOrder": [ {
  "id": "772",
  "href": "http://10.10.10.1:1111/API/orderManagement/serviceOrder/772"
} ],
"serviceRelationship": [ {
  "serviceRef": {
    "href": "http://10.10.10.1:1111/api/v1/serviceInventoryManagement/service/60",
    "id": "60"
  }
} ],
"serviceCharacteristic": [ {
  "name": "Service Type",
  "value": "ets"
}, {
  "name": "Destination STP",
  "value": "lab2.nsi.new.lab:mx1-1-7"
}, {
  "name": "Source VLAN",
  "value": "933"
}, {
  "name": "Destination VLAN",
  "value": "933"
}, {
  "name": "connectionID",
  "value": "NS-d791068874"
}, {
  "name": "Source STP",
  "value": "lab1.nsi.new.lab:mx1-1-7"
} ],
"supportingService": [],
"supportingResource": [
  {
    "href": "http://10.10.10.1:1111/api/v1/resourceInventoryManagement/Trail/1",
    "id": "1"
  }
]
B.1.3 Trail Example

```json
{
    "id": "1",
    "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/Trail/1",
    "name": "NS-d791068874",
    "description": "circuit 12",
    "version": "0.1",
    "publicIdentifier": "na",
    "category": "pilot",
    "lifecycleState": "active",
    "validFor": {
        "endDate": "2018-06-30T09:08:19.547Z",
        "startDateTime": "2018-12-12T10:08:19.547Z"
    },
    "@type": "trail",
    "@baseType": "resource",
    "@schemaLocation": "http://jra2t2.net",
    "usageState": "Active",
    "capacity": [],
    "capacityDemand": [],
    "note": [],
    "place": null,
    "relatedParty": [],
    "resourceRelationship": [],
    "resourceCharacteristic": [],
    "resourceSpecification": null,
    "resourceAttachment": [],
    "role": [
        {
            "id": "1",
            "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/role/1",
            "description": "eline",
            "name": "Eline",
            "roleSelectionMethod": "RetrieveAll",
            "roleCombination": "all"
        }
    ],
    "value": "na",
    "lrStatus": "Ok",
    "serviceState": "InService",
    "isOperational": true,
    "network": {
        "id": "1",
        "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/network/1"
    },
    "additionalInfo": "na",
    "mteAdministrativeState": "Unlocked",
    "logicalAlarmReportingEnabled": true,
    "logicalAlarmStatus": "Cleared"
}
```
"isMTEOperational": true,
"operationalState": "EnableInService",
"typeOfMTE": "Trail",
"isUniDirectional": false,
"operatingLayerRate": 250000000,
"logicalDevice": [
  {
    "id": "4",
    "href": "http://10.10.10.1:1111/api/v1/resourceInventoryManagement/logicalDevice/4"
  },
  {
    "id": "5",
    "href": "http://10.10.10.1:1111/api/v1/resourceInventoryManagement/logicalDevice/5"
  }
],
"connection": [
  {
    "href": "http://10.10.1:1111/api/v1/resourceInventoryManagement/connection/100",
    "id": "100"
  }
],
"terminationPoint": [
  {
    "href": "http://10.10.1:1111/api/v1/resourceInventoryManagement/trailTerminationPoint/1",
    "id": "1"
  },
  {
    "href": "http://10.10.1:1111/api/v1/resourceInventoryManagement/trailTerminationPoint/2",
    "id": "2"
  }
]

B.1.4 Trail Termination Point Example

{
  "id": "1",
  "href": "http://10.10.1:1111/api/v1/resourceInventoryManagement/trailTerminationPoint/1",
  "name": "dev1stp1",
  "description": "stp",
  "version": "0.1",
  "publicIdentifier": "na",
  "category": "pilot",
  "lifecycleState": "active",
  "validFor": {
    "endDate_time": "2018-06-30T09:08:18.902Z",
    "startDateTime": "2018-12-12T10:08:18.902Z"
  }
}
"@type": "trailTerminationPoint",
"@baseType": "resource",
"@schemaLocation": "http://jra2t2.net",
"usageState": "Active",
"capacity": [],
"capacityDemand": [],
"note": [],
"place": null,
"relatedParty": [],
"resourceRelationship": [],
"resourceCharacteristic": [],
"resourceSpecification": null,
"resourceAttachment": [],
"role": [
{
"id": "1",
"href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/role/1",
"description": "eline",
"name": "Eline",
"roleSelectionMethod": "RetrieveAll",
"roleCombination": "all"
}
],
"value": null,
"lrStatus": "Ok",
"serviceState": "InService",
"isOperational": true,
"network": {
"id": "1",
"href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/network/1"
},
"additionalInfo": null,
"mteAdministrativeState": "Unlocked",
"logicalAlarmReportingEnabled": null,
"logicalAlarmStatus": "Cleared",
"isMTEOperational": true,
"operationalState": "EnableInService",
"typeOfMTE": "TrailTerminationPointBidirectional",
"trail": {
"id": "1",
"href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/Trail/1"
},
"direction": "CarriersTraffic",
"typeOfTP": "BidirSource",
"vendorTPName": "mx1-1-7",
"endPointLabel": "vlan:933",
"deviceInterface": [
{
"id": "7",
"href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/device/7"
}]}
B.2 Resource Instances Examples

