SpaceOps Conferences
5-9 May 2014, Pasadena, CA
SpaceOps 2014 Conference

Evolving Mission Control System Infrastructure for an Altering Fleet of Spacecraft

Robert Messaros
Siemens Convergence Creators GmbH, 1210 Vienna, Austria

Christian Stangl
German Space Operations Center (GSOC), Oberpfaffenhofen, 82234 Wessling, Germany

and

Michael Oswald
Siemens Convergence Creators GmbH, 1210 Vienna, Austria

Space Operations of DLR (Deutsches Zentrum für Luft- und Raumfahrt) provides support to DLR's various missions using its established operational concepts from its mission control centre (GSOC, German Space Operations Center). This paper reports on the successful upgrade and development strategy along the path of technological evolution of DLR's Mission Control and Check-Out infrastructure. This includes a discussion of the main influencing variables and DLR's strategy for mitigating negative impacts.

I. Introduction

Starting in 2000 DLR established a generic infrastructure based on SCOS-2000, supporting a long row of missions to this date and beyond. During this long path of evolution several significant adaptations were added, complemented by a general evolution for the Mission Control System infrastructure baseline upgrade, demanded by the ubiquitous drag due to changes in computing hardware, Operating Systems, Customer Off-the-Shelf (COTS) software, and general software technology.

