FIDUCEO Multi-sensor Match up System Implementation Plan

Tom Block

Brockmann Consult GmbH

12/4/2015

FIDUCEO has received funding from the European Union's Horizon 2020 Programme for Research and Innovation, under Grant Agreement no. 638822

Contents

1 Introduction

This document describes the technical implementation plan for a FIDUCEO Multi-sensor matchup system. As a starting point it summarizes the requirements issued by the project partners, namely DLR, University of Hamburg and University of Reading.

The following chapters derive technical requirements based on the scientific requirements on the matchup generation process. A modular concept is developed and the modules and their technical implementation are described.

1.1 Scope

This document is thought as a living document, updates will be integrated whenever applicable or necessary. The Implementation Plan shall both serve as a project deliverable describing the ideas and concepts behind the system as well as a technical description that allows users and developers to gain a more in-depth knowledge of the MMS, in a detail level beyond the information supplied in the manual.

1.2 Version Control

Version	Reason	Reviewer	Date of Issue
1.0	Initial version	Martin Burgdorf (UoHH), Olaf Danne (BC), Chris Merchant (UoR), Jon Mittaz (UoR), Thomas Popp (DLR)	2015-12-04
1.1	- Updates of use-case descriptions - Update on AMSU-B/MHS cloud screening		

1.3 Applicable and Reference Documents

The following documents are applicable (AD) or references (RD) in this Implementation Plan document:

RD 1	SST_CCI-TN-UOL-001 SST_CCI MMD Content Specification Issue 1 (2012 05 04)
RD 2	ESA CCI Phase-II Sea Surface Temperature (SST) Technical Note - MMS Implementation Plan v1
RD 3	FIDUCEO_AVHRR_HIRS_requirements – University of Reading
RD 4	MMS-DLR – DLR
RD 5	WP3_table-revised – University of Hamburg

RD 6	Soden, B.: The diurnal cycle of convection, clouds, and water vapour in the tropical upper troposphere, GEOPHYSICAL RESEARCH LETTERS, VOL. 27, NO. 15, PAGES 2173-2176, AUGUST 1, 2000
RD 7	I. Sobol, D. Asotsky, A. Kreinin, S. Kucherenko (2011). "Construction and Comparison of High-Dimensional Sobol' Generators" (PDF). Wilmott Journal Nov: 64–79
RD 8	Rossow, W., Garder, L.: Cloud detection using satellite measurements of infrared and visible radiances for ISCCP. J. Climate, 6, 2341-2369, 1993
RD 9	Buehler et al.: A cloud filtering method for microwave upper tropospheric measurements, 2007
RD 10	John et al.: Clear-Sky biases in satellite infrared estimates of upper tropospheric humidity and its trends, 2011

1.4 Glossary

DBCP	Database Connection Pooling
DLR	Deutsches Zentrum für Luft- und Raumfahrt
EFOV	Elementary Field Of View
EPL	Eclipse Public License
GPL	GNU General Public License
IFOV	Instantaneous Field Of View
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JTS	Java Topology Suite
LGPL	GNU Lesser General Public License
LSF	Load Sharing Facility (IBM Job Scheduling Engine)
MMD	Multisensor Matchup Dataset
MIT	Massachusetts Institute of Techniology
MPL	Mozilla Public License
NWP	Numerical Weather Prediction
RDBMS	Relational Database Management System
SNAP	Sentinel Application Platform
SPI	Service Provider Interface
SRF	Spectral Response Function
UHH	University of Hamburg
UoR	University of Reading
URL	Uniform Resource Locator

2 Use Cases

The software design and the technical solutions described in this document are mainly driven by the requirements issued by the project partners concerning the generation of multi-sensor and insitu match-up datasets. This chapter summarizes the requirements for the MMS functionality that have been submitted by the project partners. The table below lists inputs received from DLR, University of Hamburg and University of Reading.

The table column captions use abbreviations for conceptual entities that are explained below where clarification might help in understanding

- *Window:* defines a rectangular geographical subset that shall be used for data extraction. The units are pixels (cross-track, along-track) in the satellite raster.
- *Win. Overlap*: defined the actions that shall be executed when the matchup dataset contains overlapping subset windows. Overlaps can either be kept or removed which results in lesser matchup pixels but avoids statistical correlations.
- *PX Delta*: maximally distance between matchup pixels. If not defined the closest pixel is chosen for matchup.
- PX Seed: defines where the pixel locations used for matchup processing are originating from. “any” – use any available pixel; “insitu” - use pixels covering insitu measurements, “random” – use pseudo-random sequences to define pixel locations

Table 1: Summary requirements on the MMS

Req. ID	Sensors	Window (px)	Win.Overlap	Time Delta	PX Delta	VZA Delta	Time Range	PX Seed
MMD-ID-001	AVHRR (GAC), (A)ATSR	11 x 11	Keep	<= 15 min	n.a.	<= 10 deg	1991-08 – 2012-04	any
MMD-ID-002	AVHRR (GAC), AVHRR (GAC)	5 x 5	Keep	<= 5 min	n.a.	<= 10 deg	n.a.	any
MMD-ID-003	AVHRR, IASI	15 x 15 AVHRR	Keep	<= 15 min	n.a.	<= 10 deg	n.a.	any
MMD-ID-004	AVHRR, Insitu	5 x 5	Remove	<= 4 h	n.a.	<= 10 deg	n.a.	insitu
MMD-ID-005	HIRS, HIRS All satellite combin.	5 x 5	Keep	<= 15 min	<= 34 km (HIRS 2/I and HIRS/3) <= 17 km HIRS/4	<= 10 deg	1978-10 - today	any
MMD-ID-006	HIRS/3 (NOAA-17), IASI	5 x 5	Keep	<= 15 min	<= 34 km	<= 10 deg	2009-04 – 2009-05	any

Req. ID	Sensors	Window (px)	Win.Overlap	Time Delta	PX Delta	VZA Delta	Time Range	PX Seed
MMD-ID-007	HIRS/4 (NOAA-18), AIRS	5 x 5	Keep	<= 15 min	<= 17 km	<= 10 deg	2005-06 - today	any
MMD-ID-008	HIRS/4 (NOAA-19), AIRS (AQUA)	5 x 5	Keep	<= 15 min	<= 17 km	<= 10 deg	2009-04 - today	any
MMD-ID-009	HIRS/3 (NOAA-16), IASI (MetOp-A)	9 x 9	Keep	<= 15 min	<= 34 km	<= 10 deg	2012-01 – 2014-12	any
MMD-ID-010	HIRS/3 (NOAA-16), IASI (MetOp-B)	9 x 9	Keep	<= 15 min	<= 34 km	<= 10 deg	n.a.	any
MMD-ID-011	HIRS, Radiosonde	9 x 9	Keep	<= 3h	<= 180 km	n.a.	n.a.	Radiosonde
MMD-ID-012	AVHRR, AVHRR	11 x 11	Keep	<= 30 min	n.a.	<= 10 deg	2002-01 – 2012-12	any (Europe, Land)
MMD-ID-013	AVHRR, MVIRI	none	n.a.	<= 30 min	n.a.	<= 10 deg	2002-01 – 2012-12	any (Europe, Land)
MMD-ID-014	AVHRR, AERONET	51 x 51	Keep	<= 30 min	<= 50 km	n.a.	2002-01 – 2012-12	Aeronet Stations
MMD-ID-015	MHS, IASI	9 x 9 IASI	Keep	<= 5 min	<= 5 km	see below	2006-11 - today	random
MMD-ID-016	MHS (MetOp), MHS (NOAA)	none	n.a.	<= 5 min	<= 5 km	see below	2006-11 - today	any
MMD-ID-017	MHS, AMSU-B	none	n.a.	<= 5 min	<= 5 km	see below	2005-06 – 2014-06	any
MMD-ID-018	MHS, SEVIRI	7 x 7 SEVIRI	Keep	<= 5 min	<= 5 km	see below	2005-06 - today	random
MMD-ID-019	MHS, ATMS	none	n.a.	<= 5 min	<= 5 km	see below	2011-10 - today	any
MMD-ID-020	MHS, SSMIS	none	n.a.	<= 5 min	<= 5 km	see below	2005-06 - today	any
MMD-ID-021	AMSU-B, AMSU-B	none	n.a.	<= 5 min	<= 5 km	see below	2000-09 - 2011-03	any
MMD-ID-022	AMSU-B, SSM/T-2	3 x 3 AMSU	Keep	<= 5 min	<= 5 km	see below	1999-01 – 2001-08	any
MMD-ID-023	MHS, HIRS/4	9 x 9 HIRS	Keep	<= 5 min	<= 5 km	see below	2005-08 - today	random
MMD-ID-024	AMSU-B, HIRS/3	9 x 9 HIRS	Keep	<= 5 min	<= 5 km	see below	1991-01 – 2014-06	random
MMD-ID-025	AMSU-B, GRUAN	5 x 5	n.a.	<= 30 min	<= 50 km	see below	2011-01 - today	GRUAN location
MMD-ID-026	MHS, GRUAN	5 x 5	n.a.	<= 30 min	<= 50 km	see below	2011-01 - today	GRUAN location
MMD-ID-027	ATMS, GRUAN	5 x 5	n.a.	<= 30 min	<= 50 km	see below	2011-01 - today	GRUAN location
MMD-ID-028	SSMIS, GRUAN	7 x 7	n.a.	<= 30 min	<= 50 km	see below	2011-01 - today	GRUAN location
MMD-ID-029	SSM/T-2, GRUAN	3 x 3	n.a.	<= 30 min	<= 50 km	see below	2011-01 - today	GRUAN location
MMD-ID-030	MultiSens, GRUAN	Sensor dependent	n.a.	<= 30 min	<= 50 km	see below	2011-01 - today	GRUAN location

Req. ID	Sensors	Window (px)	Win.Overlap	Time Delta	PX Delta	VZA Delta	Time Range	PX Seed
MMD-ID-031	n.a.							
MMD-ID-032	n.a.							
MMD-ID-033	n.a.							
MMD-ID-034	AVHRR/HIRS on same platform	75 x 41 AVHRR	Remove	n.a.	n.a.	n.a.	2006-10 – today	random

2.1 Detailed Use Case Descriptions

The following chapter lists details of the use cases supplied by the project partners. A use-case numbering scheme is introduced for this document to unify the different numbering schemes of the partners. A reference to the original issuer of the use-case requirement and the original ID is given at the head of each section.

2.1.1 Use case MMD-ID-001

Original use case ID: n.a., UoR.

AVHRR-GAC and (A)ATSR, using all available overlaps, subset size 11 x 11 pixel for each overlapping pixel, time delta below 15 minutes, VZA delta below 10 deg.

Post processing to include NWP profiles with the diurnal case time range (-96, +48 hours) and in both (A)ATSR views. Post processing to include AVHRR calibration data.

2.1.2 Use case MMD-ID-002

Original use case ID: n.a., UoR.

AVHRR-GAC and AVHRR-GAC, using all available overlaps, subset size 5 x 5 pixel for each overlapping pixel, time delta below 5 minutes, primary sensor VZA below 10 deg, VZA delta below 10 deg, border pixels removed.

Post processing to include NWP profiles with the diurnal case time range (-96, +48 hours). Post processing to include AVHRR calibration data.

2.1.3 Use case MMD-ID-003

Original use case ID: n.a., UoR.

AVHRR-GAC and IASI, using all available overlaps, subset size 15 x 15 AVHRR pixel for each overlapping pixel, one IASI pixel per window, time delta below 15 minutes, VZA delta below 10 deg.

Post processing to include NWP profiles with the diurnal case time range (-96, +48 hours). Post processing to include AVHRR calibration data.

2.1.4 Use case MMD-ID-004

Original use case ID: n.a., UoR.

AVHRR-LAC/AVHRR-GAC and Insitu Data, using all available overlaps, subset size 5 x 5 pixel for each matchup pixel, overlapping windows removed, time delta below 240 minutes, ultra-clear sky.

Post processing to include NWP profiles with the diurnal case time range (-96, +48 hours). Post processing to include AVHRR calibration data.

2.1.5 Use case MMD-ID-005

Original use case ID: n.a., UoR.

HIRS and HIRS, using all available overlaps, subset size 5 x 5 pixel for each matchup pixel, time delta below 15 minutes, VZA delta below 10 deg.

MMD shall contain:

- UTC time of measurement for both sensors
- lat/lon of footprint centre for both sensors
- lat/lon/altitude of both spacecraft
- identifier for NOAA CLASS granule for both sensors
- scanline number and scanline position for both sensors
- lat/lon for a selection of altitudes
- brightness temperatures for all channels
- additional information from the NOAA CLASS granules like telemetry information, including all available metadata

Selection of altitudes:

The instantaneous field of view for a spacecraft is a cone. Standard geolocation specifies the field of view centre at the surface. We wish to calculate how this position changes as a function of altitude. Unless the view is exactly nadir, the latitude and longitude for the centre of the field of view change as a function of altitude. To assess collocations, it is beneficial to know this centre not only at the surface, but also at the nominal altitudes for the channels we are interested in. As an approximation, we can consider 0 km, 5 km, 10 km, 15 km, and 20 km.

