Near-real-time Continuous Filtering of Sensor Measurements using Data Stream Management Systems

Sven Müller
Institute of Space Systems, TU Braunschweig, Germany
Enrico Stoll
Institute of Space Systems, TU Braunschweig, Germany

ABSTRACT

With increasing amounts of raw sensor measurements of resident space objects and the tendency to go to near-real-time processing, it becomes more and more important to discard measurements of certain objects quickly. This way, computationally intense processing can be constrained to a subset of the measurements, for example to objects on a whitelist. This lowers the burden to install processing chains that automatically work on new measurement data as soon as it is acquired. If such a filtering is done in near-real-time, it may enable new usage concepts of sensor data, e.g. to provide whitelisted measurements to certain institutions, corporations or the public instantly, without the need for human intervention. Near-real-time filtering also allows rapid sensor tasking. As soon as measurements are produced that pass the filter, the sensor (or another one) could be tasked to track the detected objects. This makes near-real-time filtering a potentially valuable mechanism for Space Surveillance and Tracking. The output side of such a filtering mechanism can be regarded as a virtual sensor, since it outputs the same kind of measurement data with roughly the same timing, only filtered. Thus, existing interfaces to process measurement data do not need to be changed for integrating this approach.

By utilizing so-called Data Stream Management Systems (DSMS), continuous near-real-time filtering can be achieved. DSMS are essentially databases transformed for the purpose of near-real-time processing of data that streams in continuously. They are not meant to be used for saving data persistently, though, and thus, do not save processed data on HDD or SSD, but only in RAM - and only as long as the data is still being processed. Because of the processing being done in RAM only, DSMS are able to cope with very high input rates of raw measurement data. Furthermore, these systems follow the data-driven processing paradigm, meaning that the input of each single data point triggers the data processing and new data results are produced. The processing steps done by a DSMS – in this case, filtering - can be modified easily by changing the data flow definition sent to the DSMS. These definitions often are saved in simple ASCII text files. Some systems even allow changes in the data flow, when the DSMS is still processing data. Data fusion is also possible with these types of systems, since they provide mechanisms for branching and joining data streams. In order to organize processing of potentially infinite incoming data streams, DSMS create "windows" on the streams, for example windows with all data elements that streamed in in the last five minutes. Data elements that drop from such a window are dismissed. Additionally, DSMS come with general processing mechanics necessary for achieving high performance, such as the scheduling of multiple parallel internal processing chains. One of the reasons, they have been invented is to be able to handle uncertain data - like sensor measurements.

This paper elaborates on DSMS and their capabilities. It is shown how a specific DSMS was adjusted for the purpose of measurement filtering. A test scenario and its execution are described, the results are discussed.

1. INTRODUCTION

In the future, it may be prudent to reduce (or at least limit) the load existing SST processing systems have to cope with so that they can continue to function as they do today without foregoing new sensor or sensor network capabilities. Since certain functions shall only be executed on measurement data of certain objects (orbit improvement or precise orbit determination, for example), it makes sense to filter measurements based on a whitelist. This approach also opens the possibility to provide the measurement data in real-time to the public without the risk of the provision of data that was not meant to be provided. This can be done efficiently by utilizing Data Stream Management Systems (DSMS). Their basic application in SST context has been evaluated in [2] by the execution of a main function (Initial Orbit Determination, IOD) on a radar tracklet and by putting it under stress produced by a realistic detection input rate. Approaches to integrate DSMS into existing SST systems have been layed out and documented in [1] as so-called SST Architecture Enhancements. One of the Enhancements called Compression was implemented and run successfully. In the Compression Enhancement, a DSMS functions as "virtual sensor" from the SST Processing System's point of view. This paper deals with the Filter Enhancement belonging to the same "virtual sensor" Enhancement family as the Compression Enhancement. The DSMS PICARD used to design and implement that Enhancement is described in section 2 – along with a DSMS' general capabilities. The software used for radar measurements generation is described thereafter. Then, the SST scenario simulated using PICARD and MWG, as well as the scenario execution results are presented and a conclusion is given.

2. DATA STREAM MANAGEMENT SYSTEM PICARD

PICARD is an extension of the Data Stream Management System (DSMS) STREAM. DSMS basically change the paradigm of data processing by viewing data as data streams. A data stream is a continuous flow of data. This perspective has been shown fitting for the new world of ever-so-small sensors producing data in large quantities. Basically, while a common database stores data permanently, a DSMS trys to store only as much data as possible for as long as necessary. While a database is queried to output a finite data set, a DSMS stores queries and continuously outputs the query results as a variable, (potentially) infinite data stream until the queries are removed. All data in DSMS thus can be described as volatile. The idea is to go from a bulk-oriented data processing to a conveyor-belt-oriented one by dividing it into self-contained and small modules ("operators") with each taking care of one simple job and to doing it as quickly as possible. Each data element input is processed and its output created separately ensuring small delays in processing. Operators are connected to form an operator graph implementing the pursued processing chain. The processing is done in main memory (RAM).

STREAM could be described as Relational DSMS. Just like a common Relational Database it works on the mathematical concept of relations which can be expressed as tables. The difference is that, in STREAM, the tables are streamed rather than stored. The data streams consist of so-called relational updates which represent either the insertion (indicated by a "+") of a tuple into that relation or the removal (indicated by a "-") of a tuple from that relation. The relation itself does only exist virtually. If one were to apply all relational updates streamed until t onto an empty relation, they would get a relation containing the tuples the virtual relation contains at t. STREAM's operator graph is determined by parsing a simple text file containing script language ("script file"). By changing the script and restarting the program a new data flow is executed.

PICARD is based on STREAM. It allows to define so-called Processing Operators that can be quickly added and arbitrarily implemented. PICARD also allows the definition of Representational State Transfer (REST) data input and output, i.e. it can receive and provide data via a web API using REST SDK.

