Self Service Campus: an online tool for network management delegation

Cláudio Teixeira, João Paulo Barraca, Carlos Costa, Dimitri Silva
Universidade de Aveiro, Portugal
claudio@ua.pt, jpbarraca@ua.pt, costa@ua.pt, dimitrisilva@ua.pt

Abstract

Self-Service Campus (SSC) goal is to enable technical field staff to tke on daily tasks within campus. To complete their tasks autonomously, staff are required to access and configure network equipment, such as switches, to define port security and access rules. In a troubleshoot scenario, access to the same switches is required to take a closer look at logs. These are privileged accesses on sensitive equipment, typically using command line tools, meaning that extensive command knowledge is required. Even experienced technicians may, unwillingly, remove previous configurations, or misconfigure ports, with potential security issues or out of service situations.

The SSC is an online tool for graphic based network configuration, encapsulating a subset of possible operations using an API developed on top of Cisco Prime APIs (Application Programming Interfaces), combined with role-based access controls, with fine-grained access control to features on network equipment.

This paper presents the overall architecture set in place to support applications like the SSC in a sustained and generic way, with special focus on this particular implementation.

1 Introduction

Every organization is full of data, buried deep across databases, applications, servers, and systems, often with no integration with each other, and no means to interact directly with a given subset of data, or with a specific set of privileges. A common way to access the data and features revolving around is to use the specifically developed user interfaces that come with the systems, each with its own integration paradigms, protocols, and schemas. API and messaging systems are two ways to integrate information from different vendors, or within a new application, but the process is not trivial and requires careful and systematic movement towards an integrated system.

Despite these sound approaches, if there are too many systems that interact with each other, it becomes cumbersome to keep track of all the developed integrations is, and they may be rapidly outdated. Moreover, ad-hoc integrations mean that when the time comes to do maintenance on one of those systems, particular care is required informing all integrating parties and have them properly on board of the new integration method. The math behind these integrations is a simple exponential, given that within *N* systems, each may be connected to *N-1* systems. Direct integration also adds the burden to keep track of the individual endpoints and exposes system failure or maintenance downtime to other systems.

In a scenario of a campus, with a large set of services being provided, Enterprise Service Buses (ESB) may play a crucial role, acting as a one-point shop for integration. Typical ESB features include API usage analytics, security offloading and API transformation, besides the fact that decouple communication endpoints from service location and availability

Figure 1 illustrates a mesh-like integration between multiple systems, which is common in complex organizations comprising multiple services. It is noticeable that the availability or change to a service will have a direct impact on many other dependent systems, sometimes in a cascade of changes or even on a deadlock. Figure 2 illustrates the same integration logic when using a ESB as a central aggregation point. In this situation, there is a decoupling between service providers and consumers, with the ESB being able to limit the impact of schema discrepancies Although the ESB based architecture shows a centralized aggregation point, the implementation always relies on a redundant cluster of instances.

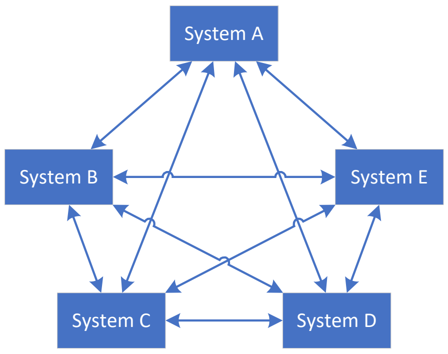


Figure 1 – Point to point systems integration

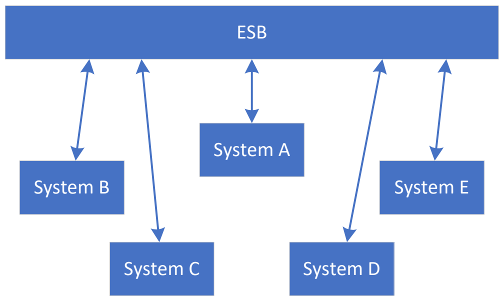


Figure 2 – ESB based integration

The reality of the information systems at University of Aveiro was a highly connected mesh, where service dependency was high. This both limited future developments and the resilience of the infrastructure, promoting the need for a leaner model to manage APIs. For this reason, we adopted an ESB driven model, currently based on the WSo2 software package. This work describes the rollout and the main outcome of the deployment, which we believe will provide insight to other similar organizations in their long-term development models.

