Improvements to SST Core Software: Advanced Software Engineering Practices, Integration of Track-to-Track Algorithm and other Algorithmic Enhancements

Diego Ramírez Rodriguez [(1)], Jack McHugh[(2)], Ben Johnson[(1)], Francisco Javier Simarro Mecinas[(3)], Alfredo Antón[(3)], Bogdan Bija[(4)], Ian Holbrough[(5)], James Beck[(5)], Nishitha Mathew[(6)], Robin Jennings[(6)], Riaz Shafi[(6)], Louise Rees[(6)], Andrea Scalabrin[(7)]

[(1)] GMV, Airspeed 2, Eighth Street, Harwell Science and Innovation Campus, Didcot, Oxfordshire OX11 0RL, UK., Emails: dramirez@gmv.com, ben.johnson@gmv.com
[(2)] GMV , Enterprise Centre, Innovation Park, Triumph Road, Nottingham, NG7 2TU, UK, Email: jmchugh@gmv.com
[(3)] GMV, Calle Isaac Newton 11, Tres Cantos, 28670, Spain. Emails: fjsimarro@gmv.com, amanton@gmv.com
[(4)] Former Affiliation, GMV, Airspeed 2, Eighth Street, Harwell Science and Innovation Campus, Didcot, Oxfordshire OX11 ORL, UK
[(5)] Belstead, 387 Sandyhurst Lane, Ashford, Kent, TN25 4PF, UK, Emails: ian.holbrough@belstead.com, james.beck@belstead.com
[(6)] Starion, Newark Works 2 Foundry Way, South Quays, Bath, Somerset, BA2 3DZ, UK, Emails: n.mathew@stariongroup.co.uk, r.jennings@stariongroup.co.uk, r.shafi@stariongroup.co.uk, l.rees@stariongroup.co.uk
[(7)] Former Affiliation, Starion, Newark Works 2 Foundry Way, South Quays, Bath, Somerset, BA2 3DZ, UK

ABSTRACT

This paper presents the enhancements made to the SST Core Software (CSW) with the aim of enhancing its capabilities by integrating innovative algorithms and modern software engineering practices. Under P3-SST-XXVI, this included reorganising the software into different GitLab repositories, refurbishing GitLab Pipelines, integrating a Nexus repository for artifact publication, parallelising DPC processes, restructuring the Maven Project Object Model (POM), implementing an automated test framework, and upgrading re-entry analysis capabilities. Under S2P-S1-SC-08 the Track-to-Track (T2T) algorithm was implemented to address the limitations of the Track-to-Orbit (T2O) method enabling the association of uncorrelated tracks, which enhances the robustness of orbit determination and reduces error margins in space object tracking.

1 Introduction

The SST Core Software (CSW) is ESA's complete software for SST data processing, including the configuration of a sensor network

It is separated in the following components/subsystems:

- Generic Services: maintains general services and tools used by the CSW.
- Data Processing Chain (DPC): processes input sensor observations and third party orbit data to perform a series of operations for new or existing objects: Initial Orbit Determination (IOD), Routine Orbit Determination (ROD), correlation, orbit quality analysis
- Planning System (PS): manages sensor network and schedules tracking and survey requests, requested by the user or the CSW itself.
- SST Services: Series of end-user products to conduct analyses in different areas of SST from the data processed by other subsystems of the CSW:
    - RPS: Re-entry Planning Subsystem
    - CPS: Collision Prediction Subsytem
    - FAS: Fragmentation Analysis Subsystem
    - CQS: Catalogue Query Subsystem
- SVT: Software Virtualisation Toolbox
- SSTDB: CSW database
- Catalogue tools

These components are deployed within different docker containers. The CSW's GitLab CI/CD pipeline is responsible for building, testing, generating documentation, creating the CSW docker images, and deployment.

The following sections and subsections shall detail the improvements made to the CSW, both structurally and algorithmically, achieved during the P3-SST-XXVI and S2P-S1-SC-08 activities.

2 SST CORE SOFTWARE STRUCTURE IMPROVEMENTS

2.1 From Monolithic to Modular Approach

The initial organisation of the CSW GitLab repository included everything needed to build and test (as well as other jobs/procedures) the components of the CSW to

finally deploy it to a specified environment, handled by its CI/CD pipeline. Since the code base of the CSW is large, this monolithic approach proved unfavourable for multiple reasons:

- Git branch management complications; poses risk of overlapping of changes requiring additional processes to ensure no data loss
- Management of contributors is limited since a developer would have access to all aspects of the CSW.
- The entire CSW repository was needed to be downloaded to perform any change, even if there was no impact between the modified component and the rest of the software
- It was required to build and test all CSW components before confirming and validating new changes. Any failure would further increase this time.

A significant improvement was therefore to transition from a monolithic approach to a modular one. It was identified that the connections between CSW components is handled through a specific component; the **SST Common Data model** (explained in section 2.2.2.2) meaning there is no direct relationship or dependency between the components themselves. As a result, to transition the CSW to be modular, the SST Common Data model was extracted to be inside its own GitLab repository that can be **compiled and distributed as a dependency** to the other components. The Nexus software repository manager is now used to facilitate this, to act as the artifact/dependency provider for the CSW.

With this, it enabled the CSW to be modularised into multiple GitLab repositories, all stored within the same GitLab group. The following subsections detail the CSW Gitlab Group contents, including their sub-groups specific purpose, the repositories and the necessary adaptation of their CI/CD pipelines.

2.1.1 CSW Multi-pipeline

The multi-pipeline is responsible for performing a full build of the SST Core Software via the CI/CD pipelines of all GitLab repositories inside the Code Base Management (section 2.1.2) and CSW Modules (section 2.1.3), in a specific order. This includes the building, testing, quality analysis, artifact publishing, and docker images generation, ready for deployment and automated testing.

This repository keeps control on the building of the CSW, eliminating risk of errors if a user/contributor executes individual repositories out of order.

Once the sst-cs-build CI/CD pipeline is executed successfully on a specific environment, then all docker images shall be stored on the defined docker registry, and all artifacts shall be stored on the Nexus repository, giving contributors the freedom to then directly execute pipelines on specific repositories with their changes.

2.1.2 Code Base Management

As part of the P3-SST-XXVI activity, these repositories were included to gain more control and management of future CSW implementations, by minimising the amount of duplicated data, spread across various subfolders. They are now stored in three different repositories, each with their own unique purpose:

- **sst-cs-parent**: maintains the Centralised (parent) POM used throughout the CSW repositories
- **sst-cs-libs:** maintains the CSW libraries such as the SST Common Data model
- **external_standards_lib**: maintains external libraries used by the CSW.

Further details on these new concepts and repositories can be found in sections 2.2.1 and 2.2.2.

Their CI/CD pipelines all follow the same structure where it builds the target folder (through maven) conducts any necessary test of the jar files, then publishes the artifacts to the Nexus repository so they are all available for use by future repositories.

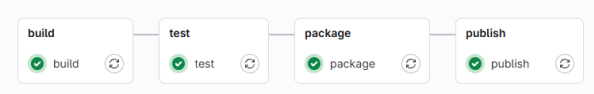


Figure 1. CI/CD pipeline template used by Code Base Management repositories

2.1.3 CSW Modules

The CSW modules contain the bulk of the software components necessary to build all CSW docker images. The following lists the repositories and what they are responsible for.

- common_modules: code base for the generic services, including space weather and expert centre components
- dpc_modules: code base for the DPC
- ps_modules: code base for the PS
- sst-web-portal_modules: code base for the RPS, CPS, FAS and CQS subsystems, as well as the CWBI webservice and SST HMI.
- svt_modules: code base for the SVT
- database_modules: code base for the SSTDB
- catalogue_modules: code base for the catalogue tools used with the CSW when live.

