Flexible and dynamic observing at the ESO Very Large Telescope

T. Bierwirth*[a], B. Amarandei[b,a], G. Beccari[a], S. Brillant[c], B. Dumitru[d,a], S. Mieske[c], M. Pasquato[a], M. Pruemm[d,a], M. Rejkuba[a], P. Santos[a], L. E. Tacconi-Garman[a], I. Vera[a]
[a]European Southern Observatory, Karl-Schwarzschild-Str. 2, D-85748 Garching, Germany;
[b]Top IT Services, Inselkammerstraße 1, D-82008 Unterhaching, Germany;
[c]European Southern Observatory, Alonso de Córdova 3107, Vitacura - Santiago, Chile;
[d]Informate International N.V./S.A., Stationstraat 46, Bus 44, B-3620 Lanaken, Belgium

ABSTRACT

Until recently, users of ESO's Very Large Telescope had to prepare Observing Blocks (OBs) with a standalone desktop tool. Tool support for automated OB mass production was mostly limited to imaging public surveys. Furthermore, there was no connection between the OB preparation software and other ancillary tools, such as Exposure Time Calculators, finding chart preparation software, and observatory schedule, meaning that users had to re-type the same information in several tools, and could design observations that would be incompatible with the Service Mode schedule. To address these shortcomings, we have implemented a new programming interface (API) and a state-of-the-art web application which provide observers with unprecedented flexibility and promote the usage of instrument and science-case specific tools, from small scripts to full-blown user interfaces. In this paper, we describe the software architecture of our solution, important design concepts and the technology stack adopted. We report on first user experience in both Visitor and Service Mode. We discuss tailored API programming examples, solving specific user requirements, and explain API usage scenarios for the next generation of ESO instruments. Finally, we describe the future evolution of our new approach.

Keywords: ESO, VLT, Observation Preparation, Automation, Programming Interface, Web Application

1. HISTORY OF OBSERVATION PREPARATION

At ESO's La Silla Paranal Observatory (LPO), which includes the 4m class telescopes, survey telescopes, the Very Large Telescope (VLT) and the largest optical interferometer (VLTI), Observing Blocks (OBs) are the atomic unit of observation definition and execution. Observations are either carried out by ESO staff on behalf of the investigators in *Service Mode*, or by themselves at the observatory site, in *Visitor Mode*. For more than two decades, the standard software application to prepare OBs for the VLT was the ESO-developed Phase 2 Proposal Preparation (P2PP) tool[1], that went through several major releases while maintaining its underlying software architecture. When P2PP was conceived, a single requirement had a tremendous impact on the chosen architecture: the investigators should be able to carry out their observation preparation offline without network connection. Online connectivity should only be required at the very beginning of an observation preparation session to download the *observing runs* for the logged-in user, which provide the context under which OBs are created, and their associated instrument packages, which define the parameters to be specified for each instrument. After that, the investigator should be able to work offline and only require a network connection again for the final check-in of the observation material to ESO. At the time when this requirement was imposed, it was certainly a wise choice given the unavailability of ubiquitous, fast network connectivity.

2. P2PP: INITIAL SOFTWARE ARCHITECTURE

As illustrated in Figure 1, the chosen software architecture for P2PP versions 2.x and 3.x is a classical three-tier "fat client" architecture, consisting of a Java desktop client application with a graphical user interface communicating to a Java application server hosted at ESO, which persists OBs into ESO's relational database at our headquarters in Garching. In addition, the P2PP client integrates a filesystem-based (i.e. server-less) relational database[2] that serves as the local storage for newly created OBs – what we refer to as the *local cache*. Once all OBs are defined in the local cache, the investigator has to take different steps to finalize the observation preparation depending on the chosen observing mode. For Service

*tbierwir@eso.org; www.eso.org

Mode, OBs have to be *checked in* into the ESO database, after which they immediately become read-only in P2PP's local cache, so that ESO's User Support Department can safely review the submitted material.

If further OB editing is required, the investigator has to check out the OB first, which deletes it from the ESO database, edit it in the local cache and eventually check it back in. Finally, the Service Mode OBs are uni-directionally replicated from the Garching database to the Paranal database, where they are read from by the Service Mode observing tool (OT) and executed.

For Visitor Mode, the investigator has to export the OBs from P2PP's local cache to a text file format (OBX) and save them on a local disk. If the visitor is present on the mountain and working on a Paranal machine (the traditional Visitor Mode scenario), those files are then transferred to the Paranal control network by the Observatory staff via ftp. In Designated Visitor Mode*, the staff receives the OBX files via email and transfers them internally to the corresponding workstation. Then, the OBs are re-imported with an additional manual action into the local cache of the Visitor Mode observing tool (vOT) for execution, again performed by the Observatory staff.

