17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.
Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

SpaceOps-2023, ID # 182

Concepts and Implementation of Generic Services and HMI Elements

F. Croce[a]*, C. Miranda[a], R. Gad[b], A. Simonic[b]

[a] *European Organization for the Exploitation of Meteorological Satellites (EUMETSAT), Eumetsat Allee 1 64295 Darmstadt Germany, francesco.croce@eumetsat.int, carlos.miranda@eumetsat.int*
[b] *Terma GmbH, Bratustraße 7, 64293 Darmstadt Germany, ruga@terma.com and alsi@terma.com*
* Corresponding Author

Abstract

As part of applications evolution roadmap, EUMETSAT is progressing in the development of a set of new generation of software elements within the Monitoring and Control Applications (M&C) functional domain. The architecture is based on multi-systems multi-mission generic elements on top of which different specific high-end applications can be defined and developed in terms of back-end cloud native microservices hosted by Platform-as-a-Service (PaaS) provider and front-end Human Machine Interface clients based on web technologies. After conceptual phase covering the definition of the roadmap strategies such as re-usability, high level of configurability and scalability, infrastructure and PaaS provided services, the activity has entered the detailed design and implementation phase. Like with any new implementation, this phase has been facing challenges and compromises but the original high level strategies have always been the reference for trade-off analysis when facing low-level decision taking. This paper summarizes the original strategies and design concepts, followed by a description of engineering and detailed design aspects, including trade-off considerations. Lastly the paper provides a description of one specific application use-case and presentation of status, next steps and evolutions.

1. Introduction

EUMETSAT has the mission to establish, maintain and exploit European systems of meteorological satellites, to contribute to operational monitoring of the climate and detection of global climatic changes.

The organization operates a number of GEO and LEO multi-spacecraft missions (including European Copernicus) while developing Ground Segments for new programs, with ESA typically developing the Space Segment.

The number of such programs has been increasing considerably in the last 5-7 years, including the intention to develop and operate a future constellation (EPS-STERNA) of mini-satellites currently under preliminary analysis prior to actual development phase approval.

The whole scenario, quite different from the time when only two programmes (MSG and METOP) were operated, together with general considerations such as relevant progress of technologies, IT service models, need of budget optimization, is clearly demanding a new approach to ground segment development and then maintenance.

The obvious considerations of: harmonization; optimization; elements reuse with multi-mission support; overall design and up-to-date technology with sustainable and predictable maintenance; short customization for new missions; minimizing expertise silos; offering functionality "as-a-service" rather than "as-a-system", have all been injected as driving strategies into the definition of a re-engineering roadmap for future Monitoring and Control (M&C) applications.

Part of the functionality "as-a-Service" aspect, is to take advantage of the Infrastructure-aaS, Platform-aaS, Software-aaS models and native cloud technologies. These have been injected into the roadmap definition as fundamental design and technology drivers. Partitioning system elements into layers of responsibility, each one offering formal services, has also driven EUMETSAT to move to an organizational structure with competence units dedicated to infrastructure, platform (including data services) and applications. Each unit operates with its service provision agreement towards the reference "upper client" layer or the end user in the case of application elements.

The drivers and the considerations above, have led to defining and developing new systems with a new approach that establishes a formal stack of layers. The application layer includes generic and specific service units (microservices) for the back-end and the front-end display elements. Any generic element is defined and designed to be multi-system and multi-mission on top of which specific elements fulfilling needs can be implemented.

In terms of IT hosting environment, EUMETSAT operates its own private data center, infrastructure and platform elements which are provided as services within the overall architecture layers stack. Although the adopted native cloud technologies allow deploying applications on different cloud providers (e.g. public), the deployment is done using EUMETSAT private data center due to constraints linked to internal policies.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

2. Design and Technologies Drivers Overview

Among the identified strategies, the reuse approach is the fundamental driver, enabling development and maintenance cost efficiencies and promoting fast cycles during the development of new ground segments.

Re-use has been identified in terms of:

- Multi-System (Horizontal re-use): generic elements that can be re-used across applications, each supporting different functional domains;
- Multi-Mission (Vertical Reuse): generic elements used as core elements combined with mission specific elements or extensions when an application is to be instantiated by a specific mission.

The multi-mission re-use clearly benefits of the multi-system re-use whenever different applications, defined and developed re-using multi-system generic elements, are instantiated within different missions.

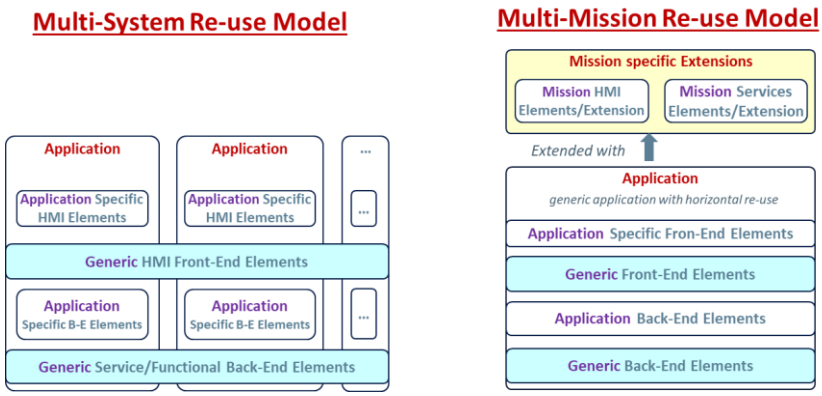

Fig. 1. Re-use Approaches

Both generic and specific elements need to follow a well-defined engineering approach in order to keep them well separated for easy maintenance and dependencies control but still compliance with the architecture foundations.

Each application associated to a system functional domain is logically associated to an **Application Family** (e.g. Flight Dynamic System, Mission Planning System, etc…) and multiple instances of an application family can be deployed in the runtime environment each one with its runtime **Context** (configuration, data sets, interfaces, …).

The concept of application family and application instances/contexts was present in the original system concept and was further extended with the possibility to have multiple instances of multiple families deployed in the same runtime environment sharing generic services and data aggregation within the same operational mission deployment.

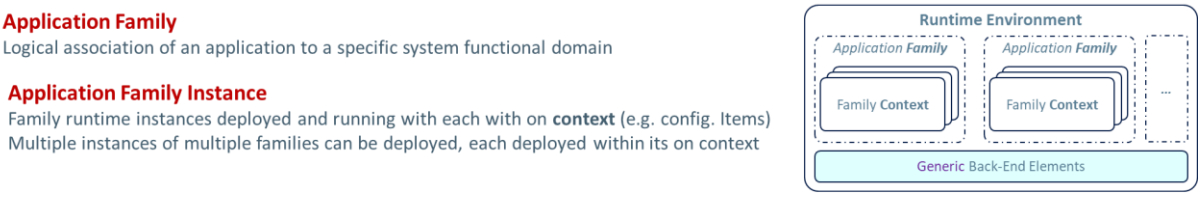

Fig. 2. Application Family and Instances/Contexts

2.1 Back-End Elements Architecture Model

Together with the re-use strategy, the back-end elements architecture model foresees the concept of self-contained units of service with each unit exposing **formal** interfaces to access the functionality they provide.

