A META Archive Providing Unified Access to all Operational Data at the German Space Operations Center

Stefan A. Gärtner* and Armin Braun

German Space Operations Center GSOC, DLR Oberpfaffenhofen, 82234 Weßling, Germany

During the lifetime of a satellite mission all kind of data are produced. This does not only encompass science and housekeeping data, but also configuration data, design documents, operational notes, anomaly reports, to only name a few. One of the challenges during spacecraft operations is to find and correlate data of various sources in order to perform successful contingency analysis and recovery, long-term analysis and situational awareness.

Collecting and consolidating data from different sources is the purpose of a distributed software system called "META archive", which is developed and used at the German Space Operations Center (GSOC) at DLR in Oberpfaffenhofen. The system does not aim at replacing the various archives or document management systems already in place, but at providing a condensed and searchable view across the whole set of mission data. As such it is suitable for starting in-depth analyses as well as for providing situational awareness.

This paper highlights the purpose of the META Archive, its design and concrete implementation at GSOC. It will be shown how META fits into the existing mission operations system and how it can be used by system engineers to ensure efficient spacecraft operations.

Nomenclature

API	Application Programming Interface
CRUD	create, retrieve, update, delete
GSOC	German Space Operations Center
HMAC	Hash-based message authentication code
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
REST	Representational State Transfer
SQL	Structured Query Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

I. Introduction

DATA being generated throughout the lifetime of a mission—usually starting years before the launch—often is available at many different locations in the mission operations system. Design documents, operations procedures, telemetry files, generated plots, and command definitions are only but a few examples of these data. Often also different versions of essentially the same document can be present. Finding a desired "product" can lead to a search in different archives, with each of them possibly exposing a different search and retrieval interface. The META archive aims at providing a unified view of these data without duplicating any of them. Instead, it allows the user to retrieve data from where they are. An easy to use yet powerful search mechanism is at the center of META also providing the possibility of cross-correlating data from different sources. The user is enabled to start an analysis or can track down causes of some data not being generated. In-depth analysis is left to specialized tools, so that the scope of META can be clearly defined. Collection and consolidation of mission data is left to so-called "generators". They extract meta-information associated with data and send it to a central service. While the central service is shared across missions—with proper security measures in place—some generators will be

*Mission Control and Data Systems Engineer, Mission Operations, DLR German Aerospace Center, Münchner Straße 20, 82234 Weßling, Germany, ORCID: `http://orcid.org/0000-0002-4077-9851` very mission-specific. A well-defined and modern REST interface[1] allows quick and easy implementation of generators for new data products and thus poses only little overhead in mission instantiation.

The central service ingests all meta-data sent to it by indexing, correlating and analyzing. System engineers and operators are presented with a web user interface that allows faceted searching and drill-downs in the data heap as well as accessing data directly if possible. The system has been designed to allow horizontal scaling to accommodate more users, future missions and higher demands for data ingestion. The central service is only central in the sense of a logical system view; physically it is made up of scalable components.

II. Definitions

Due to the overloaded and often somewhat unclear meaning of terms like "product" a definition of the META archive's underlying concepts shall be given here. How these concepts map to a working system is topic of later sections.

Back-end The central system component, which reacts to generator and user requests. It is neither physically nor logically a monolithic structure, but forms a conceptual unit.

Back-end instance A concrete deployment of the back-end. The term "back-end" is used synonymously with "back-end instance" because we are mostly concerned with a single back-end deployment. Where the difference is important, it will be pointed out.

Generator A conceptual unit that generates product metadata from product instances using a product definition.

Generator instance A concrete deployment of a generator.

Product The general notion of a certain type of data important to mission operations.

Product instance A concrete set of data representing a single atomic manifestation of a product. Usually produced during mission operations and external to the META system.

Product definition A collection of properties ("fields") and associated information used to describe a product instance in the framework of the META system.

Product metadata Concrete data conforming to a product definition. Created by a generator using a single product instance as input. This is the kind of data processed by the back-end. Often just called metadata.

Project An organizational unit usually corresponding to a mission. Generator instances, product instances and product metadata always belong to exactly one project.

The term "product" will be used synonymously with "product", "product instance", or "product metadata" in this paper. The intended meaning can be deduced from context. Where this is not possible the full terms will be utilized. Likewise "product type" and "product definition" are interchangeable, the latter being used in more formal descriptions.

III. Design

III.A. High-Level Design

The META archive is not a monolithic system but composed of several smaller components. However, in order to keep deployment and maintenance efforts comparably low and not complicate reasoning about the system we decided against a full-blown microservices architecture.[2] As such the system has a central server component at its core, implemented as a single application process using the Java programming language. It is possible to refactor parts of this application into a microservices architecture in future if need arises, which is detailed in Section V.B. The server component is accompanied by third-party services such as a database which run in their own processes. Together they comprise the back-end system. META exposes a user front-end for graphical or programmatic interaction in order to perform queries or administering the system. Furthermore, it exposes a second contact area for the so-called generators to interface with. Generators are processes that transform a product instance to its metadata representation, which they subsequently submit to the back-end system. The number of generator processes is flexible and one may view the generators as microservices. They communicate with the back-end via a well-defined interface that is not tied to any particular implementation technology, but a standard set of generators is supplied that are implemented in Java. A pictorial overview of the design concepts is shown in Figure 1. Details on the deployment can be found in Section V.