2.1.6 Use case MMD-ID-006

Original use case ID: n.a., UoR.

Matchups between HIRS and MetOp-A IASI, using all available overlaps, subset size 9 x 9 pixel for each HIRS matchup pixel, single pixel IASI extract, time delta below 15 minutes, both sensor zenith angles below 10 deg, VZA delta below 10 deg. Spatial distance maximally 9km.

2.1.7 Use case MMD-ID-007

Original use case ID: n.a., UoR.

Matchups between NOAA-18 HIRS/4 and AQUA AIRS between 2005-06 - today, using all available overlaps, subset size 8 x 8 pixel for each matchup pixel, time delta below 15 minutes, VZA delta below 10 deg.

2.1.8 Use case MMD-ID-008

Original use case ID: n.a., UoR.

Matchups between NOAA-19 HIRS/4 and AQUA AIRS between 2005-06 - today, using all available overlaps, subset size 9 x 9 pixel for each matchup pixel, time delta below 15 minutes, VZA delta below 10 deg.

2.1.9 Use case MMD-ID-009

Original use case ID: n.a., UoR.

Matchups between MetOp-A HIRS and MetOp-A IASI and MetOp-B HIRS and MetOp-B IASI, using a sobol based random sampling point matchup strategy, subset size 9 x 9 pixel for each HIRS matchup pixel, both sensor zenith angles below 10 deg, VZA delta below 10 deg. Spatial distance maximally 9km.

2.1.10 Use case MMD-ID-010

Original use case ID: n.a., UoR.

Matchups between HIRS and MetOp-B IASI, using all available overlaps, subset size 9 x 9 pixel for each HIRS matchup pixel, single pixel IASI extract, time delta below 15 minutes, both sensor zenith angles below 10 deg, VZA delta below 10 deg. Spatial distance maximally 9km.

2.1.11 Use case MMD-ID-011

Original use case ID: n.a., UoR.

Precise requirements for this collocation set are to be more fully determined later in the project. Around each radiosonde launch, we should get HIRS in a radius of 180 km, 3 hours.

2.1.12 Use case MMD-ID-012

Original use case ID: MMS-WP5.3-01, DLR.

AVHRR Level 2 Aerosol matchups, NOAA 16 – NOAA 18, 11 x 11 pixel window, using all available overlaps, time delta below 30 minutes, ultra-clear windows. Overlaps only over Europe (15W – 45E, 30N - 70N) and land. All pixels of the window shall be over land.

2.1.13 Use case MMD-ID-013

Original use case ID: MMS-WP5.3-02, DLR.

AVHRR Level 2 Aerosol and MVIRI matchups, using all available overlaps, time delta below 30 minutes (nearest time slot), ultra-clear windows. Overlaps only over Europe (15W – 45E, 30N - 70N) and land. All pixels of the window shall be over land.

2.1.14 Use case MMD-ID-014

Original use case ID: MMS-WP5.3-03, DLR.

AVHRR Level 2 Aerosol and Aeronet, using all available overlaps, time delta below 30 min, 50 km window. Overlaps only over Europe (15W – 45E, 30N - 70N) and land. All pixels of the window shall be over land.

2.1.15 Use case MMD-ID-015

Original use case ID: Requirement 1, UHH.

MHS and IASI, using a SOBOL pseudo-random pixel selection generating approx. 60000 matchups per month, subset size 9 x 9 pixel IASI, 7 x 7 pixel MHS, pixel delta <= 5km, time delta below 5 min, angle constraint $|\cos(\text{VZA-IASI}) / \cos(\text{VZA-MHS}) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$), ultra-clear sky.

IASI Level 1 data from EUMETSAT Data Centre, to be processed with AAPP scripts iasi_1c_to_pc and iasipc_to_hdf5.

Level 1 data of MHS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

Cloud screening algorithm according to the algorithms described in 2.2.1 and 2.2.4.

2.1.16 Use case MMD-ID-016

Original use case ID: Requirement 2, UHH.

MHS and MHS, cross platform, using all available overlaps, pixel delta <= 5km, time delta below 5 min, angle constraint $|\cos(\text{VZA-MetOp}) / \cos(\text{VZA-NOAA}) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$), cloud filtering as described in chapter 2.2.4.

Level 1 data of MHS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

2.1.17 Use case MMD-ID-017

Original use case ID: Requirement 3, UHH.

AMSU-B and MHS using all available overlaps, pixel delta <= 5km, time delta below 5 min, angle constraint |cos(VZA-AMSU) / cos(VZA-MHS) − 1| < ε1 (ε1 = 0.01), cloud filtering using the algorithm described in chapters 2.2.3 and 2.2.4.

Level 1 data of MHS and AMSU-B from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

2.1.18 Use case MMD-ID-018

Original use case ID: Requirement 4, UHH.

MHS (MetOp) and SEVIRI (METEOSAT-8 – 10), using a SOBOL pseudo-random pixel selection generating approx. 60000 matchups per month, subset size 9 x 9 SEVIRI pixel, pixel delta <= 5km, time delta below 5 min, angle constraint |cos(VZA-MHS) / cos(VZA-SEVIRI) − 1| < ε1 (ε1 = 0.01), cloud filtering as described in chapter 2.2.4, uniformity test STDDEV(Rad7x7SEVIRI) < ε2 (ε2 = 1 K)

SEVIRI Level 1 data from EUMETSAT Data Centre, to be processed with AAPP scripts iasi_1c_to_pc and iasipc_to_hdf5.

Level 1 data of MHS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

2.1.19 Use case MMD-ID-019

Original use case ID: Requirement 5, UHH.

MHS (MetOp-A/B and NOAA-18/19) and ATMS (SUOMI-NPP), using all available overlaps, pixel delta <= 5km, time delta below 5 min, angle constraint |cos(VZA-ATMS) / cos(VZA-MHS) − 1| < ε1 (ε1 = 0.01), cloud filtering as described in chapter 2.2.4.

Level 1 data of MHS and ATMS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

2.1.20 Use case MMD-ID-020

Original use case ID: Requirement 6, UHH.

MHS (MetOp-A/B and NOAA-18/19) and SSMIS (DMSP-F16-F19), using all available overlaps, pixel delta <= 5km, time delta below 5 min, angle constraint |cos(VZA-MHS) / cos(VZA-SSMIS) − 1| < ε1 (ε1 = 0.01), cloud filtering as described in chapter 2.2.4.

Level 1 data of MHS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

2.1.21 Use case MMD-ID-021

Original use case ID: Requirement 7, UHH.

AMSU-B (NOAA-15 - 17), using all available overlaps, pixel delta <= 5km, time delta below 5 min, angle constraint |cos(VZA-Sat1) / cos(VZA-Sat2) − 1| < ε1 (ε1 = 0.01), cloud filtering using the algorithm described in chapter 2.2.3.

Level 1 data of AMSU-B from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the flight direction (north or south), derived from Scan Line Bit Field, bit 15 (northbound/southbound flag)
- the scan position (step number), derived from NAVIGATION

2.1.22 Use case MMD-ID-022

Original use case ID: Requirement 8, UHH.

AMSU-B (NOAA-15 - 17) and SSM/T-2 (DMSP-F11/F12/F14/F15), using all available overlaps, subset size 7 x 7 pixel AMSU-B, single pixel SSM/T-2, pixel delta <= 5km, time delta below 5 min, angle constraint |cos(VZA-AMSU) / cos(VZA-SSM) − 1| < ε1 (ε1 = 0.01), cloud filtering using the algorithm described in chapter 2.2.3.

Each matchup should contain information on

- the flight direction (north or south), derived from Scan Line Bit Field, bit 15 (northbound/southbound flag)
- the scan position (step number), derived from NAVIGATION

2.1.23 Use case MMD-ID-023

Original use case ID: Requirement 9, UHH.

MHS and HIRS/4 (MetOp-A/B and NOAA-18/19), using a SOBOL pseudo-random pixel selection generating approx. 60000 matchups per month, subset size 9 x 9 pixel HIRS, pixel delta <= 5km, time delta below 5 min, angle constraint $|\cos(\text{VZA-HIRS}) / \cos(\text{VZA-MHS}) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$)

Level 1 data of MHS and HIRS/4 from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Cloud screening algorithm according to the algorithms described in 2.2.2 and 2.2.4.

2.1.24 Use case MMD-ID-024

Original use case ID: Requirement 10, UHH.

AMSU-B and HIRS/3 (NOAA-15/16/17), using a SOBOL pseudo-random pixel selection generating approx. 60000 matchups per month, subset size 9 x 9 pixel HIRS, pixel delta <= 5km, time delta below 5 min, angle constraint $|\cos(\text{VZA- AMSU-B}) / \cos(\text{VZA- HIRS}) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$)

Level 1 data of AMSU-B and HIRS/3 from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Cloud screening algorithm according to the algorithms described in chapters 2.2.2 and 2.2.3.

2.1.25 Use case MMD-ID-025

Original use case ID: Requirement 11, UHH.

AMSU-B (NOAA-15/16/17) and GRUAN radiosondes, using all available overlaps, 5 x 5 AMSU-B pixel window, pixel delta <= 50 km, time delta below 30 min, angle constraint $|\cos(\text{VZA- AMSU-B}) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$), cloud filtering using the algorithm described in chapter 2.2.3.

Level 1 data of AMSU-B from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the flight direction (north or south), derived from Scan Line Bit Field, bit 15 (northbound/southbound flag)
- the scan position (step number), derived from NAVIGATION

2.1.26 Use case MMD-ID-026

Original use case ID: Requirement 11, UHH.

MHS (NOAA-18/19 and MetOp-A/B) and GRUAN radiosondes, using all available overlaps, 5 x 5 MHS pixel window, pixel delta <= 50 km, time delta below 30 min, angle constraint $|\cos(VZA\text{- } MHS) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$), cloud filtering as described in chapter 2.2.4.

Level 1 data of MHS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the surface (Ocean, Land or Mixed), derived from MHS NAVIGATION DATA AT SCANLINE
- the flight direction (north or south), derived from NAVIGATION data Velocity Z
- the scan position (step number), derived from MHS NAVIGATION DATA AT SCANLINE

2.1.27 Use case MMD-ID-027

Original use case ID: Requirement 11, UHH.

ATMS (SUOMI-NPP) and GRUAN radiosondes, using all available overlaps, 5 x 5 ATMS pixel window, pixel delta <= 50 km, time delta below 30 min, angle constraint $|\cos(VZA\text{- } ATMS) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$), cloud filtering using the algorithm described in chapter 2.2.3.

Level 1 data of ATMS from CLASS, to be processed with AAPP scripts atovin and convert_to_hdf5.

Each matchup should contain information on

- the flight direction (north or south), derived from the UTC time when the satellite crosses the equator going from south to north (UTCDateTimeOn-Equator) in NavigationRecord Group (see Metadata for GPM Products) and the time of the scan in the ScanTime Group (see File Specification for GPM Products).

2.1.28 Use case MMD-ID-028

Original use case ID: Requirement 11, UHH.

SSMIS (DMSP-F16-F19) and GRUAN radiosondes, using all available overlaps, 7 x 7 SSMIS pixels window, pixel delta <= 50 km, time delta below 30 min, angle constraint $|\cos(VZA\text{- } SSMIS) - 1| < \varepsilon 1$ ($\varepsilon 1 = 0.01$), cloud filtering as described in chapter 2.2.4.

Each matchup should contain information on the surface (Ocean, Land or Mixed), the flight direction (north or south), and the scan position (step number). Fundamental Climate Data Record of SSMI/SSMIS Brightness Temperatures: Feedhorn data group contains the FOV surface type (sft), data group ‘platform’ contains the angular position of the spacecraft measured along its orbit (ecliptic), and global coordinate variables contain FOV across track position (across_track).

2.1.29 Use case MMD-ID-029

Original use case ID: Requirement 11, UHH.

SSM/T-2 (DMSP-F11/F12/F14) and GRUAN radiosondes, using all available overlaps, 3 x 3 SSM/T-2 pixel window, pixel delta <= 50 km, time delta below 30 min, angle constraint |cos(VZA- SSMIS) − 1| < ε1 (ε1 = 0.01), cloud filtering using the algorithm described in chapter 2.2.3.

Each matchup should contain information on the flight direction (north or south), and the scan position (step number). The scene data group is contained in bytes 133 – 468 of the data record. Raw counts of channel m in Word n belong to beam position #((n-m-1)/6).