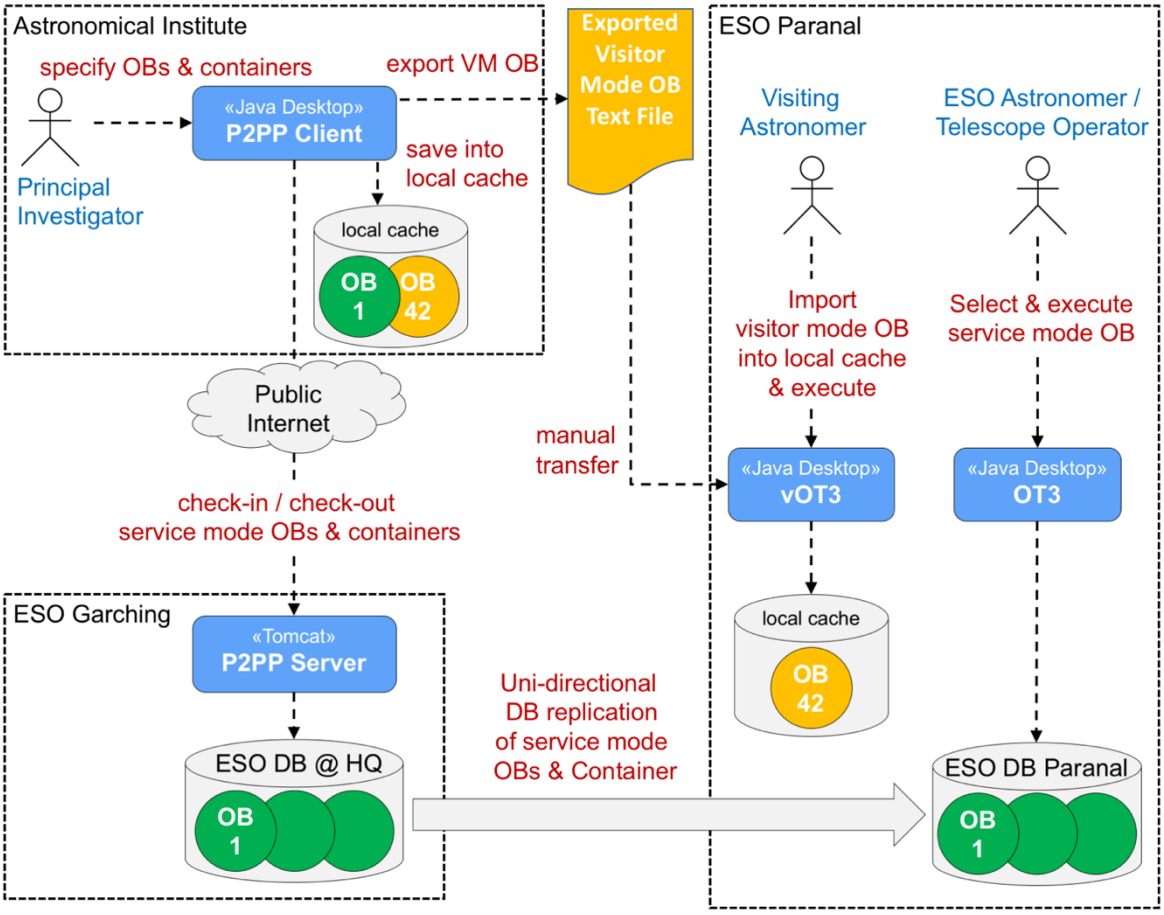

Figure 1. Initial ESO software architecture for service and visitor mode observation preparation

As long as we were only supporting so-called *loose* OBs – as opposed to OBs in scheduling containers – this architecture served us very well due to two key success factors:

- **Transactionality of relational databases** – Although conceptually the OB is an atomic unit, it has a significant underlying database schema consisting of a number of tables and relationships. Both when working against the

* Designated Visitor Mode observations on Paranal are scheduled on specific dates/slots as if they were regular Visitor Mode runs, but they are executed by an ESO staff member, in close contact (e.g. via phone, Skype or video link) with the Principal Investigator (PI), or someone the PI designates to serve as the liaison with the Observatory.

local cache as well as when working against the ESO database, we could fully leverage the transactionality of relational databases, allowing us to rely on the infrastructure to guarantee that an OB is either fully checked in into the ESO database or written into the local cache or not stored at all and thereby ensuring that no data corruption can occur.

- **Simple check-in/check-out paradigm for loose OBs** – An OB is only editable in P2PP's local cache when it is checked out of the ESO database or when it was never checked in. In both cases the OB does not even exist in the ESO database. Consequently, complex OB state synchronisation between conflicting OB changes in the local cache and in the ESO database is not needed.

3. EVOLUTION OF REQUIREMENTS

After the initial release of P2PP into production and first operational usage, a number of requirements driven by new scientific needs were incrementally implemented, improving the efficiency of ESO's user support and the observing experience at the telescopes. The arrival of the requirement for scheduling containers[3] – such as time links, groups and concatenations of OBs in order to express advanced, longer-term observing strategies – started challenging our software architecture and implementation, because we could no longer rely on the success factors discussed in the previous section. It became obvious that our software architecture eventually had to fundamentally change due to a number of new requirements and limitations:

- **Loss of database-level transactionality** – Scheduling containers used by large surveys may contain hundreds of OBs and may easily require 15 – 30 minutes to check-in into the ESO database. The sheer number of SQL inserts makes it impossible to execute such a long running action in a single database transaction with potential rollback in case of errors. We realized that we had to implement our own container-level transaction management on application level in both the P2PP client and server. For instance, loss of network connectivity during a long running container check-in, might lead to a situation in which the P2PP client would not know if the last OB being checked in never arrived at the ESO database, or if it did but the acknowledgement was never received.
- **Complex check-in/check-out paradigm for partially executed scheduling containers** – The requirement for scheduling containers also implies that an investigator has to be able to edit pending OBs of a partially executed scheduling container. Instead of checking out individual loose OBs, we therefore had to introduce the notion of checking out the entire scheduling container into P2PP's local cache. However, those OBs of the scheduling container that had already been observed, obviously had to remain in the ESO database and could not be deleted. That means with the introduction of scheduling containers consisting of >1 OB, we now had to maintain a state that could go out of sync in two locations and take care to recover a consistent situation with cross-OB dependencies.