The CSW modules CI/CD pipelines all follow the same structure and have the same stages:

- download_cots: downloads the necessary artifacts from Nexus
- build: various stages to build the source code (tools, backend, HMI)
- test: perform unit tests on the compiled code

- test_report: extract results of the tests
- quality analysis: check the quality of the code, by submitting the code onto SonarQube
- publish: further publish artifacts to Nexus for other repositories to have visibility on (required for their own pipeline execution)
- build_runtime: jobs to build the docker images.

Once the build_runtime job(s) have been executed on a pipeline, the docker image(s) will be added to a docker registry.

2.1.4 CSW Deployment

The CSW deployment is responsible for deploying the CSW following instructions from a docker-compose.yml file. This is handled through a single repository, sst-cs-deploy. Its CI/CD pipeline is composed of one stage, with different deployment jobs, for which it retrieves the docker images from the configured docker registry.

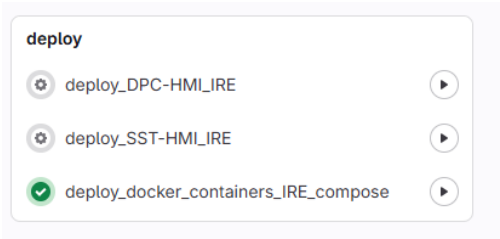


Figure 2. CI/CD pipeline for the deployment repositor.

2.1.5 Test Automation

As part of the P3-SST-XXVI activity, the CSW component and system regression tests were automated. This was achieved by creating an additional repository, the test_common_modules, which contains an automated test tool framework (tools, procedures, etc.) to significantly reduce the time needed to verify updates to the CSW. It can be executed programmatically once every night, or after a new CSW deployment to assess its status as fast as possible.

The test automation is accomplished through the GitLab CI/CD pipeline, for which it first builds and deploys a test docker container to the same environment where the CSW is deployed, and then executes the CSW tests on that environment from within the test container. The CI/CD pipeline is able to generate reports per test, as GitLab artifacts, for users to review once completed to assess the state of the current deployment of the CSW.

Further details on the automated test tool can be found in section 2.3.

2.2 The Need of a Centralised Multi-modular Project

In the context of multi-modular software projects, the usage of similar configurations, libraries and overall structure is a common approach, more even if those modules have been developed within the frame of the same project. However, if not carefully assessed, any of those submodules can diverge from the rest very rapidly, making it to the point that the configurations and libraries are not compatible anymore.

To avoid this problematic situation, the SST Core Software, an Apache-Maven structured software, has been modified to avoid any divergences in terms of libraries' usage and versions, configuration definitions and plugin usage.

2.2.1 Centralised POM

The main purpose of a parent POM in a multi-modular Maven-based project is typically to serve as a central management point that child modules inherit, promoting consistency, simplicity and maintainability in large projects.

This approach has several advantages compared to a modular-distributed approach:

- **Centralised dependency management:** all child modules can inherit the same dependency versions, preventing version conflicts and ensuring consistency across the project. This reduces the risk of different modules using incompatible versions of the same library.
- **Simplified POM files:** any module that inherits from a parent POM will avoid repeating any configuration already defined in the parent. This reduces redundancy and increases readability.
- **Consistency across modules:** configuration elements such as plugin versions, repository definitions, and property settings are centralized in the parent POM, leading to a more consistent and predictable build environment.
- **Easier version management:** dependency and plugin versions can be easily upgraded in all modules by simply changing them in the parent POM.

In this context, an additional module was added to the SST Core Software project called **SST-CS-PARENT**. It oversees defining the parent POM of the whole project, which includes:

- The definition of all the dependencies, and their versions, needed in the whole project.
- The definition of all the repositories from where the project shall download any dependencies.
- The project properties, which includes endpoint definitions, debug configurations and logging configurations, among others.
- The definition of all the plugins needed in the whole project, including not only their versions, but also the configurations needed for its use throughout the entire Maven cycle execution.

Finally, all the other modules in the SST Core Software

shall include the parent definition and point to this specific new module as shown in Figure 3:

```
<!-- Parent definition -->
<parent>
    <groupId>esa.ssa.sst</groupId>
    <artifactId>sst-cs-parent</artifactId>
    <version>2.1.0</version>
    <relativePath />
</parent>
```

Figure 3. Parent POM definition in children's modules

This way, any module that has this defined in their POM file will inherit any dependencies and plugin versions, plugin configurations and project configurations that may be defined in the parent POM.

2.2.2 Centralised Libraries

Another enhancement included in the SST Core Software related to centralisation are the internal and external libraries usage and definition.

Due to the development process of the project during previous activities, each tool was developed independently, with their own approaches, style and library usage. Although this approach increased the flexibility in terms of implementation, testing and deployment, it provoked a high amount of duplication and redundancy in the usage and definition of the software dependencies and external libraries.

In this section, two of the main sources of definition and usage duplication within the SST Core Software are presented, including the previous status and the approach followed to solve this situation.

2.2.2.1 CCSDS Standard Usage

The CCSDS (Consultative Committee for Space Data Systems) standard is a set of guidelines and specifications created to ensure interoperability and standardization in the communication and data systems used in space missions. To simplify this standard usage, the CCSDS provides a set of XSD files that allow Java-based software to automatically generate the Java classes that define and implement those standards.

In the context of the SST Core software, those XSD standards were scattered all over the modules, even using different versions of the standards.

To solve this problem, a new module was added to store all these standards. The name used is **EXTERNAL-STANDARDS-LIB,** and its sole purpose is to store and maintain external libraries used by the SST Core Software. It was generated not only to host the CCSDS standards, but any other external libraries that may be needed in the future by any subsystem in the SST Core Software.

This approach serves as a way of centralising the definition of these libraries and ensures consistency in terms of versioning, package definition and auto-generation configuration. This is controlled by the POM file defined in the module and inherits from the general parent POM.

2.2.2.2 SOAP API and Common SST Data Model

The communication between the SST Core Software subsystems is ruled by SOAP-based APIs.

A SOAP (Simple Object Access Protocol) API is a protocol for exchanging structured information in the implementation of web services. It is a message-based communication protocol that allows different systems, often over HTTP or other protocols, to exchange data in a structured, XML format. This structured XML format is based on WSDL (Web Services Description Language), which is used to describe the functionality of a web service, specifically how the SOAP service can be accessed and what operations it can perform. Finally, XSD files attached to the WSDL files ensure that the XML data follows a certain structure and defines the elements and attributes within that structure.

In the context of the SST Core Software, this SOAP API is utilised to share information between subsystems, including both front-end and back-end software subsystems. Due to the fact these WSDL and XSD files are fixed, they can be used to automatically generate Java classes that define those web services and data structures. These are generated during the build stage within the deployment of the SST Core Software, ensuring a smooth and stable software generation.

The Common SST Data Model is a set of WSDL and XSD files used by the SST Core Software that defines two main things:

- How this communication between subsystems shall be performed.
- The format and the shape of the data to be sent through the communication channel.

This includes information such as:

- space object main definition
- track structure
- conjunction message structure
- fragmentation message structure
- re-entry message structure

among others. As these WSDL and XSD files are also fixed, they can be used to automatically generate equivalent Java classes in the build stage within the deployment.

In the past, during the development of the SST Core Software, as stated in section 2.2.2, different subsystems of the entire project were developed by separated teams with no communication with each other. This provoked a duplication of both the Web Services definitions and the Common SST Data Model instances. To revert this and to avoid future divergences and incompatibilities that

could rise from this duplication, a separate module called **SST-CS-LIBS** was created. The purpose of this new module is to centralise both the Web Services WSDL files of all the subsystems in the SST Core Software and the XSD files that conform the Common SST Data Model. This does not only include the definition of those files, but it also centralises the configuration that describes how the Java classes based on those WSDL and XSD files shall be generated.

