Data Center Portal Software Architecture

Sérgio Miguel de Albuquerque Alves
Instituto Superior Técnico
Lisbon, Portugal
sergio.alves@tecnico.ulisboa.pt

***Abstract*—NOS, a fusion between Zon, Optimus and most recently Mainroad, is one of the leading corporations in the portuguese telecommunications market. Its quest for providing quality and innovative services, demands for very flexible organic structures and processes, in order to ensure that its business requirements are being fulfilled within the required time to market, and following the required quality standards. In this thesis, we will describe the agile process of designing a software architecture, which started being development in ZON age, with the goal of supporting most of the underlying processes belonging to the Department of Data Center Operations. We will also look into how the architecture evolved, in order to cope with the challenges that rose from merges between Zon, Optimus and Mainroad.**

I. Introduction

The Department of Data Center Operations is responsible for ensuring the correctness and operationality within all NOS data centers and its equipments. It also acts as service provider, which exposes data center services, to the rest of the organization. It possesses a wide range of internal clients, which are responsible for different business areas within the company, such as: the Department of Information Systems, responsible for all the applicational layers from systems that support the daily activities within the company; the Department of Product Development, responsible for all the development of systems and services that stride to provide the best TV experience to the clients; the Department of Operation and Supervision, responsible for the systems that support all internet service provider systems; All the departments related to enterprise public offers, which are responsible for creating and maintaining the solutions for external corporate clients.

The services provided by the Data Center can be seen, partly, as Infrastructure as a Service (IaaS), Network as a Service (NaaS) and Backup as a Service (Baas), where all the provisioning and operations are done manually, rather than in automatized manners, as required by the full definition of IaaS, NaaS and BaaS. This means that, according to IaaS, a Data Center client can request for the installation of physical & virtual servers, he can requests for their removal, he can request for the addition & removal of computing resources such as cpu/ram/storage, he can request for the installation of a given operating system, he can request for a given software package to be installed/uninstalled, he can request for a given user to be created/removed from the corresponding operating system, etc.

By default, due to security guidelines, network traffic within Data Centers, from one network to another network, is blocked. Therefore, as part of its NaaS strategy, the client is allowed to request the creation of new network architectures which are assigned to specific new projects, he can request for the configuration of exception within the firewall flow rules, that unlock traffic from a given source network into a given target network on a specific port, he can ask for Virtual Private Network configurations, he can ask for Virtual IP balancing of systems, he can ask for the troubleshooting of issues related to connectivities problems.

Likewise, a portfolio of backups is provided to our client to ensure that if necessary, our clients will always have a way to go back in time and recover their data. This is very useful, especially when the clients are upgrading their platforms, and want to have a roll back plan in place, just in the case of something going wrong. Thus, the clients can schedule daily/weekly backups, into given points of their file systems, databases, etc. They can also request for one time backups, which happen only once on the specified date and time. Finally, they can ask for a given scheduled backup to be skipped once, on a given occasion.

All these services are maintained and performed manually by specialized teams within the Data Center Operations Department. While some of them specialize in virtualization, others specialize in Windows systems, Linux/Unix systems, networking, backups, monitoring and management of data center rooms. The handshake, between the clients and these teams, is done by a delivery team, who is responsible for assuring the quality of the delivered services by sponsoring certain processes such as: the control of the Service Level Agreements (SLAs), the dispatching of work tasks to the intervening operational teams, the early validation of the content provided inside the clients requests, the validation between what was requested and what was implemented before delivering it back to the client, and finally, the documentation of all the necessary operational information and processes within different repositories.

All the teams mentioned behind, except for the delivery team, are also responsible for solving incidents that may arise on the infrastructure and systems managed by the area. Incidents are events which impact a given managed system, in such a way that it affects their normal operationality. They can be caused by the side effects of intervening in the infrastructure, faulty configurations, erroneous human intervention, computing resources usage reaching their limits, etc. They are usually detected by the owner of a given system and reported to team responsible for supporting it. If corrected pre-

emptively, by the team responsible for supporting the system, who detects the issue through means of automated monitoring and alarms, the issue will not be raised as an incident.

A. Information Technology Infrastructure Library Framework

Most of the methods and processes described before are based in the Information Technology Infrastructure Library [1] framework. This framework is a set of best practices for IT Service Management that is widely used in telecommunication organizations. The processes it describes are generic enough to allow their implementation on any type of organization that seeks maturity and standardization in its processes and procedures. Since the subjects of its applicability are so wide, organizations can decide which bits to implement first to achieve their goals.

One of main core concepts introduced by ITIL is the existence of a Configuration Management Database[2]. The CMDB is a repository which holds the contents and relationships of Configuration Items. Configuration items are IT entities that can range from computers, laptops, blades, racks, virtual servers, networks, to business services, technical services, or any other entity defined within ITIL universe. The idea is to have everything meaningful to the organization registered, in order to be able to nominate it and keep track of all its relationships and history of changes.

