BOSS ein verlässliches Open-Source-Betriebssystem

Sergio Montenegro, Felix Holzky
www.first.fhg.de/~sergio
FhG FIRST
Kekuléstr 7, 12489 Berlin

Abstrakt

BOSS basiert auf einer Grundregel, die die Welt vor langer Zeit vergessen hat: Einfachheit!

Wir entwickelten ein Echtzeitbetriebssystem für sicherheitskritische Anwendungen wie zum Beispiel Satellitensteuerungen und medizinische Geräte. Unsere Erfahrungen zeigen: Komplexität ist der größte Feind der Sicherheit. Wer Sicherheit benötigt, sollte nur das benutzen, was man auch schnell und klar verstehen kann. Dies war die Philosophie bei der Entwicklung von BOSS. Erstens: Erstelle alles so einfach wie möglich. Zweitens: Benutze moderne Framework-Technologie, um die Komplexität der Anwendungen zu reduzieren. Drittens: Benutze Komponenten-Technologie, um die verbleibende Komplexität bewältigen zu können.

Das Resultat ist sehr vielversprechend. BOSS arbeitet bereits seit Jahren ohne Unterbrechung im Weltraum (Satellit BIRD) und in medizinischen Geräten. Und selbst komplexe Funktionalitäten können mit BOSS sehr einfach implementiert werden.

1. Einleitung

BOSS wurde entworfen als Grundgerüst für sicherheitskritische Betriebssysteme, die sehr einfach von verschiedenen Organisationen zertifiziert werden können.

Da Komplexität der größte Feind von Sicherheit ist, wurde von Anfang an darauf geachtet, BOSS so einfach wie möglich zu halten. Dadurch ist es einfacher zu verstehen, zu überprüfen, zu benutzen, usw. Der komplette Kernel füllt gedruckt nur wenige Seiten.

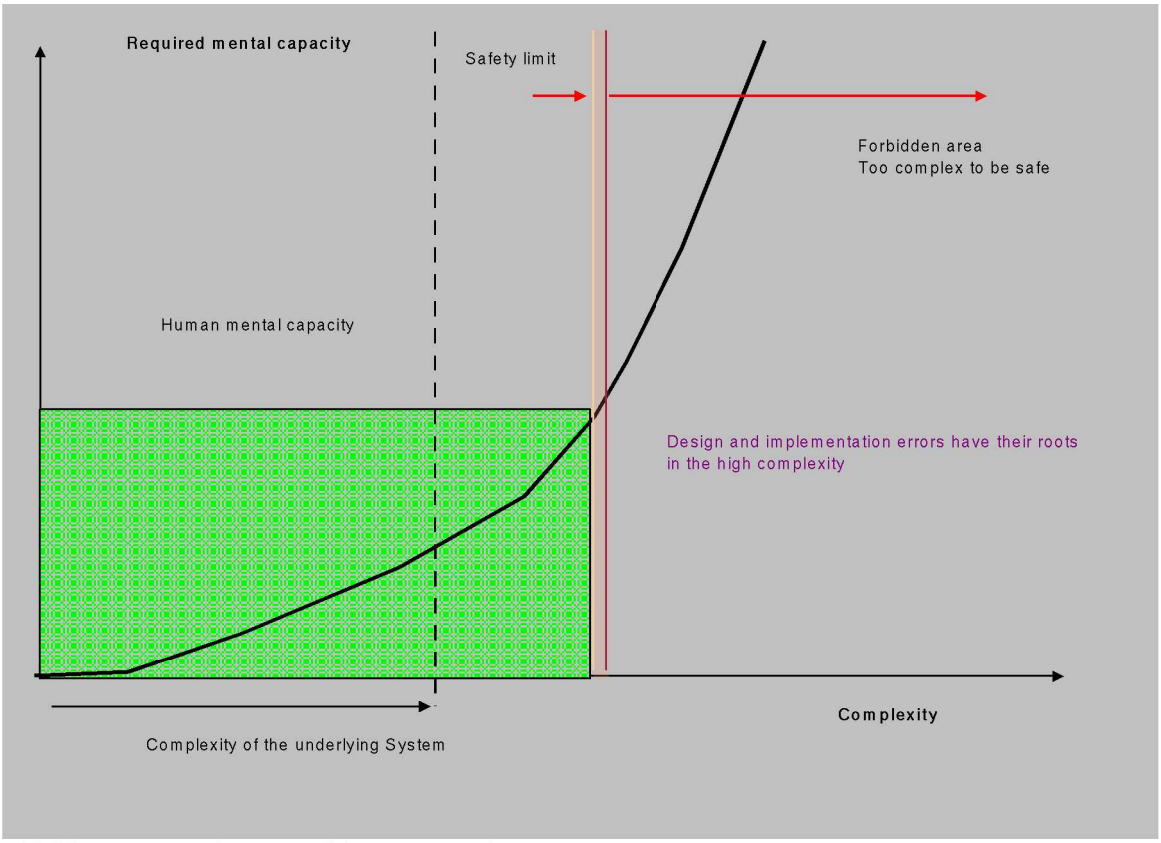

Abbildung 1: Komplexität muß begrenzt werden.