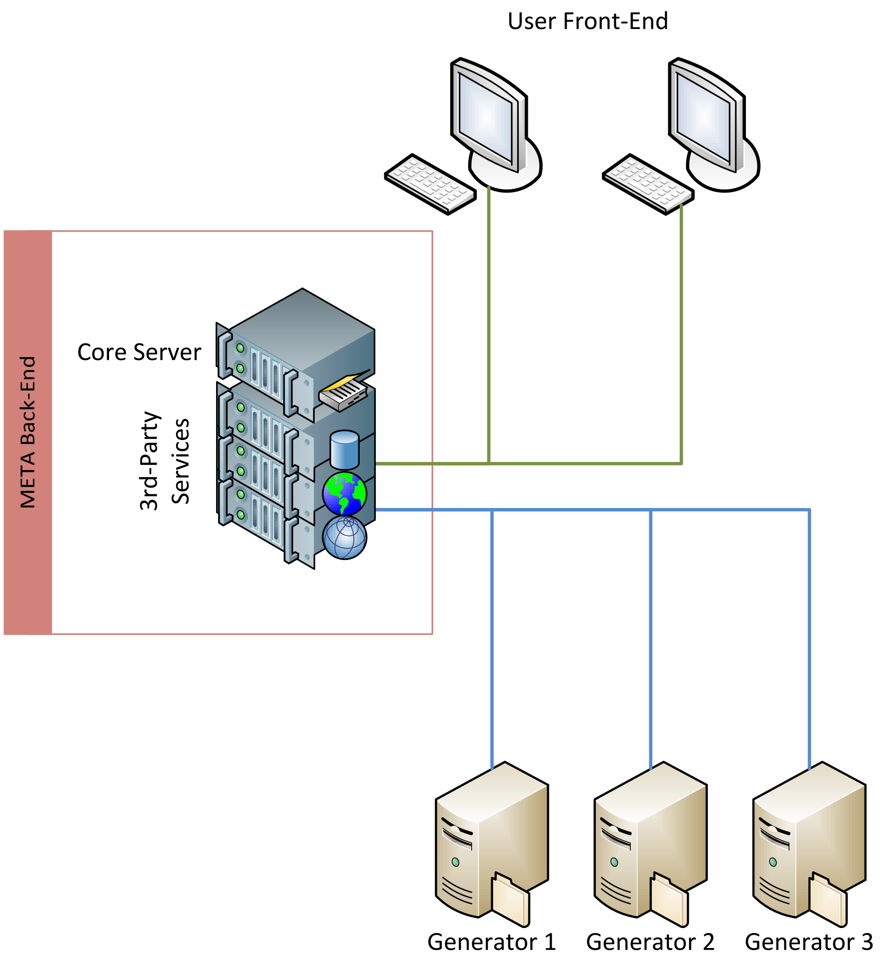


Figure 1. Overview of the META design concept. The single application process server component and supporting third-party services form the back-end. The back-end serves the user front-end for graphical and programmatic access as well as the generator front-end responsible for metadata ingestion.

III.B. Detailed Design

The META system centers around the notion of products. See Section II for a precise definition of this and related terms. In order for a mission data item to be represented in META it needs to be treated as a product instance. Each product instance has an associated product definition. A product definition describes possible metadata of a product in the form of fields. A product definition is uniquely identified by its identifier and version. Each field has at least an identifier and a data type. More information can be given for each field, e. g. a field description or additional processing directives. In order to make a product instance available to META a generator is used. A generator detects new or modified product instances and extracts product metadata using the product definition. The product metadata is then sent to the META back-end. What exactly comprises metadata is up to the user crafting the product definition and implementing the generator. A generator usually produces metadata for one or more product definitions. Because some product types are common across many different projects, a standard set of generators is supplied. This prevents that each project needs to roll their own generators for common product types.

An example for possible mission products are office documents or recorded telemetry files. A product definition for the former might include fields like document authors, creation date, modification date, keywords, full text, and possibly many more. Some of them might be marked as mandatory and always have to be supplied by the generator, others may be optional. A product definition for recorded telemetry files might include fields for the ground station used for recording, the data format, times for acquisition and loss of signal. Whereas possible in principle, not all information contained in the telemetry files should be reproduced for their metadata. Usually, an independent and dedicated telemetry processing system is in place specifically designed for this task. META is not intended to replace or incorporate such a system. However, the output of a telemetry processing system might be a number of products with their own product definitions, such that they may be represented in META using another generator. A META query can then be used to ask for all products that were generated from a specific recorded telemetry file, if such a relationship can be deduced from the metadata.

Table 1 shows all possible data types that can be used when creating product definitions. Each field may be declared mandatory or optional, as well as single- or multi-valued. It is also possible to represent missing values in a multi-value field. Usually it is the responsibility of a generator to transform a product instance into an appropriate metadata representation. However, it might be sensible to allow a server-side processing for some fields under specific circumstances, e. g. to detail treatment of full-text fields. Therefore it is possible to declare server-side processing directives for a field.