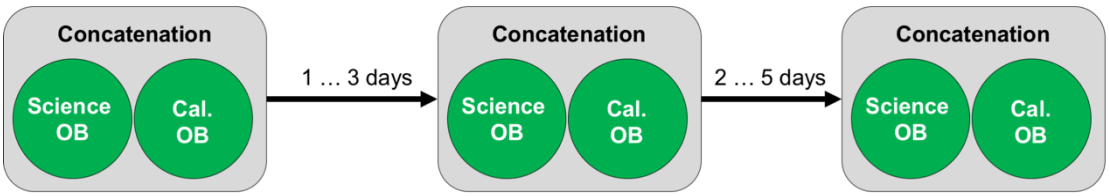

Figure 2. Example of a nested scheduling container: time link of concatenations of pairs of a science and calibration OB

- **Nested scheduling containers** – Investigators who routinely use concatenations to execute pairs of a science OB and a calibration OB (e.g. for VLTI visibility calibration or telluric correction in IR spectroscopy), would hugely benefit from being able to specify and execute time links of concatenations of OBs, i.e. containers of containers or *nested* scheduling containers, as depicted in Figure 2. For example, this would allow them to express relative time delays between pairs of concatenated science and calibrator OBs to be executed "back to back" in order to design a time sequence of observations for monitoring the scientific target's variability. However, evolving the check-in/check-out paradigm to support nested scheduling containers would be complex, expensive and not deliver a particularly intuitive and simple user experience.
- **Programmatic mass production of OBs without P2PP** – The only entry point to checking in OBs and containers into the ESO database remained P2PP. While investigators of large observations such as surveys with hundreds

or even thousands of OBs were happy to be able to use scheduling containers, the direct, fully automated mass production of valid, verified OBs into the ESO database with a dedicated script or other tool was limited. The usual workaround was to mass-produce OBs in the export file format OBX, run the P2PP client, manually import the OBX files into the local cache and finally check them in into the ESO database. This is a rather involved workflow that cannot be fully automated and comes at the risk of the OBX file format becoming invalid due to changing instrument capabilities described in an updated instrument package. While we offer a Survey Areas Definition Tool[4] for observation preparation on the VISTA and VST telescopes that produces dedicated XML files with an equally dedicated import into P2PP to simplify the definition of large surveys, this solution is specific to the definition of survey pointings for these two instruments only and the major limitation of having to manually run P2PP remains. Phase 2 observation preparation essentially remained a "closed shop" with the graphical user interface P2PP as the only entry point, whereas a part of the community increasingly expressed the need to carry out fully programmatic and automated OB preparation bypassing P2PP.

- **Dynamic OB editing throughout ongoing observing period** – The VLT's new ESPRESSO instrument has the primary scientific objective to hunt for exoplanets, requiring support for the definition of a prioritized Visitor Execution Sequence of OBs, access to the OB execution status and dynamic, unsupervised real-time editing of (Visitor Mode) OBs throughout the ongoing observing period. For instance, based on radial velocity analysis of previous observations, investigators wish to edit OBs and their priority in the Visitor Execution Sequence. This is another case where programmatic rather than user interface access to OB preparation fits perfectly.
- **Visitor mode OB transfer to Paranal** – The manual transfer of visitor mode OBs to Paranal by means of OB export to OBX, carried out by the investigator, email/ftp transfer to the Paranal control room and subsequent re-import into vOT by ESO staff, is labour intensive, not a particularly smooth user experience and late editing can only be done inside the Paranal control room. In particular for designated visitor mode, this is a showstopper for real-time OB editing by the user at home.
- **Java desktop limitations** – The implementation in terms of a Java desktop application led to a number of limitations. The number of problems caused by different operating systems and Java versions that had to be troubleshot by user support were significant. The rollout of a bugfix or new feature required users to download a new version of P2PP and copy their local cache into the new installation, or even lose the local cache and its contained OBs in case of incompatibilities with the new version. All of this led to increasing dissatisfaction of the community with P2PP's usability.

The points above were the scientific and operational drivers that forced us to entirely rethink our end-to-end phase 2 software architecture. While, in fact, we fully implemented the requirement for scheduling containers with the standalone P2PP and have been observing large surveys on various telescopes such as VISTA and VST making extensive usage of those advanced observing strategies, **we arrived at a point at which the majority of our software development efforts went into the complexity of the above sketched transaction management and state synchronisation, rather than into delivering scientific and operational value.**

4. P2: THE NEW SOFTWARE ARCHITECTURE

Figure 3 shows a structural view of the new and fully operational software architecture. It was significantly simplified by dropping the requirement to support *offline* OB preparation. Investigators are expected to have a reasonable public internet connection. The main architecture and technology decisions are as follows:

- **Web-based observation preparation** – The desktop P2PP client application is discontinued. Instead, investigators can carry out observation preparation in a moderately recent web browser (Firefox, Safari, Chrome) executing our *single page application* p2[5], a demo of which is publicly accessible at https://www.eso.org/p2demo. This approach minimizes operating system dependencies and provides a zero-install user experience, allowing us to transparently roll out bug fixes and new features without having to ask the community to download a new version of the application. This is a tremendous advantage that almost naturally allowed us to transition to a development process in which we frequently publish new features in a "devops" spirit, incrementally gathering and incorporating feedback from selected early users. Additionally, p2 allows to display and edit a Visitor Execution Sequence per user per instrument as shown in Figure 4.

