Algorithm Preservation versus Data Preservation How to Manage Value-Added Processing in Earth Observation Payload Ground Segments

M. Boettcher, B. Pruin, C.Sommer, M. Wilkniss

Werum Software & Systems AG, Lueneburg, Germany
martin.boettcher@werum.de http://www.werum.com

Abstract. Besides long-term preservation and operational accessibility of data, the preservation of operational algorithms – called processors – is an issue in earth observation payload ground segments. This contribution describes this as a choice between systematic and on-demand processing, a corresponding processing infrastructure to support several processing scenarios, the model of an interface between data management systems and processors, and guidelines for processor implementation.

Introduction

Would it not be the simplest way just to store the raw satellite data products and to generate all higher level products on demand, on customer's request? This would also avoid the repeated reprocessing of all data when a new bug in the processor has been detected or when better parameters are available. On the other hand, there are these higher level products, people do it, they compute the data sets that they consider valuable systematically. There seem to be reasons for this as well as for the other.

Taking this as a starting point this paper argues for a well-defined but open processing infrastructure to support the preservation of processing systems, or more precisely, the preservation of their status of being operational. Especially in scientific environments it is often the case that operation is tied to developers of the processors and that operation stops when a developer resigns from its job. Knowledge of how to operate gets lost.

The question is how to improve this situation. One answer is: by good, long-living processor implementations and by a flexible and powerful environment for processors to allow processors to be kept simple. The paper will discuss these issues along the lines of the Data Information and Management System DIMS of the German Remote Sensing Data Centre DLR-DFD [1][5] and a solution that is going to be used for the new ESA Multi-Mission Facility Infrastructure MMFI in the FEOMI project [4]. DIMS has been developed by DLR and Werum Software & Systems AG and is operational since 2000 starting with CHAMP and SRTM. It has integrated about 30 different processing systems, among them algorithms for ERS, ENVISAT-VA, MODIS, METEOSAT, METOP, and TerraSAR-X. FEOMI will integrate ENVISAT, ERS and all third-party missions of ESA and is expected to be used for future missions.

Processing Scenarios

Processing in general fills the gap between low level raw data products (level 0) provided by satellite sensors and higher level products (level 1B, level 2, level 3 value-added products) requested by users. The generation of higher level products can be organised in two principally different ways or scenarios, i.e. by systematic processing or by processing on-demand.

Both of these scenarios are used in payload ground segments and both have their reasonable application. Two examples are systematic production of GOME L3 products, and on-demand production of MODIS L1B products.

Systematic Processing Example "GOME L3"

The Global Ozone Monitoring Experiment (GOME) level 3 is a product with global coverage that is computed from several days of level 2 input products that cover a single orbit each (Fig. 1).

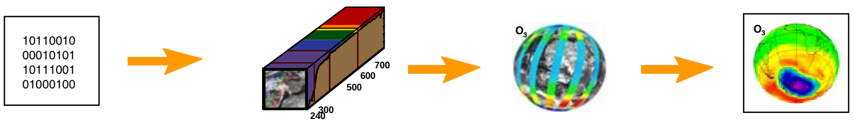


Fig. 1. Generation of GOME L3 products with global coverage from inputs with orbit coverage

To generate an ERS GOME L3 the processor composes e.g. 7 days of inputs using filter algorithms (Kalman, Rose). The generated L3 nominally gets a date at the middle of the compose interval. Daily, a new product is generated (or at least has been in the past) for the nominal date several days ago (Fig. 2).