Table 1. Possible data types usable for fields in product definitions.

Data Type	Usage	Example
`String`	for arbitrary texts in UTF-8 encoding	a parameter name
`Integer`	for any integer value	size of a file in bytes
`Float`	for any floating point value	a temperature reading
`Boolean`	for representing a traditional logical value	an error flag
`URL`	for representing a Uniform Resource Locator	for locating a product instance file on a specific machine
`Timestamp`	for representing absolute instants in time	the creation time of a product instance
`Duration`	for specific durations without reference to an instant in time	a scan interval for a file transfer service
`Timerange`	for a specific interval with reference to an absolute time	the time range of a specific ground station contact
`Text`	intended for arbitrary long texts	full text of a word processor document
`Blob`	for arbitrary binary values	a preview image
`Reference`	for referring to another product indexed in META	the recorded telemetry file used as source for this product

The distinction between a `String` field and a `Text` field shall be motivated as both are used for storing arbitrary texts: `String` fields are intended for storing short texts such as a parameter name, a file name, a user name, a short description and similar information. They are usually fully shown in the user front-end when viewing product metadata. `Text` fields shall be used for storing longer texts, possibly the text of a whole book. These fields will undergo further processing on the server side, such as tokenization and word analysis suitable for generating a proper index for full-text documents. Furthermore, a similarity score using the locality sensitive hash algorithm TLSH[3] is calculated. Because a small change in the text only leads to a small change in the hash value, this score allows searching for different versions of the same text.

Generators communicate with the back-end using a simple REST interface, providing product metadata in XML or JSON format. An XML Schema is provided to ease adoption of the interface, allow input validation, and code generation for generators. Additional more efficient formats may be used if need arises, but none of them have been specified yet. Metadata for each submitted product instance always references the used product definition by identifier and version, such that different versions of the conceptually same products may coexist.

IV. Architecture

As detailed in Section III the overall system is comprised of a distributed part—the generators—and a rather monolithic server component. Generators may employ their own architecture which might be radically different from the one chosen for the server component. The remainder of this section shall therefore describe the architecture of the server component in more detail.

We decided to utilize a so-called "ports-and-adapters" architecture,[4] also known as "hexagonal" or "onion" architecture. In contrast to traditional software layers where higher layers (usually some presentation layer) depend on the lower ones (typically the business layer in the middle and the persistence layer at the bottom), the dependency flow is very different in the ports-and-adapters architecture: As depicted in Figure 2 the business logic is at the core of the system, self-contained and not depending on any outer layer. This approach provides easy testability of the application core logic in absence of the other layers. Connection to outside layers is provided through the means of ports. They are realized as interfaces and constitute the contract between core and outer layers. The advantage is that outer layer concepts and domain objects do not creep into the core. Implementations of these interface then make up the adapters that adapt the core domain objects to outer layers and vice versa. In order to decouple ports and adapters in the application a "dependency injection" framework is employed (Google

Guice[a] in this case) that provides dynamic injection of an adapter implementation into the ports. In a test environment mock adapters or adapters to lightweight systems are injected. This allows easy testability, deterministic behavior, and quick test runs. The production environment is supplied with adapters to the production systems, e. g. to the production database.

