BECKETT – A Flight Dynamics System based on GODOT

Javier Berzosa Molina[(1)], Luning Bao Cheng[(1)], Pedro Julio Montealegre Ávila[(1)], José Cava Pérez[(1)], Roberto Sánchez Ramos[(1)] , Jaime Fernández Sánchez[(1)]

[(1)]GMV Aerospace and Defence
Tres Cantos, Spain
Email: jberzosa@gmv.com, luning.bao.c@gmv.com, pjmontealegre@gmv.com, jcava@gmv.com, rosanchez@gmv.com, jfernandez@gmv.com

Abstract

For the last 20 years, GMV has been a leading provider of Flight Dynamics Systems (FDS), developing capabilities which range from the Low Earth Orbit (LEO) to Geosynchronous Equatorial Orbit (GEO) regimes, from circular to Highly Elliptical Orbit (HEO) orbits, from single satellites to constellations, from institutional to commercial customers. Those capabilities build on top of a core of Fortran 90/95 source code which sets an upper barrier to the evolution of the software (SW) in terms of architecture, interfaces, integration with other systems, third-party dependencies, among others.

The development of GODOT (General Orbit Determination and Optimisation Toolkit) by ESOC under a permissive license grants the community the capability to exploit its potential for relevant space domains of interest. GODOT provides key Flight Dynamics and Mission Analysis capabilities both for Earth-bound and interplanetary applications under a modern combined C++ and Python framework.

BECKETT is a new FDS SW prototype currently under development which is based on state-of-the-art technologies and programming languages. At its core, the SW is based on and builds on top of the GODOT technology to implement the key functionalities. Overall, BECKETT aims at laying out the basis for a modernized FDS at GMV and explore new possibilities in terms of technologies.

The architecture of BECKETT is based on a set of independent services which (1) interact among each other as well as the user via RESTful HTTP API interfaces based on JSON, (2) make use of relational databases for data and metrics storage, (3) can be scaled up/down to adjust the computational resources to the actual workload, (4) can be configured to run functionalities on demand and/or based on events, and (5) can be deployed on a distributed architecture either on-premises or on a cloud provider.

The functionalities of BECKETT aim at covering most of the typical Flight Dynamics (FD) capabilities, including orbit propagation and determination based on satellite observations, including navigation, range, angular and Doppler measurements, event calculations involving visibilities between celestial bodies, ground stations and orbiting satellites with onboard instrumentation, and manoeuvre planning capabilities for the most common scenarios, including Launch and Early Orbit Phase (LEOP), station keeping and relocation.

This contribution presents (1) a more detailed description of the BECKETT SW, with a focus on the SW design and use cases, (2) further details on the FDS capabilities, (3) an overview of the validation activities for the implemented capabilities, and (4) the status of the implementation.

I. Introduction

As part of the standard life cycle of Flight Dynamics (FD) software (SW), evolutions to the computational algorithmic core, the interfaces, third-party dependencies, and other key elements, which may be part of the low-level architecture, are regularly undertaken to meet the stringent requirements of an evolving space community. The rapid advancement of IT technologies, exemplified by the emergence of cloud computing, as-a-service solutions, modern programming paradigms, and the increasing democratization of Artificial Intelligence through languages like Python, poses a significant challenge for the adaptation of certain SW low-level components, potentially resulting in a degradation of the overall solution.

In this context, **BECKETT** emerges as a new Flight Dynamics System (FDS) developed by GMV, which, by design principle, does not directly inherit from previous GMV solutions. Instead, BECKETT aims to leverage state-of-the-art tools to create a novel solution, with the use of GODOT [1] at the computational core of the SW being a key characteristic, along with a distributed architecture. The implementation of the BECKETT FDS began in mid-2023 as part of an ESA General Support Technology Programme (GSTP) activity, scheduled for completion in summer 2024. By the conclusion of this initiative, an initial version of the SW, featuring a set of essential Flight Dynamics functionalities, will be available.

II. Key High-Level Requirements

The design of BECKETT is guided by a set of high-level requirements, with particular emphasis on the following:

- Utilization of GODOT at the computational core: the FD SW provided by ESA/ESOC encompasses fundamental algorithms for tasks such as time and reference frame handling, orbital propagation,

observations reconstruction, estimation, and event computation, among others. Hence, GODOT serves as an optimal foundation for the computational core of BECKETT. The extensibility of GODOT as a library allows for the incorporation of custom models and refinement of the original implementation to address any missing functionalities required by the system.