3. RADAR MEASUREMENTS GENERATOR MWG

At Institute of Space Systems, a Radar System Simulator (RSS) has been developed being able to simulate a Radar-based SST System [4]. One important component is the Radar measurements generator (Messwertgenerator, MWG). It is capable of producing realistic topocentric horizon data for arbitrary ground-based sensor locations. Reflector antennas and phased arrays may be simulated and run in scanning or tracking mode. It utilizes a user-defined RSO population. For orbit propagation, a sophisticated, numerical, full-force orbit propagator named NEPTUNE is used.

4. SCENARIO DESCRIPTION

The scenario consists of the following basic components:

- Phased array sensor
- DSMS
- Pre-existing processing system

The idea is for the DSMS to be able to filter the measurements sent by the sensor in (near) real-time. The sensor shall send measurements with at least 30 Hz to the processing system. The processing system shall only let measurements of objects on a whitelist through. The sensor operates in scanning mode with nine line-of-sight (LOS) positions being cycled through. At each LOS position, 20 pulses of 1.5ms length are sent. The total opening angle amounts to 4° and no pulse integration is done. The antenna is directed to the zenith with the values listen in Tab. 1 being valid per LOS position.

Tab. 1. Phased Array line-of-sight positions in antenna-angles system cycling through in the simulation.

LOS Position ID [-]	**LOS Elevation [deg]**	**LOS Azimuth [deg]**	**Pulse Repitition Frequency [Hz]**
0	0	-16	35
1	0	-12	34
2	0	-8	33
3	0	-4	32
4	0	0	30
5	0	4	32
6	0	8	33
7	0	12	34
8	0	16	35

The antenna-angle positions given above (together with the antenna's LOS) translate to a scanning "fence" spanning 32°. The sensor is located at latitude 52.5200070°, 13.4049540° longitude and 36.19 m altitude.

The whitelist shall allow objects to be defined by Kepler element bounds. There shall be no false-positives and no false-negatives during matching, i.e. exactly those measurements shall pass the filter that belong to objects on the whitelist. The whitelist comprises the two objects listed in Tab. 2 and Tab. 3.

Tab. 2. Orbital parameters of whitelist objects at simulation start epoch.

Object ID [-]	**a [km]**	**e [-]**	**i [deg]**	**RAAN [deg]**	**AoP [deg]**	**M [deg]**
3803246	7892.8	0.0895	65.82	269.87	146.06	241.41
210126084	7792.1	0.0002	52.00	205.90	199.14	233.38

Tab. 3. Physical parameters of whitelist objects at simulation start epoch.

Object ID [-]	**Mass [kg]**	**Diameter [m]**	**m/A [kg/m^2]**
3803246	0.8175E+02	0.2954E+01	16.735
210126084	0.4500E+03	0.3864E+01	38.369

The ESA MASTER 2009 Population shall be simulated. The simulation-world simulation start shall be 2018-02-22T08:35:00.0Z with a duration of a minimum of 24 hours. Both whitelist objects pass the sensor's field of view within that time.

5. SCENARIO REALIZATION

The phased array is simulated by an MWG instance, the DSMS is realized by PICARD running on the same computer (common business laptop) as the MWG instance. The pre-existing processing system is mocked. The transfer of measurements from MWG to PICARD is done via REST in order to simulate a network connection. For this purpose, a REST source operator has been created for PICARD and the sending of measurements via REST to a hard-coded web address has been added to MWG. Some tweaking was necessary. At first, each measurement was sent in its own REST request which lead to an inefficient data transfer. However, an approach for time-based sending of measurement batches (one batch per second, for example) did not work, because the number of measurements in the batches surpassed thresholds for operating system command line lengths. Because of that, a measurement-number-based approach has been used instead. A maximum of 75 measurements are put into one batch and being shipped off to PICARD via one REST request. An aynchronous sending of the request made the operating system reach child process limits which is why they are send synchronously. Each measurement (also referred to as "detection") contains:

- detection ID (identifies a detection within a tracklet)
- tracklet ID (identifies detections that supposedly belong to the same object)
- object ID (is sent and used only for validation purposes)
- epoch
- azimuth
- elevation
- range
- range-rate (is sent, but not used)
- radar cross section (is sent, but not used)
- signal-to-noise ratio
- probability (is sent, but not used)

On PICARD side, a new REST Source operator was created based on an existing one and retrofitted to cope with a big number of elements streaming in at once. Two input queues are used now. The first one is filled by the threads processing REST requests, the second one is filled with all elements from the first queue by the main thread whenever the second queue becomes empty. This way, thread interference and blocking is reduced to a minimum.

The simulation step size of MWG has been set to 20 minutes. The idea was to achieve a flow of measurement data which was as continuous as possible without compromising simulation results. The lower diameter limit for filtering the MASTER population, the sensor's transmitting power as well as transmit and receive gains were balanced such that the measurement send rate requirement mentioned above was met (surpassed, if possible), while keeping the real-world simulation duration in check. As a result, the sensor transmitted with 1 MW and had receive and transmit gains of 75 dB. The lower population object diameter used was 1 meter resulting in 3003 population objects.

For the application of PICARD, the methodology described in [1] is used, a sequential approach starting with an SST System Architecture Enhancement idea, followed by its design and implementation. The Filter Enhancement idea is shown in Fig. 1. From the perspective of a pre-existing SST Processing System, PICARD shall act as a pre-existing sensor – only that the detections received by the sensor behind PICARD shall be filtered (in this case, based on a whitelist).

