Rethinking Ground Systems: Supporting New Mission Types through Modularity and Standardization

Stefan A. Gärtner*, Michael P. Geyer†, Stefan Hackel‡, Armin Hauke§, Corey O'Meara¶, and Yi Wasser‖
German Space Operations Center GSOC, DLR Oberpfaffenhofen, 82234 Weßling, Germany

We begin to see an increase in the diversity of today's space missions: Small student-designed satellites, unique scientific missions and fleets of commercial spacecraft are just a few of those mission types. In order to cater for new demands on the ground system and to offer customer-tailored solutions we started to rethink the foundations of the German Space Operations Center (GSOC) ground system in terms of a service-oriented architecture approach using standardized technology, mainly CCSDS Mission Operations services. We show how we modularize our ground system, identify and clearly name the mission functions present in the current system complete with timing information and data size requirements. We illustrate this process by employing a concrete prototypical mission with involvement across all departments from antenna control, data processing to mission planning and flight dynamics, and hint at the challenges encountered along the way. The chosen technical solution is motivated and explained, and aspects of deployment, performance, and security are discussed.

Nomenclature

CCSDS	Consultative Committee for Space Data Systems
CORBA	Common Object Request Broker Architecture
DLR	German Aerospace Center/Deutsches Zentrum für Luft- und Raumfahrt
GSOC	German Space Operations Center
HCC	Holistic Control Center
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
M&C	Monitor and Control
MAL	Message Abstraction Layer
MDPDS	Mission Data Product Distribution Services
MO	Mission Operations
MQTT	Message Queue Telemetry Transport
REST	Representational State Transfer
SOA	Service-oriented architecture
TCP/IP	Transmission Control Protocol/Internet Protocol
URI	Uniform Resource Identifier
XML	Extensible Markup Language

*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

†Deputy Group Leader Mission Control and Data Systems, Mission Operations, DLR German Aerospace Center, Münchner Straße 20, 82234 Weßling, Germany

‡Flight Dynamics Engineer, Mission Operations, DLR German Aerospace Center, Münchner Straße 20, 82234 Weßling, Germany

§Deputy Head of Department Communication and Ground Stations and HCC Project Lead, Mission Operations, DLR German Aerospace Center, Münchner Straße 20, 82234 Weßling, Germany

¶Mission Planning System Engineer, Mission Operations, DLR German Aerospace Center, Münchner Straße 20, 82234 Weßling, Germany

‖Flight Dynamics Engineer, Mission Operations, DLR German Aerospace Center, Münchner Straße 20, 82234 Weßling, Germany

I. Introduction

A. Current Ground System Architecture at GSOC

The current ground operations system architecture at DLR's German Space Operations Center (GSOC) was and is the basis for a successful mission legacy, such as the TerraSAR, TanDEM, GRACE, or FIREBIRD missions. As such it is flight-proven, familiar to the operations teams and—due to its multi-mission nature—under constant enhancement for usage in upcoming missions. The ground system existing today does not form a monolithic block but is decomposed into components, but how these components are distributed across systems and how these systems communicate is often driven by the organizational structure of our operations center and the historic mission legacy mentioned above. As new missions are adapted, new systems and new interfaces are devised leading to an organically growing ground system with responsibilities, interfaces and data exchange formats that are not always explicit and clear. Every component evolves at its own pace, interfaces between components are designed and implemented as needed, often reflecting what is state-of-the-art in technology at the moment the interface is designed. It is not uncommon to find a variety of interfaces in a typical GSOC ground system for a typical mission: They include the exchange of simple text files with a bespoke format or XML, CORBA (Common Object Request Broker Architecture), REST (Representational State Transfer) web services, or low-level binary formats that are exchanged over TCP/IP. However, the proliferation of interfaces leads to increased maintenance cost of existing components and increased development cost of new components that need to interact with this multitude of interfaces, which often have to be served in different versions. Some interfaces are sparsely documented and it is not always clear how to evolve them for re-use in future missions.

B. Future Demands

At GSOC we started to rethink the foundations of our ground operations system with respect to future demands. These demands stem from an increasing number of stakeholders, number of operations sites, number of space and/or ground assets, and complexity of the space segment. Not all future missions exhibit all of these traits, but the overall diversity of space missions is increasing: It is ranging from small student-designed CubeSats to satellite fleets providing commercial services and one-of-a-kind scientific missions, just to name a few mission types. As diverse as the missions is the customer base for their ground systems. Different customers have different needs: The more traditional mission types demand a turnkey solution, providing all services for operating spacecraft out of one or only a few hands. Ground systems based on this paradigm are common and it is well known how to adapt them to new missions falling roughly in the same category. However, when confronted with requirements of new mission types, setting up and maintaining these systems becomes a struggle—often resulting in ad-hoc additions and sparsely documented, hard-to-reuse interfaces for those parts that need to be broken out. The activities to rethink our ground operations systems are bundled in the "Holistic Control Center" (HCC) project, which aims at modernizing our existing ground system by modularization, deployment streamlining, documentation, and new developments where necessary. This paper describes the prototyping activities and technical backgrounds in context of HCC. For more information on HCC itself please refer to [1].