- Horizontally scalable distributed architecture: the functionalities of the FDS are divided into a set of loosely coupled, stateless components, each with well-defined objectives. While these components exhibit a certain degree of interdependency, they are independently deployed within an infrastructure composed of one or more host servers. Depending on the workload of the FDS, each component can be scaled up through replication. This means that a request can be handled by any replica of a particular component, thereby reducing the overall workload of the system.
- Centralized data management: a common centralized data model is essential to ensure a standardized approach to data interaction across the system. Given that an FDS manages a large volume of data with diverse physical characteristics, specifying the modelling of each data element facilitates the establishment of coherent interfaces and enables all components to communicate effectively using a common data language.
- REST API interfaces: BECKETT employs a RESTful HTTP Application Programming Interface (API) for synchronous communication between components. This choice simplifies interface implementation, leveraging the availability of frameworks for developing RESTful interfaces within the open-source community.

III. Use Cases

BECKETT was conceptualized to address three fundamental use cases.

The first involves utilizing the FDS within an interactive Python scripting environment. This functionality is facilitated through the provision of a Python client library that connects to the API exposed by the FDS. This allows users to interact with the data and functionalities of the system using Jupyter Notebook or similar technologies, all whilst working with intuitive Python classes and methods. Additionally, clients for languages other than Python can be generated, provided a standard API is maintained on the FDS side.

The second use case pertains to accessing the FDS via a web interface designed to be both accessible and intuitive. Through this interface, users can consult data interactively, inject new data, configure the system, and utilize any of the offered functionalities by the SW.

A third use case involves running the FDS as an autonomous service. To achieve this, BECKETT can be configured to react to specific events and execute predefined configured actions. Examples of such actions include performing Orbit Determination (OD) upon receiving new tracking data, evaluating spacecraft slot fulfillment, or planning maneuvers upon availability of a new orbit.

IV. Design and Technologies

The design of BECKETT, along with its architecture and the selected technologies for its implementation, is primarily guided by the key requirements outlined earlier.

At a high level, the proposed architecture for the BECKETT system is based on a distributed microservices architecture [2]. Consequently, the operation of the FDS is the collaborative result of one or many of these microservices. Breaking down a complete FDS into microservices allows for the identification of distinct components.

While some components are dedicated to providing the necessary business logic and data management for the system, others focus on facilitating user interaction with the system.

From the user's perspective, the distributed microservices architecture resembles the traditional client-server model, where the system can be divided into two logical levels (see Fig. 1):

- Frontend: responsible for exposing BECKETT functionalities to users through various means, including a web interface, a Grafana visualization system, and a Python client library. These tools abstract the underlying complexity of the system and interact with users through user-friendly interfaces.
- Backend: responsible for implementing the actual FD logic, managing data, and handling corresponding interfaces. To users, the backend appears as a black box that can be interacted with via the frontend tools.

At its backend, BECKETT introduces a flexible and customizable infrastructure of scalable components, forming a framework for constructing and tailoring FD products to meet specific user requirements (see Fig. 2).

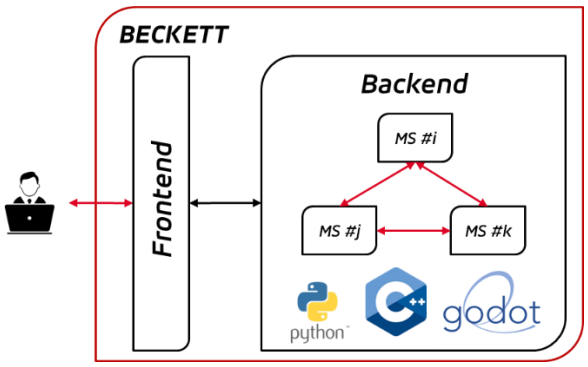

Fig. 1. High-level architecture of BECKETT

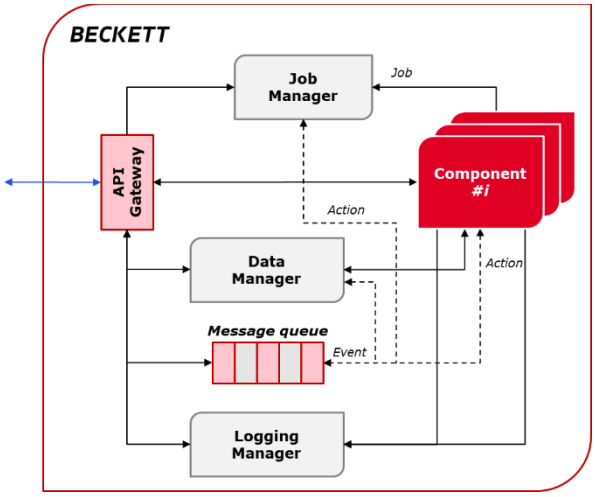

Fig. 2. Backend design of BECKETT

The key elements within this framework include:

- API Gateway: serving as a reverse proxy, the API Gateway redirects user requests to the appropriate component within the FDS, functioning as the only entry point for user requests. This component is based on Kong technology.
- Data Manager: centralizing all essential FDS information shared among multiple components, the Data Manager facilitates the addition, retrieval, and deletion of data ranging from satellite geometry information to telemetry data.
- Job Manager: centralizing the management of executions triggered within the FDS, the Job Manager enables users to monitor the status, results, availability, and traceability of jobs.
- Logging Manager: centralizes the availability of logging data.
- Message Queue: utilizing RabbitMQ technology, the Message Queue serves as a broker system, enabling the implementation of asynchronous communication channels between components. This forms the foundational principle for the autonomous operation of the system, based on event-reaction mechanisms.

The specific components responsible for implementing the core FD functionality adhere to a uniform design, as depicted in Fig. 3. Each component incorporates the following key elements:

- API: defines the endpoints enabled by the component for interaction with users and other components.
- Data: each component manages information specific to its functionality. For example, this may include execution metrics.
- Configuration: each component handles the configuration of the functional endpoints it exposes, treating it as a specific type of data.
- Core: this comprises a set of core FD algorithms that process system requests. Whenever feasible, these algorithms are based on GODOT technology, which includes the GEneral Navigation for Earth Orbiting Satellites (GENEOS) SW.

A selection of the key elements characterizing the design of the BECKETT FDS is described below.

Baseline Technology

The components are developed using Python's Django technology. Python, particularly Django, offers the foundational tools necessary for (1) interacting with databases, (2) implementing RESTful HTTP API interfaces, and (3) deploying each component as an independent server.
Python's capability to interact with GODOT and GENEOS, either in Python or C++ via bindings, makes it an ideal choice for component implementation. However, other technologies may also be used, provided that they adhere to the required interfaces.

API Standards

The RESTful HTTP API interfaces of BECKETT components adhere to the OpenAPI Specification (OAS) version 3 [3]. This ensures compatibility with Swagger tools, facilitating API documentation and auto-generation of client code for various programming languages.

Data Storage

BECKETT primarily stores data in databases to facilitate interaction via filters and query parameters through dedicated RESTful APIs. The default choice is PostgreSQL, a production-ready relational database compatible with Django.

Containerization

Components are isolated, built, and deployed using Docker technology. BECKETT is packaged into a set of container images deployable using Docker Engine, with support for compatible orchestration technologies such as Docker Compose, Docker Swarm, or Kubernetes. These technologies enable horizontal scaling of components and distribution across multiple hosts.
While initially designed for Linux systems, BECKETT shall be compatible with any Docker-compatible operating system (OS).

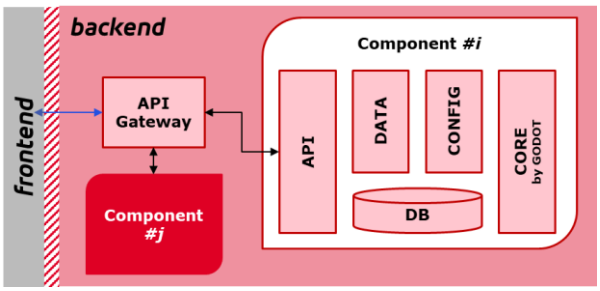

Fig. 3. Generic design of BECKETT components

V. DEVELOPMENT METHODOLOGY

The development of BECKETT is approached as a long-term endeavour. The BECKETT FDS represents an incremental product with evolving functionalities prioritized based on requirements and client needs. Embracing this philosophy, the development phase is managed using an agile methodology, where capabilities are added incrementally.

In contrast to traditional waterfall projects, the agile methodology emphasizes the availability of a functional end-to-end product throughout the development phase. Generally, the selection of new functionalities to be implemented follows the priorities listed below, in decreasing order of criticality:

1. Architecture: elements related to the overall architecture setup, deployment mechanisms, data management, and automation and monitoring of the system.
2. Basic functionality: implementations required to support fundamental FD operations, many of which are natively supported by GODOT and GENEOS. Emphasis is placed on providing a set of generic functionalities across a wide range of operations rather than implementing high-precision models and specific platform-dependent algorithms.
3. User interaction: elements aimed at improving the user experience. A minimum set of tools to interact with the system is provided early in the development phase. However, aspects such as the availability of a complete web user interface are considered less critical.