ITIL defines a Request Fulfilment process that is responsible for the handling of Service Requests. Service requests are initiated by a third party which goes by the name of the requesting party and, usually end up in the queue of an implementation party. In this context, the data center department is one of the many implementation parties for service requests, while the other departments performing the requests, are seen as the requesting parties. Examples of service requests are the creation of a new user in the a given server, the creation of a new virtual machine, the installation of a given package inside a machine. Therefore, it is quite common that service requests affect the existing configuration items. When such happens, a relationship is created between the service request and the configuration item, relation which documents the intent, so that we may audit it later if required.

The request fulfilment also states that there should be tools for effectively and efficiently handling these service requests. The service requests must have traceable implementation times, which are defined within the Service Level Agreements, so that one may be able to measure the effectiveness and efficiency of a given implementation party. The service requests can also have a life cycle, which may start from the moment they are created, and go on until the moment when they are completed, while having in-between any other state deemed necessary (eg. authorization states, pending information states). The life cycle of a request is continuously monitored so that counter measures are in place, should the SLA be breached. It is also important to evaluate and check the results of a service request, in order to provide quality control on the deliverance of them, and to record useful knowledge for future use.

This leads us to the Change Management process also described within the ITIL framework. The CM process manages the alterations which will be performed on configurations items. When such alteration is prone to happen, an event named change is initiated. This event may or may not need an approval, and may have an impact and a risk associated. As result, it will trigger changes within the status of the affected configurations items. By definition, a configuration can be anything that provides value to the business, thus, it means that the change management process itself, can also be a configuration item. Hypothetically, if we were to improve the way this process is managed, we would be able to document the improvement within the ITIL framework, by creating a change upon the change management process configuration item.

If we contextualize together, the request fulfilment process and the change management process, we have service requests that will result in the creation of corresponding changes. Likewise, we have service requests that will never trigger the creation of a change, since no changes were made to any configuration item. A service request for upgrading the operating system of a given set of configuration items fits the former, while a service request to obtain certain system information from a configuration item fits the later. Therefore, we can say that a change will document what happened to a given configuration item on a given time. It is quite common to have environments[1] associated to configuration items.

B. Early Stages

Back in 2010, ZON, one of the companies that merged into NOS, was beginning to take the very first steps away from its chaotic maturity levels. Prior to it, the IT infrastructure management was done completely by an external entity. As infrastructure became more and more critical to the core operation of its business, the decision of bringing its infrastructure management and operation, back in-house was made. This transition had its very own challenges such as, the non existent documentation of the processes and the refusal to share precious know-how. The company itself had already adhered to the ITIL framework. Many of ITIL procedures were already implemented in its different departments, while the data center was giving the first steps to join them. In the market, there were various tools available to support ITIL processes and procedures, such as BMC Remedy, HP HPSM and Easy Vista. Each one of them with its different flavours of ITIL, different pricing, different challenges to implement. In our case, ZON had acquired and implemented HP Service Manager, with all its underlying modules for service requests, change management, CMDB and others. Consequently, the new infrastructure management team supported the processes which they are introducing on this mature tool.

The HPSM CMDB was gradually populated with configuration items related to virtual servers, blades, blade centers, storages, backups, switches, firewalls, etc. Tools like HPSM

[1] such as development, tests, staging, production

need to be generic enough, in order to allow their implementation in any type of organization domain. But in reality, virtual servers have different attributes than blade servers. Blade servers have different attributes than blade centers. In fact, blade servers live within blade centers. Thus, we have the issue of, depending on the type of configuration items, different sets of attributes may be required. If there are so many differences between attributes of distinct types within a same department, these differences become even more noticeable, when considering types belonging to separate departments. Does the chosen tool allow for different classes? Or does it allow for customization? How hard is it to manually manage the classes and its relationships? How hard is it to create new custom attributes? Thus, instead of modelling a tool to answer the needs of their problems, ZON was left modelling its solutions problems in ways which fit the tool. The end result was that the configuration items were created with general attributes filled with whatever made sense in that case, and the rest of the important details were simply populated into an open description text field in the configuration item.

Another interesting limitation of these tools worth mentioning, is related to how they implement the service request fulfilment. For the very same reasons mentioned above, the content of a service request, requesting the deploy of a new virtual machine, is completely different of the content of a service request, requesting for a new laptop installation to be delivered to an employee. Does the tool allow for the creation of different forms in accordance to each service request? Or does it only allow to write a generic description of what a user requests? Can these forms be detailed enough to fit our needs? Can they be internally created and altered in a timely fashion? Can they trigger actions in other tools automatically so automation becomes possible? The service request fulfilment in HPSM was, also, only a box of text where the requester could describe what he was looking for. That became a problem when one wants to standardize their processes, and make sure that all the necessary information is present when a service request is created. Back then, ZON, who was trying to implement some sort of standardization, was using excel templates per requests, which were distributed on demand to its requesters to be filled and then attached to the service request. Whenever there was a change to that excel template, there was no way to notify its clients that it had change and that they needed to download a new version.