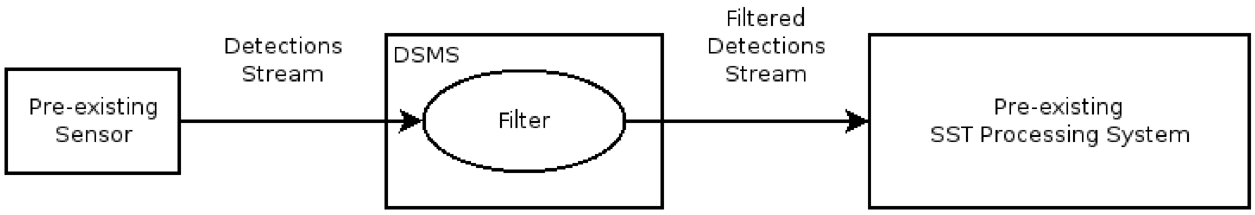

Pre-existing Sensor
Detections Stream
DSMS
Filter
Filtered Detections Stream
Pre-existing SST Processing System

Fig. 1. Idea of the DSMS SST System Architecture Enhancement "Filter".

The Filter Enhancement Design for PICARD was developed in two steps. The first step is presented in Fig. 2 and is only applicable for one whitelist object. The sensor sends measurements comprising the values listed before. The object detected is assumed to be unknown at time of detection, since the Radar is operating in scanning mode. As described in section 2, a DSMS' modules normally consists of operators. These are shown in the figure as ellipses. Detection Source operator receives detections from sensors and insert them into PICARD as a pure data stream. It also adds τ to the detection data set, the timestamp tagging data tuples in a timely manner. For this application, τ is the number of real-world (clocked) microseconds passed since PICARD startup. The data flows into the Row Window operator which produces a streamed table always containing the 20 detections that arrived the latest. These windows are key to a DSMS' functionality, since they constrain the data being hold in RAM. At first, a time-based window operator was used. However, the number of measurements contained in that window at any point in time could surpass levels at which the IOD algorithm functioned efficiently. A newly implemented Initial Orbit Determination (IOD) operator processes the window table into one which always contains the Kepler elements for each tracklet for which detections exist in the window table. Compared to the IOD operator described in [2], this one is based on Goddard Trajectory Determination System (GTDS) instead of Herrick-Gibbs and is called directly through a C-Fortran interface instead of calling an external program. Another important difference is that this operator executes IOD everytime a new detection is inserted into the input table or removed from the input table instead of waiting for the arrival of a complete tracklet. This behavior fits the conveyor belt paradigm of DSMS better than the previous approach. Based on the IOD output table, a Select operator selects only those tracklets which conform to the Kepler element bounds defined by the user of the system (keep in mind that all the data flow in PICARD is changeable via a simple text file using a script language so that a change of the bounds does pose no problem). Only semi-major axis, eccentricity and inclination are used for filtering tracklets. Here, the resemblance of STREAM, the DSMS which PICARD is based on, to usual relational database systems shows, since the Select operator will eventually be generated by a statement such as “select * from someRelationStream where a >= 5000 and [...]”. The resulting streamed table will contain the tracklet IDs belonging to the whitelist object – assuming the filter works as intended. In order to create a table containing all detections belonging to the tracklet IDs that passed the filter, a join with the original detection table produced by the Row Window operator is necessary. The tracklet ID is used for the joining. In order to join only the data attributes necessary, a Project operator is put before the join. Additionally, after the join, there are two tracklet ID attributes in each data tuple which both contain the same value. One of these attributes is thus removed by the Project operator after the join. The resulting table always contains all detections within the 20-detections-window which belong to the whitelist object. In order to mimic the original sensor, the detections shall be sent to the SST Processing System. They shall be sent each time a detection is inserted into the table which is why an IStream operator is used. Its definition is given in [3], but the main idea is to look at the input table after the time instant τ is over and emit the elements added within that time instant as stream elements. It is important to know, though, that many detections may be given the same τ value, since they arrive at the same time instant. The lower the granularity of τ is chosen, the more elements might be given the same time value. If it is set to seconds meaning τ is representing a number of seconds, it is obvious, for example, that many measurements may arrive at the same time instant. This posed a problem since the default granularity is seconds which lead to measurements being put into the window and removed from it again too quickly to be noticed by IStream. This was the main reason for increasing the granularity to microseconds.

Additionally, the following scenario must be considered:

1. Detection element is inserted into 20-detections-window and is processed by IOD operator resulting in a new IOD result element containing Kepler elements
2. IOD result element passes filter and is joined with the corresponding tracklet’s detections in the 20-detections-window.
3. Time instant passes resulting in IStream outputting the detections.
4. A new detection element is inserted into the window, resulting, this time, in the same tracklet *not* passing the filter. The tracklet and its detections are removed from the IStream input table.
5. Time instant passes.
6. A new detection element is inserted into the window, resulting, this time in the same tracklet *passing* the filter again and thus inserting the tracklet’s detections once again into IStream’s input table.
7. Time instant passes leading IStream to emit, at least in part, the same detections again.

In order to prevent the described scenario, another, bigger Row Window is attached to IStream. It’s big enough to eventually contain all possible duplicate detections. A Distinct operator is then used to get rid of the duplicates after which another IStream operator, together with a Detection Sink operator takes care of emitting a pure data stream to the pre-existing SST Processing System.