The services comprised in this folder, that is, the WSDL/XSD files that define them, are the following ones:

- authentication-service
- common-resources (Common SST Data Model)
- cps-service
- cqs-service
- dpc-service
- email-service
- fas-service
- log-service
- mmso-service
- notification-service
- planning-service
- ps-service
- rps-service
- sensor-service
- sstwp-service
- svt-service

2.3 Test Automation Framework

The cornerstone of the test automation framework is the use of JBheave, a framework for behaviour driven development (BDD). It is intended to make the testing more accessible and intuitive to enable faster and easier updates to future testing of the CSW, if desired. Each CSW test case is presented in a file that is easily readable and consists of a series of steps. Each step is mapped to methods in Java which are used to carry out the test case steps.

```
Scenario: #100.0 stop DPC Processor
Given the sstdpc machine
When terminal executes command with docker exec -w /opt/dpc/DPC_Processor/bin sstdpc ./DPCProcessor.sh stop
When terminal executes command with sleep 1m
When terminal executes command with (ignore exit code) docker exec -w /opt/dpc/DPC_Processor/bin sstdpc ./DPCProcessor.sh status
Then the operation succeed if the output string contains Service [DPCProcessor] is not running.


Scenario: #100.1 Start that the DPC_Processor and check it is running
Given the sstdpc machine
When terminal executes command with docker exec -w /opt/dpc/DPC_Processor/bin sstdpc ./DPCProcessor.sh start
When terminal executes command with sleep 2m
When terminal executes command with (ignore exit code) docker exec -w /opt/dpc/DPC_Processor/bin sstdpc ./DPCProcessor.sh status
Then the operation succeed if the output string contains Service [DPCProcessor] is running
```

Figure 4. Extract of an easily understood CSW test case story file

The test automation tool contains java projects, made to be generic and reusable to minimise repetition in the code:

- **Commanding**: Provides the ability to SSH to the IRE and execute commands on the docker instances.
- **HTTP**: Checks the availability of URL's.
- **JBehave**: Behaviour Driven Development (BDD), through story files and the underlying java code. Assertions are handled here to verify each step of a test case.
- **Properties:** A common properties project to avoid hard coding properties.
- **SOAP**: Automates SOAP service interactions.
- **SQL**: Executes queries to the SST database and returns results.
- **Selenium:** Automates web browser interactions.
- **Rich Client Platform (RCP):** An example RCP project used to automatically test Eclipse RCP clients (DPC, PS and SST-HMI), using the SWTBot.
- **SWTBot:** Automates user interactions with an Eclipse RCP Client.
- **Target**: Provides dependencies for the other projects.

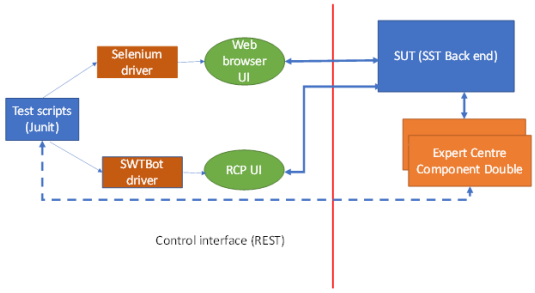
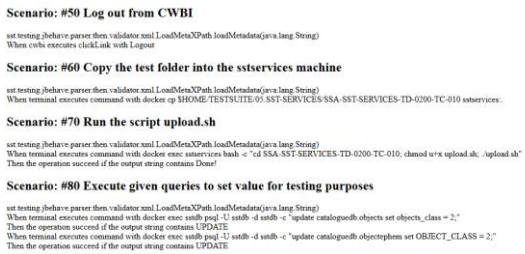


Figure 5. Automated testing for component and system regression tests

The test automation tool is deployed as a docker container onto the same environment where the CSW is deployed. This way, it ensures all necessary data is available inside the docker container, as well as has the necessary access to interact and test the CSW and the input test data folder (TESTSUITE) required for test execution.

The test_common_modules' CI/CD pipeline is responsible for test automation, from building and deploying the test container, to executing the tests, and retrieving the test reports. Once a test is completed, HTML reports (and other relevant output files of a test e.g.: screenshots, data files, etc.) are stored as GitLab artifacts for a user to download and review further.


Scenario: #50 Log out from CWBI

sst testing jbehave.parser.then.validator.xml.LoadMetaXPath.loadMetadata(java.lang.String)
When cwbi executes clickLink with Logout

Scenario: #60 Copy the test folder into the sstservices machine

sst testing jbehave.parser.then.validator.xml.LoadMetaXPath.loadMetadata(java.lang.String)
When terminal executes command with docker cp $HOME/TESTSUITE/05.SST-SERVICES/SSA-SST-SERVICES-TD-0200-TC-010 sstservices:.

Scenario: #70 Run the script upload.sh

sst testing jbehave.parser.then.validator.xml.LoadMetaXPath.loadMetadata(java.lang.String)
When terminal executes command with docker exec sstservices bash -c "cd SSA-SST-SERVICES-TD-0200-TC-010; chmod u+x upload.sh; ./upload.sh"
Then the operation succeed if the output string contains Done!

Scenario: #80 Execute given queries to set value for testing purposes

sst testing jbehave.parser.then.validator.xml.LoadMetaXPath.loadMetadata(java.lang.String)
When terminal executes command with docker exec sstdb psql -U sstdb -d sstdb -c "update cataloguedb.objects set objects_class = 2;"
Then the operation succeed if the output string contains UPDATE
When terminal executes command with docker exec sstdb psql -U sstdb -d sstdb -c "update cataloguedb.objectephem set OBJECT_CLASS = 2;"
Then the operation succeed if the output string contains UPDATE

Figure 6. Extract of an output HTML test report for a CSW test case executed through the automated test tool.

3 SST CORE SOFTWARE IMPROVEMENTS

3.1 DPC Parallelisation

3.1.1 DPC overview

The DPC is a mainly data-driven piece of software able to ingest tracklets generated by sensors and third-party orbits with the following main purposes:

- Evaluate the correlation between the input tracklets and objects stored in a catalogue.
- Update the orbital information of the correlated objects within the catalogue.
- Create new objects based on the input tracklets if no correlation was successful.
- Monitor the accuracy of the catalogue objects' orbital information and its evolution over time.
- Send tracking requests with the appropriate priorities depending on several thresholds such as the accuracy level or the time since accuracy envelop violation.

Figure 7 shows a diagram of how the data flows between the different parts that compose the DPC software. Different data flows can be distinguished and are listed below:

- **Blue lines:** processing of observational data.
- **Green lines:** third party non-correlated data.
- **Yellow lines:** third party correlated data.
- **Red lines:** data after catalogue update due to processing of any of the above-mentioned input data types.
- **Black dashed lines:** event information to be stored for data retrieval monitoring.
- **Brown dashed lines**: services that require the use of the Orbit & Covariance propagation service.
- A unique and common Database is defined and the different databases in the figure above must be understood as a logical representation of the data.

The Data Processing Chain has been re-factorized to avoid the misuse of services and improve performances, making use of basic SOAP web services and non-SOAP components. In this context, the SOAP web services provide the interfaces between the DPC and the external components: SSIM (providing simulated sensor observations), Expert Centre (providing real sensor observations), Third Party providers and Planning System. On the other hand, the non-SOAP components implement maintenance tasks and the Data Processing Chain itself.

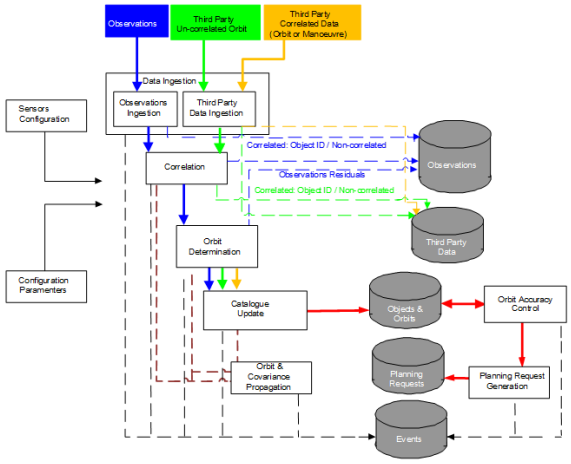


Figure 7. Data Processing Chain data flow diagram

3.1.2 Implementation in Previous Activities

The Data Processing Chain developed in the context of the DC-II project was designed as a full SOA (Service-Oriented Architecture) solution. The relationship between the system components SPS, SSIM and DPC, as well as between DPC internal elements were implemented following that Service Oriented Architecture, in which each facility dynamically interacts with the others using the COSIF SOA middleware layer.