Thus, it became clear that as the organization grew in size, so did it grew in complexity of its underlying domain. It was no longer possible to populate meaningful information, nor to support completely the intricacies of its micro processes, in just one central tool made to fit all. The main concept behind the in-house creation of the data center portal was to develop a system that serves as the main support to all the management and operational procedures of the data center department. It had to become the main entry point for its external clients and internal teams. The development itself had to follow agile and flexible methods, since the business logic to be implemented, was very domain specific and very personalized. And, all of this had to be achieved while abiding to a very important requirement: that there was only to be one official ITIL tool within the organization, the HPSM. The term System Federation will be used to describe a centralized ecosystem, where there are many surrounding systems, which have their own functions and details, but they all must feed their activities back into the centralized system. The centralized system may not be capable of receiving these feeds directly, but it must be possible to translate these feeds into its domain. These translations may imply the loss of quality or the downgrade of the information, but that is a no problem, as long as, the surrounding systems domain are merely extensions of the centralized system domain.

Following the concept of a federation of systems, we can designate the whatever is the official ITIL tool on an organization as the centralized system, and we can designate all the systems that try to extend on it, the surrounding systems. By ensuring that, no matter what specific domains and procedures are implemented on the surrounding systems, they must be translated back into the centralized one, we guarantee that we are not ditching away all the benefits of adhering to the ITIL framework. If a procedure can be partly translated back into the ITIL tool, it means that we are only expanding on the ITIL universe. Thus, the concepts behind the developing of the data center portal, are to merely expand on the centralized ITIL tool universe, and never to replace it. All the information managed by this portal had to be capable of being translated and mirrored back into the HSPM. After all, HPSM will be the source for any compliance evaluation, ISO certification and official reporting. With that in mind, the first challenge given to this portal was to systematize and standardize the reception of requests. It was important to get rid of the ad-hoc process of the service request fulfilment. Broadcasting excel templates on demand to clients was all but, elegant and efficient, since it led to all sort of unnecessary interactions, such as the ping pong validations from the client to the delivering teams.

II. Data Center Portal

The data center portal was to provide a catalogue of available services, each service having its own specific set of forms, validations and service level agreements. After creating a given request, the client was to be guided on how to access the HPSM tool, in order to follow the official service request fulfilment process, where he had to simply post the ID of the newly created and validated request from data center portal, instead of a open text description or the excel template filled by him. Later on, when the corresponding implementing team notices the newly generated SR ticket in HPSM, they were to access the organized and standardized information of the request by simply downloading it through the DC portal. This described process was a prototype, whose purpose was to test waters on how would the involved teams react to the creation of such a tool: both in terms of internal teams and external teams. This process was kept simple, as it tried to partly solve the validations issues that haunted the data center teams and its clients. And, as a trade off for that simplicity, it introduced

the extra step of requiring the usage of two tools in order to officially submit a request to our teams.

The Portal catalogue was first released with twenty one different services. They covered the all the necessities of the teams who manage the servers, who operate the operating systems, who operate the backups, who defined and configure demilitarized zones (DMZ)[3] in the network, and who deal with processes within the data center. Two months later, the number of available services had double to now include the necessities of the teams who deal with the connectivities universe. The connectivities universe included a wide range of services, such as firewall services, VPN related services, tunnelling services and dynamic network load balancing services. Throughout the first year, the varied services and forms were revisited and improved. The prototype was iterated and matured in such a way to support better modifiability and flexibility, in order to be able to answer the constant flow of new features and services. Later on, the needs of the database administration teams were also mirrored in the available catalogue, with the introduction of sixteen new services in the catalogue.

A. AD Integration & ACL Engine

There was no restriction on whom had access to the service catalogue of the data center portal. As long as the user was able to reach the server, he was able to create new service requests. Although the portal was inside a DMZ network that blocked unwanted accesses from outside networks, it was also published in the organization corporate proxy. This meant that anyone with that proxy configured in their browser was able to have unrestricted access to the portal. And, not only the user had unrestricted access through the proxy, but he also had a fake identity, since the portal web server was only able to see the proxy identity, and not the source user identity. If a user generated a service request and accidentally closed his page before copying the id, or if a user generated a service request and then decided not to map it in the HPSM, there would be no obvious way of recovering its content back to the user. So, in order to both create a sense of ownership and a sense of security, it became of the utter most importance to authenticate and authorize its users. In order to avoid the creation of yet another authentication mushroom, an integration with the organization Active Directory (AD) was followed and implemented.