Most of the implemented code is written in either Python or C++. The computational core is predominantly handled by the C++ implementation of GODOT and GENEOS, offering BECKETT users performance equivalent to the former. When necessary, GODOT is extended to accommodate missing rotations representing satellite attitude modes or the definition of specific types of visibility events, for example.

Both Python and C++ have active communities contributing to permissive open-source Commercial Off-The-Shelf (COTS) products. The development methodology of BECKETT leverages the use of COTS where feasible. A list of the most relevant COTS used by BECKETT is provided in Table 1.

VI. CAPABILITIES

At the conclusion of the General Support Technology Programme (GSTP) activity that initiated the development of BECKETT, the FDS will possess a functional foundation capable of performing basic operations encompassing the following:

- Multi-satellite handling: ability to manage one or multiple satellites within a single system.
- Data provision: ingestion and export of data in standard formats. Particularly, operations with OEM orbit and TDM tracking files are available.

Table 1. List of key COTS used in BECKETT

Name	Description
Celery	Task queue for asynchronous jobs in Python.
Django	High-level Python web framework.
Docker	Container runtime technology.
GENEOS	GEneral Navigation for Earth Orbiting Satellites ESA/ESOC software.
GODOT	ESA/ESOC flight dynamics software.
Kong	API Gateway technology.
Numpy	Fundamental package for scientific computing with Python.
PostgreSQL	Relational database system.
Python	Python programming language.
RabbitMQ	Messaging and streaming broker technology.
Swaggerapi	OpenAPI documentation tool.

- Orbit Propagation: numerical propagation of orbit and covariance data from a-priori orbit data or specific state vectors using state-of-the-art dynamic models.
- Orbit Determination: estimation of initial state vector and dynamic model parameters using ranging, Doppler, angular, and XYZ observations.
- Manoeuvre Calibration: estimation of manoeuvre calibration factors and management of manoeuvre history information.
- Event Computation: calculation of geometrical events based on the available orbit and attitude information, ground stations, celestial bodies, and onboard sensors.
- Manoeuvre Planning: computation of the set of manoeuvres required to perform orbit transfers supporting slot acquisition, station keeping, in-plane relocation and disposal scenarios for Geosynchronous Equatorial Orbit (GEO) (in-plane and out-of-plane) and Low Earth Orbit (LEO) (based on ground track deviations) satellites. The algorithms are based on semi-analytical models which make use of GODOT.
- Mass Computation: calculation of satellite mass after fuel consumption during a manoeuvre, based on the relevant thrust models.
- Attitude Monitoring: evaluation of deviations in real attitude data compared to theoretical attitude mode.

- Data Accessibility and Visualization: the data managed by BECKETT is accessible via Grafana as well as the designated APIs, along with Python-based client libraries to support the interaction from user scripts.
- Asynchronous Executions: the functionalities provided by BECKETT can be triggered by users as synchronous or asynchronous tasks. Asynchronous tasks allow users to initiate executions without waiting for completion and monitor job status until completion or implement programmatic actions upon the job completion notification arrives.
- Autonomous Operations: implementation of action-reaction mechanisms in the system enables the autonomous operation of the different functionalities based on user configuration and allows any BECKETT user to take advantage of the notifications mechanism.

As part of the future roadmap of BECKETT, the capabilities listed above will be further enhanced by:

1. Refined Models: development of refined models not currently present in GODOT.
2. Additional Functionalities: expansion of functionalities within existing components, such as extending estimable parameters or implementing further manoeuvre planning capabilities accompanied by numerical optimization.
3. Platform-Specific Capabilities: Implementation of new functionalities to provide platform-specific capabilities to the software.

Furthermore, the use of GODOT opens up possibilities for extending BECKETT capabilities beyond the Earth's regime, considering the interplanetary capabilities offered by the library developed by ESA/ESOC.

VII. VALIDATION

Validation of the software is conducted through three different methods, as described below.

Component Unit Test

Each bespoke component within BECKETT is accompanied by a set of unit tests aiming for a target code coverage of 80-90%. These tests are automatically executed at least daily using Gitlab Continuous Integration and Continuous Delivery (CI/CD).