The principle resonates with the concept of component-based approach where a component is considered a functional black box accessible through formal interfaces. This was initially considered in terms of self-contained software modules (components) to be combined into a higher level aggregation but all running within the same application runtime space and a prototype was developed adopting a formal component technology framework.

However, the approach was not considered suitable due to concerns on how long-term maintenance could be achieved effectively. Particularly when considering the presence of significant dependencies with 3rd party off-the-shelf libraries (a quite common scenario in today bespoke applications) and the intention to have distributed components across the network. In addition, the intention to take advantage of multi-languages and multi-technologies solution was clearly not possible with this approach.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

Considering the maturity in cloud technologies, containers and container's orchestration, the attention was redirected to "**microservices**" architecture. This was heavily motivated by the possibility of partitioning applications into formal independent **service units** providing functionality accessible through the defined formal interfaces, which was a high-level requirement. Service units are then configured as runtime containers deployed within a cluster with **kubernetes** as containers orchestration part of Platform-as-a-service data center provision, taking advantage of elasticity and redundancy capabilities of the combined of the PaaS/kubernetes technology stack.

2.1.1 Service Unit and Service Layering

Each **Service Units** exposes functionality through formal **REST APIs** used as interfaces when unites interact among each other as well as with Human Machine Interface (HMI) clients in the case services are required to be requested/consumed by the end-user.

REST-based interfaces are specified through the ***OpenAPI*** Specification (OAS) [3], which defines a standard, programming language-agnostic interfaces description for HTTP APIs, allowing humans and computers to discover and understand the capabilities of a service without requiring access to source code or additional documentation.

In addition, it has been recognised the need of a communication infrastructure supporting message-based exchanges with the availability of a Message Oriented Middleware (MoM) supporting publish/subscribe mechanism patterns and able to support events distributions management, critical data message exchange and in general any interactions requiring streaming of data. The architecture, therefore, includes a messaging middleware broker solution.

As further step, the evidence that the requirements spaces of any application share very similar - if not almost the same - low-level and generic functionality - such as logging, event management, data transfer, etc.. – induced to define a set of common and domain agnostic services. The architecture model then includes the formalisation of **Middleware and Generic Services** (MGS) layer where **Middleware** services provide support for mechanisms such as logging, messaging and low-level database engine, while **Generic** services provide functionality such as service registry, events handling, file systems management, sessions management, etc.

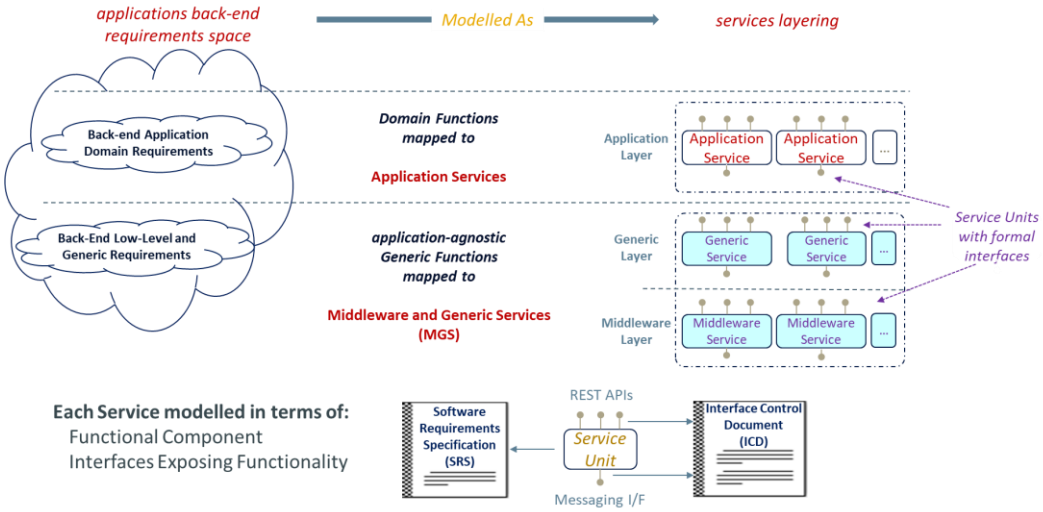

Fig. 3. Service Units and Services Layering

MGS must then be considered application agnostic that is no functional dependencies with a specific application domain and as such to be re-used across different type of functional domains.

MGS is defined, developed and maintained as an independent product with its lifecycle and technical baseline as well as deployed in the runtime environment as single instance to support multiple application families.

It is important to stress that any MGS and upper layer application specific service follow a common service unit definition paradigm but associated to their technical baseline (i.e. Software Requirements Specification, Interface Control Document, etc..). Any service is deployed into the runtime environment through the same mechanisms.

2.1.2 Application Family and Context

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

As mentioned in the introduction, an application supporting a specific functional domain is associated with the concept of **Application Family,** which is used as a logical partitioning mechanism when multiple application domains coexist in the same deployment environment.

It is possible to run multiple instances of an application family, with each instance associated with a **Family Context**. A context is an independent instance of the application family running with its dedicated configuration and properties. Examples of an application family could be Flight Dynamics System (FDS) or Mission Planning System (MPS). Contexts could be an instance supporting operations for a specific spacecraft within a constellation or an instance dedicated to testing and verification activities.

Application families with their contexts are meant to run in parallel, all using a **single instance** of MGS in support to all families and contexts. It is possible to run a single family context with its own MGS, a MGS supporting multiple contexts of one specific family or MGS supporting multiple contexts of different families.

The combination of MGS and the set of application contexts runtime scenario is driven by a deployment configuration. The level of configurability allows in any case to have a runtime scenario according to deployment requirements and/or constraints.

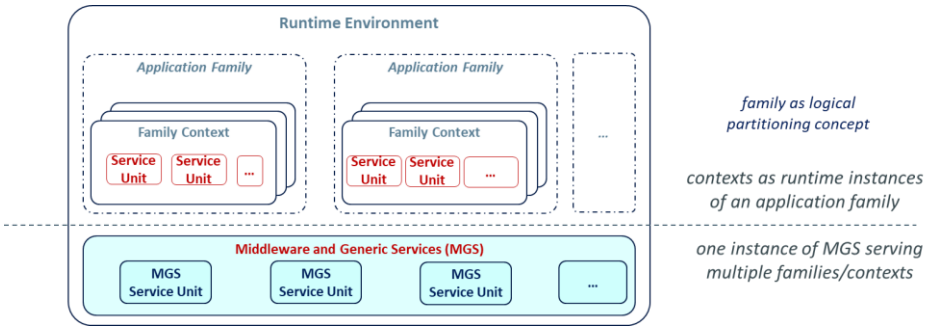

Fig. 4. MGS, Application Family, Services and Contexts

2.2 Front-End Elements Architecture Model

The architecture model for end-user clients (HMI Clients) foresees the same multi-system, must-mission reuse strategy as for the back-end services.