II. The Way to Modularization: Service-oriented Architecture

A. System Decomposition

Accommodating new mission types by exposing some systems to stakeholders or consuming some mission functions they provide becomes a challenge with the current ground system, which usually leads to unique solutions per mission. In the future, it shall be possible to "mix and match" mission functions regardless of which party fulfills them. The mission function deployment is also likely to change over the lifetime of a mission, demanding smooth transition of functions from one stakeholder to another. The service-oriented architecture (SOA) paradigm provides a possible way to accomplish this goal by breaking the current systems and components apart in smaller modules called "services". This is accomplished by looking at the ground system from a functional point of view. It becomes quickly clear that service boundaries do not necessarily coincide with the current system and component boundaries. Therefore, it is the responsibility of the HCC project to find service boundaries that provide high cohesion and low coupling between services and identify the necessary steps to remodel existing systems according to this SOA approach.

B. Distributed Systems

Once a system is properly architectured using a service-oriented approach, it becomes possible to distribute services in a flexible and natural way. This does not only mean the physical location of a service deployment (e. g. in the GSOC operational LAN or the external partner's network), but also the number of service instances. This allows scaling service capacity (like processed parameter rate) up or down as needed for a specific mission or even just a specific mission phase. Due to the black-box nature of each service—the service implementation is always abstracted away behind the service interface—maintenance becomes more controllable and cost-effective. This is enabled because of the limited scope of a service as well as the possibility of running predefined tests against the interfaces. Changes in mission requirements can also be accommodated more easily as it is expected that either new services have to be developed or obsolete services removed, both of which can happen largely independent of the rest of the system. If a service has to be changed, its previously mentioned limited scope helps in keeping costs low.

III. Ready-made Solution for Service-oriented Architectures: MO Services

With modularity comes complexity: Each module needs a clearly defined boundary, each involved party (of which there can be many) needs to agree on the interfaces and the more there are of both, the more complex these interface discussions become. The way out of this situation is provided by standardization: standardization of mission functions and standardization of data exchange. The Spacecraft Monitor & Control Working Group of the Consultative Committee for Space Data Systems (CCSDS) provides an international forum for discussing and standardizing exactly these issues. Mission functions are grouped together in so-called "Mission Operations services" (or MO services) [2], data exchange is made interoperable by so-called "technology bindings". Mission Operations services are described in a language known as "Message Abstraction Layer" (MAL) [3] and technology bindings map this language to a concrete "on-the-wire" representation. GSOC is an active participant in the standardization process together with representatives from the French space agency CNES, the European Space Agency (ESA) and the US space agency NASA. Please also see [4] on previous GSOC activities in the realm of MO.

At this point the mainly space-to-ground *Monitor and Control services* (M&C services) are published [5] as well as a number of technology bindings, e. g. to the Space Packet protocol [6] or to TCP/IP [7]. The more ground-oriented *Mission Data Product Distribution services* are on the verge of publication. Future standardization efforts are driven by the needs of agencies. Therefore, since publication of the *M&C services* in 2017 the full stack for implementing CCSDS MO is in place, already with a number of open-source implementations available*. The following sections shall give a short overview of the MO framework and how it can be used for defining and implementing not only standard but also custom services.

A. Overview of CCSDS Mission Operations

CCSDS MO is an umbrella term that consists of the MO framework and standardized MO services. MO services are services that are defined using the MO framework. The MO framework also provides the foundation to define bespoke MO services. Currently, the only published set of standardized MO services are the *Monitor & Control services*. It is expected that the next sets of standardized MO services to be published are the *Common services* and the *Mission Data Product Distribution services*. In parallel, work is ongoing to identify and standardize more services, like *File services* or *Automation services*.

The general structure of the MO framework can be seen in Figure 1. The framework is made up of four layers, with the application layer at the very top. The application layer provides implementations of mission operations services that are defined in the services layer and makes use (i. e. consumes) others. This is the layer where users interact with the framework and where the framework interacts with underlying layers. The service layer defines several services in an abstract way using data types and message exchange patterns provided by the next lower layer, the "Message Abstraction Layer" (MAL) [3]. Finally, these basic building blocks offered by the MAL are mapped to a concrete on-the-wire representation by the transport layer. The transport layer connects to remote MO framework stacks adhering to the same layered structure. The application layer of a remote stack can then consume the services provided by the local stack and vice versa.

