69th International Astronautical Congress (IAC), Bremen, Germany, 1-5 October 2018.
Copyright ©2018 by the International Astronautical Federation (IAF). All rights reserved.

IAC-18-B6.1.1

Flight Dynamics Microservices

Stefan Hackel[a]*, Yi Wasser[a], Manuel Hahn[a#], Michael Meinel[b], Ralph Kahle[a]

[a] *German Space Operations Center, Deutsches Zentrum für Luft- und Raumfahrt, Münchener Straße 20,82234 Wessling*, Germany, stefan.hackel@dlr.de
[b] *Simulation and Software Technology Intelligent and Distributed Systems, Deutsches Zentrum für Luft- und Raumfahrt, Rosa-Luxemburg-Str. 2, 10178 Berlin, Germany*
[#] *Since July 2017 with Airbus Defence and Space GmbH*
* Corresponding Author

Abstract

Spacecraft operations require smooth and reliable product transfer between the control center subsystems, and the external entities. Compared to the popular data-driven approach for exchanging products, the service-oriented architecture provides a high level of modularity, high flexibility in deployment, and scalability. Therefore, the Flight Dynamics section at the German Space Operations Center (GSOC/DLR) started to establish microservices, which provide the flight dynamic core functionality to dedicated users via network services. A set of wrapped, high level core libraries is connected to web services, which provide data products on request. Within this paper, the overall service design as well as details like the connection between high-level languages and server interfaces are described. The paper clearly shows the feasibility of dedicated flight dynamic services within the mission operation segment. Services save time, may help reducing the amount of transferred data, increase repeatability, and provide monitoring of requests for traceability.

Keywords: microservices, F2x, Fortran, Python, HTTP, Flight Dynamics

Acronyms/Abbreviations
Abstract Generation Tree (AGT)
Automated File Distribution (AFD)
Consultative Committee for Space Data Systems (CCSDS)
Flight Dynamics (FD)
File Transfer Protocol (FTP)
German Space Operations Center (GSOC)
Hyper Text Transfer Protocol (HTTP)
JavaScript Object Notation (JSON)
Message Abstraction Layers (MAL)
REpresentational State Transfer (REST)
Service Oriented Architecture (SOA)
Two-Line Element (TLE)

1. Introduction

Mission operations heavily rely on exchanging information between different components. Historically, products are generated by executables (i.e. compiled high level languages source), and results are saved on basis of data files. Product shipment or data exchange is then realized via File Transfer Protocol (FTP), among others. The service oriented architecture (SOA) is fundamentally based on modular systems, which may be located anywhere, but communicate only through open, published, and service interfaces. The Flight Dynamics Section at the German Space Operations Center (GSOC/DLR) demonstrates the usage of heritage Fortran software within the modern microservice infrastructure trough published interfaces. The present paper first describes the relevance of Flight Dynamics products within operational mission environment. Subsequently, microservices are introduced, where employed techniques and methods are described. Especially the connection between Fortran and modern Python infrastructure is in scope. Finally, an example for providing orbit related information is given, which is evaluated based on a benchmark.

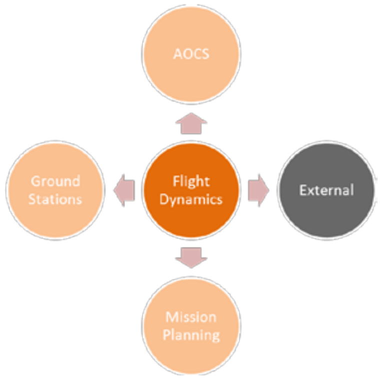

Figure 1 Example of data exchange from Flight Dynamics.