Flight direction: earth latitude and longitude: The earth location data group is contained in bytes 21 – 132 of the data record. The latitude – longitude pair in words (n, n+1) belongs to beam position #(n/2). If the beam moves westwards within a SCAN, then the sensor moves northwards.

2.1.30 Use case MMD-ID-030

Original use case ID: Requirement 12, UHH.

Multi-sensor matchups between two microwave-sounders and GRUAN radiosondes, using all available overlaps, window size sensor dependent, pixel delta <= 50 km, time delta below 30 min, angle constraint |cos(VZA- SSMIS) − 1| < ε1 (ε1 = 0.01), cloud filtering using the algorithm described in chapters 2.2.3 and 2.2.4.

Each matchup should contain information on the surface (Ocean, Land or Mixed), the flight direction (north or south), and the scan position (step number). Use sensor specific algorithm to extract the values.

2.1.31 Use case MMD-ID-031

Original use case ID: Requirement 13, UHH.

The MMS software shall be able to use already generated matchup data as input to perform a matchup processing with an additional third sensor.

2.1.32 Use case MMD-ID-032

Original use case ID: Requirement 14, UHH.

The MMS software shall be able to add a bounding polygon as input parameter that allows a region-specific matchup processing.

2.1.33 Use case MMD-ID-033

Original use case ID: Requirement 17, UHH.

For polar orbiting sensors, the MMS software shall be able to generate matchup data that contain for a given sensor on a given platform overflights over (lon/lat) for time X and X+12h.

2.1.34 Use case MMD-ID-034

Original use case ID: n.a., UoR.

Matchups between AVHRR and HIRS from the same platform, using all available platforms covering the common lifetime of both sensors, using a SOBOL pseudo-random pixel selection, generating approx. 50000 matchups per month. Subset window 75 x 41 AVHRR px size, subset size varying depending on viewing angle, AVHRR subset shall cover HIRS footprint + 5 km, subset window filled with "INVALID" outside of this area.

Only take into account where AVHRR data passes the homogeneity filter constraint

$|BT_{max} – BT_{min}|$ <= threshold

Where *threshold* is configurable and shall default to 2 K.

Post processing to include NWP profiles with the diurnal case time range (-96, +48 hours). Post processing to include AVHRR calibration data.

2.2 Cloud Screening Algorithms

This section summarizes cloud screening algorithm implementations as far as these have been defined yet. Cloud screening will be implemented as one of the several possible matchup screening algorithm available in the MMS.

2.2.1 Cloud screening algorithms for IASI

2.2.1.1 Space Contrast Test[1]

The brightness temperature in a window channel, 3.7µm, of one IFOV is compared with mean brightness temperature of the four IFOVs in the EFOV.

The pixel *i* is flagged "clear", if

$$T_{bEFOV}^{3.7} - T_{bIFOV,i}^{3.7} < t$$

[1] *This test is carried out among others in the level 2 processing for IASI. It has been optimized in the IAVISA project, but - despite some improvement in certain aspects - the number of clear scenes declared as cloudy increased with the use of optimized components. The thresholds defined above are the old ones. Unfortunately, the identification numbers of the used channels for "3.7µm" are not indicated in the IAVISA report. It is stated that it is provided by Auxiliary Dataset IASI_L2_PGS_THR_HORCO.*

with t being a threshold depending on the scene: t = 2.9137K, 4.35035K, 4.35035K for sea, land and mixed scenes respectively. It is flagged "cloudy" otherwise.

In case of unusable data: If one pixel (IFOV) in the EFOV is not usable, the test can be carried out but the result for the IFOV i must be flagged with a warning that less data has been used for the mean due to unusable pixels.

2.2.1.2 Interchannel Test

The interchannel test compares the brightness temperatures in a window and a water vapour sensitive channel. Thus, the pixel *i* is declared "cloudy" if

$$\left|T_{b,i}^{11.1} - T_{b,i}^{6.5}\right| < 25K$$

and "undecided" otherwise (according to RD 6).

The IASI channels used for "11.1µm" and "6.5µm" could be chosen such as to mimic a HIRS measurement using the appropriate SRF.

Suggested procedure: The matchup is carried out first. Having found the matching pixels, one goes back into the original data of the infrared sensor to look for the neighbouring pixels (i.e. the EFOV corresponding to the matched IFOV), moreover one extracts the brightness temperatures of the channels at 11.1µm and 6.5µm. Then the space contrast cloud test and the interchannel test are applied. Afterwards, the matched pixel is flagged according to the results of the tests, such that the matchup dataset contains the matched pixel with two cloud information flags: space test result, interchannel test result. Non-clear pixels should **not be deleted** from the data.

Taking into account further knowledge about scene, daytime, season etc. the user might then combine the two flags in an intelligent way in order to declare the pixel cloudy or clear.

2.2.2 Cloud screening for HIRS

2.2.2.1 Space Contrast Test

The brightness temperature in the window channel 11.1µm is observed in a spatial domain *D* of *n* x *n* pixels with *n* depending on the scene. The threshold Δ_1 also depends on the scene:

n= 45, 9, 9 for sea, ice-covered water and land respectively, corresponding to 450km x 450km, 90km x 90km, 90km x 90km.

Δ_1= 3.5K, 6.5K, 6.5K for sea, ice-covered water and land respectively.

The pixel $i \in D$ is classified "cloudy ", if $(T_{bMAX}^{11.1} - \Delta_1) > T_{b,i}^{11.1}$

and "undecided" otherwise. $T_{bMAX}^{11.1}$ is the maximum brightness temperature in *D*.

In case of unusable data: We allow 1% of the pixels in *D* to be unusable, i.e. 20 pixels in the 45 x 45 case and 1 pixel in the 9 x 9 pixel case. The test is carried out but the result is flagged with a warning that less data has been used due to unusable pixels.

2.2.2.2 Interchannel Test

The interchannel test compares the brightness temperatures in a window and a water vapour sensitive channel. Thus, the pixel *i* is declared "cloudy" if

$$\left|T_{b,i}^{11.1} - T_{b,i}^{6.5}\right| < 25K$$

and "undecided" otherwise (according to RD 6).

Suggested procedure: The matchup is carried out first. Having found the matching pixels, one goes back into the original data of the infrared sensor to look for the neighbouring pixels (i.e. the domain *D*; considering the matched pixel as the centred one of *D*), moreover one extracts the brightness temperatures of the channels at 11.1µm and 6.5µm. Then the space contrast cloud test and the interchannel test are applied. Afterwards, the matched pixel is flagged according to the results of the tests, such that the matchup dataset contains the matched pixel with two cloud information flags: space test result, interchannel test result. Non-clear pixels should **not be deleted** from the data. Taking into account further knowledge about scene, daytime, season etc. the user might then combine the two flags in an intelligent way in order to declare the pixel cloudy or clear.

2.2.3 Cloud Screening for AMSU-B

This algorithm is based on [RD 9] and uses the channels 18 and 20 with the following characteristics:

Ch18: 183.31 $\pm$1 GHz

Ch20: 183.31 $\pm$7 GHz

The pixel is declared cloudy, if at least one of the following criteria holds:

1) $\Delta T_B = T_B^{ch20} - T_B^{ch18} < 0$

2) $T_B^{ch18} < 240\ K$ (for nadir; see table below for off-nadir view thresholds)

Table 2: Viewing angle dependent thresholds

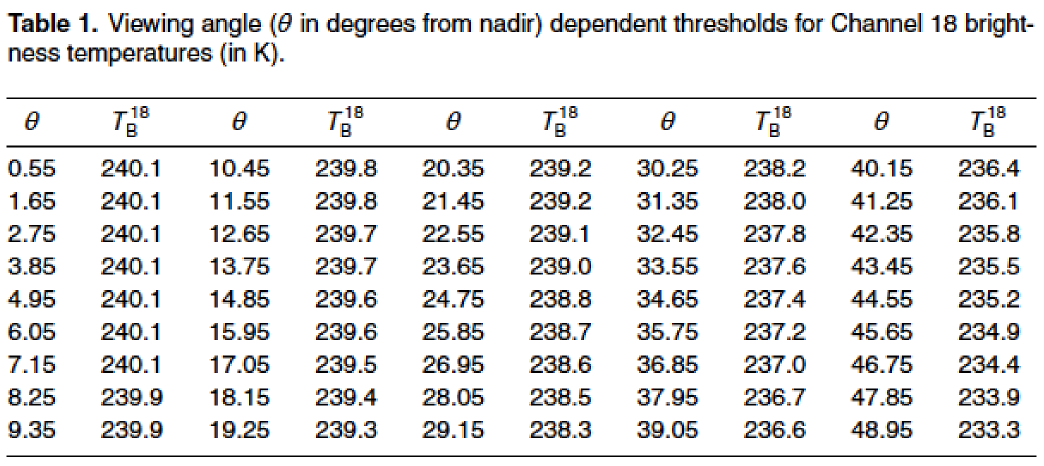

Table 1. Viewing angle (θ in degrees from nadir) dependent thresholds for Channel 18 brightness temperatures (in K).

θ	T_B^{18}	θ	T_B^{18}	θ	T_B^{18}	θ	T_B^{18}	θ	T_B^{18}
0.55	240.1	10.45	239.8	20.35	239.2	30.25	238.2	40.15	236.4
1.65	240.1	11.55	239.8	21.45	239.2	31.35	238.0	41.25	236.1
2.75	240.1	12.65	239.7	22.55	239.1	32.45	237.8	42.35	235.8
3.85	240.1	13.75	239.7	23.65	239.0	33.55	237.6	43.45	235.5
4.95	240.1	14.85	239.6	24.75	238.8	34.65	237.4	44.55	235.2
6.05	240.1	15.95	239.6	25.85	238.7	35.75	237.2	45.65	234.9
7.15	240.1	17.05	239.5	26.95	238.6	36.85	237.0	46.75	234.4
8.25	239.9	18.15	239.4	28.05	238.5	37.95	236.7	47.85	233.9
9.35	239.9	19.25	239.3	29.15	238.3	39.05	236.6	48.95	233.3

This algorithm should also be applied for ATMS and SSMT2 as they have equal channels.

2.2.4 Cloud Screening for MHS

This algorithm is based on [RD 9] and [RD 10] and uses the channels 3 and 4 with the following characteristics:

Ch3: 183.31 $\pm$1 GHz (corresponds to AMSU-B Ch18)

Ch4: 183.31 $\pm$3GHz

The pixel is declared cloudy, if at least one of the following criteria holds:

1) $\Delta T_B = T_B^{ch4} - T_B^{ch3} < 0$

2) $T_B^{ch3} < 240\ K$ (for nadir; see Table 2 above for off-nadir view thresholds)

This algorithm should also be applied for SSMIS as they have equal channels.

2.3 Data Requirements

This chapter lists the data requirements as defined by the use cases.

2.3.1 Satellite Sensor Data

Sensor	Platform	Level	Period	Format	Use case(s)
ATSR-1	ERS-1	L1C	1991-08 – 2000-03	ENVISAT-N1	MMD-ID-001
ATSR-2	ERS-2	L1C	1995-05 – 2011-08	ENVISAT-N1	MMD-ID-001
AATSR	ENVISAT	L1C	2002-04 – 2012-04	ENVISAT-N1	MMD-ID-001
AVHRR-LAC	NOAA-10	L1B	1991-07 – 1991-09	NetCDF	MMD-ID-004