*`https://ccsdsmo.github.io/`

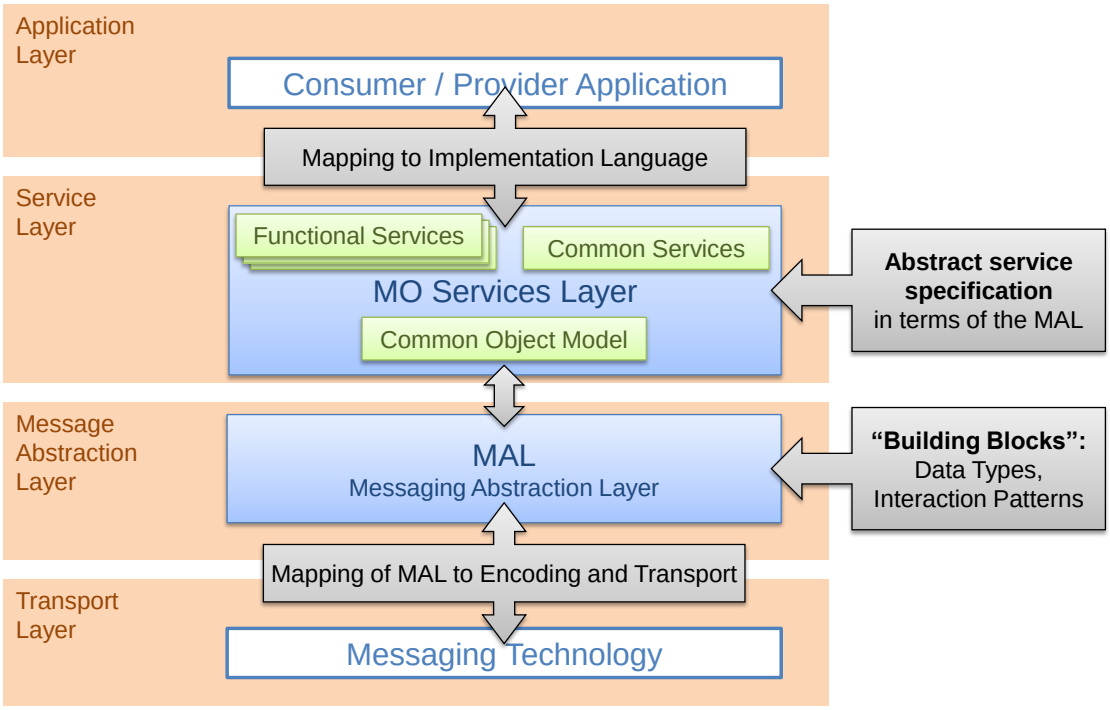

Fig. 1 Overview of the CCSDS MO framework. The application layer provides service implementations and consumes other services. The service layer specifies MO services (functional services) in an abstract way using the building blocks provided by the next lower layer, the Message Abstraction Layer. The MAL also provides the building blocks for a Common Object Model and some Common services, that can be referenced from functional services as well. The lowest layer provides a concrete on-the-wire mapping of all primitives provided by the MAL.

1. The Service Layer

Services contained in the service layer are either standardized services defined by the CCSDS SM&C working group or user-defined services for a particular mission or purpose. A number of commonly needed services have been identified by the working group and are subject to standardization. Adhering to these standardized services is one of the keys to interoperability. Users are still free to define their own services for specific needs. Service specifications themselves do not depend on any particular programming language binding. They are formulated in an abstract, technology-independent way instead. For this purpose, the MAL provides basic building blocks that allow construction of services. Specifications may be expressed as XML (extensible markup language) documents adhering to an XML Schema enforcing the MAL building blocks and composition rules. In fact all CCSDS defined services are available as XML documents.

The service layer consists of three characteristic service types: Functional services, Common services and the Common Object Model (COM) [8]. Functional services provide the high-level mission operations services the user expects from the framework, while Common services are concerned with more basic and administration tasks such as providing a service directory or proper authentication and authorization. Services may use the COM as their data model, which builds upon the MAL and introduces the notion of objects which possess certain characteristics. Using this model gives services the ability to archive and retrieve objects, generate and react to certain events or track progress of activities without the need to specify anything of that themselves.

2. The Message Abstraction Layer (MAL)

The MAL [3] provides a common abstract language that is used by service definitions. It provides primitive data types such as integers and strings without referring to any particular implementation or data representation. The MAL imposes rules on how to use these primitive types to define more complex types such as compositions or lists and already provides a number of useful complex types. The MAL not only defines rules for data type definitions, it also defines a message format and message exchange rules, allowing services to specify interfaces suited to their objectives. The message exchange rules are called "interaction patterns", where there are six types of: SEND, SUBMIT, REQUEST, INVOKE, PROGRESS, PUBLISH-SUBSCRIBE. A service defines the operations it offers using these patterns and the data it exchanges using the data type building blocks. An XML Schema is available to validate a service description in XML against.

3. The Transport Layer

Because the MAL describes messages and data types in an abstract way without reference to any particular representation, it is not directly usable on the wire. Therefore the transport layer is needed, casting the building blocks of the MAL into a concrete technology representation. Every transport technology to be used with the MAL needs to be implemented and is called a "binding" of this technology to the MAL. Usually bindings are composed of two parts: the actual data encoding and the transport of the encoded data, though this is not enforced by the framework. A second type of binding should not be confused with this transport layer binding: In order to use the abstract interfaces and data types defined by the MAL a mapping to some concrete programming language is required. This language binding is independent of the transport layer binding.