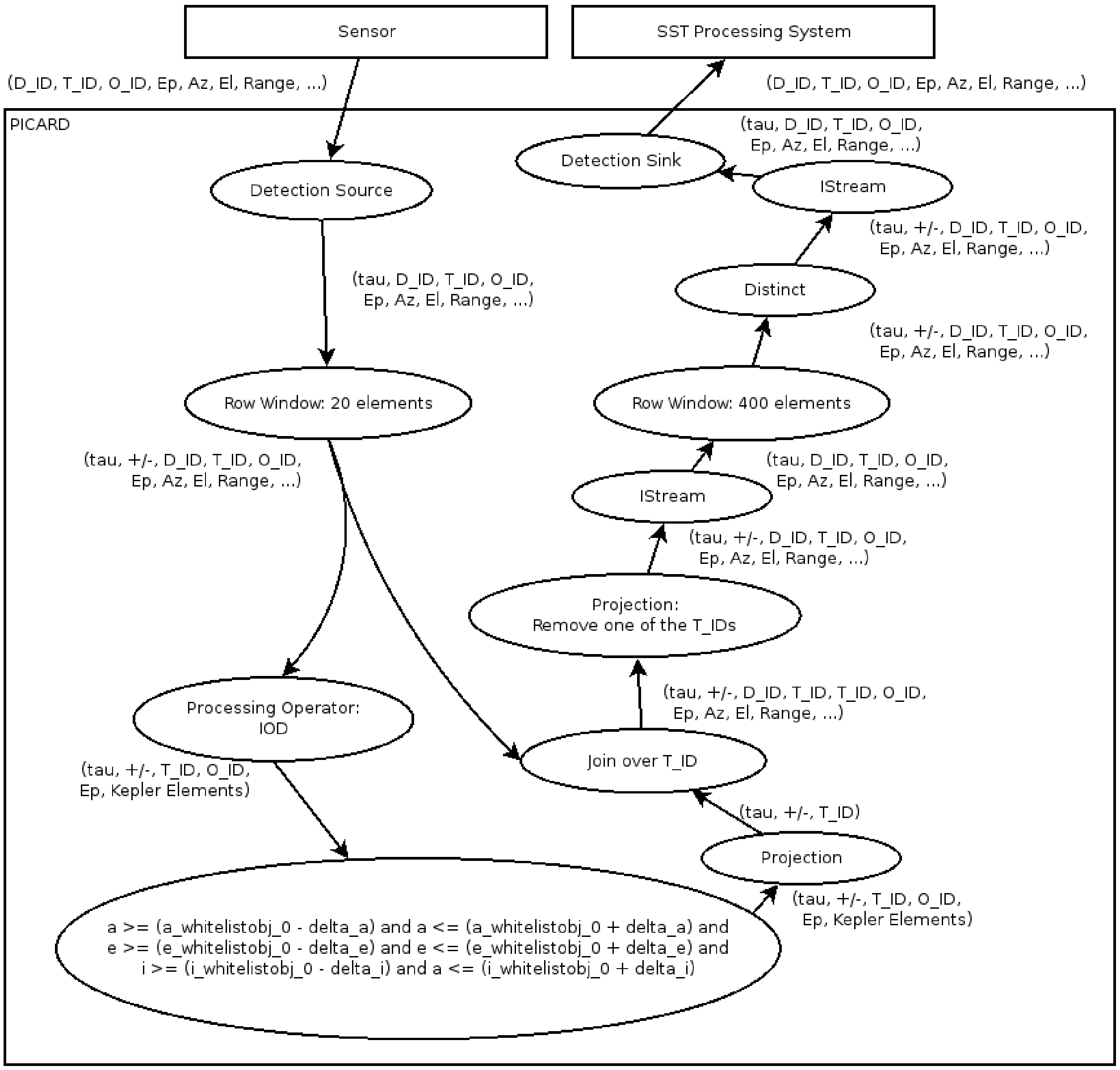
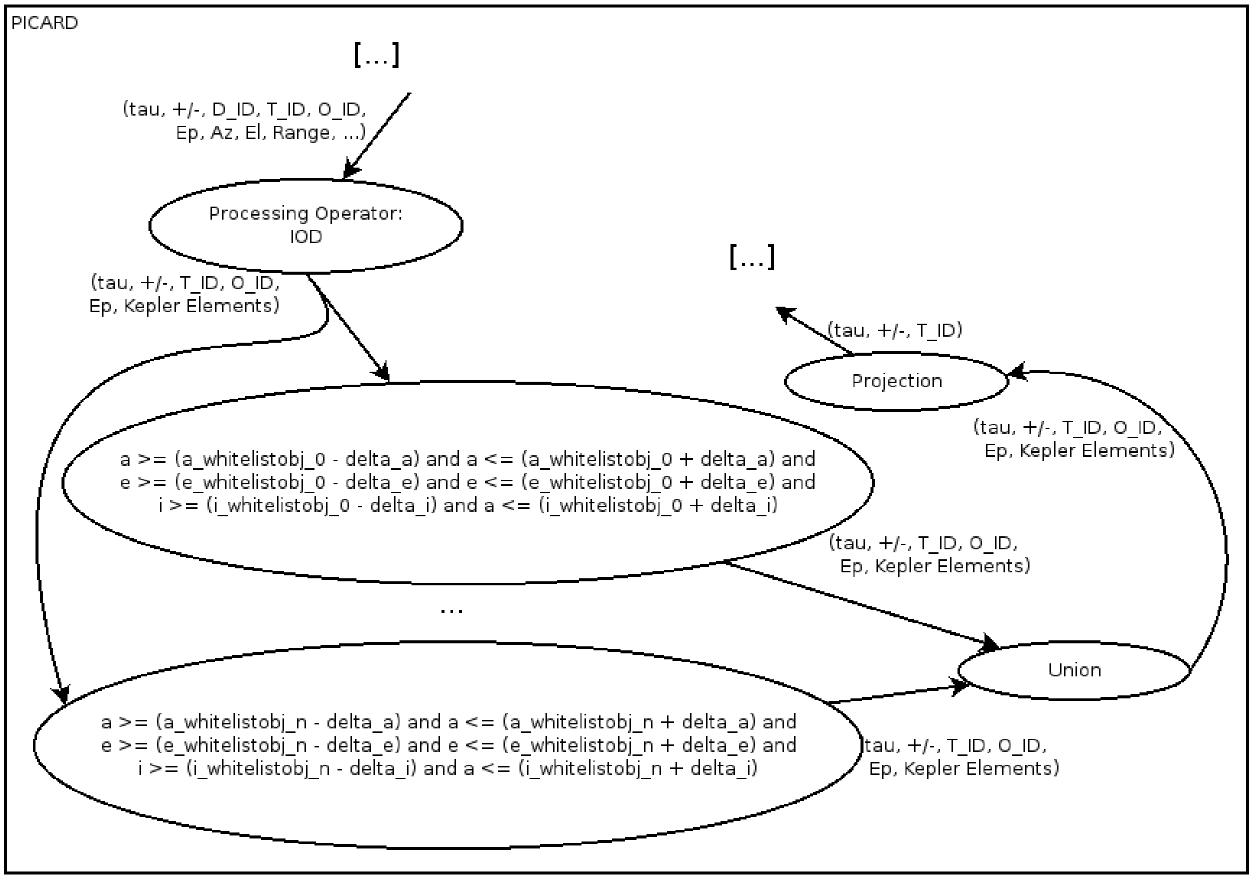