AVHRR-LAC	NOAA-11	L1B	1991-07 – 1994-10	NetCDF	MMD-ID-004
AVHRR-LAC	NOAA-12	L1B	1991-09 – 2001-03	NetCDF	MMD-ID-004
AVHRR-LAC	NOAA-14	L1B	1995-01 – 2001-11	NetCDF	MMD-ID-004
AVHRR-LAC	NOAA-15	L1B	1998-07 – 2000-07	NetCDF	MMD-ID-004, MMD-ID-034
AVHRR-LAC	NOAA-16	L1B	2001-02 – 2012-04	NetCDF	MMD-ID-004
AVHRR-LAC	NOAA-17	L1B	2002-09 – 2012-04	NetCDF	MMD-ID-004, MMD-ID-034
AVHRR-LAC	NOAA-18	L1B	2005-07 – 2012-04	NetCDF	MMD-ID-004, MMD-ID-034
AVHRR-LAC	NOAA-19	L1B	2009-04 – 2012-04	NetCDF	MMD-ID-004, MMD-ID-034
AVHRR-LAC	MetOP-A	L1B	2006-10 – today	NetCDF	MMD-ID-034
AVHRR-GAC	NOAA-10	L1B	1988-11 – 1991-09	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-11	L1B	1988-11 – 1994-12	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-12	L1B	1991-09 – 1998-12	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-14	L1B	1995-01 – 2002-10	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-15	L1B	1998-10 – today	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-16	L1B	2001-02 – today	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-17	L1B	2002-08 – today	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-18	L1B	2005-07 – today	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-GAC	NOAA-19	L1B	2009-04 – today	NOAA	MMD-ID-001, MMD-ID-002
AVHRR-LAC	NOAA-16, 17-19	L2	2002-01 – 2012-12	L1B input: re-use AVHRR-for-SST + DLR-internal (TIMELINE project) L2 output: netCDF	MMD-ID-012, MMD-ID-013, MMD-ID-014
IASI	MetOp-A	TBD	2006-10 – today	HDF5	MMD-ID-003, MMD-ID-006
IASI	MetOp-B	TBD	2012-09 – today	HDF5	MMD-ID-003
IASI	MetOp-A	L1	2006-10 – today	HDF5	MMD-ID-009, MMD-ID-015
IASI	MetOp-B	L1	2012-09 – today	HDF5	MMD-ID-010, MMD-ID-015
HIRS/2	TIROS-N	L1B	1978-10 – 1980-02	HDF5	MMD-ID-005
HIRS/2	NOAA-6	L1B	1979-06 – 1986-11	HDF5	MMD-ID-005
HIRS/2	NOAA-7	L1B	1981-07 – 1985-02	HDF5	MMD-ID-005
HIRS/2	NOAA-8	L1B	1983-04 – 1985-12	HDF5	MMD-ID-005
HIRS/2	NOAA-9	L1B	1984-12 – 1998-02	HDF5	MMD-ID-005
HIRS/2	NOAA-10	L1B	1986-09 – 2001-08	HDF5	MMD-ID-005
HIRS/2	NOAA-11	L1B	1988-09 – 2004-06	HDF5	MMD-ID-005
HIRS/2	NOAA-12	L1B	1991-05 – 2007-08	HDF5	MMD-ID-005
HIRS/2	NOAA-14	L1B	1994-12 – 2007-05	HDF5	MMD-ID-005
HIRS/3	NOAA-15	L1B	1998-05 – 2009-06	HDF5	MMD-ID-005, MMD-ID-024, MMD-ID-034
HIRS/3	NOAA-16	L1B	2000-09 – 2014-06	HDF5	MMD-ID-005, MMD-ID-024
HIRS/3	NOAA-17	L1B	2002-06 – 2013-04	HDF5	MMD-ID-005, MMD-ID-006, MMD-ID-024, MMD-ID-034
HIRS/4	NOAA-18	L1B	2005-05 – 2009-05	HDF5	MMD-ID-005, MMD-ID-006, MMD-ID-023, MMD-ID-034
HIRS/4	NOAA-19	L1B	2009-02 – 2013-02	HDF5	MMD-ID-005, MMD-ID-006,

					MMD-ID-023, MMD-ID-034
HIRS/4	MetOp-A	L1B	2006-10 – today	HDF5	MMD-ID-005, MMD-ID-023, MMD-ID-034
HIRS/4	MetOp-B	L1B	2012-09 – today	HDF5	MMD-ID-005, MMD-ID-023
AIRS	AQUA	L1B	2005-05 – today	HDF-EOS	MMD-ID-006
MVIRI	METEOSAT7	L2	2002-01 – 2012-12	netCDF lv2	MMD-ID-013
MHS	NOAA-18	L1B	2005-05 – today	HDF5	MMD-ID-015, MMD-ID-016, MMD-ID-017, MMD-ID-019, MMD-ID-020, MMD-ID-023, MMD-ID-026, MMD-ID-030
MHS	NOAA-19	L1B	2009-02 – today	HDF5	MMD-ID-015, MMD-ID-016, MMD-ID-017, MMD-ID-019, MMD-ID-020, MMD-ID-023, MMD-ID-026, MMD-ID-030
MHS	MetOp-A	L1B	2006-10 – today	HDF5	MMD-ID-015, MMD-ID-016, MMD-ID-017, MMD-ID-018, MMD-ID-019, MMD-ID-020, MMD-ID-023, MMD-ID-026, MMD-ID-030
MHS	MetOp-B	L1B	2012-09 – today	HDF5	MMD-ID-015, MMD-ID-016, MMD-ID-017, MMD-ID-018, MMD-ID-019, MMD-ID-020, MMD-ID-023, MMD-ID-026, MMD-ID-030
AMSU-B	NOAA-15	L1B	1998-05 – 2011-03	HDF5	MMD-ID-017, MMD-ID-021, MMD-ID-022, MMD-ID-024, MMD-ID-025, MMD-ID-030
AMSU-B	NOAA-16	L1B	2000-09 – 2014-06	HDF5	MMD-ID-017, MMD-ID-021, MMD-ID-022, MMD-ID-024, MMD-ID-025, MMD-ID-030
AMSU-B	NOAA-17	L1B	2002-06 – 2013-04	HDF5	MMD-ID-017, MMD-ID-021, MMD-ID-022, MMD-ID-024, MMD-ID-025, MMD-ID-030
SEVIRI	METEOSAT-8	L1	2002-08 – today	HDF5	MMD-ID-018
SEVIRI	METEOSAT-9	L1	2005-12 – today	HDF5	MMD-ID-018
SEVIRI	METEOSAT-10	L1	2012-07 – today	HDF5	MMD-ID-018
ATMS	SUOMI-NPP	L1	2011-10 – today	HDF5	MMD-ID-019, MMD-ID-027, MMD-ID-030
SSMIS	DMSP-F16	L1	2003-11 – today	NetCDF	MMD-ID-020, MMD-ID-028, MMD-ID-030
SSMIS	DMSP-F17	L1	2006-11 – today	NetCDF	MMD-ID-020, MMD-ID-028, MMD-ID-030
SSMIS	DMSP-F18	L1	2009-10 – today	NetCDF	MMD-ID-020, MMD-ID-028, MMD-ID-030
SSMIS	DMSP-F19	L1	2014-04 – today	NetCDF	MMD-ID-020, MMD-ID-028, MMD-ID-030
SSM/T-2	DMSP-F11	L1	2000-01 – 2000-08	TBD	MMD-ID-022, MMD-ID-029, MMD-ID-030

SSM/T-2	DMSP-F12	L1	2000-01 – 2001-09	TBD	MMD-ID-022, MMD-ID-029, MMD-ID-030
SSM/T-2	DMSP-F14	L1	2000-01 – 2001-09	TBD	MMD-ID-022, MMD-ID-029, MMD-ID-030

2.3.2 Insitu Data

Sensor	Originator	Period	Format	Use case(s)
TBD	TBD	TBD	TBD	MMD-ID-004
Radiosonde	TBD	TBD	TBD	MMD-ID-011
Aeronet	Aeronet	2002-01 – 2012-12	ASCII	MMD-ID-014
Radiosonde	Gruan/DWD	2011-01 - today	NetCDF	MMD-ID-026, MMD-ID-026, MMD-ID-027, MMD-ID-028, MMD-ID-029, MMD-ID-030

3 System Design

This chapter specifies the overall Fiduceo Matchup Generation System and its components using a general overview to visualize the components and their interaction. A more technical description of the system is presented in chapter 0.

The functionality of the MMS can be separated into three distinct functional phases:

- Phase I: extraction of input metadata from the various sources (satellite and in-situ), preparation of the metadata and storage of these into the Metadata Storage
- Phase II: detection of possible matchups based on the metadata, further processing on the result list (e.g. overlap-removal, cloud clearing, etc.) and MMD generation.
- Phase III: post-processing steps on already generated MMDs

Phase I operations are conducted only in the initial phase of the project or when updates to the input dataset are requested, e.g. the availability of a re-processed satellite data record or the integration of a new in-situ dataset.

Phase II operations cover the main functionality of the MMS and are expected to be executed many times during the project life-time.

Phase III operations are performed after the generation of the matchup dataset to enhance the information content of the MMD, e.g. by adding weather prediction data or other additional information not contained in the sensor data comprising the MMD.

The functionality required for each of the three phases is implemented in three separate executable programs,

- Ingestion Tool for phase I operations,
- Matchup Tool for phase II operations and
- Post Processing Tool for phase III operations.

The Ingestion and Matchup Tool share a number of common system components and especially a common data model. The data model serves to contain matchup information in a suitable way for the processing requirements of the MMS. Other system components common to both tools are

- the Data Storage
- the Metadata Storage and
- the Data Reader subsystem.

The Ingestion Tool iterates over a directory structure in the archive to open satellite or insitu data files and extract the metadata required for matchup processing. The tool is invoked using a “sensor-platform” key (see 3.1) defining the data to ingest and a time range.

It uses the Data Reader subsystem to invoke the correct data reader, reads the metadata of each file into the internal data structures, performs a geometric pre-processing and finally transfers the information gathered to the Metadata Storage system. A schematic of these steps is displayed in Figure 3-1 below:

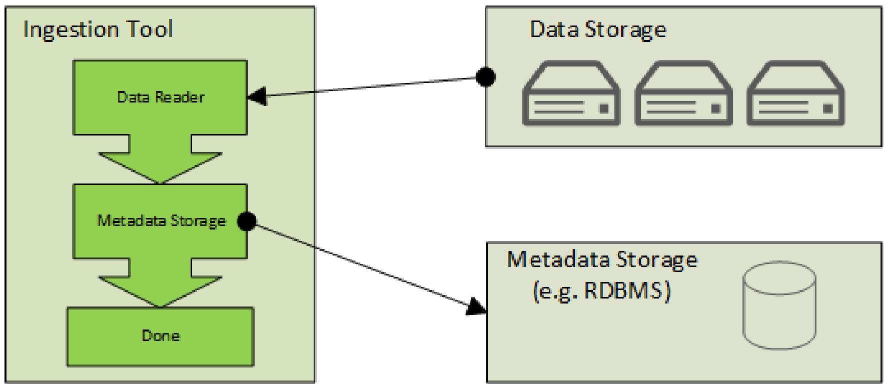

Figure 3-1: Ingestion Tool

The Matchup Tool performs the matchup processing based on the data stored in the Metadata Storage and the input parameters provided. In the Input System, the input parameters are transferred to data entities that are used for further processing. Input data can be consisting of

- random points
- in-situ data within a given time range or
- primary (and secondary, n-ary) satellite sensor data covering a given time range.

With the resulting data entities, the Matchup Tool issues a raw matchup-search on the Metadata Storage. These raw results are then filtered in the Intersection Engine, which basically removes all intersections that have a pixel time difference higher than the requested maximal time difference.

Following this, a number of further filters are applied to the matchup data, these are

- overlap removal (optional) which ensures that the MMD does not contain overlapping matchup-pixel windows
- Matchup-condition processing, e.g. checking for viewing angle or homogeneity constraints and finally
- Matchup-clearing processing, e.g. cloud screening or land/sea processing.

The remaining matchup points are considered as valid matchup-points, they are used as basis to generate the MMD. The Matchup Tool uses the Data Reader subsystem to extract all data requested for the MMD from the satellite or insitu data files. Using this input, the final MMD is assembled and written to the archive.

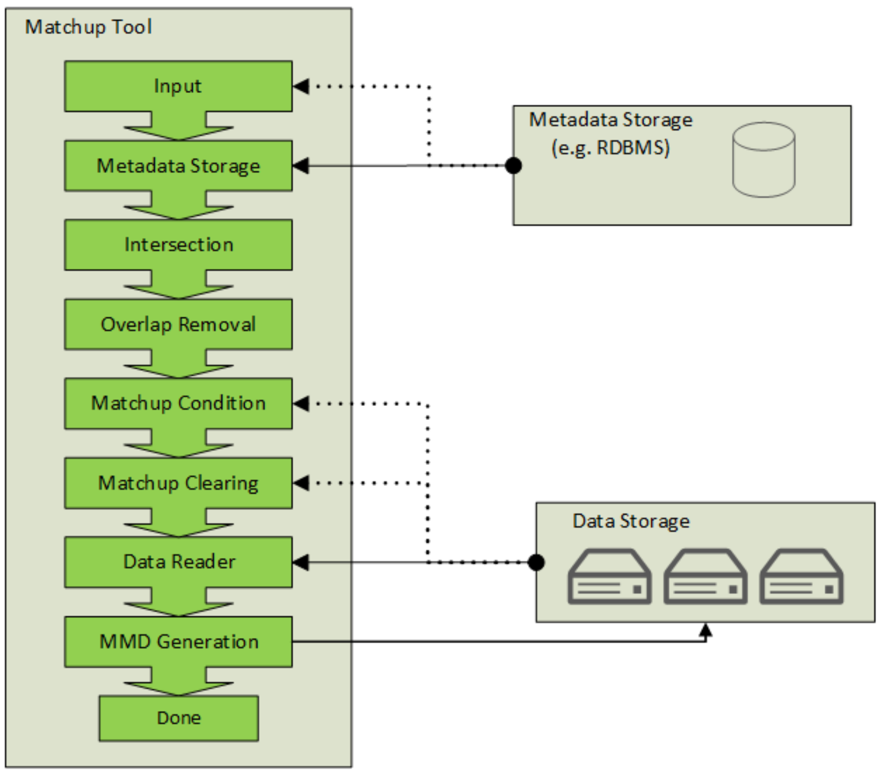

Figure 3-2: Matchup Tool

In a final step the Post Processing Tool can optionally be used to enhance the content of the MMD files generated by the preceding processing stage. This final processing stage solely operates on the Data Storage, allowing to apply a chained set of post-processing plugins.