The model foresees a generic display technology platform, a generic framework, generic UI components and application domain-specific component and the possibility to customise and extend any generic or domain-specific components for a mission specific needs.

The HMI technology platform is able to support multiple operating systems, namely *Linux*, *Windows*, and *MacOS*, although Linux is the formal baseline.

Web-based technologies have been selected for the whole HMI development in terms of *HTML*, *CSS*, *Javascript* and associated 3rd party frameworks. This is in the view of the high level of maturity of such technologies, their rich ecosystem in terms of integration with other open source and commercial solutions allowing to extend HMI applications functionality with very low effort.

On top of the technology platform, a generic HMI framework provides basic display management and services functions and on top of the framework, generic and specific components can be assembled toward a specific application. In addition, a flexible tailoring process allows to further configure and customise the HMI through contributions artefacts using features provided by the framework and generic components. Part of the tailoring is also the possibility to mashup the whole HMI application with other internal and external web applications.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

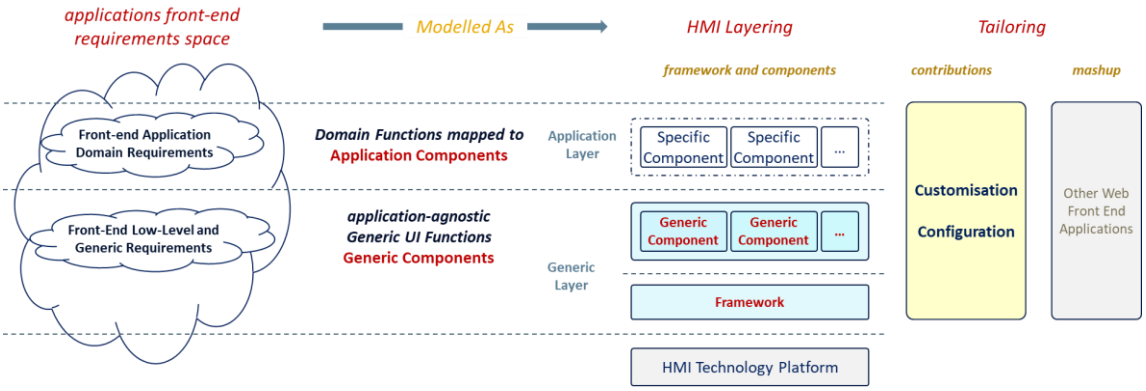

Fig. 5. HMI Architectural Elements

2.3 Overall Architecture and Technology Stack

The picture below depicts the overall stack that encompasses the infrastructure, platform, application (back-end and front-end) service and components layers.

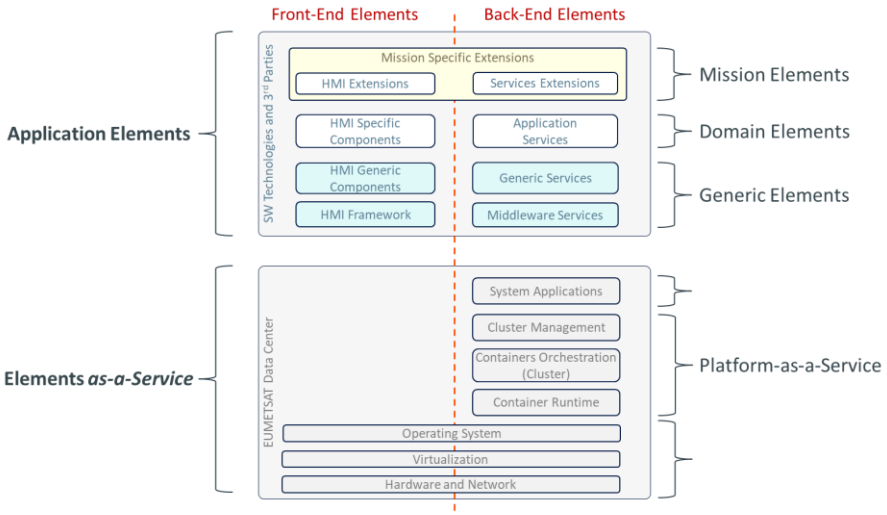

Fig. 6. Overall Architecture Stack

The data center provides the elements as a service with technologies and applications to manage all relevant application workloads, associated data and network services and required system applications.

EUMETSAT's concept of an Information Centric Service Infrastructure comprises multiple layers, reaching from IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service) to DaaS (Data as a Service). Through this service stack, different EUMETSAT internal users' needs can be addressed, reaching from online-data access, hosted processing, Web Services, Web Map Services, Image visualization to Earth Observation data format. Cloud and cluster technologies are provided natively.

In the case of applications related to the M&C roadmap, the intention is to take as much as possible advantage of the provided data center services including delegation of scalability, elasticity and most importantly, runtime and data redundancy. **Containerization** and containers **Orchestration** as part of the IaaS and PaaS data center, are used to as runtime environment for MGS and applications services.

Docker [4] is used as containers runtime engine and **Kubernetes** [5] as containers orchestrator and configured as a cluster. In particular, each service unit is built as a container image and runs within its dedicated container and additional services (sidecars) containers as needed, all deployed within dedicated kubernetes POD deployments.

Many Kubernetes features are used: container instantiation and runtime management, load balancing, cloud virtual network, local (per container) and external storage management, automated rollouts, configuration properties, and containers self-healing. The *namespace* feature logically separates application families and context when

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

deployed in the cluster, with each context running within a dedicated namespace. This applies to MGS services as well.

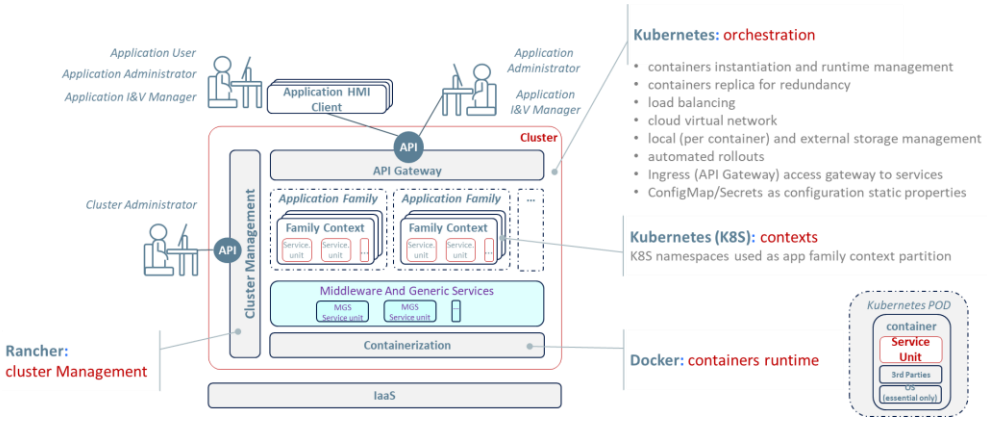

Fig. 7. Applications Hosting Cluster

As part of the PaaS layer, kubernetes services are delivered based on a customized *cloudibility* (cloud management and optimization) custom installation of kubernetes. The cloudibility approach offer integrated solution to deploy and manage multiple high available kubernetes clusters including monitoring, logging, CI/CD, version control and user management.