2. Flight Dynamics Mission Support

Flight Dynamics (FD) provides attitude and orbit related products, among others [1]. An example is two-

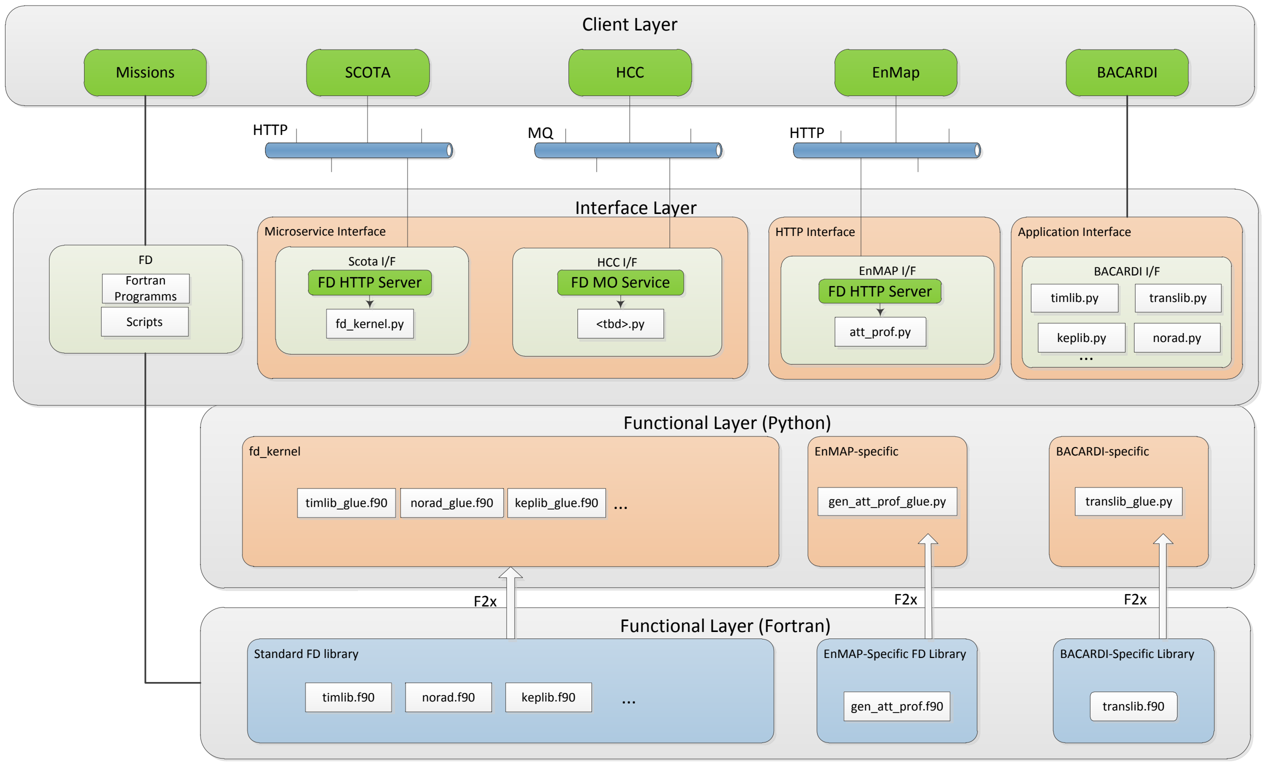

Figure 2 Flight Dynamic microservice layers.

antenna control or provision of orbit information for mission planning purposes. The latter requires orbit information for scheduling space borne image acquisition. Necessary products have to be provided quickly in order to provide a fast image acquisition. An overview of consumers (and providers) of flight dynamic products are shown in Figure 1.

In general, required data for generating the attitude and orbit related products are computed, stored, and maintained within the FD facilities to ensure a high level of quality and repeatability. Orbit and attitude information is stored in data bases; auxiliary data is available on file basis.
Historically grown, FD product generation is scheduled and computed by executables, the results are provided on file basis. Such data provision often suffers from read/write processes, and network transfers. Often, and especially when file transfers pass network boundaries, the Automated File Distribution (AFD) is employed, which often delays the product export.

2. Microservices

A modern solution for fast and reliable product generation and delivery is based on microservices. They are small, autonomous services that work together. Basically, it is a variant of service-oriented architecture, services. Companies like Amazon or Netflix successfully make use of microservices [2].