Sensor
SST Processing System
(D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
(D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
PICARD
Detection Source
Detection Sink
(tau, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
IStream
(tau, +/-, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
Distinct
(tau, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
(tau, +/-, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
Row Window: 20 elements
Row Window: 400 elements
(tau, +/-, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
(tau, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
IStream
(tau, +/-, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
Projection: Remove one of the T_IDs
(tau, +/-, D_ID, T_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
Processing Operator: IOD
Join over T_ID
(tau, +/-, T_ID, O_ID, Ep, Kepler Elements)
(tau, +/-, T_ID)
Projection
(tau, +/-, T_ID, O_ID, Ep, Kepler Elements)
a >= (a_whitelistobj_0 - delta_a) and a <= (a_whitelistobj_0 + delta_a) and
e >= (e_whitelistobj_0 - delta_e) and e <= (e_whitelistobj_0 + delta_e) and
i >= (i_whitelistobj_0 - delta_i) and a <= (i_whitelistobj_0 + delta_i)

Fig. 2. Design of the DSMS SST System Architecture Enhancement "Filter" - first step being applicable to one whitelist object.

The size of the first row window (and thus, implicitly the size of the second window) was determined empirically as 20 as a trade-off between allowing sensible IOD algorithm results and a small window size for performance considerations.

The second step of the Filter Enhancement's design being applicable to multiple whitelist objects is shown in Fig. 3. The change is rather simple: Instead of one Select operator, multiple are used on the same (relational) data input stream. A Union operator merges the streams into one without doing any kind of join. It only brings the data tuples in sequence based on τ. A Distinct operator would normally be necessary after that (cp. Quality Enhancement [1]), since, theoretically, a tracklet may pass more than one filter resulting in duplicate entries in the Union's output table. However, the existing Distinct operator at the processing chain end already takes care of this scenario. With this design, adding a new object to the whitelist only involves copying and pasting the Select part plus the change of the bound values in PICARD's script file. This approach is fine for a small number of objects. A whitelist containing more objects is possible, too, but would require a different design for convenience. For example, a text file could be read in as streamed relation containing the whitelist objects and the Kepler Element boundaries.



PICARD
[...]
(tau, +/-, D_ID, T_ID, O_ID, Ep, Az, El, Range, ...)
Processing Operator: IOD
(tau, +/-, T_ID, O_ID, Ep, Kepler Elements)
[...]
(tau, +/-, T_ID)
Projection
(tau, +/-, T_ID, O_ID, Ep, Kepler Elements)
a >= (a_whitelistobj_0 - delta_a) and a <= (a_whitelistobj_0 + delta_a) and
e >= (e_whitelistobj_0 - delta_e) and e <= (e_whitelistobj_0 + delta_e) and
i >= (i_whitelistobj_0 - delta_i) and a <= (i_whitelistobj_0 + delta_i)
(tau, +/-, T_ID, O_ID, Ep, Kepler Elements)
...
Union
a >= (a_whitelistobj_n - delta_a) and a <= (a_whitelistobj_n + delta_a) and
e >= (e_whitelistobj_n - delta_e) and e <= (e_whitelistobj_n + delta_e) and
i >= (i_whitelistobj_n - delta_i) and a <= (i_whitelistobj_n + delta_i)
tau, +/-, T_ID, O_ID, Ep, Kepler Elements)

Fig. 3. Design of the DSMS SST System Architecture Enhancement "Filter" - second step being applicable to multiple whitelist objects.

The implementation of the Filter Enhancement looks just like the second step of the design, with the following exceptions:

- The sensor is an MWG instance
- MWG instance sends detections via REST to PICARD resulting in the Detection Source operator being implemented as REST Source operator. Some details about this operator were given at the beginning of this section.
- The Detection Sink is implemented as File Sink writing the filtered detections to a text file.

The actual design and implementation contain a lot more operators than shown. They are used for the gathering of statistical data. Also, for debugging purposes, each data stream is output to a text file growing the number of operators to a total amount of 102.

PICARD has been configured to take up 2 GiB of main memory. The Kepler Element bounds used in the filter are $\Delta_a = 10\,km$, $\Delta_e = 0.001$ and $\Delta_i = 0.2°$. A section of the data flow script is given in Lst. 1 to give an impression of the resulting overall script.

Lst. 1. Exemplary PICARD data flow script sections.

```
# radarMeasurementsStr
########################
table: register stream radarMeasurementsStr(detectionId integer, trackletId
integer, objectId integer, epoch double, azimuth double, elevation double,
therange double, rangeRate double, radarCrossSection double,
signalToNoiseRatio double, probability double);
source: restAmos radarMeasurements

query: select * from radarMeasurementsStr;
dest: file output/radarMeasurementsStr
########################

[...]

# radarMeasurementsRel
########################
vquery: select * from radarMeasurementsStr [rows 20];
vtable: register relation radarMeasurementsRel(detectionId integer,
trackletId integer, objectId integer, epoch double, azimuth double, elevation
double, therange double, rangeRate double, radarCrossSection double,
signalToNoiseRatio double, probability double);

query: select * from radarMeasurementsRel;
dest: file output/radarMeasurementsRel
########################

[...]

# iodOutputFiltered0Rel
########################
vquery: select * from iodOutputRel where semiMajorAxis >= (7792.1d - 10.0d)
and semiMajorAxis <= (7792.1d + 10.0d) and eccentricity >= (0.0002d - 0.001d)
and eccentricity <= (0.0002d + 0.001d) and inclination >= (52.00d - 0.2d) and
inclination <= (52.00d + 0.2d);
vtable: register relation iodOutputFiltered0Rel(trackletId integer, objectId
integer, epoch double, semiMajorAxis double, eccentricity double, inclination
double, rightAscensionOfAscendingNode double, argumentOfPerihel double,
meanAnomaly double);

query: select * from iodOutputFiltered0Rel;
dest: file output/iodOutputFiltered0Rel
########################

# iodOutputFiltered0DiffsRel
########################
vquery: select objectId, semiMajorAxis - 7792.1d, eccentricity - 0.0002d,
inclination - 52.00d from iodOutputFiltered0Rel;
vtable: register relation iodOutputFiltered0DiffsRel(objectId integer,
semiMajorAxisDifference double, eccentricityDifference double,
inclinationDifference double);

query: select * from iodOutputFiltered0DiffsRel;
dest: file output/iodOutputFiltered0DiffsRel
########################
```