This design decision led to serious performance problems to the DPC, due to the complexity of the system, the unnecessary overload of data interchange between services and the delays provoked by the serialization of data on client and server. It was concluded that a full SOA solution was not appropriated in DPC scenario.

The new solution is focused on designing a more efficient Data Processing Chain avoiding the use of internal services for performing basic operations and preventing the detected problems.

3.1.3 New Software Implementation

The DPC is composed of basic SOAP web services and non-SOAP components as described in Figure 8:

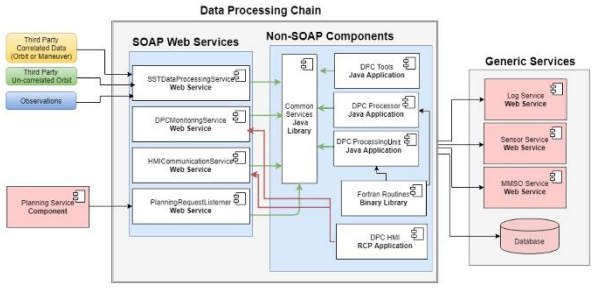


Figure 8. Data Processing Chain components

3.1.3.1 SOAP Web Services Exposed by the DPC

SSTDataProcessingService: implements the DPC's input point. The functionalities covered by the service are

the reception of observation/third party orbit data, validation of the data and the storage into the database.

PlanningRequestListener: in charge of receiving asynchronous notifications from the Planning System.

DPCMonitoringService: provides methods for starting, stopping and consulting the status of the Data Processing chain.

HMICommunicationService: internal interface only used by the DPC HMI (Human-Machine interface) to communicate with the server side of the DPC.

3.1.3.2 Non-SOAP components

DPC Tools: component implemented in Java and executed manually by one administrator. The DPC Tools implements functionalities for cleaning the catalogue database and population of the catalogue database with external TLE (Two-line Element set) catalogues. It provides scripts for start/stop and check status.

Common Services: component implemented as Java libraries which can be imported by any DPC component. It provides common utilities to the DPC components such as: configuration management, access to DPC internal database, implementation of generic services clients, etc.

DPC HMI: component that provides a graphical interface which allows the configuration, control and monitoring of the DPC.

Fortran Routines: computational algorithms which provide the processing functionalities: calibration analysis, correlation (using agendas), orbit determination, mass and area estimation, orbit and covariance propagation, generation of ephemerides, calculation of mean elements and check quality.

DPC Processor: it is the main process of the DPC. It runs as a daemon and orchestrates the processing of incoming observations and third-party data in batches. It provides scripts for start/stop and check status. The DPC Processor extracts the data to be processed from the database, issues the processing of said data and loads the results into the database.

DPC Processing Unit: is the main computational component of the DPC. It is also running as a daemon and is executing the processing (using Fortran routines) of observations and third-party data issued by the DPC Processor. Multiple DPC Processing Units can be deployed to make use of the full computational capacity of the available hardware and improve the computational speed of the DPC.

3.1.4 DPC Parallelisation

The DPC Processor is the main component of the DPC, it oversees performing several maintenance tasks and orchestrates the whole processing which provides the catalogue update by means of processing the observations coming from real sensors (or sensor simulator) and the third-party data.

Within the DPC Processor, the DataProcessingChain is a periodic task which processes requests (observations and third-party data) in batches. Some required data is loaded at the start of the batch processing as the configuration of all active sensors, the configuration of the DPC and the lists of requests to be processed.

In this activity, one of the main tasks performed by the DataProcessingChain was parallelized, that is, the observation and third-party data ingestion. The processing of this data is issued sequentially to the available **DPCProcessingUnits**, the DPC component in charge of the actual data processing.

The **DPCProcessingUnit** is able to cover all the processing jobs needed for the build-up and maintenance of a catalogue of objects, including the correlation of observations, the initial/routine orbit determination and any post-processing jobs that are needed after a new updated orbit is obtained (generation of ephemerides, OMMs, quality checks), as described in Figure 9 for the processing of observation requests.

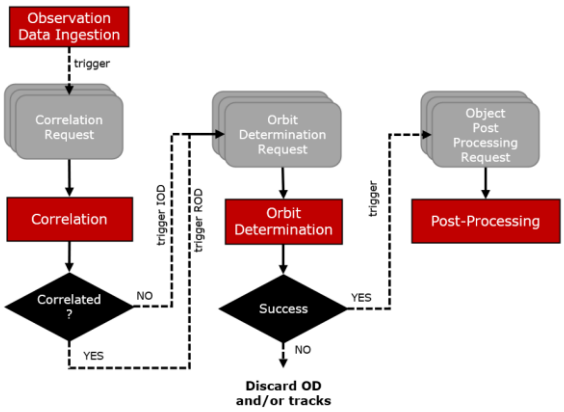


Figure 9. Observation ingestion workflow

The coordination of the **DPCProcessor** and **DPCProcessingUnits** is achieved using a combination of the orchestrator and choreography patterns, as shown in Figure 10. This means DPCProcessor works as an orchestrator-like component that issues and controls the processing of the data, while executing maintenance tasks in parallel, and the DPCProcessingUnits as the processing services.

Communication between the processor and processing units is achieved through a Kafka Message broker. Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP (Transmission Control Protocol) network protocol. It uses concepts as producers, consumers, topics, partitions and records to share messages between the different components inside a system.

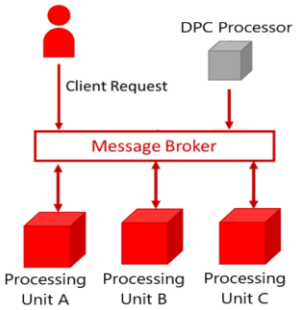


Figure 10. Orchestrator + Choreography pattern

Apache Kafka is used as the main interface between the DPC Processor and the DPC Processing Units. Figure 11 describes a high-level view of the interaction between the DPC Processor and the DPC Processing Units through Kafka, using as example two DPCProcessingUnits to illustrate the communication mechanism between the producers and consumers and how they use the topics and partitions to achieve it. As a first step, the DPC Processor queries the database for the requests to be processed in the current batch. For each request, the next steps follow the workflow described in Figure 9:

- The DPCProcessor publishes a message inside the correlation topic (using its correlation producer), informing that the correlation can be executed for the request.
- The DPCProcessingUnit uses the consumer of the correlation topic to consume the message and executes the correlation. At the end, if the request did not correlate with any catalogued object, the processing unit submits a message inside the orbit determination topic to inform that an IOD (Initial orbit determination) can be executed. Any free DPCProcessingUnit can consume the message and start the IOD. On the other hand, if the correlation was successful, a message is produced inside the successful correlation topic.
- If the correlation was successful, the DPCProcessor consumes the successful correlation topic and issues the execution of the ROD (Routine orbit determination) only if no other ROD is executed for the object. It does so by submitting a message inside the orbit determination topic.
- The DPCProcessingUnit consumes the messages from the orbit determination topic and issues the IOD or ROD. After a successful OD process, a message is produced inside the post processing topic that is consumed by the DPCProcessingUnit and executes the post-processing job (generation of OMMs (Orbit Mean Elements Message), ephemerides, quality check).