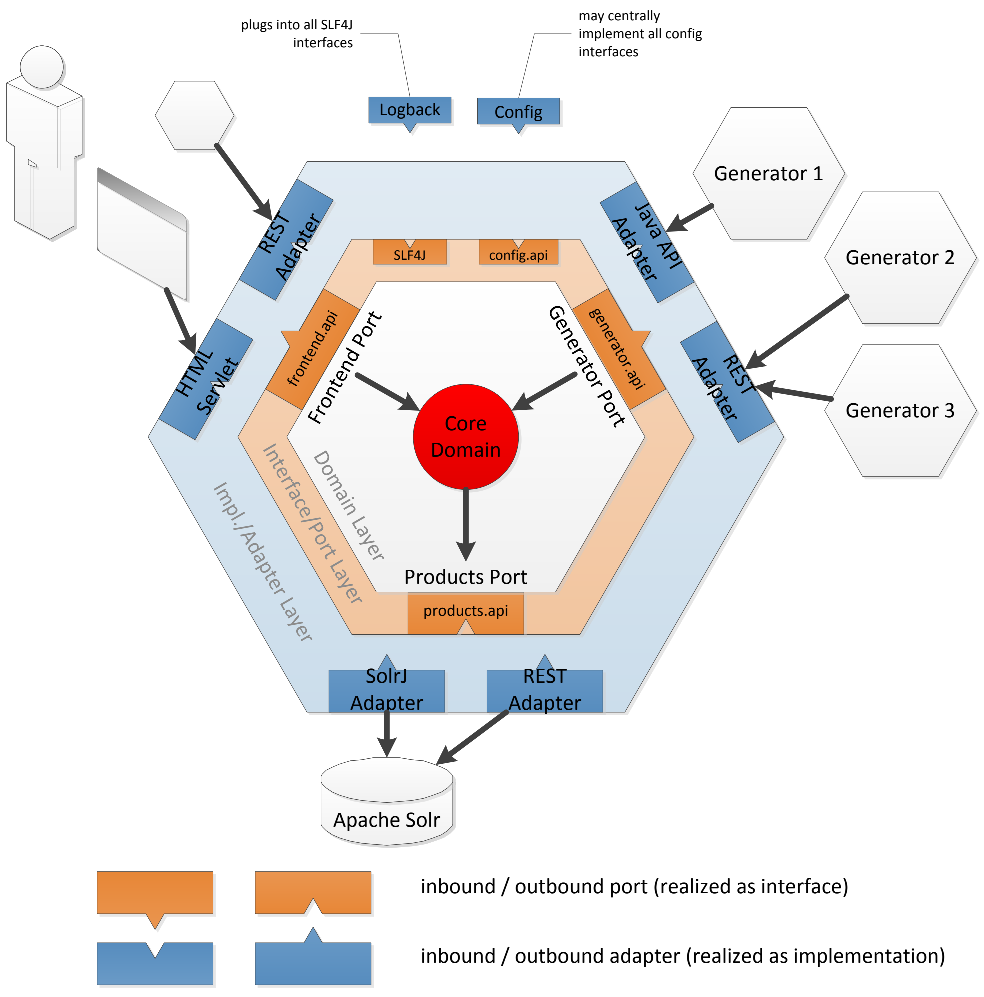


Figure 2. Ports-and-adapters architecture of META.

IV.A. Core

The system's core contains all the business objects and their behavior without any explicit dependency on any outer layer except the interface or port layer. As such the core is self-contained and is testable without any of the other layers in place. The core contains classes that represent product definitions, product metadata, query logic that is specific to META, validation logic, and in general anything that makes up the very core concepts of the application. In order to perform its work the core is dependent on its ports to outer layers. Ports are realized as interfaces so that the core is not aware of any concrete adapter that plugs into each port. It should be noted that using this architecture no outer layer objects creep into the core and establish an implicit and unwanted dependency on a specific adapter implementation. It is the responsibility of the adapter to adapt the core objects to a suitable representation to be used with the system that is plugged in and vice versa.

IV.B. Front-End Port

The front-end port describes all functionality that is needed for a user to interact with META. Two means of interaction are provided by the currently implemented adapters: (a) a graphical user interface realized as web interface

[a]`http://github.com/google/guice`

and (b) a programmatic REST interface. The current implementation performs server-side generation of a website view using the open-source templating framework Thymeleaf[b]. Internally the adapter routes view requests to the same methods made available for programmatic access using REST. This has two advantages: First, code duplication is avoided and all functionality accessible by the graphical interface is also available programmatically. Second, if at a later time server-side view generation proves to be detrimental it is easily possible to introduce a client-side framework like AngularJS[c]. This framework then may use the programmatic front-end access. In fact it is possible to perform a smooth transition from server-side to client-side view generation by mixing these two approaches.

IV.C. Generator Port

The generator port provides an interface so that the core can ingest input sent from generators. It is the responsibility of the adapter that plugs into the port to offer a specific interface to the generators. The current implementation provides a simple REST interface, with payload data in either JSON or XML format. This interface is deliberately kept simple in order to encourage generator development. It is possible to implement a complete generator in a few lines of code. While we strive for ingesting most of the mission data with a standard generator produced by the META team, we expect some very mission-specific data to demand their own generator. In this case it is desirable to have a low entry-barrier to generator development by the mission team. Should need arise new payload data formats may be added easily, e. g. an efficient binary format for decreasing required network bandwidth. Assessment of such a need requires performance figures of a real-life deployment of the system and therefore has not been done yet.

IV.D. Products Port

The products port enables functionality that revolves around managing product metadata. Its responsibilities include typical CRUD[d] operations operating on product metadata and their definitions. It persists product metadata and product definitions and it allows querying the data. The current implementation uses an adapter to the Apache Solr[e] search engine, but alternatives being prototyped as well. A two-tiered solution consisting of a traditional SQL database (PostgreSQL[f] in this case) and a reverse index (Apache Solr in this case) seems to fit well. The products port is structured in such a way that CRUD requests and query requests are handled without knowledge of each other. Viewed like this the products port is in fact two ports, yet conceptually we handle them in a unified way. This also reflects that both ports are not completely independent because the choice of persistence has a strong influence on how to query persisted data.

IV.E. Other Ports

In addition to the already presented ports some more ports are in place that follow the same ports-and-adapters principle but are mere helper ports. These include a port for logging which we use the popular SLF4J[g] API for with Logback[h] as the adapter/implementation as well as a configuration port, whose adapter is auto-generated from the interface by OWNER[i]. How to persist user-related information such as saved queries or preferences is still open: Either a new helper port is introduced or this information is treated as just another product, whose definition is provided by the core system and that is persisted in the usual way using the products port.