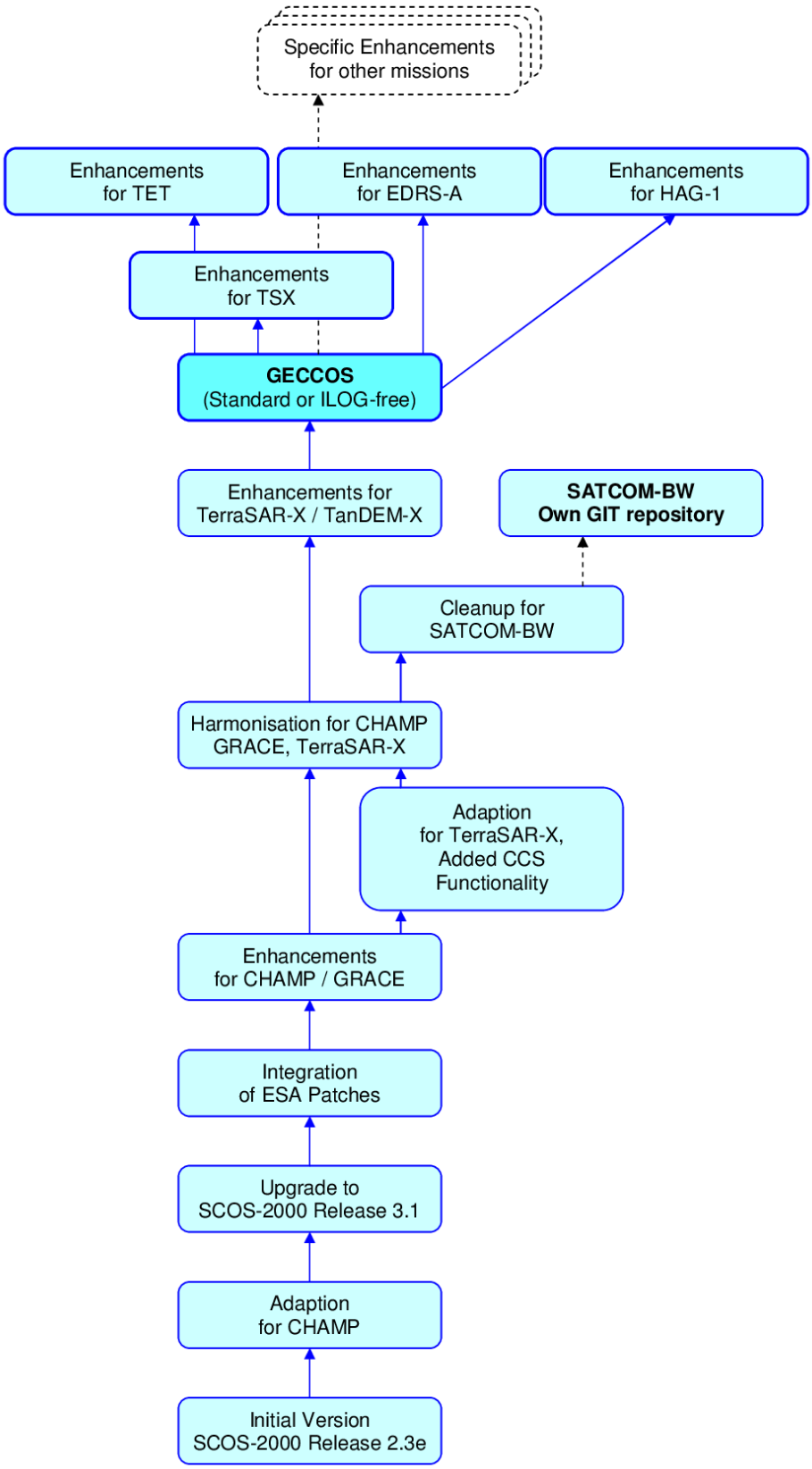

Figure 1. Main evolution steps in the DLR Mission Control System Infrastructure

II. Main Revision History

The main steps of the Mission Control System infrastructure evolution are depicted in Figure 1. It comprises the period of 2000 to 2014 (GECCOS), with further enhancements already planned for future missions.
Initially starting point was on a plain SCOS-2000 version from ESA, namely in its version 2.3e incarnation. This version had been extended and enhanced for DLR's needs into the system for the CHAMP mission, see Figure 2.

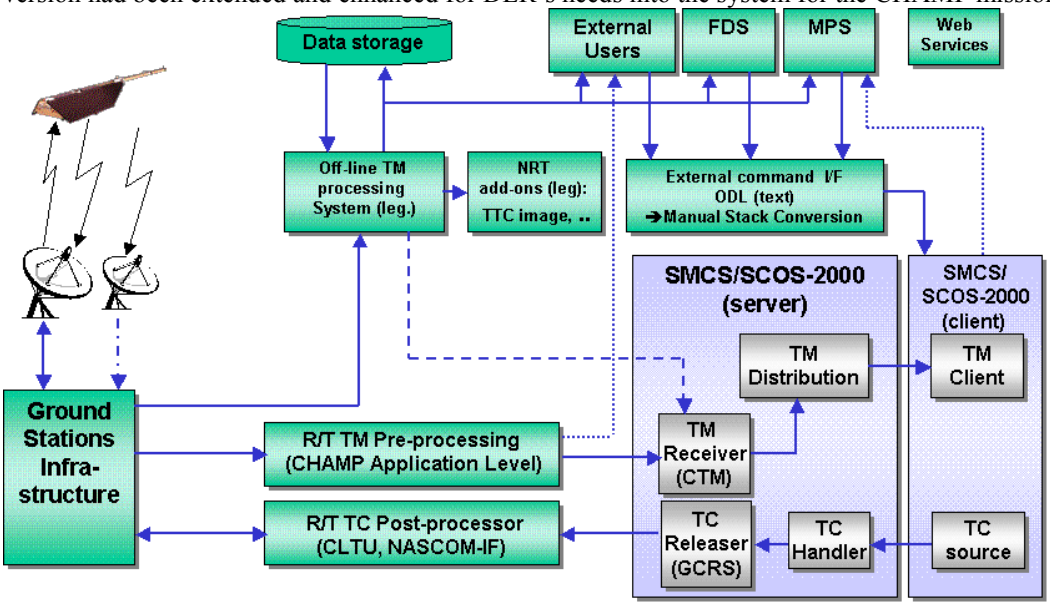

Figure 2. DLR Mission Control System configuration for the CHAMP mission

Elements drawn in blue shades represent the SCOS-2000 based Mission Control core system, while elements drawn in green shades denote proven legacy elements, which were continued to use.
Functional updates, and error corrections were fed back to the main SCOS-2000 development line, similar fixes done in the main branch of ESA's SCOS-2000 were fed back to the Mission Control infrastructure by updating the base system to the level of SCOS-2000 release 3.1. All this formed the basis for upgrading the ongoing CHAMP operations and the development of the GRACE Mission Control System. The latter required further modifications and enhancements, albeit on a minor level.