- **All content created live against the ESO databases in Garching and Paranal** – All OBs, including Visitor Mode ones, and scheduling containers are immediately created in the ESO database in Garching and bi-directionally replicated to the ESO database in Paranal. The paradigm of checking OBs or containers in and out is entirely discontinued, and consequently the concept of a local cache in P2PP and vOT is no longer needed. We implemented a major new release 4 of our Visitor Mode observing tool (vOT) for Paranal, which no longer has a local cache but reads, creates and edits OBs directly against the Paranal database. Additionally, just like p2, vOT4 allows to display and edit a Visitor Execution Sequence as shown in Figure 5. This real time editing is instantaneously mirrored into the database in Garching via the bi-directional replication. Such feature is especially advantageous for designated visitors who are observing at night while not physically present in the control room.

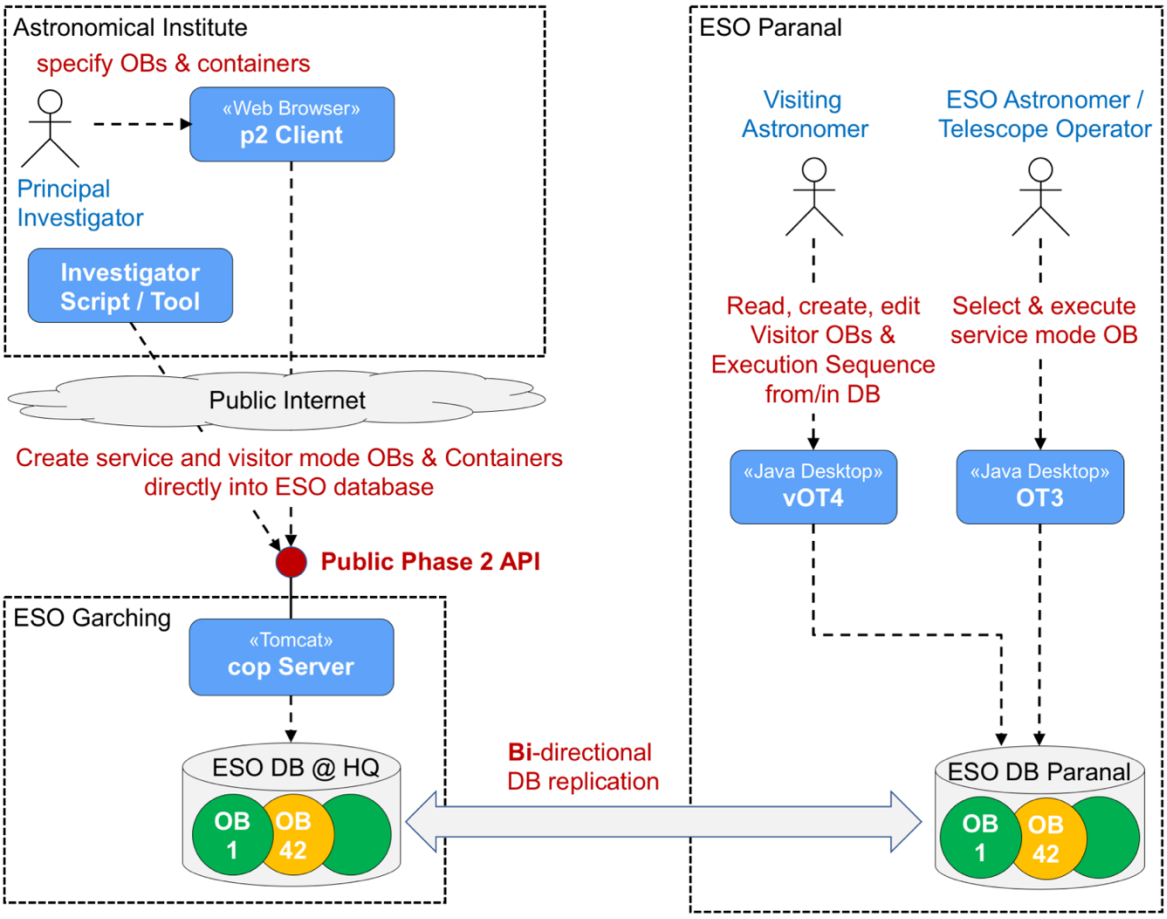

Figure 3. New ESO software architecture for Service and Visitor Mode observation preparation

- **Exposure of business logic as public APIs –** Rather than building a monolithic solution with tight coupling of user interface and business logic, we strictly separate these two layers. The business logic is exposed in terms of public application programming interfaces (APIs) using the REST[6] architectural style, which is lean and simple in both implementation and usage as a client application, very widespread and well understood by software engineers. While carefully designed business logic exposed via APIs can be long lived and stable, user interface technologies, specifically on the web, tend to age much faster. This approach ensures complete separation of obsolescence lifecycles. It is possible to implement a new user interface in a different technology without having to re-implement the APIs. In addition, REST APIs are perfectly suited for comprehensive test automation with both functional and non-functional test cases (performance, concurrency, scalability) allowing us to reach high software quality at an early stage and throughout the project by means of continuous integration.

  The business logic is implemented in a new web server application *cop* (**C**reation of **O**Bs and **P**roposals), that initially exposes the phase 2 API only, but is being extended to also expose a phase 1 API for proposal submission.