Rancher Lab is used as *cloudibility* kubernetes platform. Rancher is a cluster technology supplier in this market with main features such as: multicloud orchestration, enhanced security features, unified multicluster Kubernetes management platform supporting many public and on-premises infrastructure platforms, highly customizable access controls, policy enforcement, resource isolation, and image scanning. As part of any PaaS additional applications can be offered as a services including applications behaviour observability and monitoring.

Back-end service units are mainly developed as java applications using Spring Boot as backend framework. This is the current baseline for all services but future service units can be developed using other languages.

Regarding end user HMI applications these are deployed outside of the cluster and they access service units through the cluster API gateway. End-user applications can also be scripts or Command Line Interfaces (CLI) tools invoking APIs from a non graphical user interfaces.

3. Generic Elements

3.1 Back-End Elements: MGS

As mentioned, the Middleware and Generic Services (MGS) provide an application domain agnostic layer of services implementing functionality that has been considered part of the typical back-end requirements space of many applications and in particular applications that are and will be defined following the layered architecture model

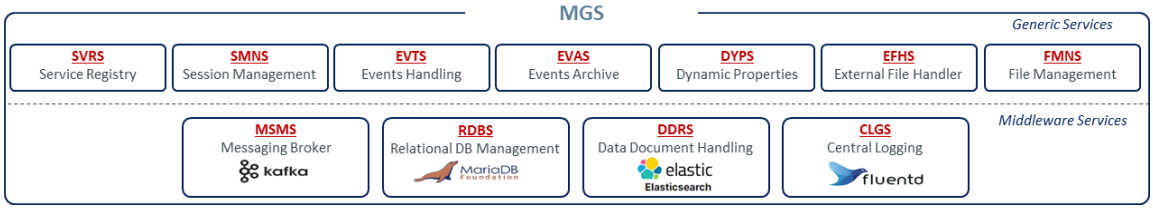

Fig. 8. Middleware and Generic Services

3.1.1 Middleware Services

Middleware services currently part of MGS are:

- **Message Oriented Middleware and Storage Management Service** (MSMS) provides message broker functionality and message persistence for message communication between loosely coupled elements and is compliant with the publish-subscribe model. In this model, entities can publish messages stored within message supporting resources – in our case topics – and other entities can subscribe to these resources to be notified when a message of interest is available by receiving a copy of the message. The publish-subscribe

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

model is implemented by message brokers component which in the case of MGS MSMS is with *Apache Kafka* [6];

- **Document centric Data Storage and Retrieval Service** (DDRS) is the central database, providing secure, scalable and highly-available storage and retrieval functionalities of unstructured data as required by other services, like CLGS (logs) and EVTS (events). This high performance, availability and resilience against data loss is implemented as an *Elasticsearch* [7] cluster and, on top, *Kibana* [8] to provide a visualization dashboard to be embedded into other HMI application components;
- **Central Logging Service** (CLGS) is responsible for collecting and storing log messages from local pods on the central archive, indexing and making them available for later analysis via queries on the CLGS storage. The interception and forward of logs from pods is implemented with *Fluentd* [9];
- **Relational Database Manager Service** (RDBS) provides the relational databases management functionality. As there is not distinction between application families and contexts within RDBS, databases include the application family and context on their naming scheme. Relational databases are implemented in the RDBS with *MariaDB* [10] but, as an off-the-self solution is applied and application services may have specific needs (or need of non-relational databases), other local databases management services can be deployed.

3.1.2 Generic Services

Generic services currently part of MGS are:

- **Service Registry**: it provides definitions of available application families and contexts, URLs/endpoints for accessing services and access to the service state/liveliness;
- **Session Management**: it provides back-end services and HMI clients with functionalities needed to identify, request and operate user sessions. It is responsible for accounts roles and privileges management, authentication, authorization as well as dynamically assigning/revoking privileges for active sessions;
- **Events Management**: events are structured messages raised by any running entity and consumed by services and HMI clients to notify something has happened or about to occur. Some properties of events can be configured as part of event definitions, for example, severity to be associated to an event ID. Applications can register for receiving selected events for which they will be notified through the messaging middleware service. In addition, events are archived through the DDRS service and can be retrieved for analysis.;
- **File Management**: file management services provide a centralised mechanism to store and access files (similar to Cloud storage service). Files are organised in the same way as conventional file system. The service then provides a level of files location transparency;
- **Data Transfer Handling**: the data transfer services are responsible for providing send and receive files functionalities between internal and external systems. It acts as a transfer gateway and mediates in sending and receiving files between source and destination locations. It connects to multiple destinations and monitors them for incoming files. Whenever a new file is detected in configured file systems specifications, a message is published using middleware messaging services;
- **Dynamic Properties**: allows storing and sharing dynamic properties (value-pair) for all services. Any change of dynamic property from an authorised entity is notified to other services (MIS or applications) that registered for receiving notifications of changes for that particular property. Through dynamic properties mechanisms, it is therefore possible to change at runtime services behaviours driven by dynamic properties values. This is in contrast to behaviour driven by static configuration, which requires a restart of the service whenever a static property is changed.

3.2 Front-End Elements: Generic HMI

The front elements (HMI) have been designed around a generic HMI framework. Components and technologies have defined to be highly customizable and adapted to a wide range of HMI applications and environments.

Key characteristics are:

- use of web-based technologies such as HTML, CSS, and JavaScript, which can be easily accessed and modified by developers and users to create custom user interfaces and functionalities;
- use of modular and flexible design patterns allowing custom libraries, scripting, adoption of 3rd parties again allowing to easily add or remove features and functionalities as needed at low cost;
- ability to connect to and communicate with other devices and systems using APIs (Application Programming Interfaces) and other web-based technologies, such as WebSockets and HTTP;
- possibility to mash HMI functional view elements with "external" web applications.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

More formally, the architecture model includes:

- A 3rd party hosting platform covering all HTML, CSS, Javascript engine and libraries. The basic HMI platform is based on Electron [11] with ad-hoc frameworks based on Angular [12]. Electron is a platform for developing desktop application leveraging web technologies. The adoption of a desktop platform was a direct consequence of user requirements addressing access capability to the local files system. However, because of the overall technology approach, migration to web browser would imply very minor adaptations;
- Generic HMI Elements developed specifically to be application domain agnostic and including:
  - Generic Framework: functional domain-agnostic building block implementing all fundamental HMI services such as layout management, extensions points, etc.
  - Generic HMI Components: set of generic display components that can be instantiated on top of the framework and serve generic functionalities
- Specific elements required by the functional domain that an end-user application is required to support and operate. This is optional and depending on the functional domain needs.

As mentioned, the overall architecture allows a high degree of customisation and configuration. As such, an end user HMI application definition, development and configuration, include three stages:

- **Components Development**: development of specific components on top of the HMI framework and generic components, toward a specific application functional and mission domain. However, this stage may not be required and depends on the specific functional or mission domain to be supported. In addition to specific components, generic components design allows their extension should this be required by a specific domain or mission;
- **Application Assembly**: assembly of generic elements (framework and components) together with application domain specific components (if present);
- **Application Tailoring**: achieved with the development of so called "contributions" items that can be referenced and run through the configuration and customisation points provided by the generic HMI framework. Part of the tailoring is also mashup with external or internal Web applications.