Use Cases

For each of functionality intended to be fulfilled by the FDS, use cases are defined. These use cases are represented by a statement describing what the user of the system should be able to achieve.
From a validation standpoint, each use case defines one or several integration tests that shall be successfully executed to consider the use case as satisfied by the SW. These tests are characterized by a set of inputs and expected reference outputs, which are typically generated using reference SW available inhouse.
Use case integration tests are implemented with the support of the Robot Framework technology, a framework for test automation. These tests are programmatically executed daily using Gitlab CI/CD and can be executed at any time for a particular version of the SW.

End-To-End

The final exercised validation mechanism involves the real-time execution of a complete end-to-end scenario mimicking a real operational environment for both LEO and GEO satellites. From a high-level perspective, the end-to-end scenario is composed of (see Fig. 4):

1. FD Segment: on the FD side, the latest version of BECKETT is deployed in a controlled secure environment which can be accessed to from the simulation segment. The resources allocated and component replication settings are adjusted based on the actual workload and number of satellites to be tested.
2. Simulation Segment: on the simulation side, a set of tools have been developed to support this validation activity. These tools are built based on ***MAORI*** [4] [5], GMV's new FD and geodesy library, whose implementation is independent of BECKETT and GODOT-related technologies. Among other responsibilities, these tools simulate orbits, as well as radiometric, Doppler, angular, and XYZ tracking data for each satellite. The simulation is self-consistent and uses previously generated data to resume the simulation at a certain frequency. The generated tracking data is published in real-time or with a simulation offset to the FDS.

During the end-to-end validation, BECKETT is provided with the necessary basic data regarding the involved satellites and stations and is configured to conduct a series of automatic operations, including orbit determination, manoeuvre planning, manoeuvre calibration, and mass estimation.

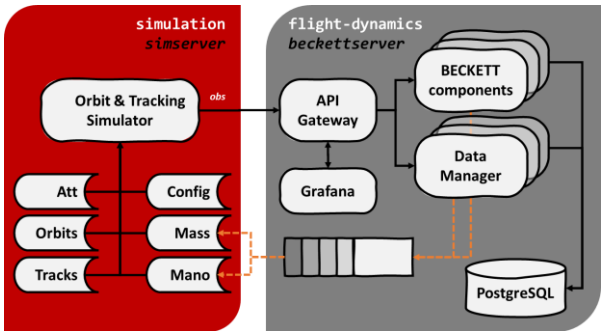

Fig. 4. End-to-end BECKETT validation infrastructure

Upon update of the satellites' mass and/or planned manoeuvre information by the FDS, the simulation

segment receives alerts of the change via BECKETT's events and takes action to update the simulated information accordingly.
The results and metrics generated by the FDS are visualized with Grafana and regularly monitored. Comparisons between the actual simulated precise orbits and the FDS-derived orbit information provides insight into the level of precision achieved by BECKETT.

VIII. CONCLUSIONS AND WAY-FORWARD

GMV has developed a new Flight Dynamics System utilizing state-of-the-art technologies, employing a distributed microservice scalable architecture, and integrating GODOT at its computational core.
The initial activity supporting the development of BECKETT is set to conclude by summer 2024. At this juncture, a foundational set of validated FD functionalities will be available to support LEO/GEO satellite operations encompassing generic orbit propagation and determination, event calculation, manoeuvre planning and calibration, as well as mass update calculations.
GMV's dedication to BECKETT extends into the long term, with the roadmap focusing on integrating refined models, adding further functionalities within existing components, introducing new features tailored to specific platform requirements, as well as enhancing the user experience by, e.g., the provision of a web user interface.

IX. REFERENCES

[1] European Space Agency, "Godot documentation," 2021. https://godot.io.esa.int, accessed: 11.04.2024 (2024).

[2] C. Richardson, "What are microservices?," 2024. https://microservices.io, accessed: 11.04.2024 (2024).

[3] OpenAPI Initiative, "OpenAPI Specification," 2021. https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.1.0.md, accessed: 11.04.2024 (2024).

[4] J. Fernández Sánchez, C. Fernández Martín, J. Berzosa Molina, "MAORI – A New Flight Dynamics and Geodesy Library", *International Symposium on Space Flight Dynamics 2024*, Darmstadt, Germany, 22-26 April 2024.

[5] C. Fernández Martín, J. Berzosa Molina, L. Bao Cheng, M. Á. Muñoz de la Torre, M. Fernández Usón, S. Lara Espinosa, E. Terradillos Estévez, J. Fernández Sánchez, H. Peter, P. Féménias, and C. Nogueira Loddo, "FocusPOD, the new POD SW used at CPOD Service", *EGU General Assembly 2023*, Vienna, Austria, 24–28 April 2023, EGU23-1908.