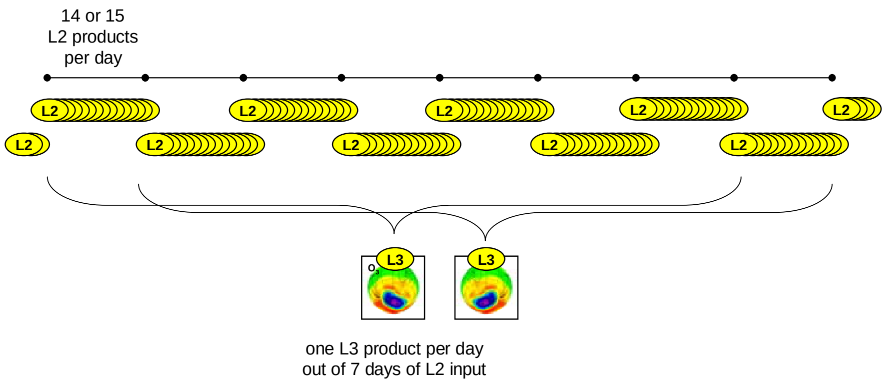


Fig. 2. Use of several days of L2 inputs for a single GOME L3 output

There are 14 or 15 orbits per day, such that about 100 L2 input products are used to generate a L3. For the next day again 100 input products are required, with an overlap of 6 days that can be cached in a processing system. Besides the fact that the GOME L3 is considered more useful than the L2, systematic processing requires only a 7th of input retrievals compared to production on request.

There are other examples of systematic processing. ENVISAT L1B and L2 products are systematically generated, archived and used for value-adding. The Shuttle Radar Topography Mission SRTM is another good example. Interferometric data sets have been used to systematically compute a digital elevation model (DEM) with about 130.000 tiles. There are constraints regarding the sequence of processing of a data take such that it would have been difficult to support production on demand for the DEM tiles, besides the large processing time and the amount of human expert interaction required.

Processing On-demand Example "MODIS Level 1B"

An example of production on demand is the MODIS Level 1B processing system (Fig. 3). The processor is able to process subsets of the L0 input such that a geo-region or the corresponding acquisition time interval can be specified by the customer in the production request. The processing system in addition retrieves suitable orbit and attitude files as inputs. It generates the L1B in a few minutes depending on the length of the selected time interval.

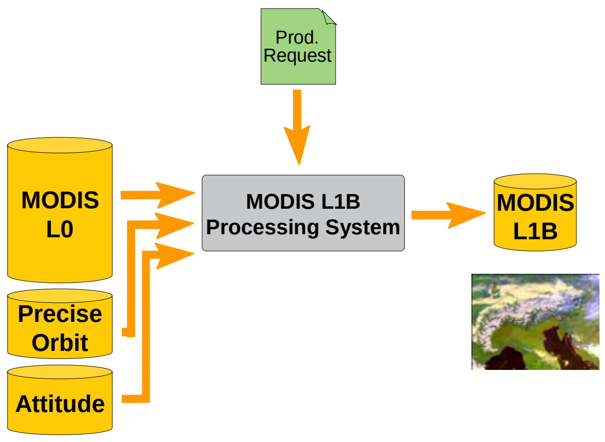


Fig. 3. Generation of a MODIS Level 1B from L0 on request

The result of processing is a delivery product that is delivered to the customer. It is not preserved. On the other hand preservation of the operational processing system must be ensured to provide L1B data in this scenario.

Comparing Systematic Processing and Processing On-demand

Table 1 lists features of the two production scenarios. The strangest feature of systematic processing is the amount of reprocessing done for many product sets in practical cases, which requires algorithm preservation, too. One of the strange features of processing on-demand is that it pretends to provide producible products but the actual generation may finally fail when it is requested, e.g. because of errors in the data or the algorithm.

Table 1. Advantages and disadvantages of systematic processing and processing on-demand

Systematic Processing	Processing On-demand
All high level products are generated *in advance* even if only some of them will ever be requested.	Only products that are requested will be produced.
High level products require additional *storage space* in the archive.	Only the low level products (L0) require space in the archive.
High level products are offered as *existing*, they may be *quality checked* before they are offered.	High level products are offered as *producible*, but *generation may fail.*
High level products can immediately be delivered.	*Processing time* delays delivery at time of request.
Standard processing options must exist. The results of processing are standard reference products.	Individual *processing options* from user requests can be supported. The results of processing may be customized delivery products.
Subsequent *processing chains* and *subscriptions* on the high level product can be supported	The same product may have to be regenerated if several customers request the same.
Reprocessing is required whenever better algorithms or parameters are available.	Better algorithms or parameters can be used as soon as they are available.
Computing power is required for processing of all products and for reprocessing.	Computing power is required for the amount of requests.
Preservation of high level products is required.	*Preservation of an operational algorithm* is required beyond end of the mission.