The Active Directory [4] developed by Microsoft is a structured hierarchy of objects. Objects have their set of attributes, and they represent entities such as users, groups, etc. Since objects are organized in an hierarchy manner, objects such as groups can contain other objects. Usually, the objects held within an AD can be grouped into Organizational Units (OUs). Organizational Units allow for one to resemble the organization real structure. If we set NOS as an example, NOS was created from the fusion between two companies which had their own set of distinct users. There was a point in time where the AD had two distinct OUs, Zon and Optimus OUs. Also, within each object, it is possible to define a set of attributes. It is possible to define user policies regarding to the password renewal process. It is also possible to authenticate against an AD, since it can check a given password against the one set on the user object attributes. The Lightweight Directory Access Protocol (LDAP) [5] is used to perform queries against directories such as the ones from the kind of an active directory. LDAP allows for an authorized user to search the hierarchy of objects in order to find the object which matches the given query. Queries can be a combination of where in the tree an object stands, and what its attributes look like. Binding is another important operation in the protocol. It allows for an LDAP client to authenticate itself as a given user object from the AD. If a binding fails, it means that the given username and password did not match. Thus, it is a way of authenticate against an Active Directory.

Next to authentication, there is authorization. Authorization allows for one to define what is a given authenticated user authorized to do. An effective way of tackling this challenge is through maintaining an Access Control List (ACL) [6]. In general terms, an ACL is a list of permissions of a given object. It defines what operations, can and cannot a user or a group perform within a specific object. Defining an ACL at the user level is the most fine grained control we can get when specifying the subject for the operation. Dealing with ACLs at such level bring a huge burden when it comes to maintain the ACL, thus it is usually ideal to perform the specification on the group level. In terms of what operations can be performed in what object, it is also important to weight its granularity. The more granular we are, the better we can control the access to an object, but the harder it becomes to maintain it. To find a balance between these two is important for both security and usability. The DC portal ACL engine kept this balance in mind, thus it was implemented in such a way that the access was controlled on a group level, while the object operations were coarse enough to allow for their auto discovery.

Finally, we need to have mechanisms in place to be able to empower the Auditing of these accesses. These mechanisms must be able to log every event generated during the authentication and authorization phases. During the authentication phase, it is important to keep track of every successful or failed login attempt along side with its details. Such as the time stamp of when it happened, which user was tried to authenticate, the cause of failure, the source of this attempt, etc. During the authorization phase, it is important to log what operation in an object was attempted by a given a user. Besides the details mentioned above, we may also include if the access was granted or denied, as well as, the time stamp of when the decision took place. The mechanisms of Authentication, Authorization and Auditing brought an end to the era of uncontrolled access. In order to avoid disrupting abruptly what the data center portal's users were used to, the access to all its components was still left open, by the means of mapping its users into a global group with access to everything. Along side with it, it was finally possible to give a sense of ownership of requests to its users.

B. Workflow Engine

Undesirably, all portal users still had to interact with two tools in order to place a request or to check on their status. An improvement in this process was due, if we were to bring the whole request fulfilment process into the portal. In order to achieve this goal, two major components were amiss, the Workflow Engine and the automatic Integration with HPSM. The Workflow Engine job was to provide ways for one to define workflows that bring automation to our manual procedures. It is possible to define the different phases of a procedure in terms of a State Machine [7]. A State Machine is a set of states and their underlying transitions. These transitions are usually represented by conditions and results, meaning that, if the condition clause of a transition is met, we can transit into the target state while producing a given result. If we place some semantics upon the definition of results, we can implement them as being the action which takes place when a given condition is met, in order to reach a new state. Thus, defining a procedure becomes nothing more than specifying its several possible states, available actions and conditions.

Another important characteristic of the Workflow Engine is to keep track of all the workflow past. It must be possible to go back in time and understand when and which transitions happened. Besides the usefulness of having an history of transitions available for general purposes, it also allows for one to build reporting strategies on its raw data. Another important extension, is to empower the Workflow Engine with the capability of understanding ownership concepts. Since workflows can be used to define the life cycle of entities, it becomes interesting to imbue it with the means to tell: who performed a given action, to which group the entity was assigned, to which group the entity will be assigned, whom currently has the ownership of the entity. This way, conditions based on the ownership become innate to the engine and do not required extra logic programming in their definition. It also helps the workflow designer to model his workflows having in mind ownership. A given state can only be achieved, if a given action is executed by the group that is suppose to own the process in that specific moment. When an action is executed, as result, the target group of the workflow may change to a new one.

C. Invoker Manager

The other major component that was amiss, in order to centralize everything in one tool, was the HPSM automatic integration. Instead of forcing a requester to manually create a service request in HPSM with the reference id of the data center portal, it was important to have this done transparently. Instead of forcing our operators to manually map the Change Management in Service Request Fulfilment, or managing the underlying relationships between configuration items and these processes, it become important to also automatize this bit. If we consider that for every service request created, we have to map at least one configuration item. And that for every service request passed into the 'Work In Progress' state, we have to create a corresponding change, we have to create a relationship between the change and the service request, we have to create a relationship between the change and the configuration items, and etcetera. These manual interactions can easily bulk up to more than 1000 weekly actions that bring no other added value other than ITIL compliance. Thus, automatizing all these interactions, not only helped us step even further in the maturity level, but they also catalysed for significant gains in the efficiency of the daily routines, since the bureaucracy burden taken from all teams was extremely high. They also allowed for the request fulfilment process to be refined in order to add better clarity on it. In HPSM, it was not clearly what phases if any, a request needed to pass in order to be completed. By migrating all this procedures into the data center portal, we were able to implement our very own phases with our clients need for clarity in mind.