The complete phase 2 API, with a total of ~70 end points, is comprehensively documented online[7] using the formal specification language RAML[8], so interested investigators can develop science-case specific scripts or other tools. The cop server is implemented using the feature-rich web framework Grails[9], a *convention-over-configuration* framework that enforces architectural styles and provides powerful abstractions for recurring implementation tasks such as object-relational-mapping to databases. The amount of technical boilerplate code is massively reduced, thereby improving code readability and allowing developers to focus on the business problem. We have been using Grails for almost a decade and experienced major improvements in implementation productivity and maintainability.

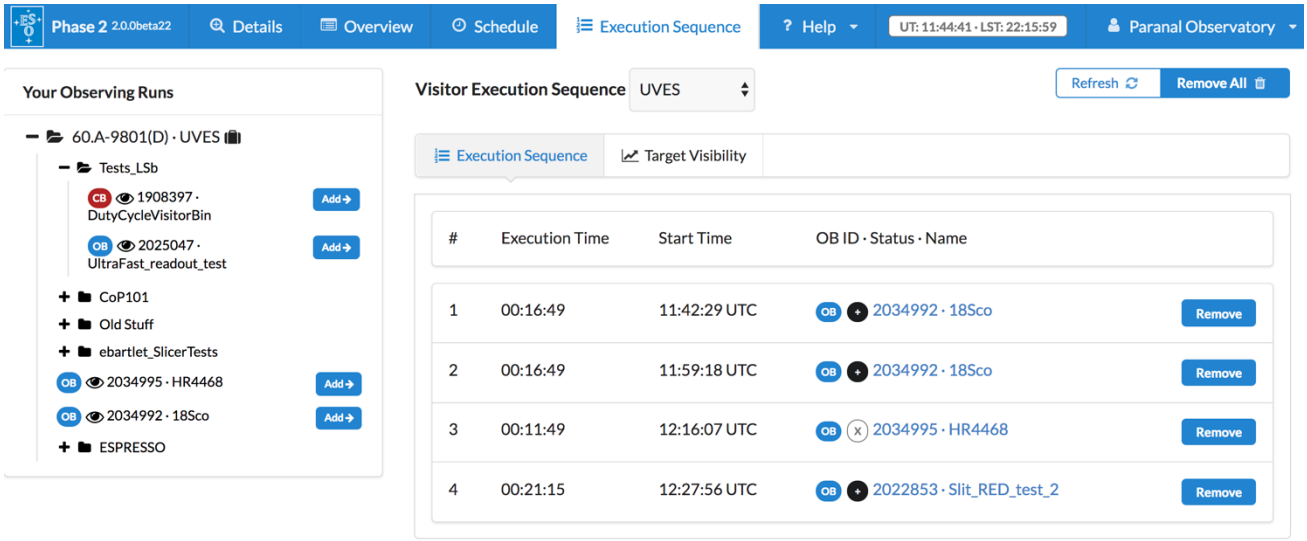

Figure 4. Visitor Execution Sequence in p2

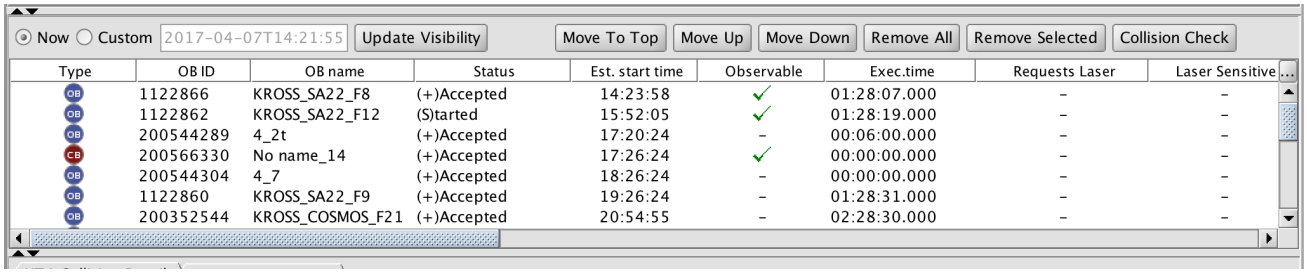

Figure 5. Visitor Execution Sequence in vOT4

- **Web user interface as single page application using the Angular framework** – The increasing adoption of REST APIs throughout the software industry led to a parallel development of very powerful client-side web frameworks allowing to develop large, dynamic so-called single page applications that are capable of bringing the rich features of desktop graphical user interfaces to web browsers. We chose to implement our p2 client in Google's Angular framework[10] starting with version 2 for several reasons. First, the framework is backed by a comparably large developer community. Considering the alternatives, it covers the widest range of required features and – probably most importantly – it allows implementation in the more strongly typed, object-oriented TypeScript[11] language, alleviating one of our biggest concerns, the difficult scalability, maintainability and testability of JavaScript code. After our first release of p2, we also quickly learned that a consistent architectural pattern for maintaining client-side state is extremely important to ensure UI consistency and minimize dependencies. Therefore, we introduced the ngrx/store client-side state container[12] throughout the application. Currently, the number of lines of TypeScript code is approximately 15000 and expected to grow much further. Programming Angular has a significant learning curve and software engineers have to learn and adapt to functional reactive programming[13]. As opposed to our server-side framework Grails, Angular did not boost our

productivity, but it allowed us to realize a dynamic and feature-rich user interface running in the web browser that was probably unthinkable only 5 years ago.