While components - generic and specific – are "fixed" elements in terms of functionality, it is through tailoring that the application can be hugely customised and extend its "services" offered to the end-user.

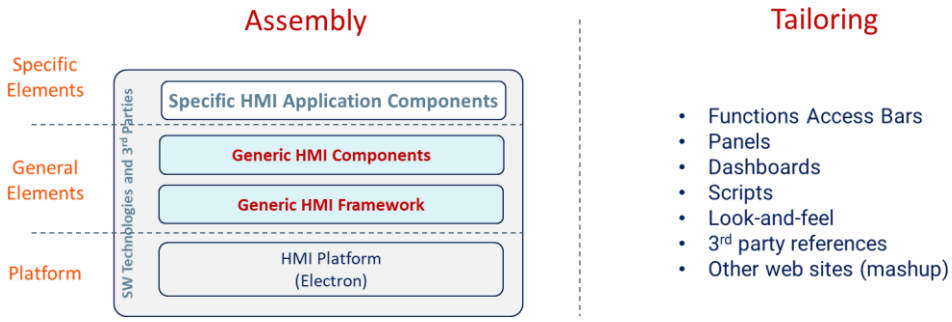

Fig. 9. Generic HMI Elements Stack and Tailoring Items (*contributions*)

3.2.1 Generic HMI Framework and Components

The generic HMI provides a set of generic components that can be instantiated and assembled by an end-user application as part of its requirements space.

Some of these components present different level of configuration and customisation access points as well the possibility to be created with multiple instances within a specific application assembly.

Following is the list of such components within logical functional groups:

- **Functions Access Bars**: Menubar, Toolbar, Status Bar, Vertical/Actions Toolbar;
- **Layout components**: Accordion, View Container, innerHTML container, iFrame container;
- **Editors**: File Editor (FED), Javascript Editor (JSED), XML Editor (XED), HTML Editor;
- **Browsers**: File Browser, Resources browser, File Comparison;
- **Static Content Viewers**: PDF, Image, Help, Property, About;
- **Supporting Views**: Problem, system console;
- **Terminals**: access to local and remote file systems through embedded terminals;
- **Miscellaneous**: login display;
- **Displays**: Panel Container.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

Among all mentioned components, Web Browser, Resource Browser and Panel Container have a particular relevance:

- **Web Browser**: used to render the UI of web applications running within the cluster (part of set of back-end functions) as well as of application external to the cluster considered relevant to complement the overall set of functions set available to the end user;
- **Resource Browser**: provides a tree hierarchical representation of items (resources) with each item configured to allow the user to invoke a variety of different functionality. As examples, a tree item can be associated to a script, a link to a panel (see next section) to be displayed in a Panel Container, a document or data file (e.g. pdf) to be rendered in the associated editor or viewer, or a link to a web page/application to be displayed in a Web Browser instance;
- **Panel Container**: allows to display user developed Panels (see next section).

Another capability of the framework is to easily integrate 3rd party elements such as javascript libraries and packages to fulfil different functional objectives such as: UI widget 2D, 3D plotting chart and special display component libraries, specific mathematical computations, data management and processing, etc.

3rd party libraries functions can be invoked and used by generic and specific components as well by most of the contribution items part of the tailoring process. The inclusion of 3rd party libraries and their direct usage within tailoring items, are a way to provide additional rich functionality not native in the framework and components with a very short deployment and availability timing.

3.2.2 Panels

Among all contribution items, Panels are the major tailoring elements. A Panel is a mini display application performing a variety of functionalities and presentation layer experience. Panels can be of two types:

- **Data Management:** forms acting on data files (or other data sources) allowing data CRUD (Create, Read, Update and Delete) operations;
- **Display Management:** general displays for different purposes and data sources representation.

Panels are implemented through html/js/css snippets rendered within an innerHTML or iFrame component within the Panel Container generic component. These technologies offer a rich variety of functionalities, for example:

- CRUD operations on data files following pre-defined schema and supporting data types, for instance: real, integer, string, time, boolean, arrays as well as records and lists of the mentioned simple data types;
- 3D and 2D images representation and animations (i.e. 3D satellite models representation or real-time Satellite orbit representation on 3D Earth model);
- Embed documentation (i.e. PDF) and additional browsing contexts (i.e. Web pages) within panels;
- Usage of JavaScript in order to:
  - Perform data validation on Data Management panels (i.e. check values correctness, etc.);
  - Customize display representation based on values contained in pre-defined data files (i.e. change display colours based on values contained in data files);
  - Develop mini-applications with different purposes (i.e. to perform calculations);
- Invoke system commands which can be tailored by each specific application (i.e. buttons triggering the execution of another HMI component or back-end services);
- Easy integration of 3rd party libraries and applications.

3.2.3 HMI Tailoring: contributions

As mentioned, after assembling an end-user application by instantiating generic framework, generic and specific components, there is the tailoring process aiming to configure and customise the display and behaviour functionality of the application. This is achieved through contribution items.

Contributions are considered configuration items towards the operational usage of the application fitting the end-user HMI overall functionality but also able to evolve with the end-user lessons learned and evolving requirements.

Through contributions it is possible to tailor: **Functions Access Bars, Resource Browser Settings, Panels, Scripts, look-and-feel (styling through custom css).**

Contributions items are developed outside of the HMI application using an external IDE, which, by the way can be launched through the HMI in order to maximize user/developer efficiency. The default IDE is MS Visual Studio Code [13] but eventually other IDEs can be used too. The external IDE serves the purpose to provide a fully featured development environment with syntax highlighting, debugger, etc.

When development/verification cycle of involved contributions is completed, contributions can be checked in into a dedicated GIT repository for contributions configuration control.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

The technologies used for panels allow the inclusion of external web applications within the HMI. For instance, it is expected that the following web applications will be embedded in the HMI applications:

- Rancher: allowing management of the cluster hosting back-end services;
- Kibana: allowing search and data visualization for data indexed in Elasticsearch;
- Altassian Jira: providing access to issues tracking management tool;
- Altassian Confluence: allowing access to workspaces hosting knowledge and allowing teams collaboration;
- Wiki: internal tool hosting information and knowledge related to applications, systems and services.

3.3 Generic Testing Framework

The original technical and development lifecycle requirements asked for a high level of automatic testing at component and system level for both back-end and front-end elements. Unit testing was required to be ensured by the test unit suite technology for the software language in use by a specific element.

The most relevant requirements for component and system testing, pointed to a *Generic Testing Infrastructure* able to define testing artefacts (plan, scenarios, cases, procedure and procedure steps) addressing the following capabilities:

- Hierarchical Structure of Test Artefacts (plan, scenarios, cases, procedure and procedure steps);
- Scripting for procedures with formal constructs to support pre-conditions, steps and pass/fail criteria assertions;
- Support to the maximum extend for automatic testing but also manual testing for all those cases where automation is not possible or not;
- Documentation generation for all testing artefacts including test results;
- Possibility to define and run test artefacts from both a test conductor IDE/Executor, from command line and from continuous integration (CI) pipelines.