Generic Functions of Processing Systems

In the introduction it has been stated that the preservation of operational algorithms can be improved by a powerful environment that allows processor implementations to be simple and thus maintainable.

There are certain functions of processing systems that reappear in many of them. Instead of implementing them individually they can be extracted to a generic component that becomes part of every processing system (Fig. 4).

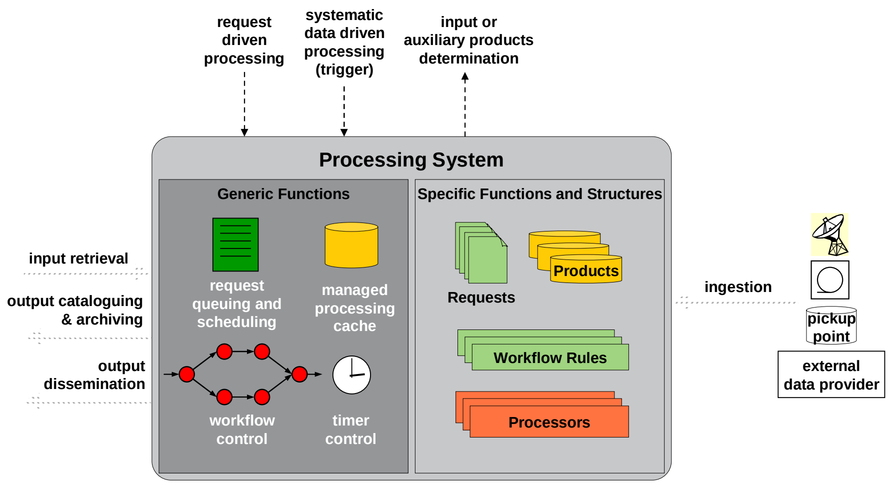


Fig. 4. Generic functions of processing systems

- request queuing

  Processing systems receive production requests from other systems of the payload ground segment (especially a production control facility), from an operator, or as batch file. Request queuing implements the multi-request capability of processing systems, freeing processors from this requirement.
- processing workflow scheduling

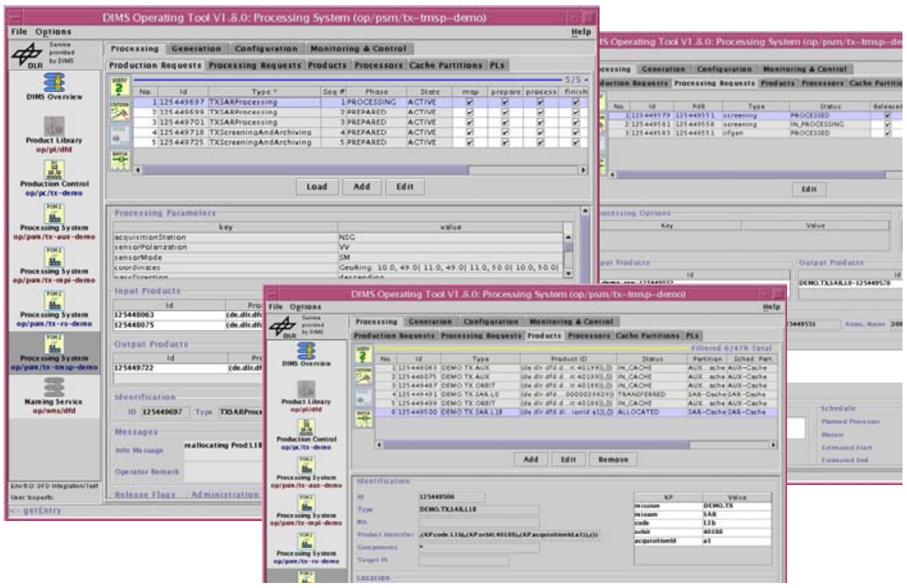
  Processing systems range from simple single-step single-processor systems to systems with complex parallelized workflows. Workflow scheduling organizes processing workflows by rules in a request-dependent way. There is a rule API or configuration capability for processing system specific extensions and for dynamic control. This function keeps processors free from the burden of step scheduling, makes processing progress visible to operators, and allows interrupting processing between the steps that are considered as transactions.
- product handling and cache management