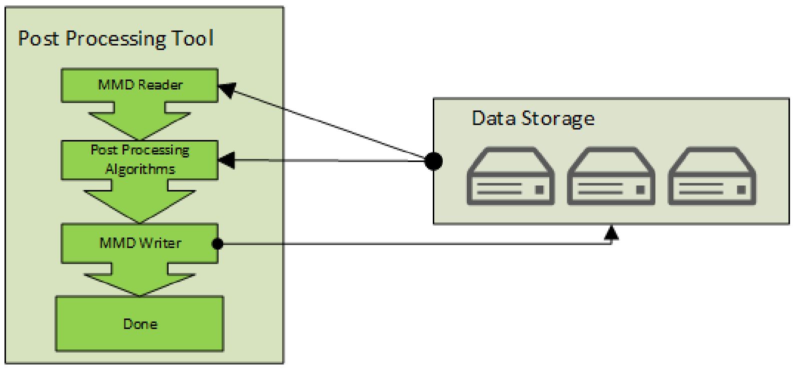

Figure 3-3: Post Processing Tool

When using a distributed processing environment like CEMS, an additional component used is a Workflow Engine that serves as intelligent interface to the parallel processing facility. This component can be seen as being "on top" of both tools as it uses the tools and ensures that a complex job (e.g. a sensor/sensor matchup covering several years) is subdivided into smaller processing steps that are executed and monitored in parallel.

A detailed description of the tools and the components used by the tools and their functionality is presented in the following chapters in a more generic way. Detailed technical specifications are presented in chapter 0.

3.1 Data storage

A large physical storage is required to complete all matchup datasets as defined by the use-cases supplied. The data storage is the central location for input and output data:

- Satellite data
- Insitu data
- MMS intermediate and output data

The organisation on this large dataset will use a common scheme that allows accessing all data sources in a common manner. This structure is a directory tree starting on a common archive-root directory and sub-directories organised by “sensor-platform”, “version”, “year”, “month” and optionally “day”:

[archive-root]/[sensor-platform]/[version]/[year]/[month]/[day]

The “sensor-platform” part of the path descriptor shall follow a dedicated naming convention in the MMS context that allows to uniquely identify satellite/insitu data collections. For example: identification of the three ATSR sensors is achieved by defining:

- atsr-e1 – for ATSR on ERS1
- atsr-e2 – for ATSR on ERS2
- atsr-en – for AATSR on ENVISAT

This pattern will be applied to all input data sources. The MMS intermediate and final data will follow a similar approach, tailored to the requirements of the Matchup Tool.

On the production system CEMS, this structure must not necessarily reflect the physical storage organisation of the underlying file system. For simplification of the MMS IO/subsystem, this structure shall be visible to the MMS, but can be consisting of a virtual directory structure composed of a system of operating-system level directory links.

3.2 Metadata Storage

A central metadata storage will be used to contain the metadata of satellite observations and in-situ measurements for all missions and sources used in the project. This resource will be a central data storage that is available to all components of the system.

For an optimum performance the satellite observations stored in the database shall be reduced to the absolute minimum of data that is required to perform a matchup search. The satellite observation table will contain:

- Sensor (and platform)
- Sensing Start
- Sensing Stop
- Data bounding polygon (geolocation)
- Ascending/Descending node flag
- Information on the acquisition time coding
- File path into the data archive.

The table for insitu data will be using a similar structure, omitting the satellite specific fields and using a simpler geometry.

The metadata storage exposes two interfaces to the system, one for data ingestion and one for data retrieval. While the ingestion interface is relatively small and incorporates not much complex functionality, the retrieval interface and the underlying functionality is critical to the overall system performance.

The retrieval interface of the Metadata Storage implements a coarse pre-selection of the possible matchups for a given time interval, geometry, sensor and node information, reducing the number of potential results from some millions to a significantly smaller number that will be processed further.

3.3 Processing components

This chapter lists all software components that are used by the tools.

3.3.1 Data Reader Subsystem

One challenge in implementing the MMS is the integration of a large number of satellite and insitu data sources. Access to the various data files is required for two purposes:

- Extraction of metadata and geo-boundary during the ingestion
- Extraction of measurement data at various stages during the matchup processing

The various reader implementations are shielded behind a reader interface to decouple classes using readers from the specific implementations. Reader classes are instantiated using a *ReaderFactory* that maintains a collection of all *Reader* classes available for the MMS.

Readers are uniquely identified by their “sensor-platform” keys as described in chapter 3.1. Each reader implementation provides one or more regular expressions that allow classes using the reader to uniquely identify associated data files in the archive directory structure.

Extensibility to new sensors is achieved by simply adding a new *Reader* class to the library of readers and register that class in the *ReaderFactory*.

3.3.2 Input System

The Input System is responsible for the preparation of the input data for the matchup process. It translates the string representation passed in by the configuration into lists of possible matchup locations.

Input sources for the matchup processing are one of

- Satellite observations
- Insitu observations
- Random points

The Input System loads the input module requested by the parameter and invokes the specified input class with the time range and types requested. The result is

- A list of geometries with associated time axes in the case of satellite and line-type insitu data
- A list of geo-locations with associated timestamps for random sources and point-type insitu data

With these entities prepared, the matchup processing can execute the further steps required.

3.3.3 Matchup Intersection Engine

The Matchup Intersection Engine is supplying the core functionality of the matchup processing. It receives the prepared input data and performs a two-step matching with these data. In the first step, it uses the Metadata Storage to retrieve a raw pre-selection of possible matchups for the given input data. A second step refines the result set by applying exact geometric intersections associated with a time analysis.

The intersection engine can perform geometric intersections based on all requested geometries, i.e.

- Polygon/polygon
- Polygon/line-string
- Polygon/point

When a successful geometric intersection has been detected, i.e. a common geometric location/area has been detected from two observations, it performs an additional timing analysis of the resulting intersection geometry. This timing analysis returns either a common time interval (where both sensors have observed the common geometry at the same time) or a minimal possible time delta between the observations.

The matchup intersection algorithm will use a configurable constraint on the maximally tolerated acquisition time difference passed in as a processing parameter. All matchup-geometry timing information will be compared to this maximal time delta. All possible matchups with a larger time difference will be removed from the matchup list.

The intersection engine accepts as optional additional parameter the definition of a time-offset that is applied to the time-delta parameter. This parameter allows to process matchups of type “same place, twelve hours later” that are required to observe the diurnal variability behaviour.

3.3.4 Overlap Removal

Most of the requirements on matchup data request that a certain window environment around the matchup pixel location is being stored along with the exact matchup location. Common window sizes vary from 3-by-3 up to 11-by-11 pixels.

When applying the window extension to all matchup center pixels, overlapping windows can be introduced. This may not be wanted for statistical analysis as it introduces high data correlation in the analysis.

The optional Overlap Removal Module ensures that all overlaps are removed from the matchup-list, avoiding duplicate measurements to appear in later stages of the processing.

3.3.5 Matchup constraints

A number of constraints on a possible matchup can be defined. These can be of general definition or sensor specific. The Matchup Constraints processing, in contrast to the Matchup Clearing as described in the next chapter, is working solely on the data of the satellite acquisition processed.

This step in the matchup processing shall ensure that only matchups passing all constraints/quality tests will be stored in the resulting MMD. The Matchup Constraints defined below are useful examples derived from the use cases, the constraints processing system will allow integration of new algorithms as well as chaining of constraints.

3.3.5.1 Viewing Geometry Constraints

When processing dual satellite sensor matchups, some use-cases require that the difference of the azimuth angles is below a (parametrized) threshold. Also, for satellite/insitu matchups, constraints on the range of allowed viewing angles can be defined.

3.3.5.2 Homogeneity Constraints

Some use cases require the matchup data to be contained in an area that exhibits only minor variations in the data. Homogeneity constraints can be used to define the evenness of the data by calculating the data mean value for the compete window and rejecting the matchup if one data value differs more than a given threshold value from the mean. This processor can be configured to process multiple variables at once, using per variable specific thresholds.

3.3.5.3 Cloud Constraints

Simple cloud screening algorithms can also be part of the Matchup Constraints processing chain, like the ones described for use case MMD-17 in chapter 2.1.17. Each sensor will have a specific algorithm for simple cloud screening. More complex algorithms will be part of the Matchup Clearing stage.

3.3.5.4 Flag Constraints

Simple flag screening that allows to exclude pixels that are flagged in a way that is not desired to be contained in the MMD. This also is a sensor specific set of constraints.

3.3.6 Matchup Clearing

The Matchup Clearing stage serves to remove matchups from the list of possible results that do not conform to requested requirements. The distinction between the Matchup Constraints processing and the Matchup Clearing is introduced to conceptually separate simple algorithms (operating on the pixels of the input data) from more complex and hence more time-consuming algorithms. The simple (and therefore fast) algorithms should be executed first to reduce the list of possible matchups before the complex algorithms are applied.

This very general clearing stage may include screenings like

- Validation against external resources, e.g. screening for sea-ice from ECMWF data,
- Validation for land/sea coverage,
- Complex cloud screening algorithms, e.g. as defined in chapter 2.2.1.

Each example listed above is defined as a single “clearing rule” that is either a sensor-specific specific rule (e.g. a complex cloud screening algorithm) or a general rule that can be applied for all matchup candidates (e.g. land/sea distinction using external mask data).
The Matchup Clearing stage is designed to allow multiple rules to be chained.

3.3.7 MMD Generation

All pixels/matchups remaining after the filtering stages can be considered as valid and shall comprise the desired result. The MMD Generation module assembles all data requested to be contained in the MMD and writes the result to the MMD. The content of the MMD is completely configurable in terms of dimensions, variables per sensor and (optional) data or format conversions applied during the MMD writing

The file format of the MMD is similar to the one used for SST CCI (RD 1). One MMD file per time interval processed is generated and stored to the appropriate location in the data archive.

3.4 Ingestion Tool

The purpose of the Ingestion Tool is to initially populate the Metadata Storage with he required meta-information about the input datasets available for matchup processing.

The inputs to the tool consist of

- The data archive base directory
- A “sensor-platform” identifier and

- A processing time range

The tool identifies all data files in the archive associated with the requested sensor-platform and an acquisition time intersecting the given time-interval. It iterates over all files, extracts the meta-information and performs a pre-processing on the geometry information. Finally it stores the results into the Metadata Storage.

The Ingestion Tool is designed to be operated in parallel to allow a quick population of the input Metadata Storage with all required data and to be able to update the storage when new sensors or reprocessed versions of currently stored sensor datasets become available.

3.5 Matchup Tool

The generation of the MMD is executed by the Matchup Tool. This software module binds all components listed in chapter 3.3 into a logical structure that can be executed.

It is invoked for a specific use-case which defines the input data and the associated time ranges. Given this information and additional configuration files, the tool generates the requested MMDs and stores the data into the archive.

The Matchup tool is designed to execute a configured number of sub-tasks in one execution. This allows the user either to execute all associated sub-tasks in one run or to split up the overall execution into separate sub-tasks that are executed one after the other. The first approach is suited for desktop processing where intermediate results and parallelism are not of primary interest. The latter approach is best suited for parallel execution environments or when the intermediate results are of specific interest (e.g. to fine-tune a cloud screening algorithm).

3.6 Post Processing Tool

A final and optional step in the processing chain is executed by the Post Processing Tool. This processing step serves to integrate additional data layers into the MMD structure that are not contained in the original sensor data sets.

The tool is invoked with a file set of MMD files, applying a configurable set of post processing plugins to the MMD. The results can be either integrated into the existing files or added to new MMD files generated from the input. The latter approach allows preserving the original data and applying the post processing algorithms many times, e.g. for algorithm tuning.

3.7 Workflow Engine

All software tools comprising the MMS software system are designed to operate as single executables on a specific task, defined by a matchup task and a time interval. As constrained by the available hardware resources, especially the amount of memory available, the processing of larger time intervals has to be broken down to e.g. monthly steps. A standard approach for realising a long-term processing assembled of smaller steps is to use the scripting capabilities of the operating system to execute the tasks in a serial chaining of processing task that operate on smaller time intervals.

Using this kind of scripting solution for bulk processing has the advantage that it is easy to use, easy to implement, and benefits from the power of a scripting language. But there are also substantial disadvantages, e.g.:

- Only sequential execution of tasks
- Missing capability to interrupt and resume
- Missing progress and status information

The MMS overcomes these shortcomings by introducing a top-level layer that is used on grid-processing environments like the one offered by CEMS.