Einige Teile von BOSS sind bereits mathematisch und formal durch Modellchecker und Theorem-prover verifiziert worden. Mit dem aktuellen Stand der formalen Verifikation lässt sich die Korrektheit komplexer Systeme (noch) nicht formal beweisen, die von BOSS jedoch schon. BOSS basiert auf wenigen, sehr einfachen Basisfunktionen, die alle sehr genau bewiesen werden können. Und diese Funktionen werden von nahezu allen Kerneloperation benutzt. Außerdem ist BOSS open-source, so dass es sich jeder anschauen, anpassen und auf andere Hardwareplattformen portieren kann.

2. Als Beispiel: Satellit BIRD

Kleinsatelliten haben hohe Herausforderungen zu bewältigen: Hohe Performanceanforderungen trotz geringer Ausstattung und vor allem trotz kleiner Budgets. Von allen Aspekten ist die Kostenfrage eine der wichtigsten Antriebe für Kleinsatellitenmissionen. Um die Kosten innerhalb des niedrigen Rahmens zu halten, ist die Demonstration neuer, nicht weltraumzertifizierter Technologien für die Weltraumsonden einer der Schlüsselpunkte für die Erfüllung der hohen Performanceanforderungen. Unter Berücksichtigung dieses Aspektes musste der Kleinsatellit BIRD (Bispectral Infra-Red Detection) der DLR seine hohe Leistungsfähigkeit Aufgrund der Benutzung neuartiger Technologie - einschließlich moderner Softwaretechnologie - demonstrieren. Das Steuerungssystem von BIRD basiert auf dem Echtzeitbetriebssystem BOSS. Der

Satellitenbus wird von einen zuverlässigen Bordcomputer gesteuert. Um hohe Zuverlässigkeit, Sicherheit und Lebensdauer zu gewährleisten, wurde der Bordcomputer aus vier identischen Rechnern aufgebaut. Wie in Abbildung 2 gezeigt, sind alle redundanten Knoten und alle vom Bordrechner zu steuernden Geräte des Satelliten durch verschiedene Bussysteme mit unterschiedlichen Protokollen miteinander verbunden.