6. RESULTS AND DISCUSSION

MWG simulation ran, simulation-world-wise, from 2018-02-22T08:35 to 11:15 of the next day, amounting to 26h, 40 min (94800 s) simulation duration. The real-world simulation duration was 15 h, 47 min and 14.479 s (56834.479 s), the ratio thus being roughly 1.7. However, it should be noted that MWG stopped in the middle of the last simulation step which ended at 10:55 simulation-world-wise (simulation-world duration being 93600 s). Until that time, MWG produced 2950 tracklets with a total amount of 5041778 detections leading to a detection rate of 53.87 Hz simulation-world-wise. The MWG execution time being taken up by the adjustments made for sending the produced measurements to PICARD (including the wait time for REST responses) was, until 10:55 simulation-world time, 17360.4688 seconds. The number of detections that were produced until 11:15 was 5108494, some of which could not be shipped for data transfer by MWG anymore due to program stop during the last simulation step. Futhermore, because of a bug, the first REST request of each data transfer after the first one could not be sent to PICARD. In consequence, 5062253 detections were received and processed by PICARD in total. The detection rate PICARD was handling during data transfer times thus amounted to roughly 291,6 Hz. Tab. 4 lists the objects that passed the filter, Tab. 5 their physical properties. The whitelist objects passed the filter as intended. However, 51 additional objects were matched to be the same as object 210126084. Their orbits are, based on the filter parameters a, e and i, nearly the same making this result not surprising. As can be seen in Tab. 6, Tab. 7 and Tab. 8, the average filter parameter differences seem to suggest that a lowering of the inclination delta could be feasible to filter out the falsely matched objects. Additionally, an inspection of the produced raw data after filter application shows that RAAN is determined by IOD with sufficient certainty to use as additional filter parameter in order to prevent false positives or, at least, decrease their number.

Tab. 4. Orbital parameters of exemplary objects that passed the filter. Also, it is listed which of the listed objects passed through which filter.

Object ID [-]	**Passed Filter as Object ID [-]**	**a [km]**	**e [-]**	**i [deg]**	**RAAN [deg]**	**AoP [deg]**	**M [deg]**
3803246	3803246	7892.8	0.0895	65.82	269.87	146.06	241.41
210125165	210126084	7792.1	0.0000	51.99	203.44	153.84	150.43
210125943	210126084	7792.2	0.0002	52.00	249.56	143.02	271.01
210126083	210126084	7792.1	0.0002	52.01	251.12	175.92	271.89
210126084	210126084	7792.1	0.0002	52.00	205.90	199.14	233.38

Tab. 5. Physical parameters of exemplary objects that passed the filter.

Object ID [-]	**Mass [kg]**	**Diameter [m]**	**m/A [kg/m²]**
3803246	0.8175E+02	0.2954E+01	16.735
210125165	0.4400E+03	0.3863E+01	37.550
210125943	0.4500E+03	0.3864E+01	38.369
210126083	0.4500E+03	0.3864E+01	38.369
210126084	0.4500E+03	0.3864E+01	38.369

Tab. 6. Semi-major axis filter parameter value differences for exemplary objects that passed the filter.

Object ID [-]	Δ_a		
	min.	**max.**	**avg.**
3803246	1.46373079973091	6.7228646918802	4.19877527870403
210125165	0.293510619882909	9.77556896093938	4.37944732592407
210125943	0.233007941992582	9.56843442394893	4.91617710319167

210126083	0.0345799516444458	9.13819972364217	4.5226553860288
210126084	0.131587107341147	9.87807110302947	4.44239660851094

Tab. 7. Eccentricity filter parameter value differences for exemplary objects that passed the filter.

Object ID [-]	Δ_e		
	min.	**max.**	**avg.**
3803246	8.64535254484211e-05	0.000733646011522274	0.000528527803026579
210125165	2.38346131141405e-05	0.000987341600926242	0.000489496946545129
210125943	2.86573887848558e-06	0.000997397339136727	0.000512737270518162
210126083	1.70909561456193e-05	0.000990395463874944	0.000487601526880916
210126084	2.36133311875826e-05	0.000997082122891875	0.000474421070722199

Tab. 8. Inclination filter parameter value differences for exemplary objects that passed the filter.

Object ID [-]	Δ_i		
	min.	**max.**	**avg.**
3803246	0.00243823908154184	0.0555450070015269	0.0287821461203563
210125165	0.000251709731621474	0.0858348053097586	0.0162863985599928
210125943	0.000322622078094525	0.0788920880737791	0.0123207593060872
210126083	0.000164705102804419	0.140567321341948	0.0153483133550132
210126084	0.000146091903957313	0.122186307536083	0.00870342843748659

Tab. 9 gives an overall summary of the filter matching results. Both whitelist objects were matched, so there are two true positives and no false negatives in this regard. Also, it should be noted that no tracklet belonging to the whitelist objects was removed from output completely; all 5 tracklets belonging to the two objects came through. But, 2691 of their 4326 detections were dropped leading to 37.8% of their measurements being forwarded. As mentioned earlier, 51 objects have been false-positively matched so that 89 tracklets containing 34672 detections came through falsely. Only about 4.5% of all detections in output belonged to the two whitelist objects, but this should be expected concerning the problematic 210126084 filter. 2857 tracklets of 1667 objects comprising 5023254 detections were correctly dropped (99.2%). The input load a processing system "behind" the filter would have needed to cope with has been reduced by 99.3%.