  Input retrieval from the archive and cache management provides processors with their inputs and working directories and handles their outputs. It feeds outputs into the next processor of a chain or archives them. It allocates space in cache before processing and cleans up afterwards. This function simplifies processors that access products as files and directories. This decouples them from the product interface of the data archive and inventory.
- processor charging and control

  The generic functions of a processing system controls processing and therefore needs adapters to processors that may be implemented as executables, shell scripts, more and more as Java classes, or as shared libraries implemented in programming languages like C/C++ or Fortran. The generic functions adapt to the processor interface, start processors, provide them with an environment and parameters and handle results and exceptions.

- uniform operating

  Though processing systems differ in their processors and the steps executed, it is a function of the generic part to provide uniform views, to some extend uniform behavior, uniform request control and uniform administration and startup for all kinds of processing systems. This is an advantage for operations.



No.	Id	Type	Seq #	Phase	State	map	prepare	process	finish
1	125449697	TXSARProcessing	1	PROCESSING	ACTIVE	✓	✓	✓	✓
2	125449699	TXSARProcessing	2	PREPARED	ACTIVE	✓	✓	✓	✓
3	125449701	TXSARProcessing	3	PREPARED	ACTIVE	✓	✓	✓	✓
4	125449718	TXScreeningAndArchiving	4	PREPARED	ACTIVE	✓	✓	✓	✓
5	125449725	TXScreeningAndArchiving	5	PREPARED	ACTIVE	✓	✓	✓	✓

Processing Parameters

key	value
acquisitionStation	NSG
sensorPolarization	VV
sensorMode	SM
coordinates	GeoRing: 10.0, 49.0| 11.0, 49.0| 11.0, 50.0| 10.0, 50.0|

Input Products

Id
125448063
125448075

Output Products

Id
125449722

Filtered 6/479 Total

No.	Id	Type	Product ID	Status	Partition	Sched. Part
1	125448063	DEMO.TX.AUX	[illegible]	IN_CACHE	AUX-Cache	AUX-Cache
2	125448075	DEMO.TX.AUX	[illegible]	IN_CACHE	AUX-Cache	AUX-Cache
3	125449487	DEMO.TX.ORBIT	[illegible]	IN_CACHE	AUX-Cache	AUX-Cache
4	125449491	DEMO.TX.SAR.L0	[illegible]	TRANSFERRED	SAR-Cache	SAR-Cache
5	125449499	DEMO.TX.ORBIT	[illegible]	IN_CACHE	AUX-Cache	AUX-Cache
6	125449500	DEMO.TX.SAR.L1B	[illegible]	ALLOCATED	SAR-Cache	SAR-Cache

Identification

Id: 125449500
Type: DEMO.TX.SAR.L1B

KP	Value
mission	DEMO.TX
mission	SAR
code	L1b
orbit	40188
acquisitionId	a1

Fig. 5. Example views of requests, products and processing steps of a DIMS processing system

- distribution and load balancing

  Processing systems get more complex and requirements are more challenging. The distribution of processing load to several hosts is a function that is increasingly used in processing systems.

What remains as specific adaptations are (and must be) done as workflow rules and processor implementations. Specific parts of the system at runtime are the individual requests and products.