V. Deployment

Traditionally web applications on the Java platform are executed in the context of an application server in order to efficiently utilize the application server's resources. With today's computing resources and the ubiquity of virtual machines (not to be confused with the Java Virtual Machine) where a single virtual machine often is dedicated to a single task, or more recently container-based deployments[j], the need for an application server decreases. The administration overhead for configuring, maintaining and tuning an application server can become a major burden that can be avoided using this model. Thus, META is bundled with its own HTTP server and infrastructure necessary for providing a web application and can be rolled out on any physical or virtual machine

[b]`http://www.thymeleaf.org/`
[c]`http://angularjs.org/`
[d]create, retrieve, update, delete
[e]`http://lucene.apache.org/solr/`
[f]`http://www.postgresql.org/`
[g]`http://www.slf4j.org/`
[h]`http://logback.qos.ch/`
[i]`http://owner.aeonbits.org/`
[j]`http://www.docker.com/`

without the need for an application server. Due to the limited number of third-party services required to run META and the limited number of back-end instances we did not look into container-based deployment yet, but as soon as deployment of the back-end needs to be streamlined this would certainly be a viable way to look into.

Generator deployment on the other hand is of greater concern because instances of them are naturally abundant due to their microservices nature. Keeping the number of deployed generators low is therefore the most effective way to control complexity. Complexity here arises due to the fact that all generator instances need to be maintained, i .e. deployed, registered (due to security reasons, see Section VI), and monitored. Keeping the number down is achieved by two means: (a) Generators are deployed at "data hubs", i. e. locations where they have access to many mission data. (b) A default generator is supplied, that already contains metadata extraction functionality for product types common to most projects. Incorporation into our in-house monitoring system then provides us with status information for each generator and the possibility to detect and react on failures.

V.A. Network Structure

Operational and office networks at GSOC are strictly separated in order to guarantee safe spacecraft operations. The only interface between both networks is a single, well-defined, and strictly-monitored file interface sporting accordingly high latencies. On the other hand, one requirement for META is to ingest data from both networks and allow users from both networks to perform queries on the whole data set. META is not expected to be used for real-time operations, meaning high latencies of the order of minutes are acceptable for ingesting metadata across network boundaries, whereas the same latency on the front-end leads to a bad user experience. This excludes the use of a single back-end instance in just one of the networks. Instead, deployment of synchronized back-end instances in each network is mandatory. Please note that this is different from the back-end scaling options presented in Section V.B that assume scaling out on the *same* network.

The synchronization problem only affects the persistence layer of the system and thus needs only to be solved for the products port (except for a possible user helper port as explained in Section IV.E). Two solutions seem feasible for implementation, which has not been carried out yet for the prototype: (a) Some database functionality may be employed for performing synchronization. The advantage of this approach is that implementation effort is low, because it uses already built-in database features. The disadvantage is that the chosen database has to offer such a feature in the first place and that synchronization becomes closely tied to the concrete database system. The latter means not being able to exercise full control over details of the synchronization, which might become important if data items are in conflict. (b) An additional adapter may be plugged into the products port. This adapter is responsible for tracing all operations performed on the database and regularly transferring such a trace file to the remote system. The remote system applies the trace file through its generator port, possibly complemented by some internal management information. The advantage of this approach is that it is independent of and can be used with any database system. Full control can be exercised over the synchronization process, being able to specify in detail how conflicts shall be resolved. Because the synchronization process is based on a trace of database operations, conflict resolution is easier than for the former solution, where only the resulting database entries are subject to synchronization. The disadvantage is higher implementation effort because it is not possible to resort to existing database functionality. Common to both solutions is the need to specify how conflicts shall be resolved with the second solution providing a more fine-grained control over this process.

V.B. Scaling

So far META is running as a prototype in order to develop concepts, evaluate operational needs and get familiar with possible performance bottlenecks. As such no major optimizations are in place yet to boost performance. However, the system design is such that it is possible to scale up, scale out, or both. As scaling up can be trivially achieved by running the system on stronger machines (more RAM, more CPU power, better network connectivity) and as there are upper limits to this approach we will concentrate on the possibilities of scaling out, i. e. of running the system on more machines. We only concentrate on the back-end system, as the generators are already distributed by their very nature and can be split up into multiple processes or can be deployed on multiple machines if they show performance problems.

The back-end can be scaled out from several starting points, depending on where a performance bottleneck has been identified. Several starting points shall be touched on here. Also see Figure 3 for a potential scaled-out deployment.

V.B.1. Products Port / Database

It is expected that the database behind the products port is the most likely source of a bottleneck because almost all use cases of the system require hitting the database. Especially for a large number of newly indexed product instances (e. g. when running a generator for the first time on hitherto unknown data) this can result in considerable