The Fiduceo MMS will reuse a reliable, Python based software component for this task that has already been successfully used in SST-CC and QA4ECV processing tasks. PMonitor facilitates concurrent execution, resolving of inter-task dependencies, and monitoring of tasks in the background. It is perfectly suited to exploit the capabilities of a parallel processing environment as supplied by CEMS.

4 Software Implementation

4.1 Overview

The MMS software will be implemented using a small set of programming languages and third party libraries. All components of the system will be freely available open source licensed libraries.

The main functionality of the MMS is implemented using the Java language. This includes all IO/facilities, database connection handling and the core modules (Ingestion, MMS processing and Post Processing). This approach ensures that the MMS core components will be operating on all major operating systems without OS specific adaptations or custom build processes.

The implementation of the CEMS workflow control will be implemented using Python and “bash” scripts. The CEMS workflow system make use of the LSF (Load Sharing Facility) implemented at the CEMS High Performance Computing facility.

4.2 External Libraries

A number of third party libraries will be used for the MMS implementation to ensure a rapid development. It is ensured that all components and libraries of the MMS are open source and freely available to allow a maximum re-use and seamless deployment for all users. The third party components used are listed in Table 3 below.

Table 3: Third-party components used by the MMS

Library	License	Link
SNAP	GPL	http://step.esa.int/main/toolboxes/snap/
NetCDF Java	MIT-style UNIDATA license	http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/
JTS Topology Suite	LGPL	http://www.vividsolutions.com/jts/JTSHome.htm
Google S2	Apache license v. 2	https://code.google.com/p/s2-geometry-library-java/
Jackson JSON	Apache license v. 2	http://wiki.fasterxml.com/JacksonHome
MongoDB Java Driver	Apache license v. 2	https://www.mongodb.org/licensing
MongoDB Server	GNU AGPL 3.0	https://www.mongodb.org/licensing
MySQL JDBC	GPL	http://dev.mysql.com/downloads/connector/j/
PostgreSQL JDBC	BSD	https://jdbc.postgresql.org/index.html
Apache H2	MPL 2.0, EPL 1.0	http://www.h2database.com/html/main.html
Commons DBCP	Apache license v. 2	https://commons.apache.org/proper/commons-dbcp/
JUnit	EPL 1.0	http://junit.org/
Mockito	MIT	https://code.google.com/p/mockito/

4.3 Data Storage

There are no specific software designs associated with this system component. The IO implementations rely on the standard Java file-system interface and the NetCDF library supplied by Unidata.

The directory tree pattern defined in 3.1 is implemented in the software components, the structure iteration tasks and file operations are shielded in a File System component to allow easy adaptation if required.

4.4 Metadata Storage

The high parallelism aimed at the CEMS HPC processing puts relatively high expectations on the performance of the metadata storage when extracting potential intersection candidates from the storage. The software design therefore reflects a relatively open design that allows a quick change of the storage implementation to find an optimal solution for the processing environment, but also opens up possibilities to easily integrate experimental storage solutions without affecting the remaining parts of the MMS.

As further benefit, this open design allows in-depth testing on the storage functionality and all software components accessing the storage by opening an interface to inject well defined metadata storage classes during test execution.

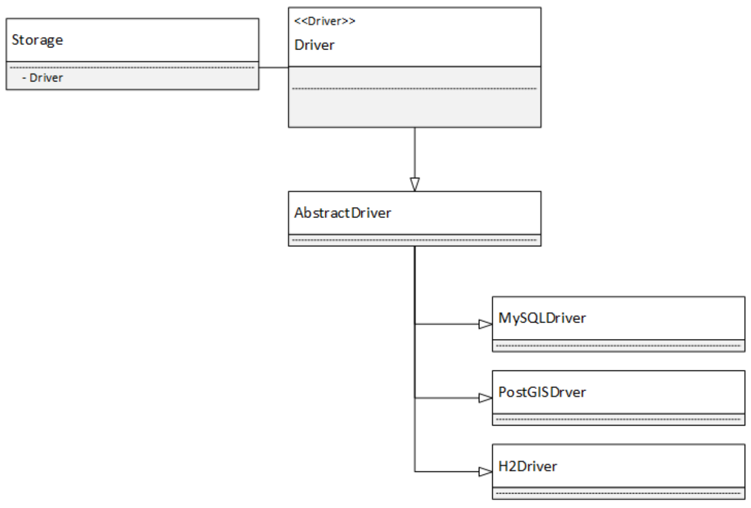

Figure 4-1:Metadata Storage class diagram

The metadata storage exposes a single class to the application, the *Storage*. This class dispatches most interactions to an *AbstractDriver* that is an abstract implementation of the *Driver* interface. Concrete implementations of a storage derive from the *AbstractDriver*.

During start-up operation, the Storage is instantiated passing in a parameter set consisting of

- Service URL
- JDBC driver class
- Username
- password

The *Storage* uses the service URL to identify the *AbstractDriver* implementation to be used. The specific driver class is then loaded dynamically using the Java SPI services framework and connects to the remote storage implementation.

During the connection operation, the driver class detects whether the remote service (e.g. the RDBMS) is prepared for usage by the driver and if not performs the driver specific initialisation tasks. When using a relational database as storage, this includes for example the creation of the database tables required and generation of indices. This initialisation stage is executed only once and only when necessary.

This design, in conjunction with the capabilities of the SPI framework, allows integrating new *Driver* implementations without recompilation of the MMS core by mere copying of an extension library to the classpath of the MMS where it is automatically detected and integrated.

Although the driver parameter set basically is reflecting the requirements of the JDBC interface, the usage of the parameter set is only defined by the implementation of the driver. The *Storage* only requires a unique URL pattern registered for a new driver implementation.

4.5 Data Reader Subsystem

A data reader is a software component that is used to read data from a satellite or insitu data file. There is always a one-to-one connection between the instantiation of a reader and a product file, i.e. one reader object is responsible for one data file.

Each satellite or insitu file format is supported by a specific reader class that is solely responsible for reading data of that type. All different data readers for the several formats and sensor inherit from a common *Reader* interface that defines the operations required on the reader class. All classes of the MMS that need to access data files operate solely on this *Reader* interface so that extensibility of the software is ensured by design.

The *Reader* interface offers the following methods

- Open: opens a file given a file location
- Close: closes the file again, releasing the operating system resources
- ReadAcquisitionInfo: reads the data acquisition information from the file. This set of data is required to uniquely identify the data acquisition in the Metadata Storage for later retrieval. Fields to be read are
    - Sensing Start and Stop times, as UTC dates
    - The bounding geometry as stored in the file, for satellite data this is either a polygon or a multi-polygon, for insitu data either a polygon enclosing all point acquisitions or a line-string for e.g. ship transects
    - The equator node type (ascending, descending or undefined)
    - (optional) Information on the time axes associated to the acquisition. This information is required for the Intersection engine.
    - The sensor and platform identifier for the data
- ReadVariableNames: extracts all variables contained in the file as String array

- ReadData: read the raw data contained in the file, given a location in either lon/lat or raster x/y. This functionality is required for the processing of the matchup constraints and the generation of the final MMD.

Each *Reader* implementation is accompanied by a *ReaderPlugin* class that is registered in the *ReaderFactory* (see paragraph below). The *ReaderPlugin* contains information about the sensor-platform supported by the Reader, a regular expression to determine the file name pattern associated with the reader and exposes a method to instantiate a Reader of the supported type. This approach ensure a minimal memory consumption and a fast software start-up.

Software requiring the use of a reader retrieve the appropriate reader class using a *ReaderFactory*. The *ReaderFactory* maintains all Reader classes in the system and loads the appropriate *ReaderPlugins* at start-up time. To retrieve a reader, the calling class calls *getReader()* on the factory, passing in the unique sensor-platform identifier for the data reader to be retrieved. The ReaderFactors searches the internal plugin registry for a plugin that supports the requested type, if found calls the *createReader()* method on the plugin and returns the *Reader* class. The following activation diagram spotlights the interaction of the classes:

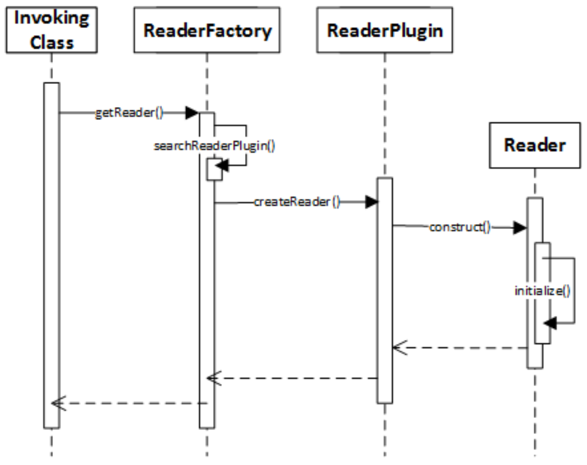

Figure 4-2: ReaderFactory activation diagram

4.6 Processing Components

4.6.1 Input System

The first step in the matchup processing chain is the generation of appropriate input data. The Input system loads/creates the appropriate input data for further matchup processing for a given set of input parameters. These are:

- Input type
    - Satellite data
    - Insitu data
    - Random points

- A time range and
- An (optional) bounding geometry.

The input type defines which input module is being invoked by the input system. For each input type, a specific implementation with an additional set of implementation specific parameters is invoked. The three input modules are described below.

When using satellite data as input, the satellite data module is invoked. It uses the Metadata Storage to request all satellite acquisitions for the requested sensor-platform within the time range (and bounding geometry) requested. It then generate the associated time axes information from the observation metadata received to prepare the dataset for processing with the intersection engine. The satellite data module receives the sensor-platform identifier of the satellite sensor requested as additional input parameter. The output of this module comprises a list of bounding (multi-) polygons with associated time axes.

The insitu data module is responsible for preparing the input data derived from insitu measurements. It accesses the Metadata Storage and requests all insitu acquisitions for the requested insitu sensor within the given time range (and bounding geometry). This module receives the insitu sensor identifier as additional input parameter. The output of this module comprises either a list of line-string geometries with associated time axes or a list of geo-points with an associated time stamp.

The Random point input module is based on a Sobol pseudo random sequence generator (see RD 7). It produces a pre-defined number of random points, equally distributed in space and time that serve as possible matchup locations. The generated list of points can optionally be filtered by

- Cloud probability, i.e. the density of points is increased in regions with a high possibility of clouds
- Land/Water pixels based on a pre-defined land/water mask

The random point input module receives the number of desired points (before filtering) as additional parameter. The output of this module comprises a list of geo-points with an associated time stamp.

4.6.2 Matchup Intersection Engine

One of the core algorithm of the matchup processing chain is implemented in the matchup intersection engine. It serves to split the large number of possible matchup locations detected by the Metadata Storage into those that fulfil both the constraints of the maximal geometric distance and the maximal time distance.

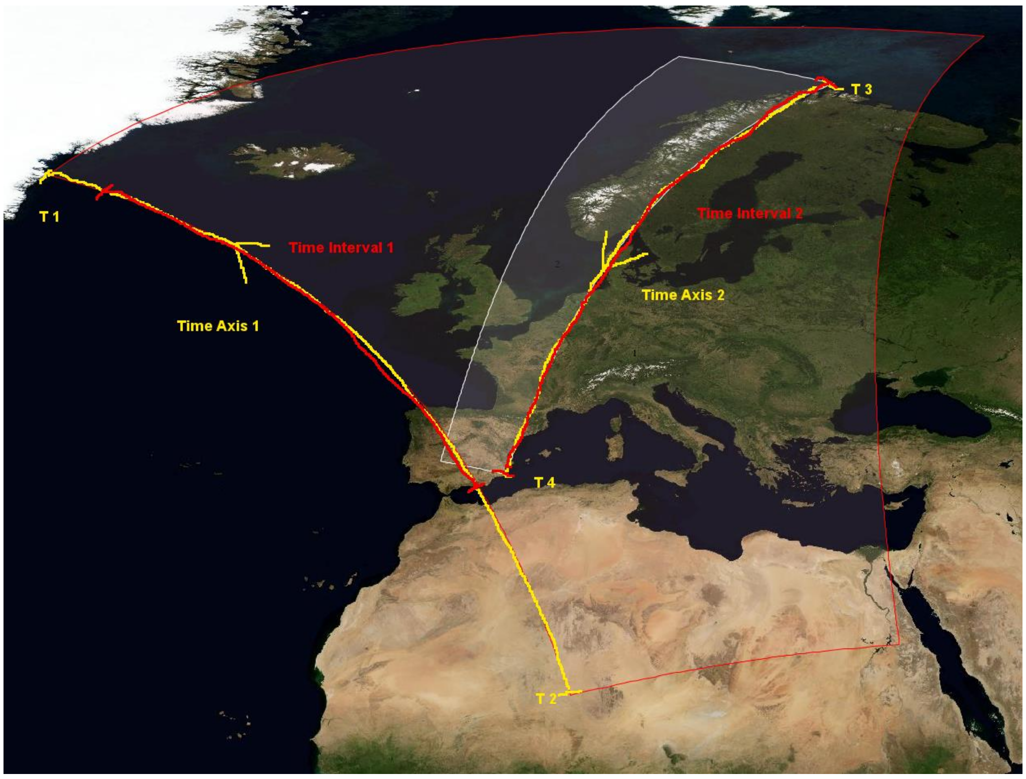

Figure 4-3: Matchup geometries and time-axes

The precise detection of intersection interval is schematically displayed in Figure 4-3. Each satellite data bounding geometry is associated with a time-axis that allows calculating the progression of the time for each associated data source over the complete acquisition geometry (Time Axis 1 and 2 in the figure).