B. Overview of the Monitor & Control Services

The *M&C services* [5] provide a consistent set of services that allow action execution and tracking, parameter reporting, alert raising, parameter checking, statistics generation, and aggregation monitoring. The services rely on the Common Object Model as their data model, and use its archiving, event, and activity tracking services. Definition of some helper services completes the picture. The services will be briefly described in the following:

Action Service This service is concerned with definition, invocation and execution tracking of actions, such as the execution of a telecommand. In principle, actions could also be defined as operations of a custom service, however using the *Action service* and thus its dynamic[†] definition of actions reduces overhead and provides a common interface without the need of implementing custom services. Actions can be checked, forwarded, their execution can be tracked, and the relationship between actions triggered by other actions can be monitored.

Parameter Service This service allows consumers to subscribe to parameter updates. Parameter information is dynamically defined (in the same sense as dynamic definition of actions). The updates can be generated periodically, when the parameter value changed more than a user-defined threshold, or ad-hoc in a user-defined way.

Alert Service This service is responsible for asynchronously publishing alerts to subscribers. Alerts are dynamically defined in the same sense as actions.

Check Service This service provides parameter checking capabilities. The supported check types are "limit check", "expected value check", and "delta check". Additional types may be supported by individual implementations.

Statistics Service This service allows association of parameters to statistic functions. The available functions are deployment-dependent. Supported are "maximum", "minimum", "mean value", and "standard deviation". Intervals for parameter sampling, report generation, and collection time between resets can be set independently.

Aggregation Service The *Aggregation service* is similar to the *Parameter service*, allowing definitions of parameter aggregations to retrieve updates for in a single message. Updates can be generated periodically, ad-hoc triggered by implementation-specific measures, or when the values in the aggregation change more than a defined threshold (or after a user-defined time-out).

[†]The term "dynamic" relates to the fact that the *Action service* itself does not fix the exact kind of actions. It does not mean that the system has to support dynamic definitions.

Helper Services The helper services consist of a *Conversion* and a *Group service*. The *Conversion service* offers structures for converting raw values to engineering values and vice versa. The minimum set of conversions includes discrete value mappings, linear conversions, polynomial conversions, and conversions from ranges to discrete values. Implementations may offer support for more conversions. The *Group service* allows definition of groups of objects of the same kind, simplifying the usage of the other services when they need to perform several operations on the same group of objects.

C. Overview of Mission Data Product Distribution Services

The *Mission Data Product Distribution services* (MDPDS) provide the means of accessing and distributing mission products. Whereas the *M&C services* provide a more direct and often real-time access to the monitor and control system, the *MDPDS* are responsible for providing processed, aggregated, value-added data, usually with less strict timing requirements than for monitor and control. They consist of two services: *Product Management service* and *Product Distribution service*. Both use COM for defining their data model and make use of its archive and event services. The *MDPDS* are currently under development and still may change until publication.

Product Management Service The *Product Management service* allows management of products, their specifications and attributes in form of a catalog. Thus, it allows to define and retrieve meta-information that is associated with a product. It also manages so-called "streams" of product instances.

Product Distribution Service The *Product Distribution service* is mainly concerned with the actual distribution of mission product data. This can be a batch request, where a batch of product instances is retrieved and delivered upon request. Alternatively, the request can be in streaming mode by subscribing to one of the streams created by the *Product Management service*, allowing a standing order. Requests can also be monitored, suspended, or canceled.

D. Overview of Custom Service Development

The MO framework does not only build the foundation for standardized MO services for interoperability, but also allows custom MO services to be developed that might only be used internally. The process is the same as for standardized MO services: An XML document adhering to the MAL XML schema is all that is needed as a service specification. This specification can either be written by hand or by use of a specialized editor application and describes in detail all services, operations, data structures, errors, and usages of the COM and Common services. All relevant documentation can and should be put right in the same specification such that it is self-contained. There exist tools[‡] that can automatically generate human-readable documentation, service consumer stubs and provider skeletons for use of services in programming languages, or even documents with concrete on-the-wire representations of the exchanged data using a specific transport technology. If a *Directory service* is deployed in a system, it can also act as a repository holding the service specification XMLs such that services can not only be discovered but also used dynamically. This might make sense if a bespoke MO service is to be offered to an external partner.

IV. Challenges

"Rethinking" a ground operations systems sounds like a daunting task, because it means taking a functional system apart, critically assessing each function, question interfaces and sometimes implementations and trying to put it back together in a more streamlined, systematic, and documented way. In contrast to a greenfield project, it has to be decided which legacy components can be reused, how they can be reused and which have to be replaced completely.

Instead of remodeling the whole system in one go, the first step is to identify a minimal, but complete and somewhat realistic, set of mission functions that can be prototyped, worked, and iterated upon in a short time. Already this is an effort spanning several departments. A dedicated inter-departmental team has been put together for the HCC project that is explicitly encouraged to think out-of-the-box and question the established way of doing things. The prototype system consists of the minimal mission functions and interfaces that are listed in Table 1, loosely modeled after typical low-earth orbiting spacecraft. This set of functions deliberately touches many departments and is supposed to provide a realistic round-trip scenario. See Figure 2 for a conceptual overview of the HCC prototype. For the first iteration the

[‡]`https://github.com/esa/CCSDS_MO_StubGenerator`

Table 1 Minimal set of mission functions needed for a functional HCC prototype providing a round-trip scenario. Functions deliberately touch many departments and are kept simple by leaving out satellite commanding for the first iteration.