2 Enterprise Service Bus rollout

The main goal with the adoption of an ESB was to fence and isolate each system, enabling a single point of contact (the ESB), decoupling API invocations from the underlying locator (an URI), while allowing to evolve, scale or split systems without an impact to availability. The perceived advantages were related to better understanding of which integrations are in place; better knowledge on the integration's usage (API analytics); systematic, cultural approach to the process of making APIs available. All these points are simply impossible or difficult to achieve in a mesh architecture. The nature of a mesh architecture grows in an organic manner, setting up new connections as needed for new features. Even if the integrations are correctly documented, it is simplistic to assume that they will be generic and have a high degree of reusability, instead of being specific for that integration. Visibility over integrations will be limited, and its comprehension will be even more limited. Standardized analytics will be another issue as the applications are developed over a long timeframe, relying on different languages and frameworks. The perceived disadvantages were related with slower pace to make the first iterations available; additional effort for systematical approach to the cultural changes required among the teams. Still, with the except of the slower initial pace, the additional effort could result in a more robust culture, where service provision is better planned, and there is higher visibility and discussion over its evolution.

Even though our main goal is the complete isolation of servers, with request to API requests not being rerouted through the ESB, this is not possible in an environment spanning some decades of developments. However, the disadvantages can be limited as a gradual approach can be followed. As the adoption of an ESB does not limit existing interactions, and, for the time being there is no limitation on what existing point-to-point integrations can do. We adopted a conservative model, to prevent service disruption, and moved towards an ESB, but where changes to endpoint systems will occur in a gradual manner.

3 Self Service Campus architecture

As with other large campuses, we have an infrastructure with tens of macro services, supported by a network following strict security policies, and with capabilities for enhanced isolation and reconfigurability. With the ESB in place, an API was set up that would interface with Cisco Prime, and, after data cleaning and handling, exposes the proper information for usage in the application.

The Cisco Prime API enables a wide range of operations, but by design there are no access roles or ways for limiting the specific features to use. The API provides either an all or nothing interface, as it is intended for operations with high visibility and control over the communication infrastructure. Therefore, the usage of the ESB enabled us to develop a proxy API whose responsibility is to act as a secure caller on specific Cisco Prime APIs, and then manipulate the data as intended. In fact, it enabled us to expose a system in a novel way, with high granularity and accounting of the actions performed, and effectively provided new services to our campuses. This way, the Cisco Prime API is out of reach of all systems, apart from this specific API proxy, which provided selected functionality.

We designed a new application, named the Self-Service Campus (SSC) application as a portal to allow interaction with academia, students, and staff to some networking aspects of the campus, following a stateless microservice approach to be feature centered, and to do not have any permanent storage or database. Some of the tasks implemented consist of rapidly extending network segments between the campus infrastructure, either through VLAN or other technology, allowing the rapid creation of limited environment for trials, practical laboratory, or lectures. This means that all information used in the application is sent through the ESB exposed API or through specific queries to switches. The switches are not fully exposed to the academic environment, and there is high visibility and accounting over the actions and the usage of the API.

Figure 3 depicts the overall architecture of the SSC. It holds a temporary database for caching data fetched from the switches, along with the switch list retrieved from the ESB API call. All commands instructed by the operator in the web interface are asynchronously executed against the respective switch, in a controlled environment. Currently we rely on an SSH executor, but others can be added as required by each switch.

The switch list is also filtered in the ESB API call, meaning that only predefined switches are available to be managed through the SSC. The predefined list is updated within Cisco Prime, and changes are instantaneous. As an added security layer, the retrieved switch list may be further filtered by the logged in user, meaning that a given user may be allowed to manage only a subset of these switches. Role based access control (RBAC) is also enforced at both port and allowed operations (commands) level, per switch.

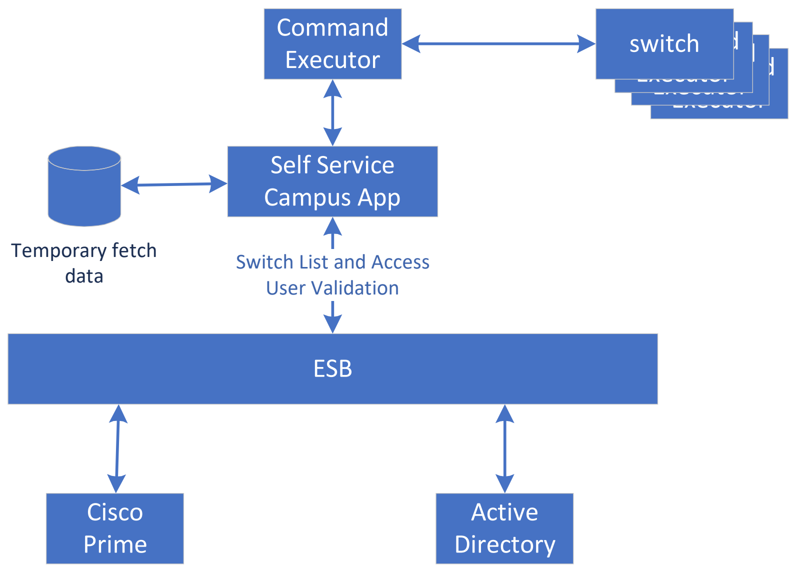
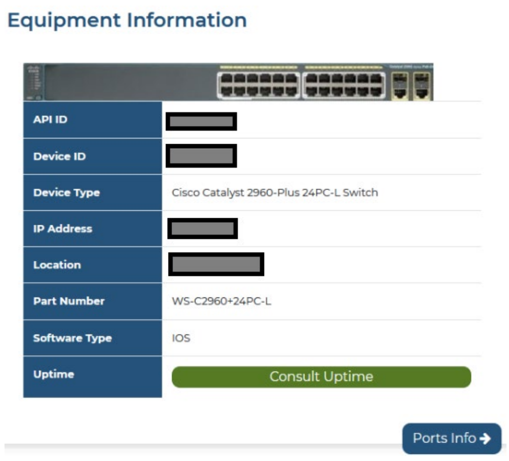