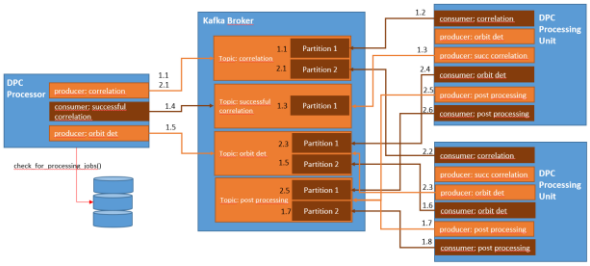


Figure 11. Kafka interface

Finally, the DPCProcessingUnits, which are DPC's main computational component, implement three main processing jobs needed for the build-up and maintenance of a catalogue of objects. These jobs are described in the subsections below.

3.1.4.1 Correlation Job

Correlation of tracks to objects based on the generation of the predicted observations (agendas) of the object in the catalogue. This correlation based on track is performed in parallel independently track-by-track.

3.1.4.2 Orbit Determination Job

The Orbit Determination (OD) job includes several algorithms for Initial Orbit Determination (IOD) and Routine Orbit Determination (ROD).

During **IOD,** orbits are computed based on an uncorrelated track. It is used for tracks that have not been correlated to an object based on the predicted observations; a new object is created from the uncorrelated track without a priori orbital information.

During **ROD,** it is executed for correlated tracks, using an initial estimation of the orbit, which could have been generated by the IOD during the correlation process. This process can be performed in parallel independently object-by-object, as the tracks are already associated to an object. This process allows to estimate the area, and the mass of the object based on the drag and radiation pressure coefficients (Cd·A/M and Cp·A/M) and the visual magnitude and RCS (Radar Cross Section) of the observations.

3.1.4.3 Post-Processing Job

This includes any job needed to be executed after a successful update of on object's orbit:

- **Orbit Quality**: This process is devoted to analysing the orbit on the catalogue and to generate tracking request for the objects violating accuracy envelope. If the catalogue was successfully updated and agendas were updated, the object's orbit calculated by the DPC is checked regarding computation of accuracy envelope and the violation of the accuracy envelope limits (which are defined at configuration level).

- **Generation of Ephemeris:** computes ephemeris and store them into the catalogue database.
- **Generation of Mean Elements (OMM):** generates mean elements by a fitting least squares process of the osculating orbit.

3.2 RPS Capabilities Enhancement

A series of upgrades were made to the Atmospheric Re-entry Prediction System (ARPS) software as part of the P3-SST-XXVI project. These improvements addressed deficiencies in the physical modelling, computational performance, and realism of risk assessments for space object re-entries. This section presents a summary of the RPS enhancements, with supporting verification analyses, however not all updates have been detailed, due to page limit constraints.

Due to the lack of extensive unit tests and system tests definitions, ESA's DRAMA software was used to verify the changes that were made to ensure no unwanted behaviour or unexpected results were output.

3.2.1 Object Initialization and Scaling

The implementation of object models has been overhauled. Template object definitions are now held in text files. This allows the redefinition or extension of the range of re-entering objects without re-compilation of the ARPS.

The scaling of all objects now adheres to the following rules:

- Object mass is scaled linearly based on the ratio of the re-entering object mass to the mass of the object prototype. Therefore, all components of a 900kg payload will be doubled in mass if the prototype object has a mass of 450kg.
- For the parent object, the object dimensions will be scaled by the square-root of the mass scaling factor. This will result in the projected area of the parent being set to the value required to ensure the original object ballistic coefficient is matched.
- For child objects, the object dimensions will be scaled by the cube-root of the mass scaling factor. This will result in the density of child objects being unchanged because of the scaling exercise.
- Child temperature and mass inheritance mechanisms were removed.

Figure 12 shows that the adjusted scaling code now correctly handles the spherical child resulting in a velocity profile in line with DRAMA. Moreover, although not apparent, this and other new test objects required for future enhancements can now be implemented without the need to recompile the ARPS.

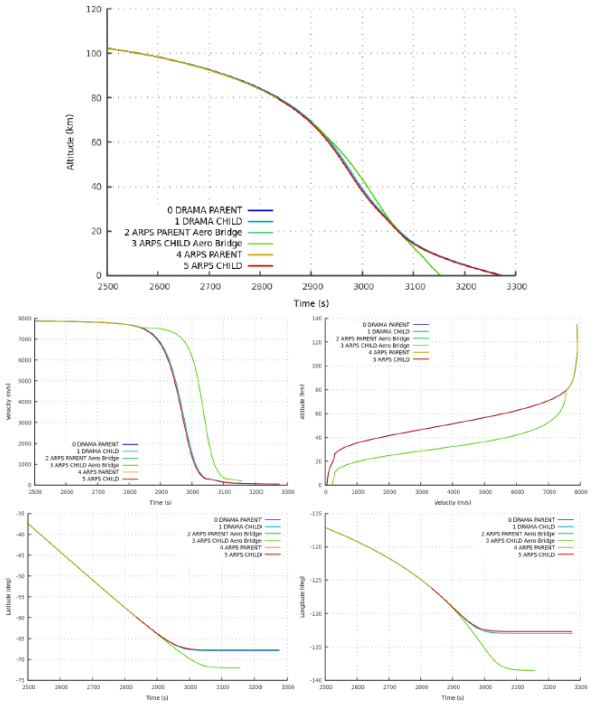


Figure 12. Initialisation and scaling post-implementation ARPS Performance Versus DRAMA 3.1

3.2.2 Updated Vehicle Models

Both the vehicle models, and algorithm used to construct vehicles, have been enhanced to improve the representation of all three types of re-entering vehicle.

The following principles have been applied when designing a single model to apply to all:

- The parent object is constructed from an undemisable material and is configured to demise at an altitude of 78km, consistent with standard practice in object-oriented analyses.
- On demise the parent object releases a catalogue of child objects covering a range of ballistic coefficients and sizes. These are constructed of aluminium, steel and titanium to represent the behaviour of objects with low, moderate and high resistance to demise respectively.

Name	Qty	Primitive	Dim1	Dim2	Material	Mass	Density	B/C	Trigger
Parent	1	Sphere	3.72		Undemise	1000	37	100	Alt78
LaLoAl	2	Cylinder	1.50	0.05	AA7075	25	283	29	
LaHiAl	2	Cylinder	0.50	0.50	AA7075	50	509	186	
MiLoAl	15	Cylinder	0.60	0.90	AA7075	10	39	19	
MiMiAl	15	Cylinder	0.38	0.57	AA7075	10	155	48	
MiMiSt	4	Cylinder	0.38	0.57	A316	10	155	48	
MiMiTi	3	Cylinder	0.38	0.57	TiAl6V4	10	155	48	
MiHiAl	15	Cylinder	0.26	0.39	AA7075	10	483	103	
MiMiTi	3	Cylinder	0.26	0.39	TiAl6V4	10	483	103	
SmLoSt	4	Cylinder	0.16	0.24	A316	0.5	104	14	
SmMiSt	4	Cylinder	0.09	0.14	A316	0.5	511	39	
SmHiSt	4	Cylinder	0.08	0.11	A316	0.5	967	60	
Ballast	294	Sphere	0.06		Ballast	1	8842	387	

Table 1 Updated Vehicle Model

Table 1 shows the updated vehicle model, where all objects are children of the Parent. Ballast objects are not actually constructed within the model as they are guaranteed to demise and will needlessly increase computational time. Approximately 5% of the mass of the model is made up of steel and titanium respectively, which is broadly consistent with the mass fractions of these materials on spacecraft.

The algorithm used to construct vehicles from this prototype has also been extended to prevent individual objects becoming excessively large or small because of the scaling operation. When instantiating a vehicle, a target mass is constructed for each object by scaling the product of the nominal mass and quantity by the ratio of the actual mass to the nominal parent mass.