Consumer	Provider	Mission Function	Data Size
Mission Planning System	Ground Station Network	Pass booking	small
Ground Station Network	Flight Dynamics System	Orbit prediction	small
Flight Dynamics System	Flight Operations System	GPS telemetry	medium
Flight Operations System	Ground Station Network	Raw telemetry	large

prototype is kept simple and no satellite commanding is planned. Later on simulator commanding is envisaged. This prototype lays the groundwork for disentangling the whole ground system in the future. The process of identifying and clearly naming the mission functions as well as categorizing timing information and data size requirements is first executed for the prototype and later expanded to the whole ground system.

A. Language Problems

Already working on the minimal prototype system taught the team many lessons, both technical as well as organizational: It is difficult to talk the same language across departments, many terms have overloaded meanings, some of them carry implicit notions that are unknowingly different between team members. As an example, the terms "service" or "parameter" mean different things to different people. A "parameter" may be a concrete bit pattern for one team member, a raw or calibrated value (an integer for example) with a mnemonic like `FWDPWR` for another or a more high-level concept like "transmission power", that always denotes the same thing regardless of mnemonic, for a third one. We started to capture common ground and common understanding in a dictionary, trying to find a ubiquitous language [9]. This is an ongoing process. Although we are working on a service-oriented ground system redesign, our dictionary does not yet include the definition for "service". This illustrates the difficulties in compiling as well as the need for such a dictionary.

B. Interface Discussions

Services are currently implemented bottom-up for the prototype: Interface partners discuss the needed interface functions directly with each other, in an agile and flexible way. This approach goes down to the actual on-the-wire representation and data exchange patterns, trying to keep things more or less in line with MO paradigms. Whereas this approach yielded a working prototype it also gave room for many fundamental discussions, often intermingling

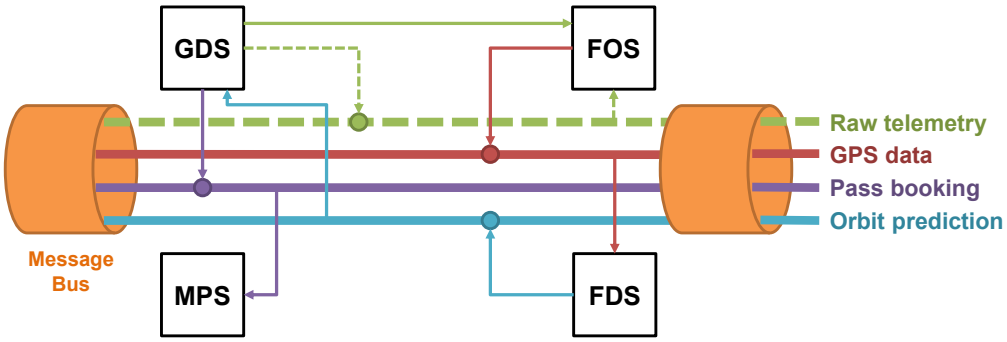

Fig. 2 Overview of the HCC prototype concept. Solid lines stand for mission functions represented as MO services and exchanged over a message bus that is provided by a message broker technology. Services are provided by the entity connected to the dot on the line and consumed by the other entities connected to the line. The dotted lines were realized in the prototype but found to be insufficient. Therefore a direct non-MO interface (represented by the direct arrow between GDS and FOS) using SLE is employed. The entities are GDS (ground station network), FOS (flight operations system), FDS (flight dynamics system), MPS (mission planning system).

lower-level transport and data representation issues with higher-level service representation problems. In order to get out a working prototype fast, the clear-cut layer boundaries provided by MO were sometimes violated by shortcuts. However, we still think this approach is justified considering the purpose of the prototype and the variety of used programming languages and platforms, for which in many cases no MAL implementation or relevant MAL tooling exists yet. After this first prototype has shown the principal feasibility and advantages of a message-broker based ground system and automated deployment processes, it will be important to address these issues. The second prototype iteration needs to replace the bottom-up developed interfaces with a top-down approach where service specifications formulated in a consistent fashion (as MO services) provide the system baseline. At this stage we expect to see MAL implementations and tooling for the platforms we use to emerge, removing any shortcuts. As much as possible, standardized services shall be used by then, e. g. using the *MDPDS* as basis for a bespoke *GPS service*.