Subsequent evolution brought a reduction of these legacy elements, replacing them with their adapted SCOS-2000 counterparts. This is shown for the TerraSAR-X mission in Figure 3 below.

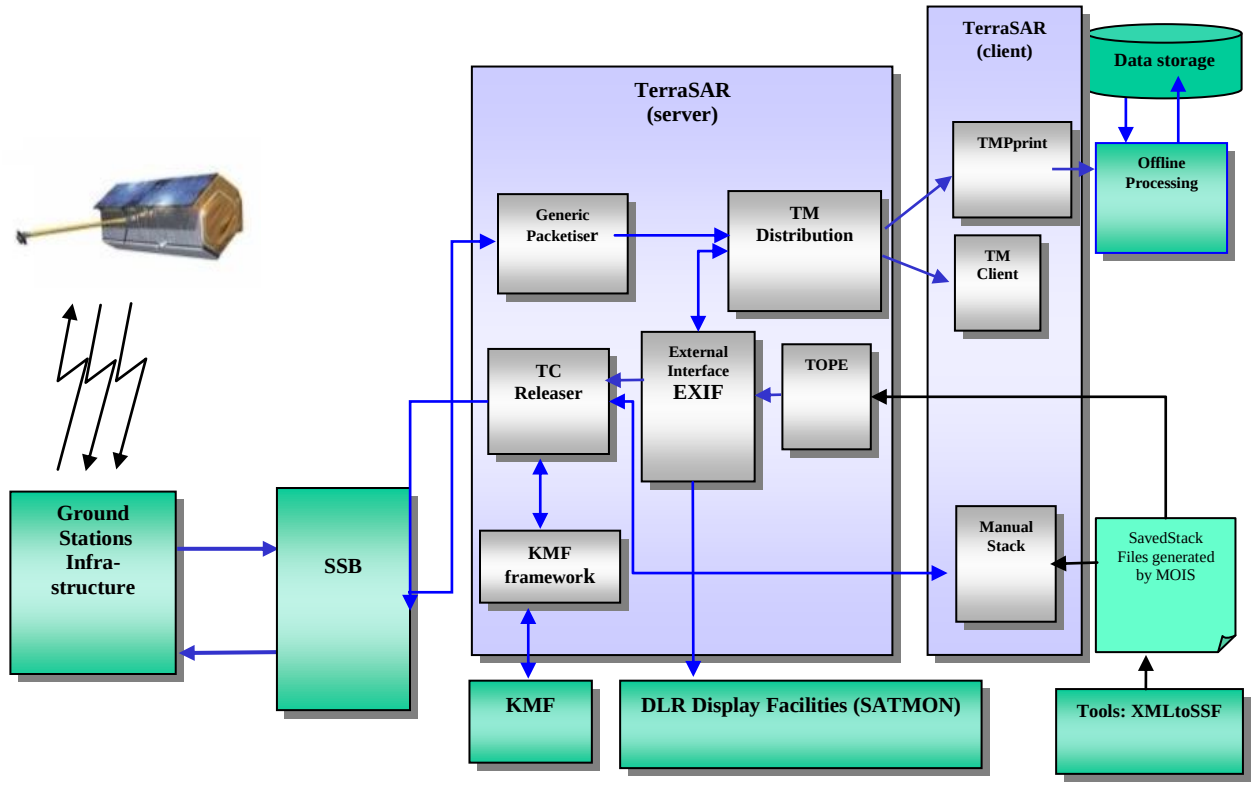

Figure 3. TerraSAR-X configuration with new TM/TC interfaces on packet and on discrete levels

A further enhancement for the TerraSAR-X was the integration of the Central Checkout System with the Mission Control System. This allowed for a continued utilization of the same system for checkout and operations, yielding not only a validated spacecraft database, but also a vast set of validated Flight Control procedures, validated during AIT, in advance of the spacecraft launch.