To guarantee that all interactions got mirrored in HPSM, these integrations took advantage of the Workflow Engine mentioned before. After the new request fulfilment workflow was designed, a way to map its state machine into the one in HPSM was drafted. It involved calling the available HPSM web services on each equivalent action. Therefore, if in the DC portal, there was an action where the request finally became assigned to an implementing team, the action would have to additionally call the corresponding set of HPSM web services that mirrored the states. Using this strategy, it became possible to map distinct but yet somehow similar workflows to best of the efforts. Despite of, the portal service request fulfilment having the responsibility of knowing how to map itself in HPSM through web services calls, it was not its responsibility to invoke them. This responsibility was passed down into yet another component that we will call of Invoker Manager. This component performed the role of a Message Queue[8]. The role of a Message Queue is to provide a way for asynchronous communications protocol. Asynchronous communications are characterized by the fact that applications sending messages, do not need to lock themselves and wait for responses. They can continue processing whatever work they have left, while providing a callback method which will be called once a response for that given request arrives.

The Invoker Manager was developed to function in similar ways. Its job was to receive an object that encapsulated an web service call (invocation) and perform it himself. Once, it received the response, it was then responsible for execute whatever was defined in the invocations callback. This way, the data center portal was able to proceed serving its clients, while in the background, the Invoker Manager made sure that all requested integrations by the portal were being invoked. The Invoker Manager was also responsible for dealing with orders and faults. The submitted invocations carried an unique group id which was used to tell that, within the group of all the invocations which carried that group id, order of arrival was to be respected. No invocation can be invoked before the previous one within its group has succeeded. At the same time, any invocation belonging to a different group id, may be invoked in parallel since they do not overlap one with another. This

property is of extremely usefulness especially when dealing with state machines models such as the ones implemented in HPSM. As we saw before, certain actions are only available in given states, thus, we must keep track of the order on which we call these actions.

In the event of a remote service being invoked becoming unavailable, or in the event of a remote service malfunctioning and returning unusual errors, we must ensure that the invocation is periodically retried until it is successful invoked or intentionally dropped. Thus, the Invoker Manager must be able to evaluate the received responses and according to their values, enqueue the invocation again for retry or call the corresponding callbacks for it was successful. This type of events may sound a bit unlikely to happen, but they do happen, more often than rarely. It is easy to come with simple scenarios where even though HPSM is functioning correctly, it might return error messages to its web services invocations. Lets take as an example the scenario where all the licensed sessions are currently being used in HPSM, if the Invoker Manager was to trigger an invocation on that interval, it will receive a response error reporting that the maximum number of sessions were exceeded. Lets say that the Invoker Manager wanted to invoke an update on a given HPSM service request, and that the given HPSM service request was currently locked by another user, the result of that invocation will be yet another error stating such. Therefore, it is of the utmost importance for the Invoker Manager to be well prepared to deal with this type of events.

Integrating with HPSM improved many of the internal and external processes, but there was a little detail that it overlooked. The internal teams still had to interact with HPSM to perform activities related to the task management. Let the task management be the process where the actual work is dispatched to the varied teams. This means that a given service request created in the data center portal will have to be sliced in one or more tasks for the different implementing teams. On that point in time, the HPSM task module was being used to achieve such. And just like any other module in HPSM, it suffer from the vary same efficiency issues. This led to the adoption of yet another tool that facilitated this process by the operational teams. Since the effort to develop such a module inside the data center portal did not make sense on that particular moment, the COTS of choice became Easy Vista [9]. There was a period where the data center delivery team was manually dispatching every new portal request into Easy Vista requests, requests which were then further split into tasks. Therefore, in order to improve the process by eliminating this manual creation process, an integration with Easy Vista was also sought and implemented taking advantage of the mechanisms created to integrate with HPSM.

D. Data Center CMDB

In 2013, ZON and Optimus were merged into a new company that temporarily went by the name of ZON Optimus. During that period only minor improvements which did not affect the software architecture of the data center portal were made. In the likeness of ZON, Optimus also had its own data center infrastructure, teams, tools and processes. For about an year, those two teams kept working separately, on their own old way and infrastructure. Meanwhile, the processes of both and possible synergies between each other were being studied. Optimus as an organization also based its processes in the ITIL framework. Unlikely ZON, Optimus supported all of its ITIL processes on top of an old version of the BMC Remedy tool [10]. The main entity of this tool was the Trouble Tickets (TT), and they can be seen as the equivalent of incidents tickets in HPSM. These trouble tickets were also being used as service request tickets. Additionally to Remedy, there was another tool that mimicked many of the features found in an ITIL tool, named Infranet. This tool had been developed in a custom manner to fit Optimus IT necessities by an outsource company. It included modules for incident management, service request fulfilment, task management and CMDB. These modules were partly integrated with Remedy, since corresponding TT were created but not updated by the tool. Teams used both tools seamless, depending on whether the source of a TT had been Remedy or Infranet. There were no defined service level agreements and no defined key performance indexes nor reporting being collected.