Name	Qty	Max Qty	Mass	Min Mass	Mass Max
Parent	1	1	1000	N/A	N/A
LaHiAl	2	5	50	30	250
LaLoAl	2	5	25	15	125
MiMiTi	3	6	10	8	50
MiHiTi	3	6	10	8	50
MiMiSt	4	8	10	7	50
MiLoAl	15	30	10	5	50
MiMiAl	15	30	10	5	50
MiHiAl	15	30	10	5	50
SmHiSt	4	8	0.5	0.1	2
SmMiSt	4	8	0.5	0.1	2
SmLoSt	4	8	0.5	0.1	2

Table 2 *Updated Vehicle Model Mass and Quantity Limits*

The degree to which the mass of each component can be scaled is constrained to an upper and lower value provided by part of the model, while the number of objects to be instantiated is also constrained by an upper value, as described in Table 2. The constrained object scaling is then used to evaluate the number of objects required to achieve the target mass for the object type. Moreover, any mass remaining within the target once objects are instantiated is carried forward to the next object definition.

The effect of this algorithm is that as the overall mass of the re-entering object is varied both the size and number of each component type can vary within sensible bounds. This can result in large components being dropped from the model if the actual object is substantially smaller than the prototype.

The performance of the post-implementation vehicle model and algorithm is shown in *Table 3*:

Parent Mass	#Objects	#Impact	%Objects	Impact Mass	% Mass
100	35	1	3	7.0	7
200	35	2	6	15.0	8
300	45	9	20	33.6	11
500	68	11	16	49.0	10
700	71	19	27	78.5	11
1000	71	22	31	125.4	13
2000	71	56	79	321.1	16
3000	71	71	100	603.0	20
5000	74	74	100	1301.1	26
7000	106	106	100	1796.7	26
10000	142	142	100	2605.3	26

Table 3 Post-Implementation Vehicle Model Performance

The benefits of the revised model are clear. Both the number and mass of impacting fragments increase monotonically as the size of the re-entering object increases, which is in line with expectations. A 25% proportion of parent mass for large objects that impacts the ground falls within the bounds of 10% - 40% reported by Aerospace Corporation (1). The fact that only items that have the possibility of impacting the ground are modelled, and the ballast is not, limits the total number of fragments modelled leading to execution times being capped at approximately 30seconds for 10,000kg vehicles. Whilst this model is not backed up by significant research or simulations and therefore should be reviewed and refined in a later activity, it is clearly a significant step forward over the existing vehicle models.

3.2.3 Gravity Harmonics

The ARPS used a simple spherical model of gravity. This has proven to be inadequate in other codes, and gravitational models with zonal harmonic terms to at least J2 have been found to be required to provide correct along track distances from the beginning of re-entry to landing. The solution to this issue was to add a new spherical harmonic gravitational model with zonal terms to J4.

Figure 13 shows how the correction of the gravity model has a significant impact on the ARPS, resulting in the trajectories matching at high altitudes (>100km) where the gravity terms are dominant relative to the drag forces.

3.2.4 Atmospheric Model (US76)

Whilst sophisticated, the MSISE atmosphere requires significant computation time to evaluate and is typically not the model used for destructive entry analyses. To solve this problem and serve as an alternative model, US 1976 Standard Atmosphere model was added to the ARPS.

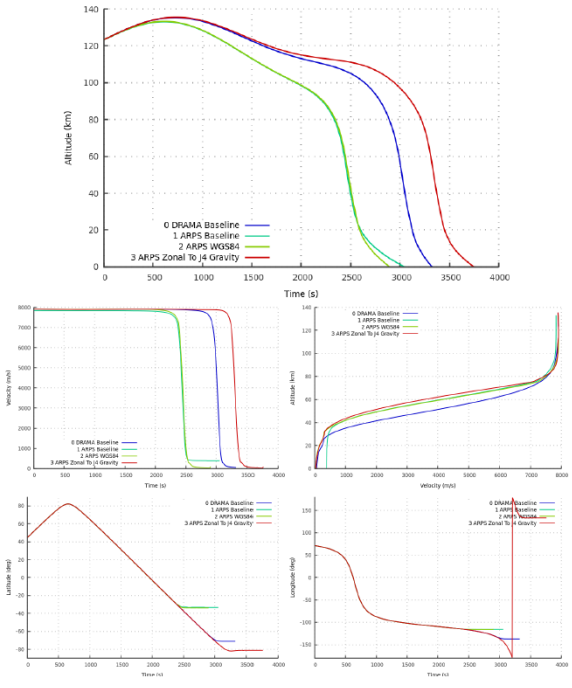


Figure 13. Gravity harmonics post-implementation ARPS Performance Versus DRAMA 3.1

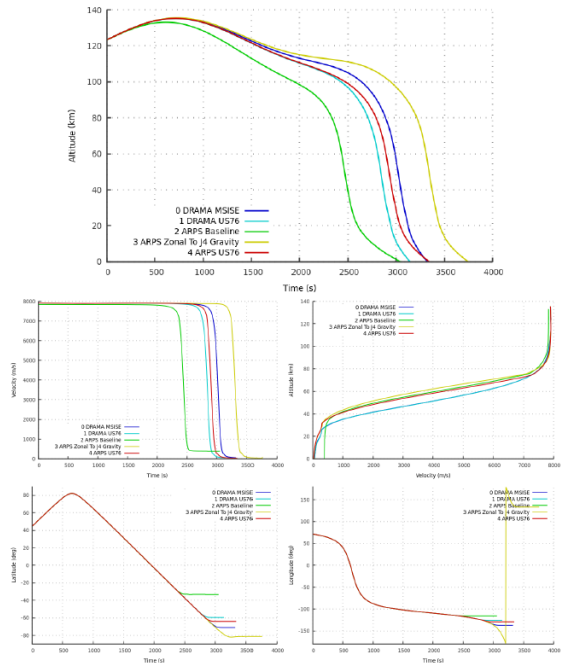


Figure 14. Atmospheric model post-implementation ARPS Performance Versus DRAMA 3.1

Figure 14 shows how the correlation of the US76 atmosphere results between DRAMA and the ARPS are significantly better than those associated with the NRLMSISE. The trajectories generated by DRAMA and ARPS are now aligned for the fully free-molecular region (>100km) and only diverge as the object enters the transition regime.

3.2.5 Earth Rotation Inclusion

In several places, the evaluation of parameters including the Mach number and kinetic energy were based on velocities in the inertial frame, rather than the rotating frame of the planet and atmosphere. As the use of these values in all instances is to gain a value relative to the atmosphere or the ground, the assessment of these values is incorrect. To solve this, A new function has been implemented to encapsulate the conversion of an inertial velocity into one relative to the planet / atmosphere.

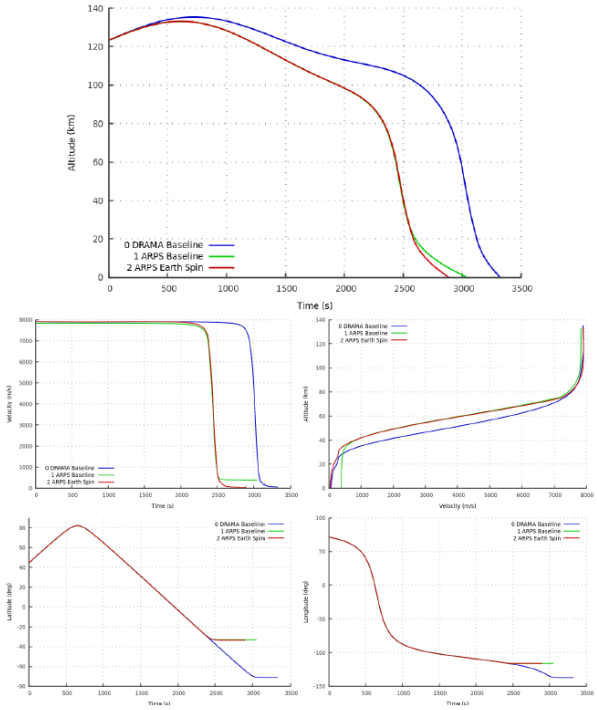


Figure 15. Earth rotation post-implementation ARPS Performance Versus DRAMA 3.1

Figure 15 shows the impact of these updates on the results. Note the small reduction in ARPS flight time and the better correlation of the velocity altitude profile at low altitudes. The large erroneous ground velocities have been removed.