Figure 6 shows p2's main, desktop-style master-detail view showing observing runs, folders, scheduling containers and OBs on the left for navigation purposes, and details of an OB on the right, in turn structured into a number of navigable tabs. Figure 7 shows an interactive target visibility plot in terms of the target's airmass over time, the moon elevation and whether requested observation constraints are fulfilled.

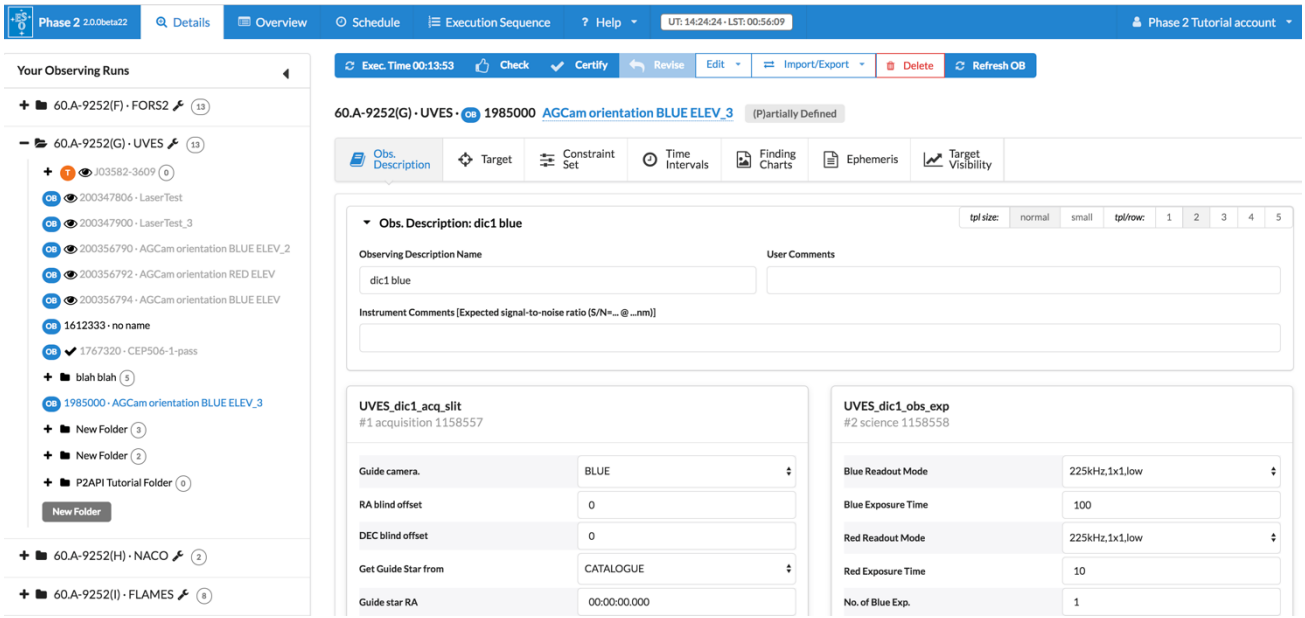

Figure 6. p2's main master-detail view showing runs, folders, containers and OBs on the left and OB details on the right

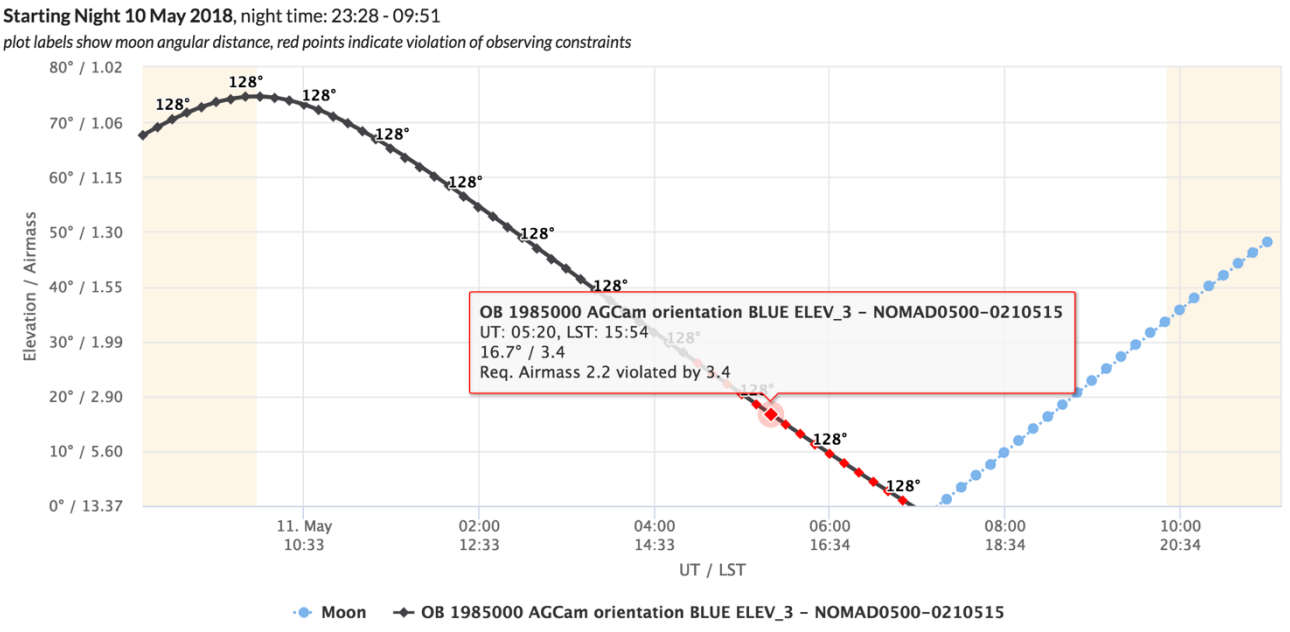

Figure 7. A target visibility plot in p2 showing whether requested observing constraints will be fulfilled