In terms of technical solution, the requirements pointed to an open-source solution to be used *as-is* or through integration or modification of suitable 3rd party packages.

A survey of possible ready-to-use solutions resulted with a high level of compromises hence considered not acceptable. Consequently, the direction was to integrate and customize available packages and libraries with a test artefacts definition and test conductor environment based on Microsoft Visual Studio Code [13].

The whole integration/customisation process was performed in a semi-agile approach and, similarly to the HMI development, detailed needs combined with possibilities offered by the adopted technologies, allowed to refine and extend the framework with many features not originally envisaged.

A description of the generic testing framework is available at [2].

3.4 Changes During Implementation

As part of generic elements implementation, a number of changes have been addressed with respect to the original requirements. The major ones were experienced in the Generic HMI and Testing Framework.

For example, the HMI tailoring capabilities and in particular Panels, access points and integration with other web applications, were not defined with the described level of rich flexibility and possibility.

Similarly, the testing framework experienced extensions to meet more and more the needs of the integration manager and testing team.

These changes became evident during the implementation phase of the different elements, when the combination of a semi-agile development process, the adopted detailed design and the set of technologies they all made apparent that moving to enhanced features and capability was a relative minor step.

4. Application Case: Flight Dynamics System

The current operational Flight Dynamics Systems (FDS) deployed in the different EUMETSAT Ground Segments are based on very dated technologies, monolithic architecture and, as such, it imposes severe constraints in terms of maintenance and evolutions.

A new generic FDS solution has been defined adopting the new described architecture model. Flight Dynamics (FD) application layer services and FD HMI application have been designed and developed by re-using MGS and the generic HMI. The resulting generic system is planned to replace all current Flight Dynamics Systems for LEO and GEO missions.

The new generic FDS architecture consists therefore of the following logical elements:

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

- **FDS Application Services Layer**: it provides the processing back end, acting as central interface between the FDS Client and the FDS Computational Programs and FDS Data Model (see below). The layer includes a set of back-end services – using MGS - managing the access to the FD data model and related resources;
- **FDS Client**: it implements the user front end (Human Machine Interface - HMI). It reuses the generic HMI Framework and MGS components together with additional domain-specific components. It interacts with the back-end services through cluster ingress;
- **FDS Computational Programs:** they implement FD algorithms used for operations and products generation. It includes functions for: Orbit Determination, Orbit propagation, Manoeuvres calculation, Products computation and generation, etc. The FDS Application Services Layer and the FDS Computational Programs interact through a well-defined and dedicated interface which allows independent evolution and maintenance of both subsystems;
- **FDS Data Model**: set of table entities compliant to a formal database Entity-Relationship schema, and used to store data related to FDS Computational Programs configuration and Mission model entities (i.e. Satellites, Ground stations, etc.). The relational database is hosted and managed by the MGS RDBS component.

4.1 FDS Back-End Elements

The FDS Application Services Layer include:

- **Environment and Mission Model Service (EMMS):** it provides FDS with a central service for accessing and managing the FDS Data Model. Any request to read or modify the data model shall go through the EMM services. Such approach guarantees that the data model is accessed in accordance to central access rights and conflict resolution rules implemented by EMM;
- **Activity Execution and Scheduling Service (AES):** it is responsible of scheduling and managing activities execution such as FDS Computational Programs execution, scripts execution, etc. AES provides the FDS Application Services Layer with functions such as:
  - Schedule preparation, based on a specific scheduling logic. This allows the execution of activities (i.e. FDS Computational Programs) based on certain time logic (i.e. every day, every day of the week, with certain periodicity, based on ground stations visibilities, etc.);
  - Sequencer allowing also the definition of set of FDS Computational Programs (or other application scripts) which are executed in sequence based on certain logic defined by the user via scripting language (*Javascript*);
  - Activity Scheduling and Execution engine, whose implementation is based on an open-source solution (*Quartz Scheduler*).
- **Other Services:** In addition to the two main services mentioned above, other services are part of the FDS Application Services Layer which provide different functions such as: managing flight dynamics operational files and configuration (CTX service), generation of graphical plots based on files produced by the FDS Computational Programs (GRG service) and controlling input files reception and sending (INF and OUF services respectively).

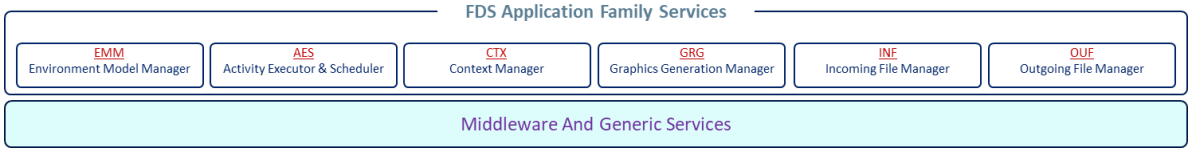

Fig. 10. Flight Dynamics Application Family Services

4.2 FDS Client

The FDS Client is based on the full reuse of the Generic HMI Framework and its generic components. Due to domain-specific requirements, it has been necessary to develop a set of specific HMI components. Major ones are:

- **Chain Editor**: HMI component used to configure activities (i.e. FDS Computational Programs) scheduler and sequencer. It allows the user to configure the logic for executing activities based on temporal rules, orbital events, or other rules defined by the user;
- **Activities Executions and Schedules View:** HMI Component allowing the user to check the status of the activities executions and schedule. The user is able to explore the status of past, present and future (scheduled) executions through Gantt chart view or list view;

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

- **Events Gantt Chart View:** HMI Component allowing to represent in a Gantt Chart the Orbital events generated by the FDS Computational Programs;
- **Browsers:** generic resource browser provided by the Generic HMI are customized and extended such that browser items allows user to trigger FDS specific functions.

As from the HMI architecture model, the FDS Client is assembled using the generic framework, the generic components and the FDS specific components. In addition, tailoring is applied through development of specific panels, mashup with other relevant web applications and overall customization in terms of menu-bar, toolbar, action bar and help files. In particular, the tailored process includes:

- **Panels** supporting a number of data displays and CRUD (create-read-update-delete) operations related to specific Flight Dynamics data and configuration;
- **Resource browser** for referencing any relevant resource (data file, panels, documentation, etc..) and associated action;
- **Function Access Bars:** Menubar, Toolbar and Vertical/Actions toolbar is tailored in order to provide access to specific Flight Dynamics Functions;
- **Scripts** implementing functionality accessing data file and data model entities. Scripts can be invoked independently, through menu/tool/action bars items or from panels. Scripts can be javascript and shell.

In addition, the user can define **Dashboards** which are panels with specific presentation layout defined to present access points to many FDS functionality within an aggregating homogeneous operational experience.

Panels, dashboard and scripts can make use of 3rd party libraries and packages that needs to be dropped in the HMI installation directory and use directly as part of the tailoring process.