B.2.1 Logical Device Example

{
  "id": "4",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/logicalDevice/4",
  "name": "lab1-mx80",
  "description": "mx80",
  "version": "0.1",
  "publicIdentifier": "na",
  "category": "router",
  "lifecycleState": "active",
  "validFor": {
    "endDateDateTime": "2018-06-30T09:08:18.390Z",
    "startDateDateTime": "2017-12-12T10:08:18.390Z"
  },
  "@type": "logicalDevice",
  "@baseType": "resource",
  "@schemaLocation": "http://jra2t2.net",
  "usageState": "Active",
  "capacity": [],
  "capacityDemand": [],
  "note": [],
  "place": null,
  "relatedParty": [],
  "resourceRelationship": [],
  "resourceCharacteristic": [],
  "resourceSpecification": null,
  "resourceAttachment": [],
  "role": [
    {
      "id": "1",
      ...
B.2.2 Connection Termination Point Example

```json
{
  "id": "250",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/connectionTerminationPoint/250",
  "name": "dev1stp1",
  "description": "stp",
  "version": "0.1",
  "publicIdentifier": "na",
  "category": "pilot",
  "lifecycleState": "active",
  "validFor": {
    "endDateTime": "2018-06-30T09:08:18.902Z",
    "startDateTime": "2017-12-12T08:18:18.902Z"
  },
  "@type": "connectionTerminationPoint",
  "@baseType": "resource",
  "@schemaLocation": "http://jra2t2.net",
  "usageState": "Active",
  "capacity": [],
  "capacityDemand": [],
  "note": [],
  "place": null,
  "relatedParty": [],
  "resourceRelationship": [],
  "resourceCharacteristic": [],
  "resourceSpecification": null,
  "resourceAttachment": [],
  "role": [
```
{  
  "id": "4",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/role/4",
  "description": "stp",
  "name": "STP",
  "roleSelectionMethod": "RetrieveAll",
  "roleCombination": "all"
},
{
  "id": "1",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/role/1",
  "description": "eline",
  "name": "Eline",
  "roleSelectionMethod": "RetrieveAll",
  "roleCombination": "all"
}
],
"value": null,
"lrStatus": "Ok",
"serviceState": "InService",
"isOperational": true,
"network": {
  "id": "1",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/network/1"
},
"additionalInfo": null,
"mteAdministrativeState": "Unlocked",
"logicalAlarmReportingEnabled": null,
"logicalAlarmStatus": "Cleared",
"isMTEOperational": true,
"operationalState": "EnableInService",
"typeOfMTE": "ConnectionTerminationPointBidirectional",
"mappingMode": "MappableAtLowerRates",
"ctpState": "BidirConnected",
"supportedConnectionRates": "250000000",
"connection": {
  "id": "100",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/connection/100"
},
"connectionTerminationPoint": [],
"direction": "CarriersTraffic",
"typeOfTP": "BidirSource",
"vendorTPName": "mx1-1-7",
"endPointLabel": "vlan:933",
"deviceInterface": [
  {
    "id": "7",
    "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/deviceInterface/7"
  }
B.2.3 Connection Example