Figure 3 - Smart Switch Campus architecture

Current available approaches to handle this scenario do not support RBAC with such granularity. Also, this approach is not vendor specific. If properly deployed connectors are in place, the SSC can manage any vendor.

4 Look & Feel

The following figures illustrate the look and feel of the SSC. Figure 4 presents the summarized equipment information (model, IP address and location amongst the visible properties), whilst Figure 5 presents the interface list for a given equipment. As depicted in this figure, users can access state information regarding each port of the switch, as well as the active VLAN. Some of the information can be changed if the users' permissions allow such changes, which is an important departure from the access model followed by Cisco Prime.

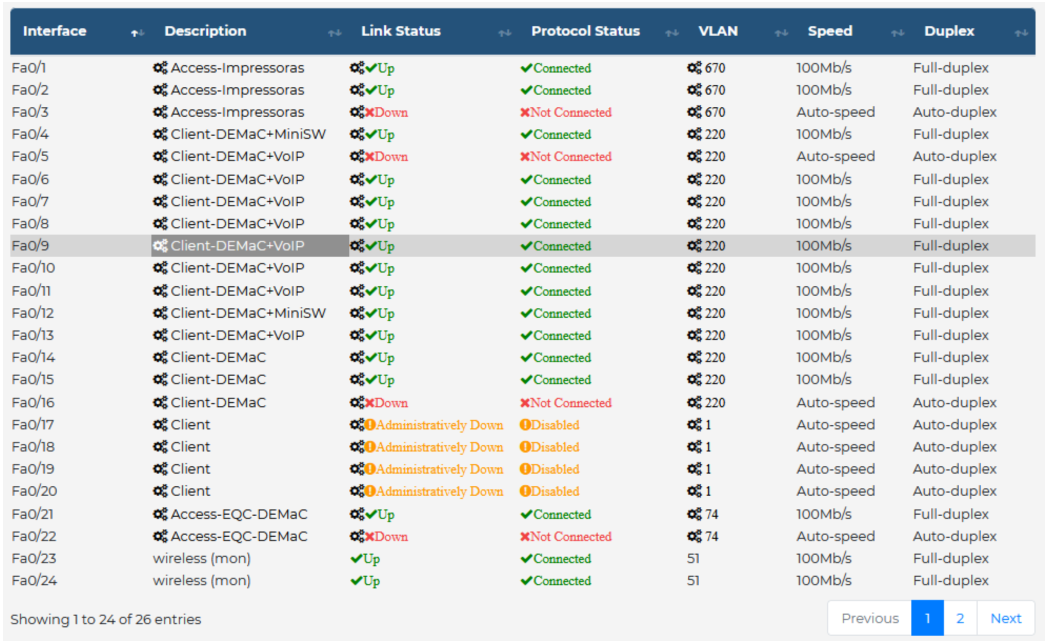


Equipment Information

API ID	[redacted]
Device ID	[redacted]
Device Type	Cisco Catalyst 2960-Plus 24PC-L Switch
IP Address	[redacted]
Location	[redacted]
Part Number	WS-C2960+24PC-L
Software Type	IOS
Uptime	Consult Uptime

Figure 4 - Equipment information

Using SSC, operators may only change information or act upon interfaces with a given subset of descriptors, all related to client port management, such as specific client VLANs or VOIP.