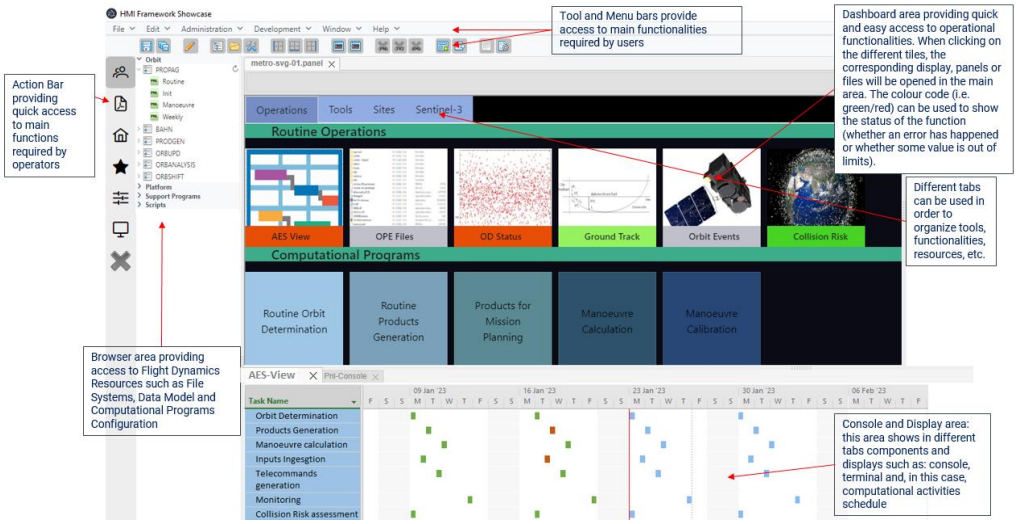

Fig. 11. Snapshot of Flight Dynamics HMI displays and panels

It is expected as well that Flight Dynamics panels will provide not only the possibility to act on data and functions access points but also, to embed documentation (PDF) and external web sites.

4.3 FDS Operational Use-Cases and System Configuration

At EUMETSAT, several instances of Flight Dynamics System are deployed within the ground segment of each Mission in different environments, each environment with a specific role. These are:

- **OPE:** Operational Environment for routine and special operations: orbit determination and propagation, ground track monitoring, manoeuvres computation, telecommands computation and products generation;
- **VAL:** Validation Environment used to validate new releases of flight dynamics system;
- **VER:** Verification Environment used to verify new releases of flight dynamics system;
- **ENG**: Engineering Environment used to develop operational configuration.

Within each environment, it is expected to deploy an independent cluster, each cluster containing several **Family Contexts**. The objective is to have, in each environment, different Contexts associated with different FD activities. As an example, within the OPE environment (cluster), the following Contexts are foreseen:

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

- **Routine Operations Context:** used to perform flight dynamics routine operations such as: spacecraft telemetry reception and processing, orbit determination, orbit propagation and routine products generation and dissemination;
- **Special Operations Context:** used to perform flight dynamics special operations such as orbital manoeuvres computations, special spacecraft telecommands generation, instruments calibration operations, etc.
- **Operations Monitoring Context:** used to perform some monitoring activities of different flight dynamics processes such as orbit determination performance analysis, etc…;
- **Anomalies Investigation Context**: used to perform any required trouble shooting or anomaly investigation of a problem occurred in any of the previous contexts.

Within each **Family Context**, it is expected to have a full independent deployment of the Flight Dynamics back-end Services and Data Model (i.e. avoiding conflicts among them).

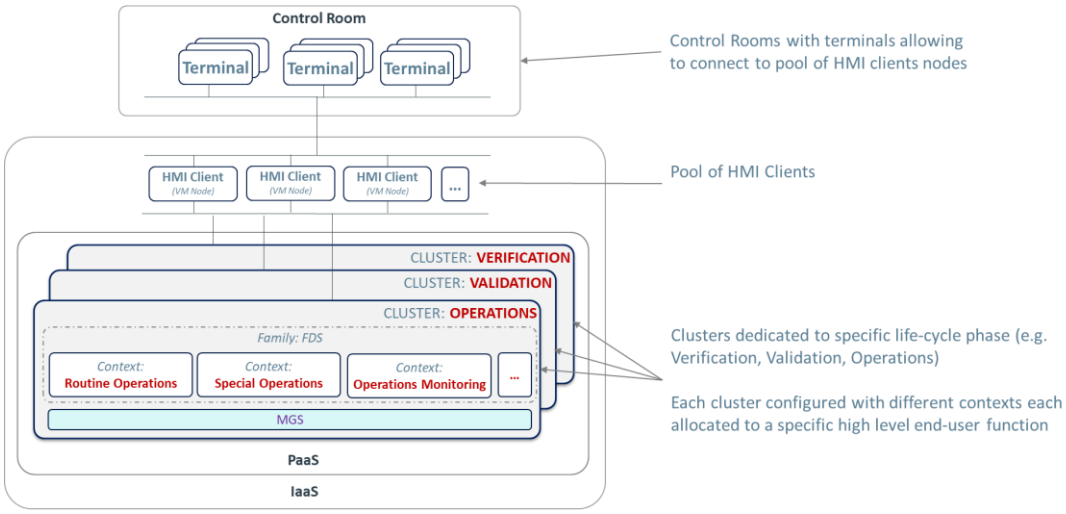

Fig. 12. FDS Environments (clusters), Contests and HMI

In order to access the back-end services, there will be available a number of terminals which can be used to connect to a pool of FDS Clients. In principle, it is expected that an FDS Client belonging to certain Environment can only access Family Contexts defined in that Environment, for security reasons. Nevertheless, the architecture does not impose any restriction at this respect and it may be decided that a Flight Dynamics Client belonging to any Environment (or even outside any of the environments) could also access Family Contexts defined in other Environments. For instance, it could be possible that a FDS Client installed in VER, could access Family contexts from ENG.

5. Status and Evolutions

The development phase of the described elements is approaching the final delivery covering the original MGS, Generic HMI and FDS family baseline requirements.

The decision to develop the generic elements together with a specific application family (FDS), in contrast to a sequential approach - MGS and Generic HMI first and then FDS - has allowed concurrent and integrated design definition and implementation across all layers with the possibility to tune the generic elements detailed design with a real application family case.

The objective has always been to have a stable and verified generic elements ready to be used by other applications families' - part of high level re-engineering roadmap – resulting in an optimised, effective and as much risks free as possible, application families' development life cycle.

While clearly targeting overall functional, stability, robustness and performance verification of the current developed elements, there are already identified a number of next steps as well as MGS and Generic HMI evolutions.

In terms of next steps, additional application families are approaching their definition phase (requirements) taking full advantage of a delta development on-top of the generic elements. Candidate families are related to: mission planning, payload data production performance monitoring, mission control system data remote monitoring and operations preparation.