In the Data Information and Management System DIMS a component named Processing System Management (PSM) [2] implements the functions listed above. It is a small workflow engine with additional functions adapted to the task in the environment of other components like Operating Tool as GUI, Product Library [3] as inventory and archive, and Production Control.

Among the functions supported by the PSM are also the supported scenarios, usually not all of them in the same system, but used as generic patterns:

- *ingestion*

Ingestion transfers data from external sources as products into the Product Library. These products are then available to other processing systems and to ordering and delivery. External sources may be receiving station systems, external system including legacy applications to be replaced by the new data management system, tape drives and pickup points of FTP servers that are systematically scanned. Polling and listener models of ingestion can be supported. By implementing them in processors the solution is decentralized and open to new interfaces of additional types of systems.

In ingestion scenarios the PSM starts the ingestion processor, cares for cache space, starts additional processing steps, handles exceptions, and archives the products. Typical processing steps of ingestion systems are input verification, metadata extraction and quicklook generation.

- *systematic processing*

The PSM supports systematic processing by subscription/trigger for new products in the Product Library (e.g. L1 from all inserted L0) and by timers and queries (e.g. daily composites of atmospheric products). On trigger or timer events the PSM determines and retrieves new products, initiates processing and archives the outputs. For systematic processing the PSM ensures seamless product generation even after maintenance down times.

- *on-demand production and post-processing*

PSM supports request-driven processing. Production requests are typically submitted by Production Control to PSM. Requests either stem from a user order or they are part of a production chain controlled by Production Control. Requests specify inputs, processing parameters and optionally the desired output. PSM retrieves the input products from the Product Library, charges processors and usually stores outputs in the Product Library again. PSM signals status back to Production Control. The output can be a reference product, or it is a delivery product as output of post-processing only stored for delivery.

- *near-realtime production and delivery*

By chaining ingestion or systematic processing with post-processing, the PSM is able to run autonomous near-realtime processes requiring no intermediate external resources. The near-realtime processing chain can include a final online delivery step transferring generated delivery products to an FTP pick-up point. Processing results can additionally be stored in the Product Library without affecting the near real time process.

Making these functions generic simplifies processors to filter programs with files input and files output. Whether they are used in systematic processing scenarios or for processing on demand makes no difference. Note that integration of processing systems is not for free. There is an effort of reaching the status of being operational, e.g. for workflow definition, product modeling in the inventory, and processor interfacing.

An Interface Model for Processors

Often processing algorithms are implemented already before their integration into an operational processing system is planned. Secondly, there are many ways to implement a processor. Consequently, the generic part (PSM) should provide adapter patterns how to integrate processing algorithms. Fig. 6 shows different types of adapters currently used with the PSM.

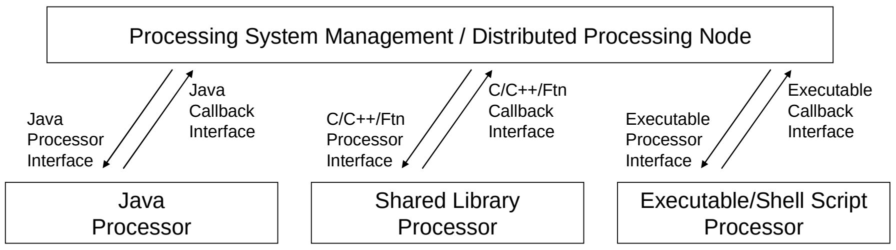

Processing System Management / Distributed Processing Node
Java Processor Interface
Java Callback Interface
C/C++/Ftn Processor Interface
C/C++/Ftn Callback Interface
Executable Processor Interface
Executable Callback Interface
Java Processor
Shared Library Processor
Executable/Shell Script Processor

Fig. 6. Variants of processor adapters

The Java interface means that the processor is implemented in a class with a method with parameters. The method starts the processor for one request. The parameters transfer information about inputs, outputs and processing parameters to the processor as a processing request upon start of processing of a product. Shared library means a function in a compiled and bound software library loaded into the processing system. Executable or shell script means a program executed as a child process with information about inputs, outputs and processing parameters provided as command line argument, environment variables or in a processing request file.