With a system composed of multiple, collaborating services, different technologies can be used inside each one. The technological heterogeneity is a key advantage of microservices. Within each service, different software or scripting languages can be employed. In addition, they are characterized by a high level of resilience. If one component of a system fails, but that failure does not cascade, the problem can be isolated, and the rest of the system can carry on working. Scalability is another key word, often heard in combination with microservices. If a system can be decomposed in small entities, as usually done in a monolithic approach, it is possible to run parts of the system on smaller, less powerful hardware. Ease of deployment is guaranteed, since a single service can be changed and deployed independently of the rest of the system, which ensures pretty fast code deployment [2]. Service to service (or to consumer) communication is realized using the Hyper Text Transfer Protocol (HTTP), which defines some useful capabilities to communicate with the REpresentational State Transfer (REST) style. HTTP verbs like GET, POST, and PUT have already well understood meanings in the HTTP specification. GET for instance retrieves a resource in an idempotent way, and POST creates a new resource.

In addition, HTTP brings a large system of supporting tools and technology. Examples are HTTP caching proxies and load balancers, or monitoring tools.

The use of standard textual formats gives clients a lot of flexibility as to how they consume resources, and REST over HTTP allows a variety of formats. The FD microservices make use of the popular JavaScript Object Notation (JSON), which uses human readable text to transmit data objects consisting of attribute-value pairs.

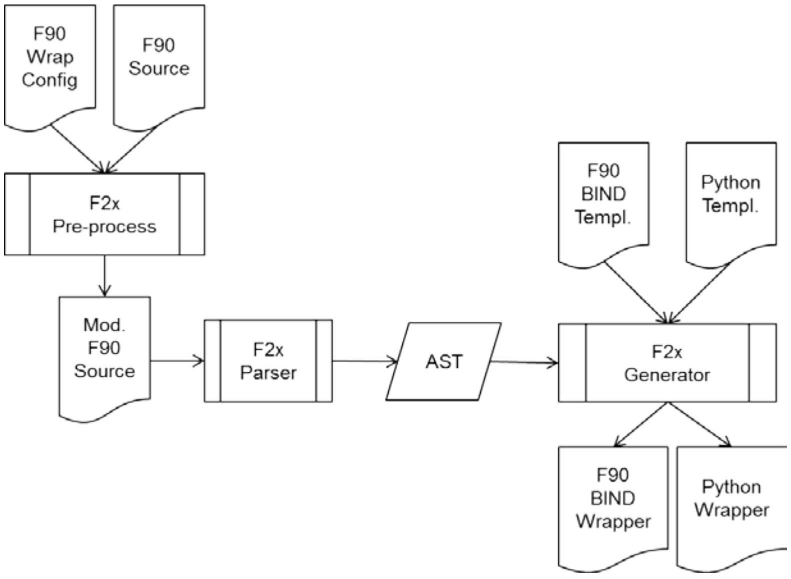

Figure 3 Wrapping Fortran modules with F2x.

3. Service Layers

The FD microservice design considers three layers, as illustrated in Figure 2. Flight Dynamics functionality is settled on bottom, consumers on top. The bottom and core layer is called fd_kernel, and consists of basic FD software, originated in Fortran code. The code basis is identical to the operational executables, which is a key advantage of the employed approach. For instance the orbit integrator, a key FD tool, is available as Fortran source, compiled as executable, and accessible in Python through wrapping Functions, like an orbit integrator, are modularized and available in Python through F2x (cf. section 4). Key feature is a common branch of source files, which are either directly compiled as executables, and wrapped modules. Updates only have to be performed centralized in fd_kernel.

Above is the interface layer, which is connected to the kernel modules and accessible by consumers through defined interfaces. The interface layer is either supported through HTTP or as application interface with direct access to wrapped Python modules. In any case, the interface is the frontend, which has to be defined together with the service users. Principally, the microservice interface architecture allows also implement Consultative Committee for Space Data Systems (CCSDS) Message Abstraction Layers (MAL) for a standardized interface between service providers and consumers [3].

If no HTTP interface is requested, also direct access to Python modules is supported to dedicated users through the application interface.

4. Technologies

Compared to the classic Fortran executables, microservices call for modernized infrastructure. As already stated in the previous section, the FD services require a connection between the Fortran source, and the modern Python scripting language. Solving this issue is of vital importance for establishing microservices.

4.1 Connection High-Level Languages

The gap between Fortran and Python is bridged via F2x, a tool to automatically make Fortran modules available for use in Python. In contrast to f2py [4] it supports the full Fortran 2012 standard and especially supports derived types. F2x uses a template-based approach to generate the wrapper layer. This allows easy extension to support other integration targets than Python.