Looking at the phase 2 API from a user interface point of view, we managed to keep the required network bandwidth low and provide a smooth user experience. Even from Paranal, p2 is reasonably usable with a roundtrip delay to Garching of ~260ms and a bandwidth of 1–2 Mbit/s. If needed, the user interface performance can be significantly improved by executing independent API calls concurrently rather than sequentially.

- **Bi-directional DB replication** – The transition from uni- to bi-directional database replication was a major challenge. Our fundamental replication approach is asynchronous, i.e. in case of network outage or congestion, pending changes on either side of the Atlantic are buffered in replication queues. This ensures that both sides can continue operations in such situations rather than waiting for the opposite side to respond. However, since all Visitor Mode OBs are now also in the databases in Garching and Paranal and can be freely edited either with the p2 web application from anywhere or with vOT in the Paranal control room, there is a certain likelihood that concurrent changes to the same OB or to the Visitor Execution Sequence are conflicting, specifically in Designated Visitor Mode, when the investigator is remotely connected for a visitor night and uses p2. What if the vOT user on Paranal removes a template from an OB while the p2 user at home changes a value in that template? What if the p2 user moves an OB to first position in the visitor execution sequence while the vOT user removes said OB from the sequence? Our two main rules for replication conflict resolution are (a) delete wins against conflicting edits regardless of which side triggered the delete and (b) in case of other conflicts Paranal wins. This required implementation of significant conflict resolution logic in the DB replication infrastructure layer. This solution has been used for operations at the telescopes since October 2016 and it works very reliably. The advantage of this solution is that any OB created or edited with the p2 web application or the phase 2 API against the Garching database is almost instantaneously visible on Paranal. Vice versa, any change carried out on Paranal using vOT is immediately visible via web application and API. Therefore, the API allows for monitoring of OB status and execution progress.

5. PROGRAMMING THE PHASE 2 API

The phase 2 API has to be programming language agnostic so that it can be programmed against in any language, the only prerequisite being the availability of an HTTP protocol implementation. Following a REST[6] architecture, the API addresses each resource by a dedicated URL, maps the creation, editing, retrieval and deletion of a resource to the respective HTTP request methods POST, PUT, GET and DELETE, and documents each end point in detail[7].

```
# login
api = p2api.ApiConnection('demo', '52052', 'tutorial')

# create folder under an observing run
runContainerId = 1538878
folder, folderVersion = api.createFolder(runContainerId, 'P2API Tutorial Folder')
folderId = folder['containerId']

# create OB
ob, obVersion = api.createOB(folderId, 'My First OB')
obId = ob['obId']

# edit OB's user priority
ob['userPriority'] = 9
ob, obVersion = api.saveOB(ob, obVersion)

# attach Acquisition Template
acqTpl, acqTplVersion = api.createTemplate(obId, 'UVES_blue_acq_slit')

# attach Science Template
scTpl, scTplVersion = api.createTemplate(obId, 'UVES_blue_obs_exp')
```

Figure 8. A simple Python script using the phase 2 API

While this would be enough to start programming, we felt the need to better encourage and promote API usage by providing a Python-specific binding[14], i.e. a small glue layer with API methods such as *createOB()* that simply execute the appropriate HTTP call with the correct method and URL. Given an existing Python installation, the binding can be installed with a single terminal command *pip install p2api* and the user is ready to start coding. Figure 8 shows a very basic Python script that first creates a folder under a given observing run, then creates an OB inside that folder, changes the user priority of the OB and attaches an acquisition and a science template. We also publish a comprehensive tutorial on how to program the API in Python[7], covering the creation of OBs and scheduling containers, the editing of OBs and their templates, the attachment of finding charts and ephemeris files, the population of the Visitor Execution Sequence, the verification steps and the final notification of submission to ESO.

Some example numbers to illustrate API performance are as follows: in one hour, we managed to import ~3000 complex OBs (with attached finding charts and parameter files) using a typical private, asymmetric DSL connection with ~6 MBit/s downstream and ~500 kBit/s upstream. To achieve this performance, parallel API calls must be made to saturate the upstream network. Most API calls take less than 250ms, many even less than 100ms. Some bulk calls may take several seconds up to minutes. The key to good performance is the concurrent execution of independent API calls with up to 4 connections to the server.

6. PHASE 2 API USAGE

The primary user of the phase 2 API is our own p2 web application. Both the API and p2 have been incrementally rolled out into production. Visitor Mode is already fully operational for all Paranal instruments, while Service Mode is operational on UT2, VISTA and VST since early 2018, and will be extended to the remaining instruments by Q3 2018. We will soon be able to decommission the P2PP version 3 client and server. Work is already underway to also upgrade our La Silla observatory site to the infrastructure described in this paper, which will allow us to finally decommission P2PP version 2 as well.