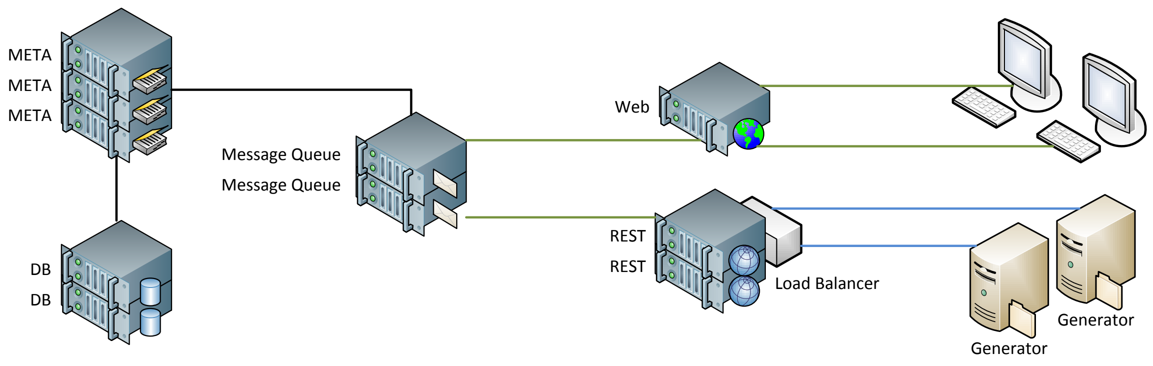


Figure 3. Potential deployment of META, scaled out at different starting points in order to boost performance. Here two database nodes are in use, as well as a load-balanced interface for the generators. Please note that in this picture the additional decoupling step mentioned in the text is shown as a message queue, allowing to have three core systems in place (labeled with META). We do not expect to take scaling to such an extreme, except for boosting database performance.

load on the database. Depending on the complexity of queries and the number of concurrent users, querying the database might also become a bottleneck. Scaling out a database is mostly transparent to META and even to its specific database adapter. In the case of Apache Solr additional so-called "nodes" can be added without any change to the application. These nodes can be distributed across different machines and Apache Solr deals with the added operational complexity all by itself. In the two-tier approach of one database as product metadata store and another as reverse index, the same considerations apply: With Apache Solr as reverse index additional nodes can be added any time. PostgreSQL as product metadata store allows partitioning of data, e.g. by project or generation timestamp, such that most queries only need to hit a small number of partitions. One might even consider employing one of the recent NoSQL databases that are specifically designed to handle many concurrent read and write accesses, however no evaluation has been done yet. It is important to note that except for the specific adapter no change to the application is necessary in order to scale out a database or even exchange the underlying database system.

V.B.2. Front-End Port and Generator Port

Though we do not expect a bottleneck at the front-end due to the restricted number of users of the system (it is not open to the general public), also the front-end part is capable of being scaled out. This can be accomplished via two ways: First, the server-side generation of the web site view can be replaced by a client-side generation using a client framework that directly hits the REST interface provided for programmatic access. Thus the server load is decreased. This kind of optimization only works for the web site access. The second way for scaling out addresses both graphical and programmatic access: Here we use the defining property of a REST interface that in each request a state representation is transferred from client to server and no state has to be kept on the server. Thus each client request may be routed to a different server instance by a load balancer. Because the generator port is also implemented as REST interface the same considerations apply to this port as well.

Because the server component is a monolithic application scaling out the front-end port means actually scaling out the whole server application. This is deemed unproblematic because the system behind the products port is already capable of serving multiple cores and the generator port just scales as well. Should this pose problems contrary to expectations[k] an additional decoupling step mediating between front-end and core has to be added. In this case the transition to a full-blown microservice architecture on the server side should be carried out.

VI. Security

META has an underlying security concept that allows fine-grained control over who can do and see what. Users are authenticated using our GSOC-internal LDAP system, preventing the need to securely store user credentials in META. Users are authorized on a role-based model. Read rights can be granted on project, product, and time range level, i. e. users will only be able to query and see those results they have clearance for. Product modification and deletion rights can be assigned with the same granularity. Administration rights can be granted on a system and project level. System administrators have the right to create new projects or modify existing ones. Project administrators have the right to create, modify or delete product definitions belonging to their project.

[k]This means that the system is put to use in a completely different environment than originally intended, i. e. deployed on large—potentially global—scale. These dimensions are clearly out of scope for now.

Generators are authorized in the same way as users with the exception that they do not get granted any administrator rights. Authentication of generators is not performed via LDAP but by META itself. Each generator instance needs to be registered by a user with appropriate administration rights. During this registration process the generator gets assigned a unique instance identification number and a key which both need to be made available to the generator instance by configuration. Each generator request is cryptographically signed using a hash-based message authentication code (HMAC) using the generator key and SHA-256[5] as hash algorithm. This information is used by the back-end to ensure data integrity, authenticate the generator request, and look up proper authorization. In case a generator key is compromised a user with appropriate administration rights can revoke the key and issue a new one.

All data sent to and received from the back-end are transparently encrypted using HTTPS. Web browsers displaying the user interface automatically check back-end certificate authenticity in order to prevent man-in-the-middle attacks. Generators are required to do the same check, however projects may lift this requirement for project-specific generators considering the closed network infrastructure especially in the operational context. The security concept allows the user front-end to be made available for use outside of GSOC, possibly with restricted user roles, albeit generators may only access META GSOC-internally. However, currently there are no plans to further pursue such a deployment.

VII. Usage in Operational Context

META is specifically designed to help subsystem and data engineers to get an overview of available mission data. It enables the start of detailed analyses, e. g. in case data is missing or cross-correlation between different products is desired. Out of scope is detailed analysis of satellite parameters, because that relies on availability of all data, not just metadata, as well as health monitoring of ground systems. We have dedicated tools like Satmon,[6] RootVis,[7] Nemo, or Icinga[1] for approaching these problems.