{
   "id": "100",
   "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/connection/100",
   "name": "LS-xxxx",
   "description": "active circuit segment",
   "version": "0.1",
   "publicIdentifier": "na",
   "category": "pilot",
   "lifecycleState": "active",
   "validFor": {
      "endDateDateTime": "2018-06-30T09:08:19.547Z",
      "startDateDateTime": "2017-12-12T10:08:19.547Z"
   },
   "@type": "connection",
   "@baseType": "resource",
   "@schemaLocation": "http://jra2t2.net",
   "usageState": "Active",
   "capacity": [],
   "capacityDemand": [],
   "note": [],
   "place": null,
   "relatedParty": [],
   "resourceRelationship": [],
   "resourceCharacteristic": [],
   "resourceSpecification": null,
   "resourceAttachment": [],
   "role": [
      {
         "id": "1",
         "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/role/1",
         "description": "eline",
         "name": "Eline",
         "roleSelectionMethod": "RetrieveAll",
         "roleCombination": "all"
      }
   ],
   "value": "na",
   "lrStatus": "Ok",
   "serviceState": "InService",
   "isOperational": true,
}
"network": {
  "id": "1",
  "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/network/1"
},
"additionalInfo": "na",
"mteAdministrativeState": "Unlocked",
"logicalAlarmReportingEnabled": true,
"logicalAlarmStatus": "Cleared",
"isMTEOperational": true,
"operationalState": "EnableInService",
"typeOfMTE": "Connection",
"connectionTerminationPoint": [
  {
    "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/connectionTerminationPoint/250",
    "id": "250"
  },
  {
    "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/connectionTerminationPoint/251",
    "id": "251"
  }
],
"isUniDirectional": false,
"operatingLayerRate": "250000000",
"logicalDevice": [
  {
    "id": "4",
    "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/logicalDevice/4"
  },
  {
    "id": "5",
    "href": "http://10.10.1.1111/api/v1/resourceInventoryManagement/logicalDevice/5"
  }
]
References

[ACAPI]  TMF 640 Activation and Configuration REST API
[CCS-SD]  CCS service definition document
[D8.5]  Service Provider Architecture Pilot v1.0
[ELSUG]  E-Line service user guide
[eTOM]  GB921 Business Process Framework (eTOM) R17.0.1
[EPL]  E-Line: Ethernet Private Line defined by MEF
[JM]  Apache JMeter
[NFV]  TMF NFV API documentation
[NSI2]  GFS.212: NSI Connection Service 2.0
[OBM]  TMF GB988 OSS/BSS Metrics definition document R16.5.0
[ONSA]  Open-source implementation of NSI
[PMAN]  Postman API Development Environment
[RIAPI]  TMF 639 Resource Inventory Management REST API
[RIADD]  TMF GB922_Service_Overview_R16.0.1
[TMFSS]  TMF service specification and catalogue
[SIAPI]  TMF 638 Service Inventory Management REST API spec R16.5
[SSPUG]  Self-service portal user guide
[TOA]  TMForum Open API Table (accessible by members only)
## Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BPF</td>
<td>TMForum Business Process Framework</td>
</tr>
<tr>
<td>BPM</td>
<td>Business Process Management</td>
</tr>
<tr>
<td>BSS</td>
<td>Business Support Systems</td>
</tr>
<tr>
<td>CCS</td>
<td>Consolidated Connection Service (also known as E-Line)</td>
</tr>
<tr>
<td>CFS</td>
<td>Customer Facing Service</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>CSP</td>
<td>Communications Service Provider</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DDS</td>
<td>Dynamic Data Specification</td>
</tr>
<tr>
<td>EDA</td>
<td>Event Driven Architecture</td>
</tr>
<tr>
<td>EMS</td>
<td>Element Management System</td>
</tr>
<tr>
<td>EPL</td>
<td>Ethernet Private Line</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>eTOM</td>
<td>TMForum Business Process Framework old naming</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>ITSM</td>
<td>IT Service Management</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>MEF</td>
<td>Metro Ethernet Forum</td>
</tr>
<tr>
<td>NMS</td>
<td>Network Management System</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Function Virtualisation</td>
</tr>
<tr>
<td>NSA</td>
<td>Network Service Agent</td>
</tr>
<tr>
<td>NSI</td>
<td>Network Services Interface</td>
</tr>
<tr>
<td>ONSA</td>
<td>OpenNSA Implementation</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>OSS</td>
<td>Operations Support Systems</td>
</tr>
<tr>
<td>OTRS</td>
<td>Open-source Ticket Request System</td>
</tr>
<tr>
<td>P2P</td>
<td>Point-to-Point</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFS</td>
<td>Resource Facing Service</td>
</tr>
<tr>
<td>SD</td>
<td>Service Definition</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Networking</td>
</tr>
<tr>
<td>SIAPI</td>
<td>Service Inventory API Data Model</td>
</tr>
<tr>
<td>SID</td>
<td>TMForum Shared Information Data</td>
</tr>
<tr>
<td>SLA</td>
<td>Service-Level Agreement</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SPA</td>
<td>Service Provider Architecture</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SSP</td>
<td>Self-Service Portal</td>
</tr>
<tr>
<td>STP</td>
<td>Service Termination Point</td>
</tr>
<tr>
<td>TAM</td>
<td>TMForum Application Mapping</td>
</tr>
<tr>
<td>TMF</td>
<td>TeleManagement Forum</td>
</tr>
<tr>
<td>TOA</td>
<td>TMF Open APIs</td>
</tr>
<tr>
<td>VC</td>
<td>Videoconference</td>
</tr>
</tbody>
</table>