Tab. 9. Categorization of filter matches and non-matches.

Match Category	**Number of Objects**	**Number of Tracklets**	**Number of Detections**
True Positive	2	5	1635
False Positive	51	89	34672
True Negative	1667	2857	5023254
False Negative	0	0	2691
Total	**1720**	**2951**[1]	**5062252**[2]

[1] Note that the script used for the calculation of these numbers only considers relation stream entries for complete τ instances, i.e. entries for the newest τ are ignored. Also, some tracklets may have been lost, because of the bug mentioned earlier. Moreover, some tracklets have been sent in the last (cancelled) transfer that were not counted in the total tracklet number mentioned before.

[2] The loss of one detection is unclear and needs to be investigated.

For PICARD performance evaluation, the metric “Operator Lag” is used which tells how much time a DSMS operator “lags behind” at a certain DSMS execution time [2]. At the moment, lags smaller than 1 millisecond can’t be detected accurately. Above that threshold, however, the lag time resolution is one microsecond. This has nearly no influence on the following results interpretation based on the lag definition in [2], but will affect another lag definition presented further beneath.

Fig. 4 and Fig. 5 show the lag of three operators in the first simulation hour representative for all other operators’ lag behavior. The first figure shows the lag of the REST Source operator which injects measurements into the system. MWG sends the measurements in intervals of roughly 500 seconds. While waiting, there is no input data update and thus no output data update. The data output “time” thus lags behind further and further. Upon arrival of the measurements, there is nearly no lag anymore, since elements are output as quickly as possible. While the current measurement batch is processed by the operator, the lag is very volatile, but stays lower than 500 ms. After the batch is processed, the lag builds up again and another “cycle” begins. The second figure shows the lag of an operator far down the processing line. The general appearance is comparable, but the lag at data input times is not that volatile and generally at a higher level, while not surpassing 500 ms either. The time intervals in which this operator is busy, are also a bit shorter as the ones of REST Source Operator. The third figure shows the lag of an operator far down in the processing line with fewer busy times. These types of figures are the ones the are produced from operators behind the filter for object 3803246 which does not match objects in each cycle.

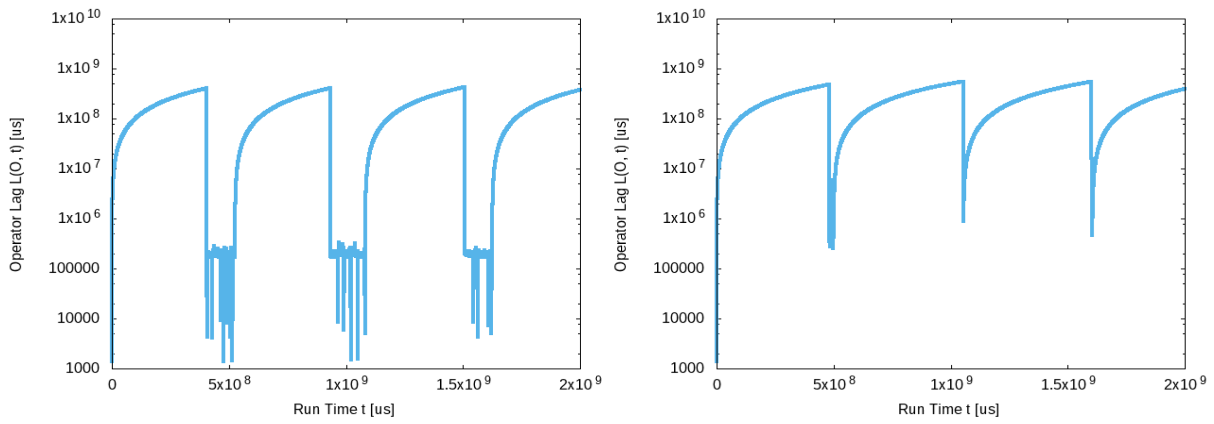


Fig. 4. Left: Source Operator lag. Right: Lag of an operator further down the processing line.

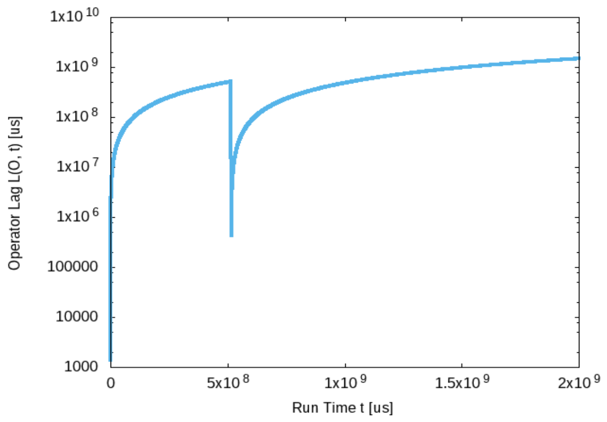


Fig. 5. Lag of an operator in the processing branch for object 3803246‘s filter.