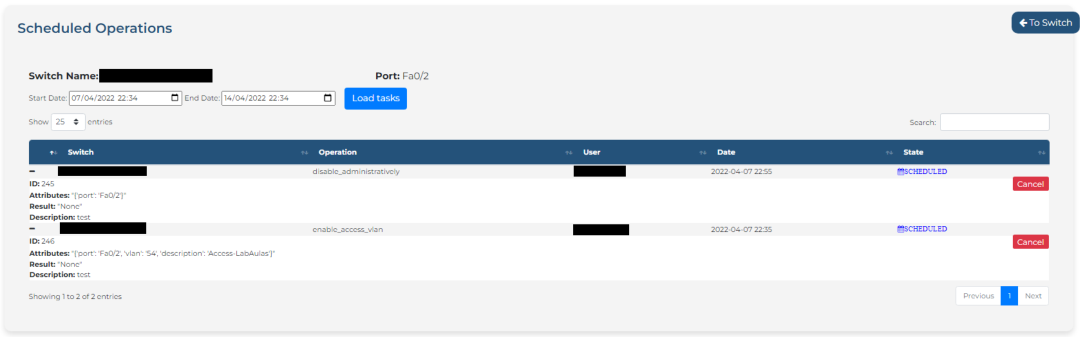
Interface	Description	Link Status	Protocol Status	VLAN	Speed	Duplex
Fa0/1	Access-Impressoras	✓Up	✓Connected	670	100Mb/s	Full-duplex
Fa0/2	Access-Impressoras	✓Up	✓Connected	670	100Mb/s	Full-duplex
Fa0/3	Access-Impressoras	✗Down	✗Not Connected	670	Auto-speed	Auto-duplex
Fa0/4	Client-DEMaC+MiniSW	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/5	Client-DEMaC+VoIP	✗Down	✗Not Connected	220	Auto-speed	Auto-duplex
Fa0/6	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/7	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/8	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/9	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/10	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/11	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/12	Client-DEMaC+MiniSW	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/13	Client-DEMaC+VoIP	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/14	Client-DEMaC	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/15	Client-DEMaC	✓Up	✓Connected	220	100Mb/s	Full-duplex
Fa0/16	Client-DEMaC	✗Down	✗Not Connected	220	Auto-speed	Auto-duplex
Fa0/17	Client	Administratively Down	Disabled	1	Auto-speed	Auto-duplex
Fa0/18	Client	Administratively Down	Disabled	1	Auto-speed	Auto-duplex
Fa0/19	Client	Administratively Down	Disabled	1	Auto-speed	Auto-duplex
Fa0/20	Client	Administratively Down	Disabled	1	Auto-speed	Auto-duplex
Fa0/21	Access-EQC-DEMaC	✓Up	✓Connected	74	100Mb/s	Full-duplex
Fa0/22	Access-EQC-DEMaC	✗Down	✗Not Connected	74	Auto-speed	Auto-duplex
Fa0/23	wireless (mon)	✓Up	✓Connected	51	100Mb/s	Full-duplex
Fa0/24	wireless (mon)	✓Up	✓Connected	51	100Mb/s	Full-duplex

Showing 1 to 24 of 26 entries

Figure 5 - Interface list

All actions are mandatory to have a description, which is sent along the instructions to the switch, and can be later seen within Cisco Prime or SSC.

One interesting feature is the scheduled tasks. Scheduling tasks enables operators to temporarily assign a given permission on a given port switch for a specific amount of time. Figure 6 presents the schedule task interface.



Switch Name: [redacted] **Port:** Fa0/2

Start Date: 07/04/2022 22:34 End Date: 14/04/2022 22:34

Show 25 entries Search:

Switch	Operation	User	Date	State
[redacted]	disable_administratively	[redacted]	2022-04-07 22:55	SCHEDULED

ID: 245
Attributes: "{'port': 'Fa0/2'}"
Result: "None"
Description: test

Switch	Operation	User	Date	State
[redacted]	enable_access_vlan	[redacted]	2022-04-07 22:35	SCHEDULED

ID: 246
Attributes: "{'port': 'Fa0/2', 'vlan': '54', 'description': 'Access-LabAulas'}"
Result: "None"
Description: test

Showing 1 to 2 of 2 entries

Figure 6 – Scheduled tasks list

5 Conclusions

We briefly presented the Self-Service Campus, an online tool for network management delegation, demonstrating the use of an ESB to integrate a restricted, user specific set of functionalities. The main goal of this application is to ease the load on expert network administrators, by providing means for field staff to carry on everyday activities and diagnostic, without requiring additional access to the administration interfaces of the switches. This application was designed on top of an ESB, enabling rapid adoption of security features like role-based access control and scope definition in the APIs, without changes to the underlying control API.
The application in being used at Universidade de Aveiro to manage all the switches and ports used for client connections, with new features are being suggested for future versions next version by receiving feedback from its users.
The usage of an ESB enabled us to properly handle and isolate the feature rich Cisco Prime API, and safely releasing just the required data. By having all access centralized in the ESB, we hold full control, from a management point of view, of all the applications that are accessing this API, and rich analytics. Using the same approach, Cisco Prime API encapsulation has been used on other applications, enabling a read-only access to specifically tailored and completely anonymized network metrics related to access points, Wi-Fi connections and network bandwidth usage on the *campi* buildings.