The system is accessible from both the control room environment during operations as well as from the office. Also, mission data from both contexts (control room and office) are fed into the system and metadata made available at both locations. As non-real-time system ingested products may take a few minutes before showing up in META. META offers its services through a web front-end that can be displayed by any recent web browser. Thus no additional set-ups are necessary for the clients and the system can in principal be made available to external partners or home-office workers as well. See Section VI for the security concept allowing such a deployment. A screenshot of the live application web user interface is presented in Figure 4. Please keep in mind the prototype status of the project, which means that some functionality presented here might not have been implemented yet or made available through the user interface.

VII.A. Query Submission

The starting point for using META is the submission of a search query. A query input text box is provided, allowing the user to construct a query without having to learn special syntax to get started. This lowers the entry barrier for adopting the system in the engineer's workflow. By default the entered text is looked up in any metadata field belonging to products of the user-selected active project. Results are displayed in order of their relevance, taking into account the number of occurrences of the search term in each product, the product generation time and other information.

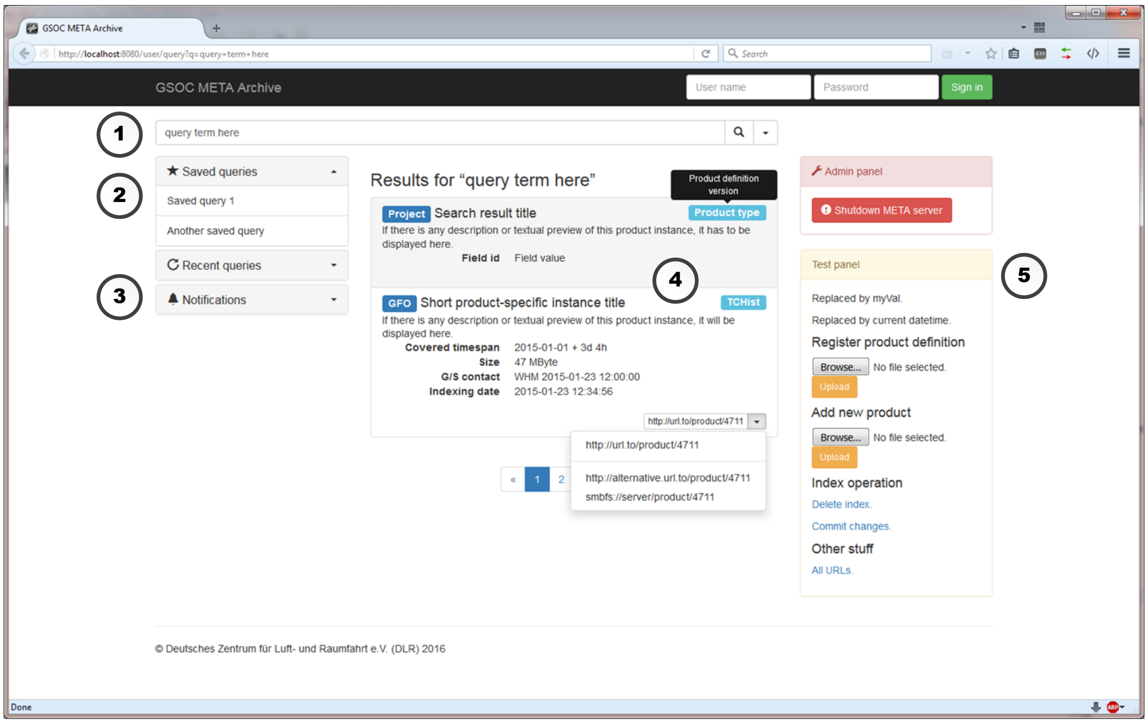
The user is not limited to submission of simple terms for search queries, but can provide more information in order to narrow down the result list and get more relevant results. It is possible to restrict searching to specific product types or to restrict the metadata fields used for the search. Search terms need not be entered exactly but may contain wildcards or may be automatically transformed to their word stems. Still, exact searching is possible by enclosing the term in quotation marks.

Found results can be used as starting point for new queries or to access the actual product instance represented by a result. This is facilitated by accessing URLs from the result metadata. Not all product instances can be accessed from everywhere because of network separation or special URL protocol handlers that require the existence of specific applications on the user's machine, e. g. Satmon.[6] In this case the user is hinted at possible ways to retrieve the wanted data.

VII.B. Data Drill-Down

The META system is designed to be used interactively, i. e. queries can be refined successively. The first query submission might be too broad and return a number of unwanted results. The engineer is guided through the refinement process by displaying a set of so-called facets in the user display that can be used for drill-down into

[1]`http://www.icinga.org/`


GSOC META Archive
http://localhost:8080/user/query?q=query+term+here
Search
GSOC META Archive
User name
Password
Sign in
1
query term here
2
Saved queries
Saved query 1
Another saved query
Recent queries
3
Notifications
Results for "query term here"
Product definition version
Project
Search result title
Product type
4
GFO
Short product-specific instance title
TCHist
Admin panel
Shutdown META server
Test panel
5
Replaced by myVal.
Replaced by current datetime.
Register product definition
Browse... No file selected.
Upload
Add new product
Browse... No file selected.
Upload
Index operation
Delete index.
Commit changes.
Other stuff
All URLs.
http://url.to/product/4711
http://alternative.url.to/product/4711
smbfs://server/product/4711
« 1 2
© Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) 2016
Done

If there is any description or textual preview of this product instance, it has to be displayed here.