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

In terms of generic elements evolutions, extending MGS with data analysis and diagnostic services is a clear need to provide end users with capability to derive real-time and post-processing information and diagnostics rather simple records of data requiring post human processing. This in relation to data generated by one family as well as correlation across families. Information and diagnostic extraction is a critical aspect considering the volume of events and other type of data recording handled by logging and documentation services. Machine learning and data analysis services will play a key role.

Regarding generic HMI elements, evolution will be toward smart components and exploiting all possibilities already part of the tailoring process. Having adopted a web technologies stack, the intention is to take full advantage of their ecosystem evolution including 3rd party libraries toward a rich user experience that can be delivered with a very fast customisation life cycle, all through a tailoring process rather than software components development.

In relation to PaaS, it is also clear that platforms evolve too and they will expose more services and system software as a service that can be integrated with the MGS layer at back-end end side and having their front-end mashup with the HMI clients.

6. Summary and Conclusions

This paper describes an architecture model and related elements following a service layer approach, using PaaS, native cloud, containers, orchestration (kubernetes) and web technologies.

The model includes the concept of service units deployed as microservices on the back-end with the formalisation of application agnostic middleware and generic application services (MGS) on-top of which specific application family's services can be defined, developed and deployed in the cloud.

Similarly, the front-end HMI elements include generic and specific components with two main characteristics: first is the enrichment of user operability, enhancement of the user experience and in general, improvement of efficiency while operating the systems. And second, efficient and rich tailoring of HMI capabilities in order to provide high level of user experience customisation targeting the operational context and efficient interaction with the whole system within the different application families.

The overall model is not new in today´s software applications development offered as-a-service (SaaS) hosted by infrastructure and platform with cloud and clustering technologies.

The challenge of the roadmap has been defining an architecture and technology framework allowing the migration of selected M&C functional domain to an integrated and common model. This implies maximizing re-use of elements at different level of abstraction, define common mechanisms such as sessions, events file management, data management in terms of data units structure, storage and retrieval mechanism and processing.

Combining MGS features, which offers common services to multiple application families each with multiple instances, together with the Iaas/PaaS capabilities such as scalability and elasticity, resilience, agility and added-value services, allows deployments with different level of flexibility and performances.

Besides the technical and engineering considerations, a common approach across functional domains target also avoiding silos of knowledge currently existing among engineering and end-user teams of different domains using heterogeneous solutions. This is a non-negligible factor that has typically high cost and implications whenever there is a human-in-the-loop with engineering and operations responsibilities. The possibility of sharing a common architecture model in terms of concepts, artefacts and data, create a common engineering and operational culture and ecosystem at organisation level which allows major effort savings for knowledge management and its sharing. This is clearly extremely relevant when an organisation like EUMETSAT experiences a relevant grow in terms of missions and complexity.

Any development of new approach and concept come with an initial effort caused by the change of paradigm definition, new engineering processes and new technologies, which typically imply and because of new technologies which typically imply steep learning curves.

Some of these aspects are not necessarily negative but indeed require effort to be fully assessed, understood and applied in the most effective way to maximize their effectiveness within the overall adopted roadmap strategy. Some have to be addressed as specific hot topics and even as show-stoppers which, once solved, become intrinsic part of the new paradigm culture and solution. Others are simply relevant lessons learned.

The implementation of the described model and elements have not been exempt from this scenario. Some relevant considerations at this stage of development can be summarised as:

- **Development Phases**
  - **Analysis**. The requirements and services definition phase is typically the most relevant one. The allocation of requirements to service units with their interface definition require a higher level of attention than in the case of nominal client/server requirements analysis. Also, interface invocation

Copyright 2023 by EUMETSAT. Published by the Mohammed Bin Rashid Space Centre (MBRSC) on behalf of SpaceOps, with permission and released to the MBRSC to publish in all forms."

mechanisms and orchestration towards a system use-case have to be "assessed" upfront with dedicated analysis sessions;
    - **Architecture Design**. Once requirements allocation to service units and related interface specifications are consolidated, the design is essentially at system level with the assessment of selected technologies and allocation of responsibilities;
    - **Detailed Design**. The high level of possibilities of the selected technologies ecosystems and the fact that the team was evolving in their understanding were the reasons why new solutions or different ones, not initially envisaged, emerged during this phase and applied. As mentioned in the paper, examples are the Generic HMI tailoring capability and testing framework. The adoption of an agile-like approach, a well thought detailed design and technologies, were all instrumental in enabling this to happen;
    - **Testing**. Automatic testing is highly required in order to cope with the different services functionalities, their interaction and system behavior. Defining extensive test plans, reproducing anomalies and verify any regression cannot be performed manually especially for critical operational systems.
- **Maintenance**. It is envisaged that once the initial development is completed, the adopted layered approach in combination with microservices, will allow to isolate areas of intervention for the back-end (even re-engineering) without impacting other services. This is quite a relevant aspect compared to monolithic or even client/server architectures. The same is related to inject new service units covering specific needs, again without impacting exiting units. In practice the service unit becomes the formal unit of maintenance.
- **Cloud Technologies**. When approached for the first time, a prototype is essential to assess the high scale of possibilities and features these technologies offer but also the level of configuration and tooling they require. The assessment is not only technical but also to qualify and quantify the level of effort to allocate to areas of expertise the project team must have;
- **Engineering/Tooling**. Software engineering requires support from tools each having its responsibility during analysis, development, continuous integration and deployment (DevSecOps), product assurance. In addition, cloud technologies and cluster environment require support for services management, behavior observability and monitoring. Learning the whole set of tools, being aware of their evolution and availability of new ones, require learning curves and knowledge maintenance. This is considered inevitable in order to take full advantage of the cloud and cluster benefits on one side but also to inspect the runtime behavior of the system toward improvements and corrections. PaaS provider provides great support in the management of such tools;
- **Multidisciplinary Teams**. Flight dynamics engineers involved in this development are typically space mechanics experts. They have faced the need to adapt to new paradigms and technologies with the advantage of acquiring a multidisciplinary expertise highly valuable within the organization.

Acknowledgements

The authors of this paper would like to acknowledge the contributions from EUMETSAT management, engineering and operations teams as well as industry high level expertise, all involved at different level of responsibilities in the definition and development of the presented architecture model and elements.

References

[1] ECSS-E-ST-40C – Space Engineering Software, European Cooperation for Space Standardization (ECSS), 2009-03-06, https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/, (accessed 2023-01-27).
[2] Ruediger Gad, Francesco Croce, Jessica Cowley, Ales Simonic, Meriem Drira, Carlos Miranda "Integrated Framework for Software Testing and Verification based on Open Source Software", 17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023
[3] OpenAPI Specification (OAS): https://spec.openapis.org/
[4] Docker: https://www.docker.com
[5] Kubernetes: https://kubernetes.io
[6] Apache Kafka: https://kafka.apache.org/
[7] Elasticseach: https://www.elastic.co/elasticsearch
[8] Kibana: https://www.elastic.co/kibana
[9] Fluentd: https://www.fluentd.org
[10] MariaDB: https://mariadb.org
[11] Electron: https://www.electronjs.org
[12] Angular: https://angular.io/
[13] Microsoft, Visual Studio Code, https://code.visualstudio.com/, (accessed 2023-01-27).