Later in 2014, NOS the new official brand for Zon Optimus was publicly announced. Internally, the distinction between ZON and Optimus teams, was supposed to cease to exist. This meant that a new strategy was in need to finally merge both realities into a single one. Until that moment, the data center portal had been adapted to allow requests incoming from the Optimus reality, being such requests then manually dispatched to the corresponding teams. The main strategy was maintained, it involved having the data center portal as the entry for everything. While having the data center portal mapping its information onto the relevant systems. It became known to all the areas that a new ITIL tool, different from HPSM and Remedy, will be officially implemented. Thus, the data center portal was to maintain its integration with the HPSM while ignoring any integration with Remedy, until the appearance of the new tool. This represented a problem itself, since HPSM CMDB had configuration items related to ZON universe, while Infranet CMDB had configuration items related to Optimus universe. Also, the data center portal was validating itself upon the HPSM CMDB, and if we were to simply import Infranet CMDB upon HPSM CMDB, a great amount of relevant information would get lost in the process. This was mainly because that Infranet CMDB was very specific to the data center universe, where HPSM CMDB was way too generic to any universe.

In reality, the Infranet CMDB schema had been developed internally by the ex-Optimus team. Infranet was simply a front-end that fed on top of that information, pretty much in the image of how the DC portal fed on top of HPSM CMDB. We decided to take ownership of this Infranet CMDB and through the methods of federation mentioned previously, to federalize whatever information was possible back into HPSM CMDB. This way complying with the ITIL procedures remained

possible while both universe became merged seamless and in a controlled way. To make this possible, the schema of the Infranet CMDB had to be updated to include attributes that were not intersected when comparing to the HPSM CMDB model. Then, an unified configuration item front-end was developed within the data center portal. Thus, the Infranet CMDB became a key new component within the data center portal architecture. Whenever a new configuration item was created or changed through the newly developed front-end, the equivalent invocations were sent to HPSM through web services, in order to maintain both universes consistent. This meant that the DC portal became the only component allowed to perform changes within the items of this Infranet CMDB.

Additionally, this Infranet CMDB was also augmented on its schema of the attributes necessary to introduce the concept of ownership of a configuration item. This was of extreme importance since configuration items started having a workflow life cycle themselves. Until this point, there was no notion of state within a configuration item, they either existed or did not. When in reality, there are many phases through which a configuration item can pass and different actions that can be executed depending on which phase it is. So, we had to expand the influence of the Workflow Engine in order to reach this new type of entity within the Infranet CMDB. From this point on, we will start to referring to this Infranet CMDB as simply being the Data Center CMDB. It became possible to easily track through the DC portal which servers one owned as a requester. Which operating systems they had, what were they specifications, what were their management IPs, what were their service IPs, how much storage was deployed on that host, etc. It became also possible to dive into the history of a given server and visualize which portal requests affected it directly. This type of functionalities greatly improved the user experience of both internal and external teams to the data center. After all, a data center operator troubleshooting an incident had now access to very detailed information of a given configuration item without having to access it directly. And by consulting its history, one can try to figure out if there is any relation between the incident and its most recent past of requests.

E. Task Management Module

Easy Vista licensing model was revealing itself too expensive, especially when considering that within the new NOS universe, we needed to double the access allowance licenses. If investing on the in-house development of a task management module did not make much sense before, with the definitive merge of companies it was now a main priority. It was important to normalize the way internal teams manage their work and while at it, to reduce the costs derived from the tools supporting that process. Former Optimus team members were still receiving their work in the previous defined ways except for the requests that followed the new standard processes. Those type of requests were arriving to those teams through email. That method of dispatching was not without its very own problems: they easily got lost in the heap of mails, there was no way of measuring their times, it was hard to understand the contents of the request. A possible solution to this problem was to bring these new teams under Easy Vista too, but as mentioned before, the costs of doing such were not justified. Plus, by developing this module in-house, it was possible in a flexible and agile way, to improve the whole process of how people receive, dispatch and perceive work. As processes grew more complex, perceiving the work that needs to get done was no longer a trivial matter.