Development and early operation of TerraSAR-X occurred in parallel with the ongoing CHAMP and GRACE operations. In a proactive step the code-basis for the missions was harmonized, such that diverging code-branches could be avoided. The resulting harmonized code-basis was used as the starting point for the ongoing operations, as well as for the further development for SATCOM-BW, a classified mission with its own branch.

Main development continued with updates for supporting multi spacecraft missions, which can be operated from a single Mission Control System. The overall system kept the core elements separate for individual spacecraft, a deliberate step to avoid increasing overall complexity. The operational context was unified on the level of the Mission Planning System.

Said enhancements were used for the GRACE mission (two spacecraft right from the start) and for the TerraSAR-X & TanDEM-X combined mission (launched in 2007 and 2010, respectively).

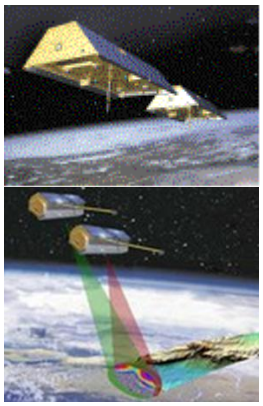

Figure 4. GRACE and TerraSAR-X TanDEM-X dual spacecraft missions

Based on this version the current development foresees a further proactive development cycle, which is targeted at removing legacy COTS elements, which are getting (too) close to their end-of-life from the code-base, and which therefore are prone to reduce the maintainability of the overall code-basis.

This GECCOS system forms the basis of all continued operation of existing missions – missions in their operational phase just as well as missions prepared for launch – and likewise upcoming missions. Recent systems, which were created with Siemens's involvement, are the Mission Control Systems for TET-1, EDRS-A, EDRS-C, and HAG-1, as well as TerraSAR-X enhancements. Future systems will be for EDRS-C, PAZ, BIROS, GRACE-FO, EnMAP, NMAP, and H2Sat (Heinrich Hertz).

III. Main contributors to the overall update drag

The idea of establishing a generic base-system, which can be used without changes over a long period of time, is generally understandable, but unfortunately quite unrealistic. There are several factors, which in the long run dictate an approach of constantly updating the overall system, even in cases, where no additional functionality is gained.

The main observed contributors in this respect are discussed in the following.

A. Outdating of Platform Hardware

The low-level hardware elements of the computing platform being used constantly change. Typically chipset generations are released these days every 14 months, with older products disappearing from the market quickly. This results in the necessity to update at least the operating system kernel, Linux in the case of DLR's systems, in order to cope with this hardware evolution. Virtualization only provides a partial solution, creating other issues (see point H below).

B. Outdating of Platform Software

Even when spending the extra effort of maintaining separate kernel versions the kernels will from time to time require to use updated compiler versions, Operating System tools, and Operating System libraries. Consequentially this will result in a demand to upgrade to an up-to-date version of the Linux platform. Again, virtualization only provides a partial solution, creating other issues (see point H below).

C. Outdating of Customer Off-The-Shelf (COTS) Software, license issues

A similar drag is exerted by the COTS software, though with fewer possibilities of mitigating it effects. While initially the benefits of COTS software is appreciated and the compromise of trading in externally provided functionality against license fees, typically the final stages see the COTS software as a foreign body in the overall system, with little to no maintenance by its supplier and often for horrendous license fees or even licenses no longer being sold at all. With much-needed adaptations by the supplier to new compiler versions and library versions getting out of the picture cost-intensive re-implementation of that functionality needs to be undertaken, which originally had been needed at the time of inclusion of the COTS, often accompanied with significant overall architectural modifications, which in turn need further effort down the tree of dependencies.