Field id Field value

If there is any description or textual preview of this product instance, it will be displayed here.

Covered timespan	2015-01-01 + 3d 4h
Size	47 MByte
G/S contact	WHM 2015-01-23 12:00:00
Indexing date	2015-01-23 12:34:56

Figure 4. Screenshot of the web front-end to the META system (currently in prototype status). ① Query input box. ② Queries saved by user and recently executed queries appear here. ③ Query subscriptions that cause a notification for new matching data are listed here. ④ The main content is a list of found product metadata. ⑤ Data drill-down is made possible by context dependent query refinement boxes (this user interface component is not yet implemented at the time of writing and dummy boxes are shown instead).

the result set. A facet represents a cluster of items in the search result set. As an example suppose a query returns many different product types. One possible facet then is the product type, i. e. the user is presented a list of product types contained in the result set. By selecting one of these product types a new query is submitted with results constrained to the selected product type. The user then might be interested only in products indexed during a specific time interval and thus selects this interval in the corresponding facet. This process allows a data drill-down using dynamic clustering, which is especially useful for cases where the user does not yet have a clear idea of the query needed to return the wanted results.

VII.C. Saved Queries and Notifications

As queries might start simple and get more complex when using interactive drill-down techniques or manual refinement of the query, the user can save a query at any time for later reuse or refinement. The query itself is saved, not the results returned by submitting this query. Saved queries are accessible by their own interface item. At the moment no further categorization or organization of saved queries is provided. In addition to queries explicitly saved by the user, each single query submission is saved as well. Similar to saved queries these recent queries are accessible by their own interface item. By purging the oldest queries the number of them is limited in order not to clutter the interface. This allows the user to try out different query refinements, go back and save the one giving the best results.

Saved or recent queries can be actively used by users to perform a new search by just clicking them. However, sometimes the user does not wish to actively start a search but wants to be notified of new products conforming to a specific query. As an experimental feature it is therefore possible to mark a query as a "notification query". Each new product indexed by the system is checked against the notification queries and the user is notified by an unobtrusive pop-up box in the user-interface in case one of the products matches a query. As this is a performance-intensive feature, the number of notification queries is limited and only queries of logged-in users are checked against.

More advanced use cases for saved queries can be imagined, e. g. sharing them with team members or building high-level status displays for monitoring data-flows, but this is currently out of scope.

VIII. Conclusion and Outlook

The principal design decisions behind META have been shown as well as the concrete architecture chosen to achieve this design. It has been outlined how to deploy and potentially scale the system, respecting a thorough security concept. The problem space has been detailed as well as how the system is put to operational use within a mission operations system. The META archive is currently being prototyped and developed. Experience with a test deployment at GSOC will flow back into the development process, providing better insights into performance figures and a handle to further assess possible scaling options. User wishes and expectations that are currently not or incompletely covered by requirements will be identified leading to a system actively used during mission operations. On the implementation side more generators for commonly used products are in the process of being developed, the query system is refined and synchronization between multiple server instances will be designed and implemented, as well as the data-drill down user interface. If need arises further adapters can be implemented. Especially the programmatic front-end access is subject to extension, e. g. in order to provide a CCSDS Mission Operations[8] service interface for standardized access to a mission data product catalog.

References

[1]Fielding, R. T., *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. thesis, University of California, Irvine, 2000.

[2]Newman, S., *Building Microservices*, O'Reilly Media, Feb. 2015.

[3]Oliver, J., Cheng, C., and Chen, Y., "TLSH – A Locality Sensitive Hash," *Proceedings of the 2013 Fourth Cybercrime and Trustworthy Computing Workshop*, CTC '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 7–13.

[4]Cockburn, A., "Hexagonal architecture," `http://alistair.cockburn.us/Hexagonal+architecture`, June 2008, retrieved 15 March 2016.

[5]Eastlake, D. and Hansen, T., "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," RFC 6234, Internet Engineering Task Force, May 2011, `http://www.rfc-editor.org/rfc/rfc6234.txt`.

[6]Noguero, J., Reid, S., Gil, J. C., Honold, A. P., and Ceballos, A., "SATMON - A Generic User Interface for Satellite Control," *SpaceOps Conferences*, American Institute of Aeronautics and Astronautics, May 2004, pp. –.

[7]Faltenbacher, L., Göttfert, T., Grishechkin, B., Braun, A., and Kumar, A., "RootVis telemetry analysis framework," *SpaceOps Conferences*, American Institute of Aeronautics and Astronautics, May 2014, pp. –.

[8]CCSDS, "Mission Operations Services Concept," Dec. 2010, Green Book, 520.0-G-3, `http://public.ccsds.org/publications/archive/520x0g3.pdf`.