C. Limits of CCSDS MO

Already now we begin to see that probably not all mission functions can or should be modeled as MO services. Most notably the raw telemetry interface from the Ground Station Network to the Flight Operations System has peculiar requirements such that it makes more sense to employ more specialized, but still standardized, means of communication. In this concrete case, CCSDS Space Link Extension (SLE) [10] provides the better solution. Still, MO fits nicely in this picture and even acknowledges its own limits by allowing different paradigms to be used when necessary, reducing its own role to the higher-level management of services. This becomes particular evident for file transfers, which most probably will still be needed for external data exchange: Many tried-and-tested protocols and techniques exist for transferring files, which is why MO does not try to provide yet another transfer mechanism, but provides a unified transfer management instead. These *File Transfer Management services* are currently under consideration for standardization by CCSDS. One should note that by far the major share of the interfaces does not require such special solutions and MO can be applied perfectly fine throughout. Even where a specific transport technology (such as a broker-based message transport system) does not provide the required performance, the flexibility of MO allows to just switch transport protocols for this interface (e. g. to a point-to-point connection) without throwing all advantages of MO, like consistent service descriptions, over board. This also allows the future evolution of a system by switching to better technologies as they become available.

V. Technical Solution

A. Mission Bus Backbone

The MO framework can be used in conjunction with any transport technology provided that a mapping exists for all MAL primitives to the technology. The MAL provides a service bus view of the system to the upper layers, though the technical implementation is not necessarily realized using a bus system. There are a number of standardized or soon to be standardized technology bindings available [6, 7], all of which rely on direct connections between the service entities. For the deployment at GSOC we found these bindings unsuitable for a number of reasons that will be detailed in the following sections. Instead, we chose a custom binding to a message broker protocol with deployment of a number of message brokers. Each message broker forms its own message bus. The brokers are interconnected and thus allow a unified view of the bus. Whereas such a setup is beneficial for our internal message exchange, the next sections also clarify why we plan to use one or several of the standardized bindings for external message exchange. See Figure 3 for a graphical representation of the broker-based transport concept es envisaged for use at GSOC.

The message broker system we chose for the prototype is MQTT 3.1.1 [11]. MQTT is a standardized lightweight messaging transport protocol specification that originated in and is mainly used by the “Internet of Things” community. We have chosen this protocol mainly for its simplicity and the availability of numerous open-source client libraries for different programming languages and broker implementations. The systems that shall be connected in the prototype are written in C++, Fortran, Java and Python. We are open to exchange the protocol later if the protocol or the implementations do not meet our requirements or fail to convince otherwise during the prototyping phase. These requirements not only include performance, but also reliability, availability, usability, to name just a few. A change away from the paradigm of a broker-managed message bus is not intended, however.

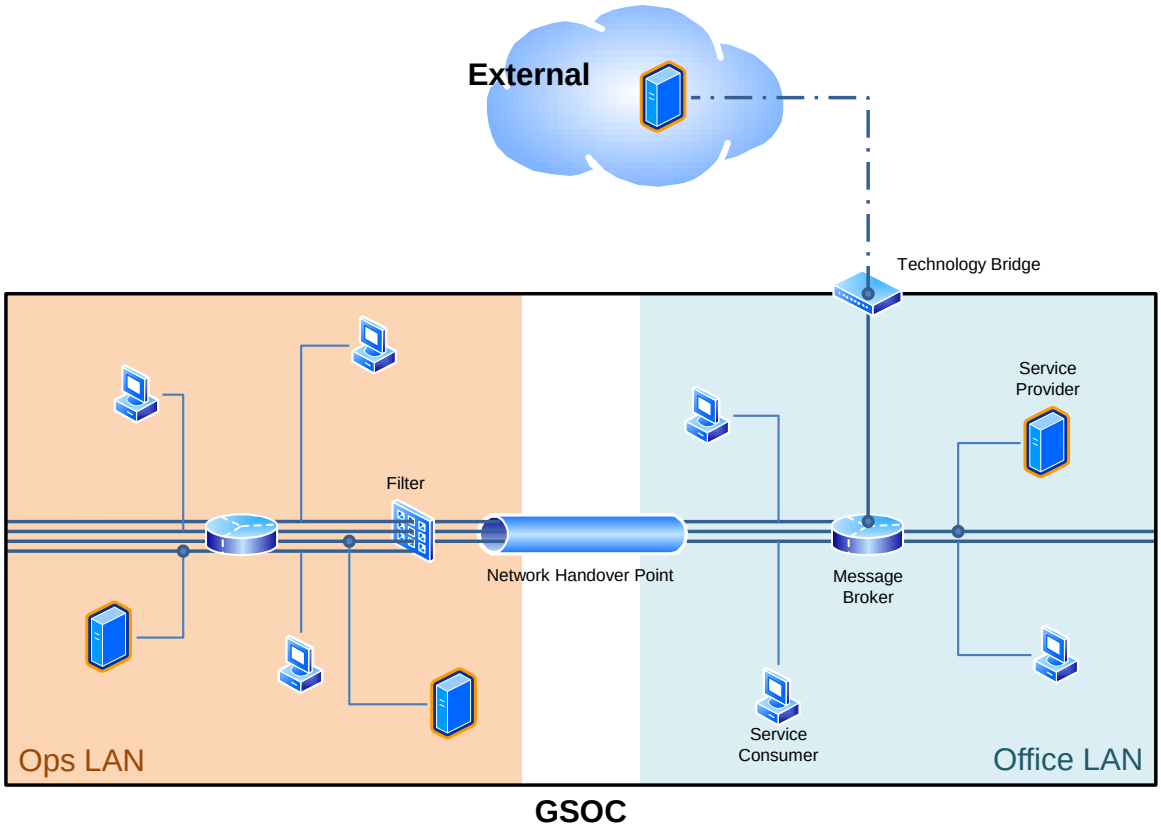

Fig. 3 Overview of the HCC broker concept. A message broker consisting of a high-availability cluster is deployed in each LAN at GSOC. Part of the messages are piped through a single monitored network handover point between the networks. Solid lines represent connections using the message broker technology, which currently is MQTT. External partners are connected using a technology bridge, such that the dash-dotted line represents a TCP/IP connection. Filters can inspect and discard messages. For simplicity reasons only one filter is shown. In practice each message broker would be equipped with one. More can be deployed (e. g. at the technology bridge), if deemed necessary.