As the new requests arrived in an orderly manner, they had to have passed through some phases of approval before they reached the point where they start getting implemented. It was when they reached such phase, that the corresponding tasks were then created. By corresponding tasks, we mean that this new system had to map templates of tasks into services. So, a given service had its own set of tasks, with their title, description, OLA, delivery group, target group and attributes. Around the moment that the system was going to instantiate these tasks from their template, it had to define what extra time to add to their OLA due to the number of configuration items affected on that concrete request. If the request affected many configuration items, the extra time was to be partitioned between all the tasks. A dashboard with pending work was then required, in order for the teams affected by those tasks to be able to identify easily the pending work. Then, when they concluded, cancelled, paused, resumed their tasks, it was important for the corresponding request to show up in the dashboard related to requests, as pending an evaluation of progress. This was mainly because delivery teams needed a mechanism that enabled them to see requests where further actions were required from their part. Take for instance the scenario where a given task was completed within a given request with multiple tasks, the delivery team needed to be able to know that a task had been completed and that the following tasks were pending a submission by their part. There was also the cases where a request had more priority over the others, even though it had been chronologically requested after them. The system needed to allow for one to mark it as urgent, and to change its target date while updating the target date of its corresponding tasks.

It also needed to empower the cases where a given request or a given task was unable to carry on because some necessary details were still missing. In these cases, where the requester needed to be enquired, it was important to update all the SLA, OLA and estimated dates to take into account all the time spent in external teams. A new estimate was to be calculated based on it, and reported back to the original client. Thus, perceiving the existent work was no longer a matter of managing a first in first out (FIFO) queue. There were lots of variables and micro processes that needed to be weighted before making the decision of what was to be implemented next. Therefore, it was important to buff the system with the capability of providing all this information in an easy to read way, in order to make the decision making process straight forward. The engine that empowered the DC portal was by then mature enough to easily implement all these kind of functionalities. Thus, removing

TABLE I
VOLUME OF PORTAL REQUESTS SINCE THE BEGINNING OF TIMES

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
2012	-	-	17	111	162	136	197	169	255	334	293	220	1894
2013	322	270	265	286	333	245	398	250	414	453	396	266	3898
2014	324	358	361	350	375	328	426	275	363	364	414	362	4300
2015	440	470	589	522	410	420	521	399	505	-	-	-	4276

Easy Vista, not only became a matter of reducing costs but also a matter of simplifying the architecture and computational resources. Since, it was no longer needed to have a connector that implemented the integration with the EV, as well as, the computational resources upon which the EV ran.

III. USAGE DATA

In table I, we have a representation of the volume of requests, made within the data center portal, since the time it was first put into production. The first release dates back to 12nd March 2012. We can see that throughout the first month, only a dozen of requests were made through the data center portal. This mainly because the data center clients were still being informed and educated of the new process. Consequently, in the following months, we saw a steady increment of the requests performed via the data center portal. By the end of its first year, the mark of 1894 requests had been achieved, which amounts to an average of 189 requests per month, 9 requests a day.

TABLE II
CLASSES OF PORTAL REQUEST VOLUMES MAPPED TO THE NUMBER OF USERS WHO FIT IN THOSE CLASSES

No of requests	No of users
$]0-10]$	259
$]10,30]$	27
$]30,50]$	39
$]50,80]$	20
$]80,120]$	14
$]120,180]$	9
$]180,250]$	5
$]250,+\infty[$	2
Total	375

In 2013, the number of requests performed through the data center portal doubled, with an average of 324 requests per month, 16 requests per day. This is explained by the fact that some of our clients did not adopt the system right away when it was released. And only by the end of the 4rd quarter of 2012, we had everyone requesting through the portal. In 2014, the year where ZON and Optimus became the new brand NOS, we had a mark of 4300 requests, with an average of 358 requests per month and 17 requests per day. The increment is not as many as expected for a fusion of two companies, mainly because internally, both ex-Zon and ex-Optimus still existed. Finally, by the end of the 3rd quarter of 2015, we have already achieved the mark of requests of the previous year. This is mainly because 2015, is the year where the processes within the data center department started to get standardized for both areas.

In table II, we have the volume of requests divided in classes, and we have the number of users which fit within each class. The important to retain from this data is that until this date, we have had 375 distinct employees or collaborators within NOS, performing requests inside the data center portal. Also, we can see that we have a couple of users which fit inside the classes with a large amount of requests. This is justified with the fact that, just like the data center department has its own delivery team, there are other departments in the organization which have their corresponding delivery teams, too. Thus, instead of giving the access to all the employees within these departments, these departments opted to pipe those requests through their delivery teams.

IV. CONCLUSION

When the project first started, we did not put too much emphasis on its architecture. Its early stages were all about speed, and none about software quality. The immediate result of that was a double edged sword: lots of functionality appeared from night to day, in terms of presentation and usability, it was very user friendly and attractive, but, on the other edge, code was just being copied and repeated all over. Thus, the system grown at the fast pace on which the stakeholders came up with new ideas but, as soon as the stakeholders started wanting to change functionality instead of adding, the house of cards fell. Since code was duplicated all over, a simple change, a simple bug correction, would have to be repeated throughout many modules. This led us to stop, sit and think on how to bring qualities that had been overlooked in the conception of the system. We learned that even a prototype should have a well defined underlying architecture, especially the sooner, the better. That because, sooner or later, if the prototype is successful, it will no longer be seen as a prototype, but rather as system in production. And, introducing an architecture into a chaotic system, is way more complex than simply put some effort in drafting the architecture from its early stages.