The geometric intersection of the (multi-) polygons of the satellite acquisitions involved either is empty (which indicates that there is no geometric intersection) or it results in another polygon that is contained in both geometries. The projection of this intersection polygon onto both sensor time axes results in timing information of the satellite overpasses over the common area.

Given this timing information, the intersection engine can detect whether there is a common time interval where both sensors haven been observing the common area and if none present can at least determine the minimal time difference possible for the area. Comparing this time information with the constraint of maximal matchup time delta allows to precisely remove all “false” matchups before accessing the raw satellite data in the archive (which is a relatively time consuming task).

For intersections of satellite and insitu data, the situation is a bit simpler as the input data for insitu measurements either consist of point/time pairs (e.g. buoy measurements) or data that belongs to a transect with aligned data acquisition points (e.g. GRUAN data). For point acquisitions, the time information can simply be detected by projecting the point location onto the satellite time axis. When using line-string data, the proceeding is similar as in the satellite case, but simpler as the insitu data is already forming a time-axis. In this case, the geometric intersection results directly in two time-axes (one insitu and one for the satellite) that can be further processed by the intersection engine.

As a final step, the resulting list of possible matchups is ordered by primary input source, so that all subsequent processing steps can operate with a minimal number of disk accesses by processing all matchups of a single data file in one run.

With this file-ordered list, a last (optional) check is made to ensure that each matchup location with the surrounding window is covered completely by the satellite orbit file(s) associated with the matchup. All matchups where the surrounding x/y window is only partially within the orbit raster are rejected.

4.6.3 Overlap Removal

This step and all the following processing steps operate on a different coordinate system. While the processing up to this point has been performed only on geo-locations, the subsequent steps operate mainly on data raster x/y coordinates.

The overlap removal module will re-use an algorithm already successfully used in the SST-CCI MMS implementation. The removal algorithm operates on lists of matchup windows that are aligned by primary sensor data file. It iterates over all primary sensor data rasters, ensuring that only non-overlapping windows for each file raster remain after the removal stage.

In a first step, the algorithm identifies all sequences of overlapping windows in the list, i.e. chains of two or more overlapping windows that form a connected chain.

For each chain, the algorithm detects whether the chain contains branches. If so, it removes the windows at the branching locations to ensure that only simple, unbranched chains remain.

For each of the chain, the algorithm repeatedly removes the central window, splitting the chain into two smaller segments. This iteration stops, when each chain has been discomposed to single windows that hav no common overlap with other windows.

The algorithm is displayed schematically in Figure 4-4, without the branch removal algorithm.

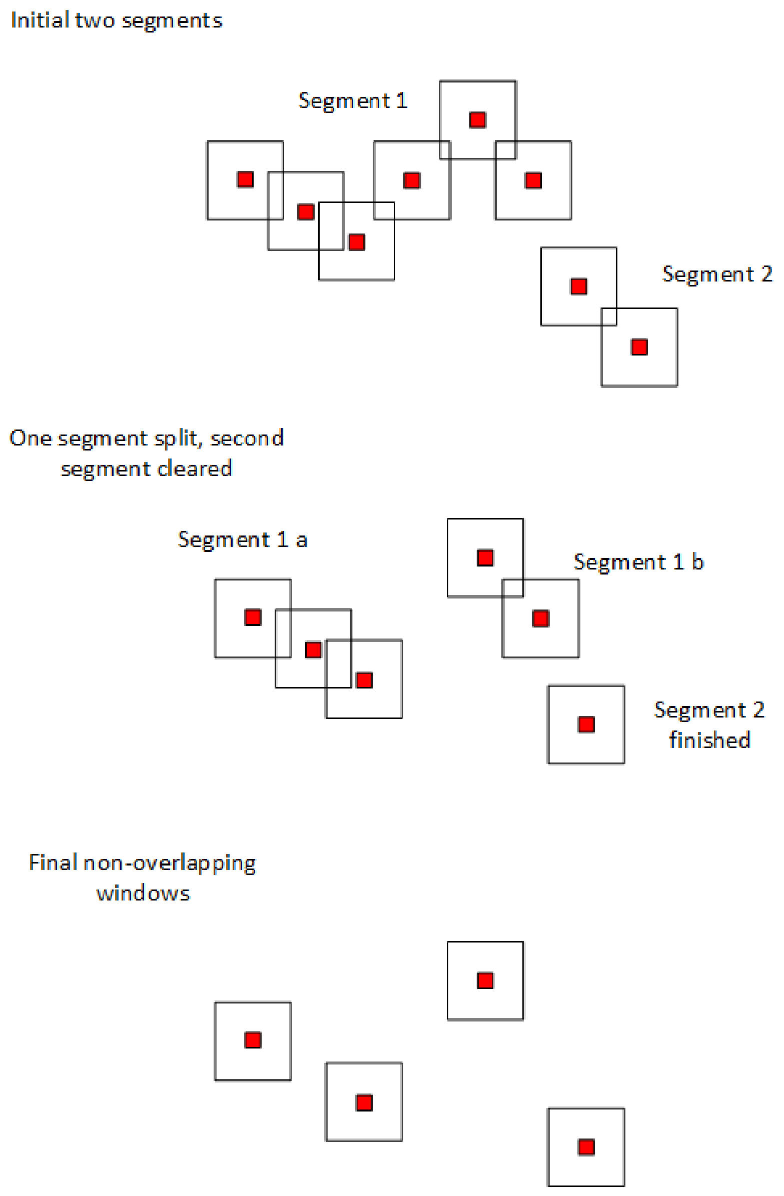

Figure 4-4: Overlap removal algorithm steps

4.6.4 Matchup Constraints and Matchup Clearing

This chapter combines the description of both clearing stages as they are technically very similar and the implementations differ in minor details. Conceptually, the differentiation is made to separate relatively lightweight and therefore fast matchup clearing algorithms from heavier ones. This approach shall ensure that the computationally expensive algorithms are already fed with a reduced list of possible matchup candidate locations.

Matchup clearing and the constraints processing both aim to identify those matchups in the list of possible matchups that conform to certain rules. The result of both stages is a Boolean keep/remove decision for each single matchup location.

The processing is performed using a plugin concept where each plugin executes a specific condition testing. The plugins can be chained, the information of the chain of plugins to use and the parametrization of each plugin is stored in an external configuration file.

Each plugin receives a list of possible matchup locations with geolocation, raster x/y coordinates and input file identifier, ordered by input file. Additionally, the plugin receives a processing context class, allowing access to the global resources of the processing system, i.e. data storage, metadata storage, logging context etc.

4.6.5 MMD Generation

The creation and storage of a MMD file is an extremely complex operation as it involves several data extraction and conversion steps. The Fiduceo MMS will re-use and adapt the technology developed for the SST-CCI MMS.

Each MMD contains all variables and metadata for all matchups detected in a pre-defined time-interval, i.e. the result of one processing slice of a larger time series. The content and format of the MMD file will be similar to the format developed for SST-CCI which is documented in (RD 1).

In summary, the MMD is a netcdf file that contains variables extracted from the input data (satellite/insitu) and additional metadata. The extensions of each variable are configurable, they are generally the desired window size in x and y dimension and the matchup count in z dimension.

The MMD generation process is controlled by a rule-engine that assembles the input data, performs (optional) chained transformations and finally writes the data to the output file. The rule-engine has separate rules for

- Parsing and creating dimensions
- Parsing and writing variable attributes
- Reading and storing satellite and insitu data
- Rules for data type conversion (e.g. float -> int)
- Rules for conversion of physical values (e.g. deg. Celsius -> deg. Kelvin)

The operations of the rule-engine are configured using an external configuration file. This is a large Java properties file that contains all information required to assemble an MMD.

Target dimensions can be defined on a per-variable basis, which allows the creation of variable window sizes around the matchup point, to reflect e.g. the different ground resolutions of the sensors. An example:

```
atsr.nx             = 11
atsr.ny             = 11
avhrr.nx            = 3
avhrr.ny            = 5
seaice.nx           = 1
seaice.ny           = 1
```

The content of each variable is defined in a second part of the configuration file that incorporates three columns. The first column specifies the name of the output variable as it appears in the MMD file. The second column specifies the name and origin of the data source for the variable. The third column defines the input-to-output transformation rule, usually a composite of several rules (separated by comma). An example:

```
matchup.id          Implicit  MatchupDimension,MatchupId
matchup.time        Implicit  MatchupDimension,TimeType,ReferenceTime
matchup.latitude    Implicit  MatchupDimension,LatType,MatchupLat
matchup.longitude   Implicit  MatchupDimension,LonType,MatchupLon
...
atsr.3.time         Implicit  Atsr3Sensor,MatchupDimension,TimeType,ObsTime
...
atsr.3.reflectance_55_nadir  atsr_orb.3.reflec_nadir_0550  MatchupDimension,
                                                           AtsrImageDimensions,
                                                           FillValueShortMin,
                                                           FromPercent,
                                                           ToReflectance
```

4.7 Ingestion Tool

The first step in all matchup processing tasks is the extraction and storage of all data locations and times that should be treated later. This is the duty of the Ingestion Tool.

It is invoked with a sensor-platform identifier and an (optional) time range. As a first step, the tool detects all files that fulfil the requirements in the archive, using the file name regular expression supplied by the associated data reader.

In a long iteration task, each of the files detected is opened and the reader is invoked to extract the acquisition information contained in the file (see chapter 4.5).

The geometry and the time axes received from the data reader are then pre-processed to suit the requirements of the processing stages that perform geometric operations. This step is necessary to represent the geometric data correctly in a two-dimensional longitude/latitude space.

Pre-processing steps applied are

- Segmentation at anti-meridian crossings, a simple polygon geometry is split into a multi-polygon
- Specific polygon corrections for acquisitions that contain one (or both) poles.

During the pre-processing, it is ensured that the associated time-axes of the acquisitions are updated accordingly.

The Ingestion Tool checks before inserting new data if the metadata-set being entered into the system is already stored to ensure that no double entries are generated. The identification can safely be done using the file path of the input file that is unique by file system constraints.

Finally, the meta-information assembled is stored to the metadata storage system. Depending on the implementation, this may be performed in multiple transactions. Optimization will be used to determine the optimal strategy, the solutions may also be depending on the metadata storage implementation.

The ingestion tool will be designed to allow the Workflow Engine to make optimal use of the parallel processing capabilities of the target hardware system.

4.8 Matchup Tool

The Matchup Tool is used to process either the complete chain of processing steps or a subset of them. It uses a configuration file that allows to define which steps should be invoked as well as the configuration of all processing steps.

It binds all processing components to form an executable entity. In most stages of the processing chain, it uses a plugin-approach that allows easy exchange of the components or adding/removing of components. The framework embedded in the Matchup Tool is designed to allow maximal flexibility to be able to react on new requirements as well as possible optimization steps or algorithm updates.

The chain itself is pre-defined by its logical structure:

1. Input Tool: receive input data for the chain (Satellite, insitu, random)
2. Intersection: perform a finer selection of the results retrieved from the metadata storage
3. Overlap removal
4. Matchup Condition processing: may involves chain of condition plugins, see chapter 4.6.4
5. Matchup Clearing processing: may involves chain of condition plugins, see chapter 4.6.4
6. MMD generation

The tool can be invoked using just a subset of the steps listed above. This allows splitting the matchup processing into smaller steps, allowing the Workflow Engine to make an optimal use of the parallel processing capabilities of a larger processing cluster. The Matchup Tool performs a logical analysis of the steps invoked and rejects erroneous configurations (e.g. intersection without associated input data). This behaviour is supported by functionality that allows this “sliced” processing. i.e.

- loading of previously stored intermediate results
- saving of intermediate results when not finishing with “MMD generation”

In the first case it detects the input data for the requested first processing stage that have been previously stored in the data archive and loads the appropriate data. In the latter case, the Matchup Tool ensures that the result data of the last processing stage is written to the appropriate archive location for later retrieval.