The lag definition used until now is sufficient to understand the propagation of data within a DSMS. In order to get an idea of the general lag of the system at all times, a variant of the lag definition is introduced, the Heartbeat Lag $L_H(O,t)$. It not only takes into account the times at which data is output by an operator, but also the times at which so-called Heartbeats are output by an operator. Heartbeats are used to progress general time in the system – if no data input is given. For this simulation, REST Source Operator outputs one Heartbeat element per millisecond. This is also the reason why lags smaller than 1 ms can't be determined: If no data input is present, a general time update is done only once per millisecond and there is no accurate information during each millisecond. It is possible to increase the Heartbeat rate, but one must keep in mind that each element that needs to be processed by operators takes time to process. With a Heartbeat rate of 10 microseconds, for example, the Source Operator would produce 50 mio. Heartbeats for each MWG cycle, while only roughly 10-100 k actual elements would be sent by MWG to be processed per cycle. Fig. 6 shows the REST Source operator Heartbeat Lag in the first hour of simulation. As can be seen, there always exist a very small lag – even at the waiting times. However, it stays below 250 ms. The clearly identifiable blips indicate the MWG data input times in which the lag rises up, but does not surpass 500 ms. The figure also shows the lag of another operator far down in the processing line, again, representative of all other operators. The general volatility is higher and there is a minimum lag of roughly 5 ms. The lag, however, stays below 1 s. Fig. 7 presents the Heartbeat Lag over the whole simulation time. As can be seen, lag does surpass 1 s at times, but normally stays below 1.25 s and does not reach 1 s most of the time. The maximum delay of all operators over the whole simulation time was 2112769 microseconds, i.e. roughly 2.1 seconds.

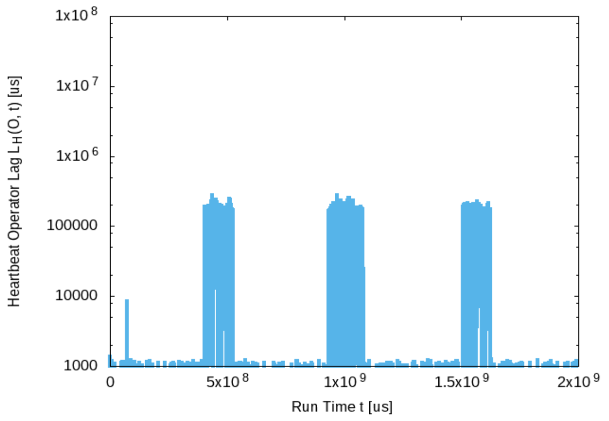


Fig. 6. Left: Heartbeat Lag of Source Operator. Right: Heartbeat Lag of an operator further down the processing line.

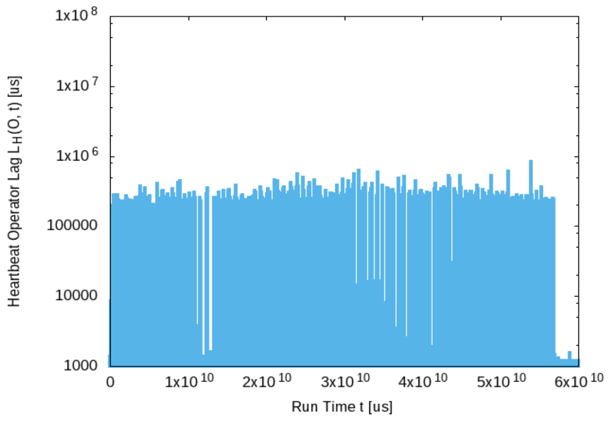


Fig. 7. Left: Heartbeat Lag of Source Operator over whole simulation time. Right: Heartbeat Lag of an operator further down the processing line over whole simulation time.

7. CONCLUSION

This paper presented a continuous near-real-time Radar measurement filtering approach using a Data Stream Management System (DSMS). The test scenario utilized a sophisticated Radar measurement generator (MWG) and the DSMS PICARD, both being in development at Institute of Space Systems of TU Braunschweig in Germany. In the simulated SST scenario, the detected objects’ orbits are determined without a-priori information as soon as each measurement arrives at the system. The elements are matched against Kepler Element bounds of two objects on a whitelist and are forwarded to a pre-existing SST Processing System (mocked by file system for this paper). From the perspective of the pre-existing Processing System, PICARD acts as a sensor – only that it is supposed to send solely those measurements which belong to objects on a whitelist. Based on the 1-meter MASTER population, the results show that measurement filtering is possible using a DSMS. The two objects were correctly matched and all tracklets of them were forwarded. It has to be noted though, that only about 37.8% of their measurements came through and that they made up only 4.5% of all measurements forwarded. This is due to the fact that one of the whitelist objects belongs to a constellation with very similar filter parameter values. It is assumed that by adding another parameter and lowering the inclination filter parameter value, most of the false-positives can be prevented. The number of measurements that a processing system “behind” PICARD needs to cope with was reduced by 99.3%. The processing delay stayed below 1 second most of the time with a maximum of 2.1 seconds over the whole simulation time, while handling 291.6 detections per second during data input.

ACKNOWLEDGEMENTS

The work described in this paper was achieved during the research project ”RSS” (Fkz.: 50 LZ 1404) funded by German Federal Ministry for Economic Affairs and Energy. All responsibilities for the contents of this publication resides with the authors. Also, a thank you is directed at Christopher Kebschull for the many discussions being had about the subject.

REFERENCES

1. Müller, S. and Stoll, E., Data Stream-Centric SST System Architecture Enhancement, Proceedings of the 1st IAA Conference on Space Situational Awareness (ICSSA), Orlando, FL, USA, 2017, paper IAA-ICSSA-17-00-01.
2. Müller, S., Kebschull, C. and Stoll, E., Evaluation of Data Stream Management Systems in the Context of Space Surveillance and Tracking, Proceedings of the 7th European Conference on Space Debris (ECSD), Darmstadt, Germany, 2017.
3. Arasu, Arvind, Babu, S. and Widom, J., The CQL continuous query language: semantic foundations and query execution. The VLDB Journal 15.2 (2006): 121-142.
4. Kebschull, C., Reichstein, L. and Stoll, E., A Simulation Environment to Determine the Performance of SSA Systems, Proceedings of the 18th Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS), Wailea, HI, USA, 2017.