The interaction between PSM and processor is not complete with its start. A principle sequence (Fig. 7) comprises intermediate status callbacks from the processor to the PSM and optional command calls from the PSM to the running processor. Finally the result of processing is returned to the PSM by finishing the start function. The return value usually only distinguishes success or failure. Processing results are provided as output products.

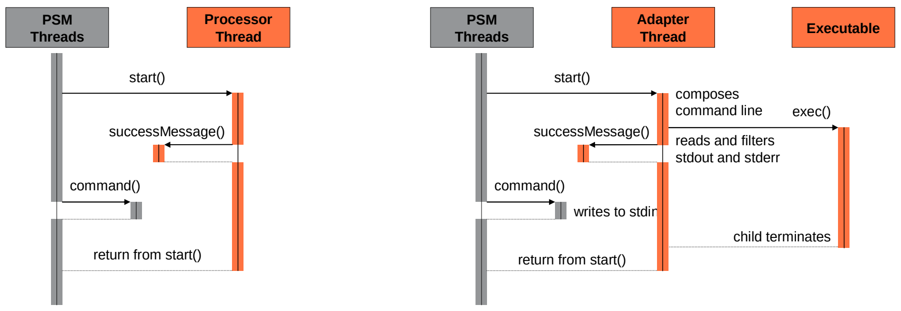


Fig. 7. Processor call interface for in-process execution or child process generation

The right side of Fig. 7 depicts the interaction in case of executables or shell scripts. Here, callbacks and commands may be exchanged via stdin and stdout which unfortunately is not trivial.

The interface between the generic PSM and the processors or more generally between the generic functions of a processing system and the processors is defined by the call interface including signatures and the product interface. There could be said more about the call interface e.g. to support ingestion which requires additional methods but it is dropped as a detail here. The product interface defines the structure of the data products. It is an interface between a processor and its environment. The products are referenced as paths to directories in the methods and functions of the call interface. Fig. 8 depicts this for inputs, outputs and intermediate working directories for a single request.

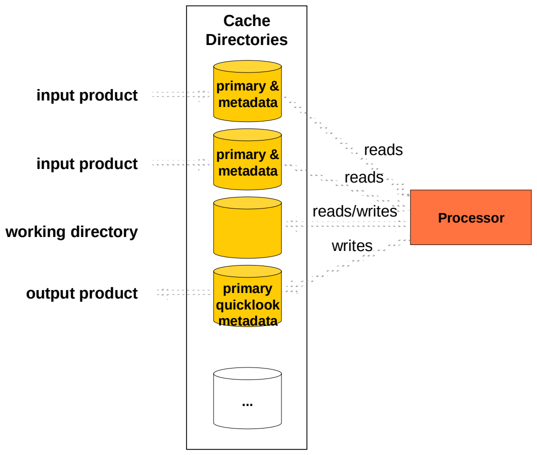


Fig. 8. Processor's file interface to products (example with two inputs and one output)

There are at the same time other directories in the cache for other requests. The structure and content of the directories is the product interface for processors. It must be defined. It should be flexible enough to support different formats, different number of files and different amount of metadata accompanying each product. An example of this is the product model of the DIMS Product Library with its Item Information File as exchange format for metadata and as index to contained components and files [3].

There is a need for such models because

- products are exchanged between different processors, one's output is the other one's input
- products are long-term stored in the product library
- products are searched for, which requires a certain amount of meta data to be available about each product

A product model is commonly developed together with the processing system that generates this type of product. Though sometimes not defined explicitly also intermediate products and working directories have an implicit model. It is the agreement between the processors that write and read from such products what they shall generate or can expect to find in a product directory.

Guidelines for Processor Implementation and Integration

Cancellable Processors

A processor is cancellable if it reacts on the cancel command by interrupting processing. It is a desirable feature of processors with sensible execution times (more than a second).