The general internal workflow of F2x is shown in Figure 3. To facilitate the parsing of Fortran source code, a preprocess step is done during which all executable code and ambiguities are removed. Afterwards, the modified source is parsed into an abstract representation called the Abstract Generation

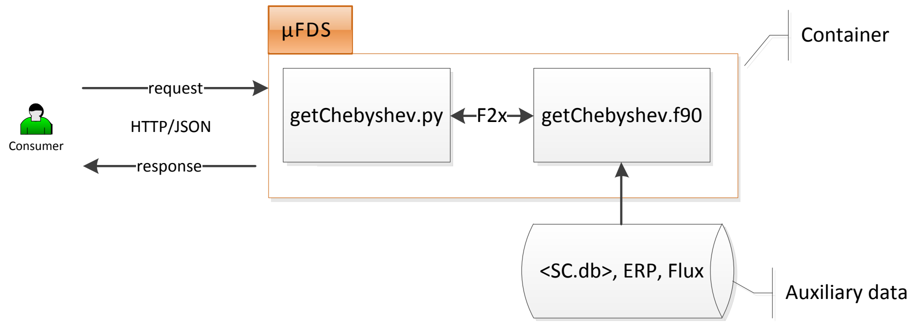

Figure 4 Example service request to get Chebyshev polynomials.

Tree (AGT). The AGT is similar to an abstract syntax tree but contains only information about the interface as this is enough to generate the required wrapping layers. The AGT is then passed to the generation step which applies templates to the tree to generate the sources for the interface layers. In contrast to f2py, there is no automatic compilation of the wrapping code yet.

Currently, there are two layers generated. One that exposes the exported Fortran functionality in a form that is compatible for use in C code. Therefore, the BIND(C) feature of modern Fortran is used to avoid the requirement of compiler-specific interface code. The second layer generates a Python module that consumes the exported C interfaces exposing Python types and functions that are equal to the Fortran interfaces.

Currently, the Python layer uses the ctypes library included in Python. This however produces very slow libraries. To compensate for this, a second-layer template using Cython [5] is currently in development. First benchmarks show that this new template will produce code equally fast as the one produced by f2py.

4.2 Setup and Containerization

The services are provided by a server, based on Python. Wrapped fd_kernel modules are accessible through defined interfaces using the HTTP server. Request or general errors are trapped directly in Python, additionally also the fd_kernel (i.e. the high level) error codes are forwarded.

Deployment of the FD microservice architecture is based on Docker, building lightweight containers [6]. Docker manages and runs containers, which is similar to a virtual machine, but uses a lot of the underlying operating system (host) to work. Instead of building the whole operating system with emulated hardware and its own kernel, a container uses everything it can from the underlying machine, and, if well designed, implements only the bare essentials to run the application or service you want to run. Contrary to virtual machines, containers are usually designed to run specific jobs.

Required fd_kernel modules, as well as the server and corresponding setup are packed into the container. They are only accessible form outside through the specified protocol, address, and interface. Connections to required FD databases, which for example contain spacecraft orbit information, are accessible by mounting the required databases and table files directly into the container.

5. Example: Orbit Trajectory Reconstruction

An example of the service is aimed at reconstructing the orbit trajectory at a certain time interval, and for a certain spacecraft, which states basically the input for the service request. Precondition is history knowledge of the spacecraft's position, which is stored in FD databases, and updated, on regular basis. A common way for providing orbital information over a dedicated time period is the provision of Chebyshev coefficients [7]. The coefficients are the response of the FD service, and allow the consumer reconstructing the spacecraft`s trajectory within the requested time interval.