3.2.6 Rarefied Bridging via Knudsen Number

The previous ARPS implementation did not model the transition between the free-molecular and continuum aerodynamic regions with an appropriate bridging regime. A proper rarefied flow model bridging between free-molecular and continuum aerodynamics is required for the ARPS to be able to obtain credible results.

The free-molecular and continuum coefficients for the three basic shapes supported by the ARPS are taken from DRAMA. In the transition regime these will be bridged based on the Knudsen number, Kn, in the same manner to DRAMA as follows:

- Knudsen number > 1.0: Free molecular drag coefficient ($C_{D,free-molecular}$) is used
- 0.01 < Knudsen number <= 1.0:

$$B = \left(\sin\left(\pi(0.5 + 0.25 \cdot \log_{10} K_n)\right)\right)^3 \quad (1)$$

$$C_{D,rarefied} = C_{D,continuum} + B\left(C_{D,free-molecular} - C_{D,continuum}\right) \quad (2)$$

- Knudsen number <= 0.01: Continuum drag coefficient ($C_{D,continuum}$) is used

The trajectory performance for a spherical object of ARPS is now closely aligned with that generated by DRAMA, as can be seen in *Table 4*:

Variable	MSISE Baseline			Current Performance		
	DRAMA	ARPS	Difference	DRAMA	ARPS	Difference
Flight time	3326 secs	3054 secs	272 secs	3146 secs	3162 secs	16 secs
Impact latitude	-71.23⁰	-33.08⁰	39.15⁰	-59.62⁰	-60.38⁰	0.76⁰
Impact Longitude	-137.24⁰	-115.93⁰	21.31⁰	-126.06⁰	-126.54⁰	0.48⁰

Table 4 Comparison of ARPS and DRAMA Trajectory for a Sphere

At the completion of this fix, the distance between the impact locations reported by DRAMA and ARPS is 89km, which translates into the removal of 98% of the 4429km difference seen in the original baseline comparison at the outset of this activity. The correlation now seen is well within the uncertainties in aerodynamics of the actual vehicles, and therefore, the performance of aerodynamics of spheres is deemed acceptable.

3.2.7 Standard Aerodynamic Coefficients

ARPS coefficients differed from DRAMA, causing trajectory drift. Therefore, the free molecular and continuum coefficients used by DRAMA v3.1 for spheres, cylinders and boxes have been extracted from the NetCDF databases within the application. These have then been implemented with the ARPS as detailed in *Table 5*.

Primitive	Free Molecular Cd	Continuum Cd
Sphere	2.07	0.91
Cylinder	2.07	0.914
Box	2.07	0.919

Table 5 New ARPS drag coefficients

3.2.8 Simplified Knudsen Number Evaluation

The prior implementation was unnecessarily complex and not easily maintainable.

A straightforward hard sphere Knudsen number model is used. The Knudsen number

$$K_n = \frac{\lambda}{L} \quad (3)$$

where λ is the mean free path and L is the reference length of the object. The hard sphere model gives the mean free path as:

$$\lambda = \frac{k_B T}{\sqrt{2}\pi d^2 p} \quad (4)$$

where k_B is the Boltzmann constant, T is the temperature, d is the molecular diameter and p is the local pressure. Using the ideal gas equation, this becomes:

$$\lambda = \frac{k_B}{\sqrt{2}\pi d^2 \rho R} \quad (5)$$

where ρ is the local density and R is the species gas constant. For the Earth's atmosphere, the molecules of interest in the rarefied region are nitrogen and oxygen. The molecular diameter, d, is taken as 3.64x10-10m, and the gas constant, R, is taken as 287J/kgK. All the values in this equation are now constants, except for the local density.

This allows implementation of the Knudsen number in ARPS as:

$$K_n = \frac{8.13 \cdot 10^{-8}}{\rho L} \quad (6)$$

which is a significant simplification over the previous implementation.

3.3 Track-to-Track Association

The Track-To-Track (T2T) association aims to reliably detect and initialise new resident space objects (RSOs) to help with cataloguing space objects that are not currently within the catalogue. This can be a time consuming and computationally expensive process to conduct, which is where the T2T algorithm described in this article steps in.

3.3.1 Track-To-Orbit Drawbacks

Track-To-Orbit (T2O) association can be effect if certain conditions are met, but it does have some drawbacks. Firstly, it is very dependent on the models used to predict a catalogued object's position in the future. Highly complex and accurate models are time consuming and with so many tracks that could be fed to the system, this is undesirable. Secondly, the success of T2O algorithms is also dependent upon the size of the current catalogue and the accuracy of information inside it. Small initial catalogues will make it hard to associate tracks from sensors to objects in the catalogue, leading to a low success rate. Additionally, if the information in the catalogue is rarely updated or inaccurate then this will lead to more false positives and false negatives being associated as well as making the orbit prediction processes require more work as the error in the prediction relies on the period that the prediction is made over.

Lastly, for tracks that do not get associated via T2O, these tracks may identify a new object or an existing object that has conducted a manoeuvre. In this case an IOD may be conducted to update or add the object to the catalogue. However, a singular IOD for an object will not be sufficiently accurate to be used later when associating it to other tracks that may belong to the same object, which can incur false results.

3.3.2 The Need for T2T

The T2T association algorithm described in this paper addresses those limitations outlined in section 2.1. This is not to say that T2T should be used in replacement of T2O algorithms, but rather they can be used together as the T2T picks up the pieces where the T2O lacks.

T2T uses models of varying complexity and accuracy to help speed up computations where accuracy is not the top priority and maintains that accuracy when it is needed for further inspection or validation. It also contains thresholds that involve the number of tracks and figure of merit (FOM) that ensure the abundance and quality of the tracks are sufficient to reliably associate the tracks. This avoids unreliable results and could help rectify some error that may already exist in the catalogue

It is also independent of the current state of the catalogue as it does not need to refer to it at any point during the prediction of the objects orbit, only when checking if it currently exists to add to or update the orbit. This avoids the drawbacks of the amount or quality of data in the catalogue.

3.3.3 Definitions

Below are some definitions that can be commonly mistaken, so are shown here for clarity in the context of this paper.

- Measurement: Single value of a geometrical or physical property of an object observed by a sensor at certain epoch (e.g. Azimuth).
- Observation: A set of measurements taken from a single sensor at a common epoch and originated from the same object.
- Track: A set of observations taken by a single sensor during a period of continuous observation of an object.
- Hypothesis: Association of N tracks assumed to have been originated from a common object.

3.3.4 Methodology

Ref. [1] gives an in depth look at the algorithm that was used as the main inspiration for the algorithm that was implemented into the CSW. In this section, a rough outline of the methodology is shown as well as any other tweaks that were made to suit the implementation into the CSW.

The methodology for this T2T algorithm can be broken down into 7 main processes which are:

- Generation
- Estimation
- Scoring
- Pruning
- Promotion
- Merging
- Confirmation

These steps link together as shown below in Figure 16.

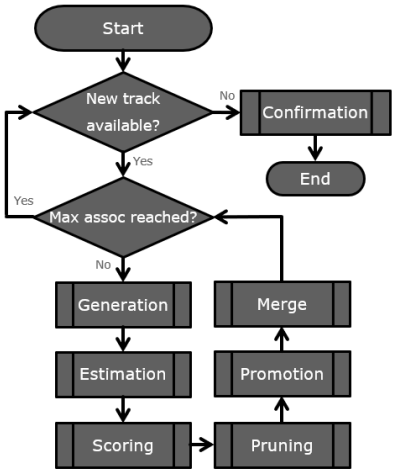


Figure 16. T2T overview schematic

3.3.4.1 Generation

The Generation step is the first major step in the T2T algorithm and is responsible for generating hypotheses from new and previously uncorrelated tracks. The generation of hypotheses are subject to thresholds which limit the computational burden that comes when creating hypotheses from tracks that obviously wont correlate. These thresholds are:

- Upper- and lower-time span thresholds
- State thresholds

The time span thresholds ensure that the tracks are not too close or too far in time to correlate with one another. The state thresholds which check semi-major axis, eccentricity and inclination ensure that the tracks have roughly the same orbits so that it can avoid any obvious tracks that won't correlate.