The intermediate files will be stored in suitable file formats. For the output of most intermediate steps, template formats and algorithms already developed for the SST-CCI MMS will be re-used and adapted. The main output format used there is JSON, because it allows a balance between the file-size and the requirement to store data in a complex, structured way.

4.9 Post Processing Tool

The Post Processing Tool operates on MMD files generated by the preceding stages. It operates on a set of MMD input files defined by the input parameters

- Time range,
- Source directory tree and
- MMD identifier

to enhance, modify or complete the content of the MMD. Algorithms applied to the input data set are encapsulated into dynamically loaded algorithm plugins. The Post Processing Tool allows chaining of algorithms to perform multiple steps in one run.

A command parameter allows to switch the tool between two operational modes:

- Inject additional variables or attributes into the existing files or
- copy the source files to a target location and perform the post-processing on the copied files, preserving the original input.

In which of these two modes the post processing tool is operating is shielded from the plugins.

In a similar concept as has been developed for the product reader support, post-processing plugins can be loaded dynamically on demand using the Java SPI framework. In addition, plugins can be added and removed without the need to recompile the complete software system.

4.9.1 NWP Plugin

This plugin adds variables to the MMD that are derived from the ERA Interim dataset for numerical weather prediction. The extraction of the data from the global reanalysis dataset is based on the matchup time and location. Calculation of the additional variables spans a time configurable time-range around the matchup time instant.

4.9.2 AVHRR Calibration Data Plugin

The AVHRR Calibration Data Plugin serves to integrate AVHRR calibration metadata into the MMD. The plugin extracts for each matchup from the associated AVHRR L1 file a number of attributes and integrates them into the MMD in a suitable way. The extraction covers for each matchup the calibration data spanning a configurable number of scan-lines before and after the matchup (default: 25). It extracts from each scanline:

- Space counts for each spectral channel
- Blackbody counts for each spectral channel
- BB PRT values
- The observed counts for each spectral channel for the complete matchup window
- Solar contamination flag where available
- average BB temperature over an orbit

4.10 Workflow Engine

The Fiduceo MMS system will re-use and adapt the workflow engine P-monitor that has already been used successfully in the SST-CCI and the QA4ECV projects. The P-monitor software is a constraint-based workflow engine that can be used in scripting environments to achieve control of concurrent bulk production jobs on the 'operator side'. P-monitor is implemented using the Python language.

The purpose of the P-monitor engine is to execute a sequence of tasks, along with their dependencies, on a pool of computing resources. The tasks and the dependencies between individual tasks are formally described as task inputs (i.e. pre-conditions) and task outputs (i.e. post-conditions). Post-conditions set by one task can be used as pre-condition for another task. By analysing the declared dependencies P-monitor can detect when the pre-conditions for an individual task are met and the task is ready for being executed on the resource pool.

The P-monitor engine maintains and operates on:

- A thread pool with a queue of 'ready-to-be-executed tasks', which is a list of all tasks for which all pre-conditions are met (i.e. a list of those tasks for which all required inputs are available).
- A backlog of tasks not yet 'ready-to-be-executed'
- A report file that records all completed tasks and, optionally, the paths of produced file (set) names mapped to file paths
- A list of tasks that had been executed in previous runs (e.g. in a run that was aborted). This list is obtained from the report file produced by the previous run and facilitates resuming production at where it had been interrupted.
- An optional map of bound file (set) names (the same as those used in the report file)
- A map of file set names to the number of yet missing outputs from (non-collating) tasks.
- A list of host resources with capacity (i.e. number of concurrent tasks by host) and current load, ordered by capacity minus load. This list is only relevant if tasks are computed on multiple hosts within a local network; it is not relevant when P-monitor is used as front-end to a cluster using grid engine or other cluster middleware
- A list of task-type resources with capacity (i.e. number of concurrent tasks by task) and current load
- A semaphore to protect concurrent access to the complex system status in order to maintain lists and reports consistently

4.10.1 Workflow Definition

P-monitor is invoked using Python scripts that define the workflow to be executed. These scripts have a common structure that usually consists of:

- A declaration part
- A workflow definition part and
- An execution instruction

In the declaration part all parameters that are used by different input sets are defined with their appropriate values. For bulk production this usually comprises a list of sensors and a time range to be processed.

The workflow is defined by a call to the P-monitor ‘constructor’, with the initial inputs and resource declarations, and several calls to the P-monitor ‘execute’ method, each call defining a new task, with pre-conditions and post-conditions in order to declare task dependencies; the order in which tasks are defined does not matter syntactically.

Finally, the execution instruction is constituted by a call to the P-monitor ‘wait-for-completion’ method, which starts the concurrent execution of workflow previously defined.

4.10.2 Input Parameter

P-monitor is configured using a set of input parameters that define the tasks to be executed and the execution environment. The input parameter are listed in the Table 4 below:

Table 4: P-monitor input parameter

Parameter	Description	Type	Default	Example
inputs	A set of pre-conditions satisfied a priori. Usually the inputs of the bulk production.	List of strings	n/a	['/archive/atsr.3/v1/2004/01', '/archive/atsr.3a/v1/2004/02', ...]
request	Name of the request, <request>.status and <request>.report will be name of status and report file. Must be unique in a directory if there is more than one script in the instance.	string	n/a	'mms-10'
hosts	Pairs of host name and capacity of the host. Defines overall concurrency for steps.	List of pairs of string and number	[('localhost', 4)]	[('localhost', 6)]
types	Pairs of task and concurrency capacity for a task type. Restricts concurrency, if provided. Alternative to setting 'weights'.	List of pairs of string and number	None	[('ingest', 2), ('matchup', 2), (clearing, 4)]
weights	Pairs of step and concurrency number of slots required for a task type, restricts concurrency if provided. Alternative to setting 'types'.	List of pairs of string and number	None	n/a

Parameter	Description	Type	Default	Example
swd	Software directory containing task implementations. If provided, the steps can be simple file names instead of absolute paths	string	None	'/fiduceo/mms/inst/bin'
cache	Root directory for working directories of steps.	string	None	n/a
logdir	Directory where to place log files of steps into	string	'.'	'/fiduceo/mms/inst/log'
simulation	The tasks are not executed, but all bookkeeping is done. Can be used for a dry-run of the workflow during development.	boolean	False	True
delay	Number of seconds to wait between subsequent additions of 'ready-to-be-executed' tasks to the worker pool in order to further process first what has been produced first. Experimental parameter.	float	None	0.2

4.10.3 Processing Parameter

Processing of a single task is performed in the "execute" method of P-Monitor. This execute method accepts a number of additional parameters to fine-tune the behaviour of the execution. The minimal and required parameter set for a single task consists of

- Task name ("call")
- Pre-conditions and
- Post-conditions

The full parameter set is described in Table 5 below:

Table 5: P-monitor processing parameter

Parameter	Description	Type	Default	Example
call	Script name to be executed for the task, must match one of the names declared in the types or weights parameters of Table 4	string	n/a	'mmd-generate'
inputs	Set of pre-conditions of the task. Usually (but not necessarily) the inputs of the task	List of strings	n/a	['/archive/atsr-en /v1/2004/01']

Parameter	Description	Type	Default	Example
outputs	Set of post-conditions of the task. Usually (but not necessarily) the outputs of the step	List of strings	n/a	['/archive/mmd_09/v1/2008/mmd -2008-01']
parameters	Set of parameters of the task. Must match to what the task implementation script would expect, if called from the command line	List of strings	[]	['2004', '01', 'atsr-en']
priority	Relative priority of the task, considered when not all 'ready-to-be-executed' tasks could be started concurrently because of resource limitations. Higher numbers are executed first.	integer	1	3
collating	If there is a set of more than a single input, collating means a single call of the script with all inputs as parameters. Non-collating means one call per input, each with a single input only. This feature can be used in combination with output reporting by the script. Lines issued on standard output by the script in the form "output=<path>" are recognised by P-monitor. Several such lines are collected to a set of outputs. This set replaces the formal post-condition in the script. A subsequent non-collating call will be split into calls for each product provided by the previous call.	boolean	True	n/a
logprefix	File name prefix of the task log file in the log directory. The names of log files default to the *call*, i.e. the name of the task, with the extension '.sh' stripped off. If several execute calls in a workflow definition use the same task script, this parameter could be used to distinguish the log files	string	None	'ingest-2004-01'

4.10.4 Workflow Execution

All tasks defined by using P-monitor have a set of pre-conditions and post-conditions; there always is an initial set of conditions satisfied a priori to allow the first steps to be executed.

For determining which tasks are ‘ready-to-be -executed’, P-monitor maintains a set of conditions. A condition in this sense is represented as plain string. A pre-condition of a task is satisfied if there is a string literally equal to a condition in the set of conditions maintained by P-monitor. Initially, this set is empty. The set is filled with initial conditions (usually the file paths to the inputs of bulk production) in the beginning of the workflow definition part of the workflow script.

A workflow script usually defines the set of tasks to be executed by means of several calls to the ‘pm.execute(...)’ method, where each task is supplied a set of pre-conditions and a set of post-conditions. A task is ‘ready-to-be-executed’ when all its pre-conditions are satisfied, i.e., if the set of conditions maintained by P-monitor contains all pre-condition strings associated with the task.

P-monitor selects task for execution also by considering resource constraints. For this purpose it maintains a list of resources per task type with capacity and load. Capacities are defined in the workflow script. P-monitor maintains the current load by bookkeeping of all tasks executed concurrently. If the load is lower than the capacity for a task type, then the task is put on the worker pool and executed.

The order in which P-monitor selects ‘ready-to-be-executed’ tasks is mainly determined by the order the tasks have been defined. Only if the sum of step capacities is larger than the overall (host) concurrency, the priority of steps (see above) matters.

5 Testing

A complex software system like the MMS using multiple interacting components and a combination of two programming languages (Java and Python) is extremely prone to errors. In addition, the overall complexity does not allow to test the end-to-end functionality in all possible combinations of parameters and data types. Therefore, efforts will be made to ensure that the complete system is at least assembled of well tested components that react in an expected way, both in normal processing as well as in error cases.

During development, a classical agile development process is aimed to be used for the MMS system. This includes at one of the core components a test-driven development scheme forcing the developer to always write an automated software test for a component under development before adding production code to the system. When used consequently, this approach leads to a test coverage of the production code well above 90 percent of the lines.

We will rely on two well-known test frameworks for Java, namely JUnit and Mockito. JUnit offers all functionality that is required to set up large collections of tests and execute these. The framework contains a large number of “assert” methods that allow verifying method return values, as well as means to test software behaviour in error cases. Mockito is a framework that is used in conjunction with JUnit to “mock” classes, i.e. create classes with a behaviour that can be defined by the developer specific for each testcase during the test set-up phase. It allows decoupling the test case from the environment by, e.g. allowing to mock file-system behaviour without the need to physically access the file system.

The test hierarchy implemented for the MMS distinguishes between

- Unit tests that cover the functionality of a single class
- Integration tests that cover the functionality of a collaboration of a small number of classes and
- Functional tests that cover functionality of a complete component (e.g. Ingestion Tool)

All Unit tests and most of the Integration tests are performed on each build step, as the build system used (Maven, see chapter 6) fully supports JUnit test integration. A failing tests automatically leads to a failing build, unless this behaviour is explicitly switched off. Thus, the majority of the automated tests is executed many times a day.

For the Python components of the MMS, a similar approach is used. Python itself contains full support for Unit testing, a functionality that will be used for testing the workflow engine. Also here, we define two hierarchies of tests:

- Unit tests that cover the functionality of a single class and
- Dry-run tests that test the behaviour of the complete workflow engine.

6 Version Control and Building

The development of the MMS system will be backed by a version control system. We will use git as version control system as this has been used reliably for a number of projects developed by Brockmann Consult.

To reflect the open-source nature of the project, a public open repository on Github will serve as the master source repository. This service is provided by Github for free, as long as the code is publicly available.

Using git allows maintaining different versions of the software at the same time; this concept is called branching. We will use the “HEAD” branch for the actual development, using a “SNAPSHOT” version number indicating that this branch is in development. Every time, a stable and reliable state is reached that shall be released, a branch is introduced to the version control system. The software gets a final version number assigned, is built from the branched version and released. The branch is then used for maintenance and bug fixing only. Development continues on the “HEAD” branch. Changes and improvements implemented on a branch can always be migrated to the “HEAD” branch by using a git merge operation.

The software build and packaging will be executed by maven. Apache Maven is a freely available and open source build management system that allows executing all tasks required for software build and release in one run. Maven is configurable and maintains in the configuration file all tasks, software dependencies and build and package steps required. These include

- Compilation
- Dependency resolving
- Testing
- Module assembly
- Packaging to distributable archive

Although maven is primarily designed for use with Java based software, the package and assembly steps can easily be extended to use also Python based software in conjunction with Java sources.