Figure 4 provides an overview of the service. Core is the getChebyshev Fortran module, which makes use of further fd_kernel modules that were hidden for simplicity. The module has direct access to spacecraft databases, and auxiliary data. The service is containerized and only accessible through specified interfaces and requests. Usage of the service is provided for many high level, and scripting languages. The example request below makes use of the curl command and requests the Chebyshev polynomials for the spacecraft with identifier (ID) TSX-1, a selected orbit reference, the time interval between GPS start and stop date, and the frame ITRF2000:

```
curl -i -X POST -H 'Content-Type:
application/json' -d '{"id":TSX-1,
"orbitRef":"ref", "startDate":1261900818,
"stopDate":1261900818, "frame": "itrf2000" }'
http://microservice/getChebyshev
```

The service response states key indicators of the request, as well as the polynomials for position and velocity separately:

```
{
 "frame": "itrf2000", "orbitRef": "REF",
"segments": [{"coeff":{"pos": {
 "x":[-1619134.152, -158667.196],"y":[-
2850331.747, -128924.608],"z": [-6216337.776,
100498.168]}, "vel": {"x": [-5286.923,
36.430], "y": [-4296.068, 120.388], "z":
[3348.801, 213.639]}}, "startDate":
1261900790.8857002, "stopDate":
1261900850.8856997}]}
```

The response, displayed above, is usually intended to be directly further processed without displaying. The service may be directly employed by an image planning software written in C#, which is capable of handling HTTP. At this stage, there is no need for providing and exchanging files anymore, direct access provides fast processing.
A simple benchmark compares the processing times of executing the compiled source code (including the data write process), and through the HTTP interface (including displaying). The request parameterization is stated in the above, the response contains one set of Chebyshev polynomials. Performance results are shown in Table 1. On localhost, execution runtime of the compiled Fortran source is at 0.37 s, access through the web interface takes 0.41 s. When network connection is required, i.e. the source of request and response are located on different hosts, which are connected via fiber wire, the Fortran result is copied via scp without any compression. Time for product generation including data transfer takes 0.59 s. The HTTP response is delivered in 0.47 s.

5. Discussion and Conclusions

During prototyping the FD microservice, we faced several challenges. One of the major ones is the Fortran to Python connection, especially the provision of arrays, derived datatypes, or error handling. The latter provides also passing the Fortran error codes to the interface, whereas also top-level Python error checks are embedded. Repeatability and transparency is ensured through logging. In case a consumer request fails, assistance can be provided easily.

Since the Fortran code is already existing, and mostly heritage software, source code changes should be avoided. Only condition is that functions, which have to be accessible in Python, have to be Fortran modules.

Table 1 Process runtime [s] comparison.

	Request source:	
	localhost	Network
Fortran	0.37	0.59
microservice	0.41	0.47

Interfaces have to be chosen properly, and aligned with the consumer`s needs. Containers provide a fast and reliable way for software deployment. Access to local databases is possible, which is a condition for running FD processes within a container.

In general, the system architecture has to support technologies like containerization or the utilization of Python 3.6.

However, when processing requests stemming from different hosts or networks, the microservices response is faster compared to the classic product generation, and data transfer. Microservices and service oriented architecture clearly have benefits when products have to be provided across different networks. Granular system design allows a high level of modularization.

The presented FD microservice for orbit trajectory reconstruction clearly showed the potential. Further services for providing flight dynamic products to ground stations, but also to receive GPS information extracted from telemetry are currently being implemented

References

[1] T. Uhlig et al. (eds.), Spacecraft Operations, Springer-Verlag Wien, 2015, 10.1007/978-3-7091-1803-0_2.

[2] A. Newman, Building Microservices, O’Reilly Media, 2015

[3] Mission Operations Services Concept, CCSDS 520.0-G-3, Green Book, Issue 3, December 2010 https://web.archive.org/web/20130531013416/http://public.ccsds.org/publications/archive/520x0g3.pdf

[4] P. Peterson, F2PY: a tool for connecting Fortran to Python programs, International Journal of Computational Science and Engineering 4, 2009, 10.1504/IJCSE.2009.029165.

[5] S. Behnel, R. Brandshaw, C. Citro, L. Dalcin, D. Seljebotn, K. Smith, Cython: The Best of Both Worlds, Computing in Science and Engineering, 13(2), 2011, 10.1109/MCSE.2010.118.

[6] D. Bernstein, Containers and Cloud: From LXC to Docker to Kubernetes, in IEEE Cloud Computing 1(2), 2014, 10.1109/MCC.2014.51

[7] Barrio R., Elipe A., Integration of Orbital Motions with Chebyshev Polynomials. In: Wytrzyszczak I.M., Lieske J.H., Feldman R.A. (eds) Dynamics and Astrometry of Natural and Artificial Celestial Bodies. Springer, Dordrecht, 1997