Another major API client is ESO's unified GuideCam Tool[15], our platform to realize instrument-specific observation preparation requirements that go beyond what can be accomplished in p2, such as guide star selection, stepping through telescope offsets and the production of finding charts. For the foreseeable future, the GuideCam Tool remains a Java desktop application, since it integrates and heavily relies on the Aladin Sky Atlas[16]. Currently, the GuideCam Tool supports the VLT instrument VIMOS, VISIR, HAWK-I and MUSE, and support for more VLT instruments will incrementally be added. In order to work with an existing OB, the user selects the OB in p2 and then opens the GuideCam Tool. In the GuideCam Tool window, there is a button that fetches the OB from p2 (also connecting to P2PP version 3 and vOT version 4). Upon pressing this button, the OB details (pointing coordinates, instrument, observing mode) are retrieved and GuideCam displays the image of the sky and the focal plane setup, allowing the user to select the guide star, adjust the pointing coordinates, position angle, offsets, or blind acquisition parameters for a faint target. When the user has completed the work in the GuideCam Tool, all produced information, such as guide star coordinates, telescope offsets and finding charts, can be pushed back to the OB with a single button press. This entire workflow is enabled by the phase 2 API.

Another API usage scenario is the Python script fcmaker[17] that produces custom MUSE and HAWK-I finding charts and attaches them to OBs.

For future survey facilities at ESO – e.g. MOONS, 4MOST – it is expected that mass OB production via API will be a very useful and indeed indispensable feature.

We would like to emphasize that both a p2 user interface demo[18] and a phase 2 API demo[19] with a dedicated tutorial account are publicly available. This demo environment is entirely separated from our operational one for reasons of security and production data integrity. It can be safely used to learn how to program against the API, to experiment with observing strategies and feasibility, and to train new users.

7. SUMMARY & OUTLOOK

We have made a major progress in realising our technical strategy to prefer web over desktop development whenever possible, to separate UIs from business logic and to make our business logic publicly available for creative and science-case specific usage by exposing it via carefully designed APIs. All OBs, regardless of whether they are Service or Visitor Mode, are available in the two ESO databases in Garching and Paranal which are fully synchronized in near real time, so that edits on either side are almost instantaneously visible on the other side. Visitor Mode OBs are now also seamlessly

transferred to the Paranal control room and can be quickly and easily modified. A Visitor Execution Sequence per user and instrument is available to support dynamic adjustment of observation priorities, for on-site and designated Visitor Mode.

We have now reached a modern, sustainable and extendible software architecture that enables us to implement a wide range of observing strategies in a user friendly, adaptive and scalable way. The introduction of the phase 2 API paves the way for also exposing instrument-specific observation preparation features (e.g. optimize adaptive optics performance, find instrument guide stars) implemented by the community as scriptable APIs, rather than shielding such functionality away in a desktop tool inaccessible for scripting. For Service Mode OBs, the introduction of nested containers will allow us to much more easily enable typical observing strategies for VLTI and NIR spectroscopy. Furthermore, we will update our tools to more adequately prioritize adaptive optics observations on site.

An important future development is to expand the integration with other tools, such as the upcoming new Phase 1 Proposal Submission and the Exposure Time Calculators (ETCs). Data already specified in the phase 1 observation proposal will be seamlessly transferred to phase 2 and the ETCs, thereby providing a seamless workflow experience to further simplify the observation preparation. Finally, we plan to port the new phase 2 end-to-end solution to our La Silla observatory site, such that the next generation La Silla instruments can take advantage of the same benefits.

REFERENCES

[1] ESO User Support Department, "The P2PP Tool (version 3)", https://www.eso.org/sci/observing/phase2/P2PP3.html
[2] "H2 Database Engine", http://www.h2database.com/html/main.html
[3] Bierwirth, T., Szeifert T., et. al., "New observing concepts for ESO survey telescopes", Proc. SPIE 7737 (2010)
[4] ESO User Support Department, "SADT – The Survey Areas Definition Tool", https://www.eso.org/sci/observing/phase2/SMGuidelines/SADT.html
[5] ESO User Support Department, "p2: a web-based tool for the preparation of your Phase 2 material", https://www.eso.org/sci/observing/phase2/p2intro.html
[6] Wikipedia, "Representational state transfer", https://en.wikipedia.org/wiki/Representational_state_transfer
[7] ESO, "Phase 2 Programmatic Web Interface", https://www.eso.org/copdemo/apidoc/index.html
[8] "RAML – The simplest way to design APIs", https://raml.org
[9] "Grails – A powerful Groovy-based web application framework for the JVM built on top of Spring Boot", https://grails.org
[10] Google, "Angular", https://angular.io
[11] Microsoft, "TypeScript – JavaScript that scales.", https://www.typescriptlang.org
[12] "@ngrx/store – RxJS powered state management for Angular applications", https://github.com/ngrx/platform/blob/master/docs/store/README.md
[13] "RxJS: Reactive Extensions for JavaScript", https://github.com/reactivex/rxjs
[14] Bierwirth, T., "Binding for the ESO phase 2 programmatic API", https://pypi.org/project/p2api
[15] ESO User Support Department, "The unified GuideCam Tool", https://www.eso.org/sci/observing/phase2/SMGuidelines/GUCT.html
[16] CDS, Strasbourg Observatory, France, "Aladin Sky Atlas", http://aladin.u-strasbg.fr
[17] Vogt, F., "fcmaker", https://fpavogt.github.io/fcmaker
[18] ESO, "p2 demo", https://www.eso.org/p2demo
[19] ESO, "Phase 2 API demo", https://www.eso.org/copdemo