The cancel command is to be implemented in the command method of the processor implementation. Because processors are executed in separate threads or processes there are in principle no secure ways to abort them immediately by the command method executed from another thread. The reason for this is that the processor thread may have locked resources that are never released in case of aborts. Therefore there is a proposed implementation pattern that uses a semaphore and notification for synchronization.

This way the processor aborts gracefully on a cancel command by terminating where the implementation has foreseen it. Note that the delay between the command and the reaction of the processor will in some cases not suffice to abort a processor gracefully when the PSM is shut down.

Restartable Processors

A processor implementation is restartable if wherever it has been aborted it can be started again with the same parameters on the same inputs, working directories and outputs and it will be able to succeed in processing. This does not require that partial results of the aborted run are necessarily used. It simply means that the modifications done before does not hinder a restart.

There are features of processor implementations that support this property:

- initial cleanup: Processors should be prepared to find files in their output directory from preceding runs with the same parameters that had been aborted. The processor should not fail by IOException when creating a file just for the reason that there is a file with this name already. It should overwrite it.
- no modification of input files: The processor should usually not modify an input but write to an output instead. If this must be done for any reason the processor should expect that modifications may have been done partially before.
- look for newly generated products: If the processor uses a cache allocation callback method for ingestion it should check whether the product created is among its parameters already. It should usually be re-used in this case.

To be restartable is a strongly recommended property of a usual processor. For ingestion processors it is not required to be restartable in this sense. Instead, they must ensure that an inspected product is first approved by the corresponding ingestion callback method and then marked as being ingested, e.g. by removing it from the pickup point.

Re-entrant Processors

A processor implementation is re-entrant if it is save to run it concurrently without mutual disturbance. The important prerequisite that inputs and outputs of several concurrent processors are separated in the cache is assured by the PSM. This property is often easy to achieve, leads to cleaner implementations, and allows running the processor concurrently on different requests.

To be re-entrant the processor implementation should

- use re-entrant versions of libraries and functions
- avoid static global variables
- do not write to places except for those provided as output directories

Processors that write anything to their local processor software directory are candidates to miss this property.

To summarise this, Table 2 lists some practices how to make processor implementations short-living. The list is non-exhaustive.

Table 2. Guidelines for writing bad processors

Features of processor difficult to maintain
• writes intermediate files into local directory of the processor implementation • runs for hours without possible breakpoints • crashes if it is restarted without prior cleanup • runs for hours without progress feedback • fails without error handling • generates lots of debug output in operational mode • requires some files from the implementers home directory

References

1. Mikusch E., Diedrich E., Göhmann M., Kiemle S., Reck C., Reißig R., Schmidt K. Wildegger W., Wolfmüller M.: The Data Information and Management System for the Production, Archiving and Distribution of Earth Observation Products. In *Data Systems in Aero-Space (DASIA)*, EUROSPACE, Montreal, 2000
2. Boettcher M., Reissig R., Mikusch E., Reck C.: Processing Management Tools for Earth Observation Products at DLR-DFD. In *Data Systems in Aero-Space (DASIA)*, EUROSPACE, Nice, 2001
3. Kiemle S., Mikusch E., Göhmann M.: The Product Library – A Scalable Long-Term Storage Repository for Earth Observation Products. In *Data Systems in Aero-Space (DASIA)*, EUROSPACE, Nice, 2001
4. Pinna G. M., Mikusch E., Bollner M., Pruin B.: Earth Observation Payload Data Long Term Archiving - The ESA's Multi-Mission Facility Infrastructure. PV 2005, ESA ESRIN, Edinburgh, 2005
5. Kiemle S., Bilinski C., Buckl B., Dietrich D., Kroeger S., Mikusch E., Reck C., Schmid F., Schroeder-Lanz A.-K., Wolfmueller M.: Data Information and Management System for the DFD Multi-mission Earth Observation Data. PV 2005, ESA ESRIN, Edinburgh, 2005