License agreements covering the COTS in source code, which could be maintained and recompiled by the customer whenever needed, are rare in the industry, and if provided, they usually come at exorbitant cost.

D. Changes in Platform Mechanisms

In order to exploit the established multi-core computing hardware to the maximum extent possible, the system architecture was adapted following a multithreaded approach. Particularly the threading and synchronization infrastructure inside the Operating System kernel and the support libraries changed at a notable rate. This frequently resulted in architecture level updates.

E. Mission specific updates, spacecraft platform specific updates

Supporting another spacecraft bus or supporting new mission types, typically requires some updates to the core system. Wherever possible such changes were done to the main development line. Where this was not feasible the architectural changes were fed back at a later phase, in order to settle diverging development branches into a single entity.

F. Functional upgrades

Adding new functions, which were not available before, or re-implementing functions previously supplied by systems, which already had reached their end-of-life, requires careful planning in order avoid interfering with the overall release scheduling. The preferred points for these functional upgrades were those periods, which were deliberately reserved to overall mending of the system baseline.

G. Upgrades due to changes of Operations Concepts

Improving and optimizing the overall usage concepts is a continuous effort, particularly with increased mission demands and spacecraft bus peculiarities. The main are of such upgrades concern Human Computer Interfaces, many times demanding further architectural updates. Such upgrades were therefore particularly well planned, in order to avoid too many changes at the same time, while at the same time permitting operation of missions with both, the established system version, as well as with the upgraded system version. This possibility of stepping back always provided a high level operational immunity against newly introduced malfunctions.

We want to exemplify this by the experience gained with updating the TerraSAR-X system to the dual spacecraft TerraSAR-X & TanDEM-X system. The spacecraft perform a formation flight at a radial distance of 250–500 m at a velocity of about 7 km/s, which mandates performing many contacts in rapid succession and careful monitoring of the spacecraft orbital parameters. Loosing several contacts in quick succession could mean total mission loss. This was avoided by keeping backup single-mission systems available in cold standby for rapidly taking over mission control in case of problems.

H. Virtualization side effects

Virtualization allows to abstract away much of the platform hardware induced drag, yet not all of it. DLR bases its hardware infrastructure on VMware ESXi, a bare-metal hypervisor running its own microkernel (vmkernel). On top of the provided basic infrastructure one or more guest systems can be run, each of which runs on an abstracted virtual hardware with configurable number of CPUs, memory, disks, and so on. While this has the benefit of mitigating the drag induced by platform hardware and software (see points A and B above), new problems have been induced.

One such problem is that the virtualized system clock can get consistently out of sync (SUSE Linux Enterprise 11, 32bit version). Therefore the 64bit version host system had to be used, however to the effect that some utility programs with graphical user interfaces cannot be run using remote displays. Furthermore the build process cannot be done on the 64bit system itself, it needs to be done on a separate 32bit system and the produced items need to be transferred to the 64bit system.

IV. Upgrade Methodology

In order to achieve the envisage goals, the development within DLR's Mission Control System infrastructure adopted an Agile Software Development approach, inspired by Scrum. In this set-up DLR plays the role of the Product Owner, defining the business value of the Iteration/Sprint and updating the Product Backlog. The development team then performs all involved activities needed to produce the Shippable Items of the Iteration/Sprint. This work effort comprises analysis, design, development, testing, and documentation activities.

V. Conclusion

DLR looks back on 14 years of successful mission operations, using the evolving GECCOS system. Right from the begin the elements of change, upgrade, and extension have been incorporated by adopting an iterative and incremental approach. This overall evolution strategy deliberately foresees mending steps, allowing to re-unify the software basis to the benefit of continued development, with many missions lying ahead.