When we finally started drafting the architecture for the system, we felt that not only it was very important consider the stakeholders requirements, but as well as prepare for the future by embedding the architecture with modifiability and extensibility qualities. If the stakeholders asked for a certain functionality, there is a high chance that they will alter it in the future, and then go back to what it was, over and

over. When we are talking about in-house development of a tool that supports processes, it has to be flexible enough to empower the changing of the processes. The processes should not need to adapt to the system, instead, the system should adapt to the processes. These intra-department processes are in a constant quest for the self-improvement, and only by having flexible system which can evolve easily with them, can they be implemented efficiently. Another topic worth mentioning is that throughout the development cycle, we were influenced into introducing external systems maintained by third parties into our architecture. Downtimes on these third party systems do happen, and when they do, most likely, we will not be informed of them. Thus, it is important to architect these integrations with the assumption that the attempts to integrate will fail quite often. Only with a fault tolerant integration components, we will have our system safeguarded.

As a final remark on the conception of this system: we believe that introducing continuous integration to our architecture is something that we need to consider in the near future. The system so far has been designed and developed by just one person, but having the logbook completely full with new stories to develop, more and more the need of expanding the team becomes noticeable. Thus, we believe that introducing testability and continuous integration into our architecture is going to become very relevant in the near future. Once again, we can only hope we are not too late for that.

V. Future Work

As future work, a series of well defined requisites have already been identified. These requisites span a wide range of subjects, and they are mainly focused on the expansion of both functionalities and catalogue. A catalogue, tweaked towards supporting the areas which operate the systems belonging to NOS external clients, is being drafted and implemented. These development will not impact the architecture since all the required qualities are already set in place. In other words, it is the simple introduction of a new family of services, just like happened in the very first iteration of the architecture.

One of the most urgent requirements is the enrichment of the `CMDB DB` component with new classes of configurations items. Yet again, the architecture is equipped with the means to empower this development without any change on it. Similar to what was done in the `CMDB` phase after the architecture was defined, new data access objects classes will be defined for the new classes of configuration items, new workflows will be also defined for them, new controllers which expose the workflow actions for each new entity will be created, and the corresponding views will be implemented.

With some initial trials already under belt, there is the integration with automation orchestrators, in order to empower uses cases which allow for the automated provisioning of virtual servers. This development will require a new interoperability scenarios to be introduced, which means that new web services integration packages will have to be developed while adhering to the already defined architecture.

Somewhere in the future, the system will have to support the managing of available network segments. It will also have to be capable of managing the knowledge base of the existing data center client platforms. In order to achieve that, it is predictable that no additional changes will have to be performed inside the defined architecture. Instead, in the very image of the Task Management phase, the current portal domain will have to be extended in order to support these new business logics.

There is also a very important integration, pending at the moment, which is the migration of all the underlying integrations, from the HPSM, to the new ITIL tool, which is slowly being implemented within the organization. This integration will imply deprecating the integrations introduced for the HPSM, by eliminating this component from the architecture, and at the same time, it will imply introducing the corresponding integrations with the final tool, which will replace HPSM as new component inside the architecture. Thus, one may say that this development will be very similar to the HPSM part of the integration phase described before.

Finally, at the bottom of the queue, with no estimated date of release, there is the budget management module, which will be responsible for creating the notion of charge back within the requests performed by the data center users. No architectural changes are predicted on this phase, since the implemented systems already provide the necessary qualities in order to implement the underlying domain logic of this budget management module.

Acknowledgment

I would like to thank all the members, that are or have been part of the data center team, for their continued support, trust and opportunities given. It is nice to be part of a project which can directly improve other people daily work-life. I would also like to thank for all the support and feedback given by professor António Rito Silva.

References

[1] itSMF UK, ITIL Foundation Handbook. The Stationery Office; 3rd ed. (2012) edition (January 31, 2012), 2012.
[2] G. Donnell, The CMDB imperative how to realize the dream and avoid the nightmares. Upper Saddle River, N.J: Prentice Hall, 2009.
[3] M. Thomatis, Network design cookbook : architecting Cisco networks. Raleigh, N.C: Lulu, 2012.
[4] R. Allen and L. Hunter, Active directory cookbook. OReilly Media, Inc., 2006.
[5] K. Zeilenga, Lightweight directory access protocol (ldap): Technical specification road map, 2006.
[6] R. S. Sandhu and P. Samarati, Access control: principle and practice, Communications Magazine, IEEE, vol. 32, no. 9, pp. 4048, 1994.
[7] P. Adamczyk, The anthology of the finite state machine design patterns, in The 10th Conference on Pattern Languages of Programs, 2003.
[8] P. Avgeriou and U. Zdun, Architectural patterns revisiteda pattern, 2005.
[9] "EasyVista ITSM, ITAM, IT Consumerization" http://www.easyvista.com/.
[10] "Remedy - IT Service Management Suite - BMC", www.bmc.com/it-solutions/remedy-itsm.html.