B. Performance and Reliability

As can be seen easily, performance in terms of message and/or data throughput of directly connected service entities is usually higher than having an additional intermediate entity for brokerage. One part of our prototyping studies is to find out whether the performance penalty has any noticeable negative effects. Therefore, we broadly categorize the service interfaces on the two axes data size and timing pattern. Data size can be small (order of bytes to 10 kBytes), medium (order of 100 kBytes to megabyte) or large (order of 10 megabytes or more). Timing pattern mainly captures how often data is expected, i. e. message rate in the broadest sense. This can be rare (e. g. once a day), medium (every few minutes), or often (every second or less). Interfaces with small or medium data size and small or medium data rates can be handled well by our broker system. We have been able to exchange several tens of message per second between two service entities over a Mosquitto[§] message broker. High data rates or large data sizes demand a direct connection and in case of large data sizes might not even be viable to exchange via MAL but need dedicated protocols. One such interface is the raw telemetry interface from ground stations to control center, both in real-time mode and offline mode. In our prototype, we first modeled this interface as MO service, but are now switching to CCSDS Space Link Extension [10].

[§] `https://mosquitto.org/`

Most of the interfaces fall on the small to medium scale regarding data size and data rate, especially if function partition into services is done systematically with these limitations in mind. For example, one would not use an MO interface for a function that needs to be called several hundred times per second as can occur in optimizations problems. In such a case one should rethink the service boundary, because service coupling is too high.

Reliability of the message system is of paramount importance for satellite operations. The disadvantage of a message broker system is that it introduces another component that can fail. In a direct connection one of the service entities or the network can fail. In a broker-moderated connection additionally the broker can fail. Therefore the broker will be made up of a high-availability cluster, such that downtimes are kept to a minimum. The advantage of a broker system is that intermittent connection losses of the service entities can be mitigated by buffering the messages until proper reception is ensured.

C. Inter-network Connections

Due to security reasons, GSOC employs a network structure of separated networks, mainly an operational network (Ops LAN) and an Office network (Office LAN). There is only a very limited number of exchange points between the networks, which are strictly monitored. Whereas operations can be conducted from a completely separated Ops LAN alone, for everyday operations it is much more convenient to have some Ops services available in the Office LAN and vice versa. For example, telemetry display should be possible from the Office and access to project documentation and other non-mission critical information should be possible from the Ops LAN.

From a security point of view, direct connections between two networks should be minimized as much as possible, which is why MO technology bindings requiring direct connections are not feasible in this environment. By deploying a broker (or broker cluster) in each LAN, intra-LAN communication becomes possible (though for performance reasons a direct connection might be used in exceptional cases). Intra-LAN connectivity is provided by relaying or mirroring part of the broker topic tree into the other LANs. The exact technical details depend on the broker protocol and the concrete implementation and are still subject to change for our prototype.

D. Security

The previous section already touched on security issues and the resulting separation of networks. The remaining handover points between message brokers in different networks are very few and allow installation of additional security measures. The structure of MO services allow for a natural way of inspecting the exchanged traffic and filter it according to global or project-specific rules. The MAL message header, whose structure is the same across all messages, provides the possibility of fine-grained filters. If not actively realized as its own service, it is not possible to exchange arbitrary data but only data pertaining to certain services. These security features are not implemented in the prototype yet, but can and will be added later via correct assignment of the MAL header fields and filters installed at the message exchanges. Filtering cannot only be performed at the exchanges but also for all messages received by the broker. Some of these security features are listed in the following paragraphs.

Certain services can be completely blocked at the exchange points by filtering on `Service Area`, `Service`, and `Area version` header fields. As an example, if you do not want to expose any *Action service*, that allows command execution, all messages belonging to this service can be filtered out. One can also filter single operations, such as *submitAction*, by inspecting the `Operation` header field and still allow messages for operation *preCheckAction* to pass.

Authenticity of messages can be checked by evaluating the `Authentication Id` field. Currently we have not yet decided how to exactly use this field, but some kind of message signature is discussed. This check can be performed by the exchange filter or broker, but also by each service provider or consumer if there is need for extra security.

If needed, only certain types of sessions can be allowed through the filter, such as simulation sessions. This is achieved by using header fields `Session` and `Session Name` and can also be used to prioritize certain sessions in order to ensure smooth operations.