Further to these thresholds, there is a requirement that for a hypothesis to be generated it must come from two child hypothesis. These child hypotheses must have all but one track in common in order for the new hypothesis to be generated which will result in the generated hypothesis having one more track than the child hypotheses (All common tracks plus each unique track from each child hypothesis).

3.3.4.2 Estimation

The estimation step is responsible for the execution of the IOD/OD for each hypothesis within the new hypothesis list created by the generation step. The IOD is called for hypotheses containing only one track and OD for all other hypotheses. The outputs of this step consist of the state vector, covariance matrix and measurement residuals, all of which help with the scoring of hypotheses.

For the propagators used within the T2T algorithm, it is desirable to have the ability to use both numerical and analytical/semi-analytical propagators. Since there is a trade-off between accuracy and speed between the types of propagators used it is important to use the correct one in the correct place. The analytical/semi-analytical propagators are best used for when hypotheses have fewer tracks due to its greater speed and larger convergence windows which will help to avoid non convergence on tracks that could possibly correlate when the hypothesis has more tracks. On the contrary, when it comes to validating and confirming that a hypothesis is true, then it is desirable to use a numerical propagator since the importance lies in the accuracy and not so much the speed for a step as important as this.

Despite this plan being made, the Orbit determination service in the DPC did not have an analytical/semi-analytical propagator implemented. Due to this falling outside the scope of the activity, it was not implemented, and the tests were run with only the numerical propagator. This will make the system less effective in the problems it tries to solve, but it will not make results any less reliable, bar the speed of course.

3.3.4.3 Scoring

This step is where the hypotheses are evaluated by a 'Figure of merit' (FOM) which is a value that represents how well the tracks within a hypothesis correlate. In this case the lower the FOM value the more likely it is that the tracks within the hypothesis correlate. The FOM can be seen as a Mahalanobis distance in the measurement space demonstrated in Eq. 1.

$$d^2(H) = \frac{1}{|H|}\sum_{T\in H}\frac{1}{|T|}\sum_{z\in T}(z-\hat{z})^{\mathrm{T}}(\boldsymbol{P}_z^0)^{-1}(z-\hat{z}) \quad (7)$$

Where d is the FOM, H is the hypothesis, $|H|$ is the number of tracks in the hypothesis, T is the track, $|T|$ is the number of observations in the track, z is the measurement, $\hat{z}$ is the a-posteriori computed measurement and $\boldsymbol{P}^0{}_z$ is the a-priori measurement covariance.

This equation corresponds to the weighted root mean squared (WRMS) and allows each track to contribute equally to the FOM, else, this could encourage the production of false positives or false negatives depending on the situation.

3.3.4.4 Pruning

The pruning step is a filtering step which will only allow certain hypotheses through and all those that don't meet the criteria are removed from the system (the tracks are not removed, just the hypothesis). The criterion for passing this step is for hypotheses to have a FOM value lower than a threshold configurable by the user. A different value can be configured for hypotheses with 1, 2, 3 and 4+ tracks in a hypothesis. It is important to not configure this value to be too low for hypotheses with low number of tracks as it is not desirable to filter out hypotheses that potentially could correlate too early due to them having a high FOM. On the flip side, it is also not good to have these thresholds too high so as to allow through so many hypotheses that it becomes a computational burden.

3.3.4.5 Promotion

The promotion step is where hypotheses that meet certain criteria can be considered true and correlated. To be considered for promotion, the criteria that a hypothesis must meet are that a hypothesis must contain a minimum of 4 tracks, and it must have a FOM value lower than a user defined threshold value, similar to that from the pruning step. The hypotheses that pass is then sorted by increasing FOM and are promoted in order starting with the lowest FOM.

When a hypothesis is promoted, it is considered that a promoted track cannot belong to more than one hypothesis which could suggest that it belongs to more than one object. This is not possible so when a hypothesis is promoted, all other hypotheses that contain a track that belongs to the promoted hypothesis are invalid and removed from the system.

3.3.4.6 Merge

This step can be seen as optional as all hypotheses that make it to this step can be deemed accurate enough to add to the database. However, it has been included in this implementation. The objective of the merging process is to attempt to combine two already promoted hypotheses. This helps to avoid adding duplicate hypotheses to the database. The two selected hypotheses must pass time span thresholds and state thresholds similar to that in the generation step, but with more strict values. If the two hypotheses pass this then they are merged, estimated and scored as one hypothesis in the same way as in the previous steps. Then if the merged hypothesis results in the lower FOM value than that of its two child hypotheses then it is validated, and the child hypotheses are removed from the validated list.

3.3.4.7 Confirmation

The confirmation step involves using an OD model that is more accurate that what has been used already in order to fully confirm and have even more confidence that the promoted hypotheses are correlated. However, this was

not possible to do with the current state of the OD service as it did not have the capability of using such an OD model at the time of implementation.

3.3.5 T2T Integration within SST Core Software

In the context of the SST Core Software, the tool DPC is the one in charge of processing observations, performing object correlations, computing orbit propagations and assigning planning requests to sensors based on the status of the objects catalogue.

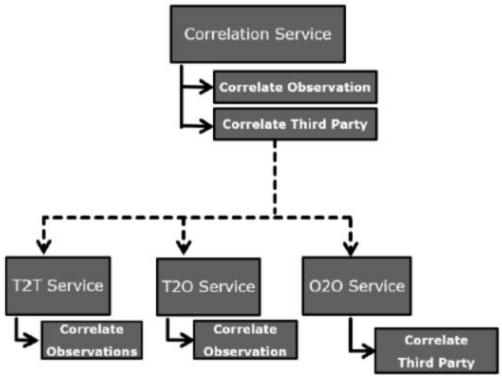


Figure 17. T2T implementation within DPC Correlation Service

Figure 17 shows how the T2T algorithm has been included in the Correlation Service within DPC Processor execution chain. As explained in section 3.1.4, Once an observation is ingested and its processing starts, a DPC Processing Unit will try to correlate it with an object that already exists in the catalogue.

In the previous implementation, if the observations present in the input tracklet are not correctly correlated to any object in the catalogue, a new object would be generated with only the orbital information from that uncorrelated tracklet. However, with the new implementation, once a tracklet could not be correlated to an object in the catalogue, its information is now sent to the T2T algorithm. Once ingested, the algorithm will send the tracklet observations through all the steps defined in section 3.3.4, trying to correlate them to already existing tracklets that did not have enough information to conform a valid new object. From this point three different outcomes can be faced:

- The tracklet is associated to a hypothesis with enough information to generate a new object: in this case, this information will transcend into the database by creating a new object in the catalogue and removing the hypothesis.
- The tracklet is associated to a hypothesis without enough information to generate a new object: in this situation, the hypothesis will remain incomplete waiting for more tracklets to be ingested. No new objects are generated, and no hypotheses are removed.
- The tracklet is not associated with any hypothesis: in this case, a new hypothesis is generated with a single tracklet. This new hypothesis will remain incomplete until new information is ingested.

4 REFERENCES

1. Pastor, A., Sanjurjo-Rivo, M. & Escobar, D. (2022). Track-to-track association methodology for operational surveillance scenarios with radar observations. *CEAS Space Journal (2023)*, 535-551.
2. Comparison of Reentry Breakup Measurements for Three Atmospheric Reentries. Feistel, A, Weaver, M, Ailor, W. s.l. : 6th IAASS Conference, 2013.