User authorization is possible by relying on the *Login service*. We plan to integrate this as frontend to our central LDAP server and fill the MAL `Authorization Id` header field based on login information in a yet to be specified manner.

Confidentiality is achieved on the transport layer: The MQTT message broker system we chose has the capability of encrypting all traffic between broker and service entities. In order to provide filtering, no end-to-end encryption between two communicating service entities will be established, only between entity and broker. Thus, it is of extra importance to properly secure the brokers against unauthorized access. In the unlikely case that end-to-end encryption is required, one can think about encrypting the message body independently from the rest of the message.

E. Providing Services to External Partner

As shortly mentioned before, externally exposing the message brokers is undesired. We prefer to use a standardized technology binding, such as MO to TCP/IP [7] or MO to HTTP/XML for exposing services outside of GSOC. This is achieved by technology bridges that can translate from our internal broker system to a standardized transport. The same filtering and security mechanisms as mentioned before can be applied. Services will not be addressed directly, but service URIs are translated from internal to external form and vice versa at the technology bridge. In addition to the exposition of an MO interface, other interfaces (like a web user interface) can be exposed as well, that will interally map to MO.

F. Deployment

All of our services are developed in a way that they can be easily packaged into a container format. For the prototype this is the Docker[¶] format. A central repository holds the Docker images. Although each service is supposed to be deployable to any mission and made mission-specific by configuration, this might not always be possible. This holds especially true for legacy systems, that are put behind a service interface. Still, a central repository that holds the executable services provides a one-stop shop for all multi-mission and mission-specific services in a version-controlled way. Configuration of services is provided by a dedicated *Configuration service*. Although we are not there yet, the idea is that the whole ground-system can be deployed with just a few mouse clicks in an automated way that becomes possible by container orchestration technology. This not only allows a quicker system setup (compared to weeks it takes now), it also makes changes to the deployment painless: Systems can be moved to different networks or even to the customer's site and vice versa. Ideally, service endpoints are not hard coded but retrieved from a *Directory service* (the only one that needs to be hard coded), which knows about service locations.

VI. Conclusion and Outlook

We have shown how we started to remodel our existing ground operations systems in terms of MO services by implementing a prototype system. This system consists of four services, which are realized on an MQTT message bus. Already now, a few challenges become apparent: Technical challenges demanded the exchange of the raw telemetry service interface with SLE, communication challenges led to a dictionary defining a ubiquitous language. Next steps include the harmonization of the interface descriptions using MO, better integration of legacy systems like GECCOS (GSOC Enhanced Command and Control System for Operating Spacecraft, based on SCOS-2000) and future systems like EGS-CC (European Ground System – Common Core), and gaining experience with container-based deployment and orchestration. Automatic configuration and security measures will be put in place, network bridging between multiple brokers is going to be tested and high-availability concepts will be worked out. Finally, more services need to be identified, defined, and implemented in order to provide a smooth and efficient experience for future missions and customers.

References

[1] Hauke, A., and Geyer, M. P., "Towards a Modular and Flexible New Ground System," *SpaceOps Conferences*, American Institute of Aeronautics and Astronautics, 2018. Same proceedings as this paper.

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

[3] CCSDS, "Mission Operations Message Abstraction Layer," , Mar. 2013. URL `https://public.ccsds.org/Pubs/521x0b2e1.pdf`, Blue Book, 521.0-B-2.

[4] Gärtner, S. A., Hartung, J. H., and Wendler, M., "Implementation of CCSDS Mission Operations Services at the German Space Operations Center," *SpaceOps Conferences*, American Institute of Aeronautics and Astronautics, 2014, pp. –. doi: 10.2514/6.2014-1869, URL `http://dx.doi.org/10.2514/6.2014-1869`.

[5] CCSDS, "Mission Operations Monitor & Control Services," , Oct. 2017. URL `https://public.ccsds.org/Pubs/522x1b1.pdf`, Blue Book, 522.1-B-1.

[6] CCSDS, "Mission Operations - MAL Space Packet Transport Binding and Binary Encoding," , Aug. 2015. URL `https://public.ccsds.org/Pubs/524x1b1.pdf`, Blue Book, 524.1-B-1.

[¶] `https://www.docker.com/`

[7] CCSDS, "Mission Operations - Message Abstraction Layer Binding to TCP/IP Transport and Split Binary Encoding," , Nov. 2017. URL https://public.ccsds.org/Pubs/524x2b1.pdf, Blue Book, 524.2-B-1.

[8] CCSDS, "Mission Operations Common Object Model," , Feb. 2014. URL https://public.ccsds.org/Pubs/521x1b1.pdf, Blue Book, 521.1-B-1.

[9] Evans, E., *Domain-Driven Design: Tackling Complexity in the Heart of Software*, 1st ed., Addison Wesley, 2003.

[10] CCSDS, "Cross Support Concept - Part 1: Space Link Extension," , Mar. 2006. URL https://public.ccsds.org/Pubs/910x3g3.pdf, Green Book, 910.3-G-3.

[11] OASIS, "MQTT Version 3.1.1," , Oct. 2014. URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, edited by Andrew Banks and Rahul Gupta.