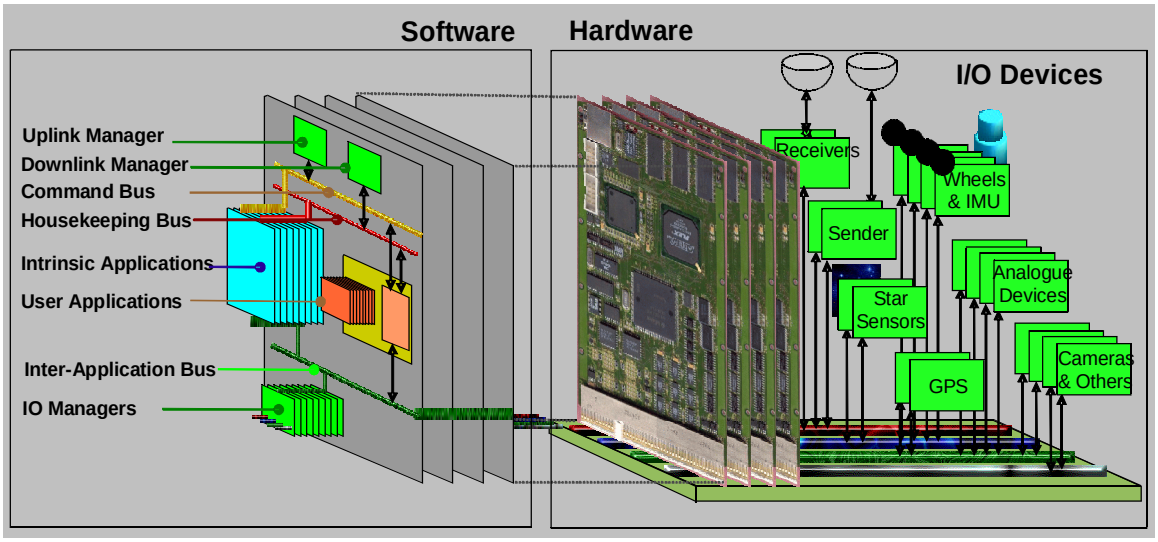

Abbildung 2: Redundanzmanagement

Die Architektur der redundanten Steuerrechner ist total symmetrisch, das bedeutet, dass jeder dieser Knoten imstande ist, alle Steuerungstasks auszuführen. Ein Knoten (*worker*) kontrolliert/steuert den Satelliten, während ein zweiter Knoten (*supervisor*) den korrekten Arbeitsablauf des *worker*-Knotens kontrolliert. Die beiden anderen Knoten sind Reservekomponenten und abgeschaltet. Sobald eine Unregelmäßigkeit des *worker*-Knotens durch den *supervisor*-Knoten festgestellt wird, übernimmt der *supervisor* die Rolle des *workers* und kontrolliert den Satelliten. Der alte *worker*-Knoten wird gezwungen eine Wiederherstellungsfunktion zu durchlaufen und, falls dabei kein permanenter Fehler gefunden wird, übernimmt dieser Knoten die Überwachungsfunktion. Wenn jedoch die Wiederherstellungsfunktion fehlschlägt oder ein permanenter Hardwarefehler entdeckt wird, wird der fehlerhafte Knoten abgeschaltet und durch einen Reserveknoten ersetzt. Durch diese Strategie können bis zu drei permanente Knotenfehler verkraftet werden, und der Bordcomputer bleibt dennoch funktionstüchtig.

Das hochmodulare Betriebssystem wurde unter Verwendung neuester Softwaretechnologie implementiert, und die kritischen Abschnitte wurde formal verifiziert. Die Applikationen, die auf BOSS laufen, wurden mit Hilfe objektorientierter Technologie implementiert, so dass eine hochmodulare Anwendungssoftware entstanden ist. Das BOSS Betriebssystem ist nicht nur für das Zielsystem verfügbar, es ist ebenfalls als User-Programm unter Linux implementiert. Dies ermöglicht das Implementieren und Testen der

Anwendungssoftware auf Linux-Computern und das Portieren auf das Zielsystem ohne jede Änderung.

Um ein gut strukturiertes Applikationssystem zu erhalten, haben wir eine „*software backplane*" aus zwei Softwarebussen geschaffen. Jede Applikation implementiert ein Interface zu jedem dieser Softwarebusse. Der eine Softwarebus wird dazu benutzt, Befehle, die von den Applikationen ausgeführt werden müssen, zu verteilen; der andere Bus sammelt die Statusinformationen der Applikationen, welche an die Bodenstationen auf der Erde gesendet werden müssen. Das Prinzip der *Software backplane* erlaubt eine einfache Konfiguration des Systems durch simples ein- und ausstöpseln der Softwarekomponenten in die bzw. aus der *software backplane*.

3. BOSS Beschreibung

Folgende Eigenschaften sind erwähnenswert:

- Multithreading
- Präemptiv
- Prioritätsmanagement
- Echtzeit
- Fehlertoleranzunterstützung
- Kommunikationsunterstützung
- OO-Design und -Implementierung
- C++ Interface
- Zeitrasterung: 1 Mikrosekunde
- Zeit für Threadwechsel: 3 Mikrosekunden (PPC mit 48 Mhz.)
- Reaktionszeit: < 3 Mirkosekunden (PPC mit 48 Mhz)
- Bootzeit von Flash: < 1 s

Derzeit sind drei Implementierungen von BOSS auf verschiedenen Plattformen verfügbar: PowerPC, x86 (PCs) und Linux-Applikation. Anwendungen, die für BOSS entwickelt werden, können ohne jede Änderung auf all diesen Plattformen laufen. Die Linux-Variante hilft den Entwicklern bei ihrer Arbeit, da sie ihre Arbeit lokal am Arbeitsplatz testen können, ohne das Zielsystem nutzen zu müssen. Um die entwickelte Applikation auf dem Zielsystem zu bringen, muss der Code lediglich für das Zielsystem neu kompiliert werden. Das Verhalten der Anwendung ist identisch mit Ausnahme der Zeitanforderungen und der Zeitauflösung, die unter LINUX nicht so exakt wie auf dem Zielsystem sein können.

4. Entwicklungsumgebung

Die Applikationsentwicklung und -ausführung eines simulierten Systems kann unter LINUX geschehen. Für die Ausführung auf dem Zielsystem bieten wir ein Debugger-Interface und serielle Verbindungen zum Laden und Debuggen des Systems an. Unter Benutzung von LINUX als Front-end für des Zielsystem kann die ganze Arbeit natürlich auch über das Internet erledigt werden. Ausserdem besteht die Möglichkeit, alle Aktivitäten unter BOSS auf dem Zielsystem aufzuzeichnen, mit vorhandenen Werkzeugen läßt sich das so aufgezeichnete interne Zeitverhalten visualisieren.

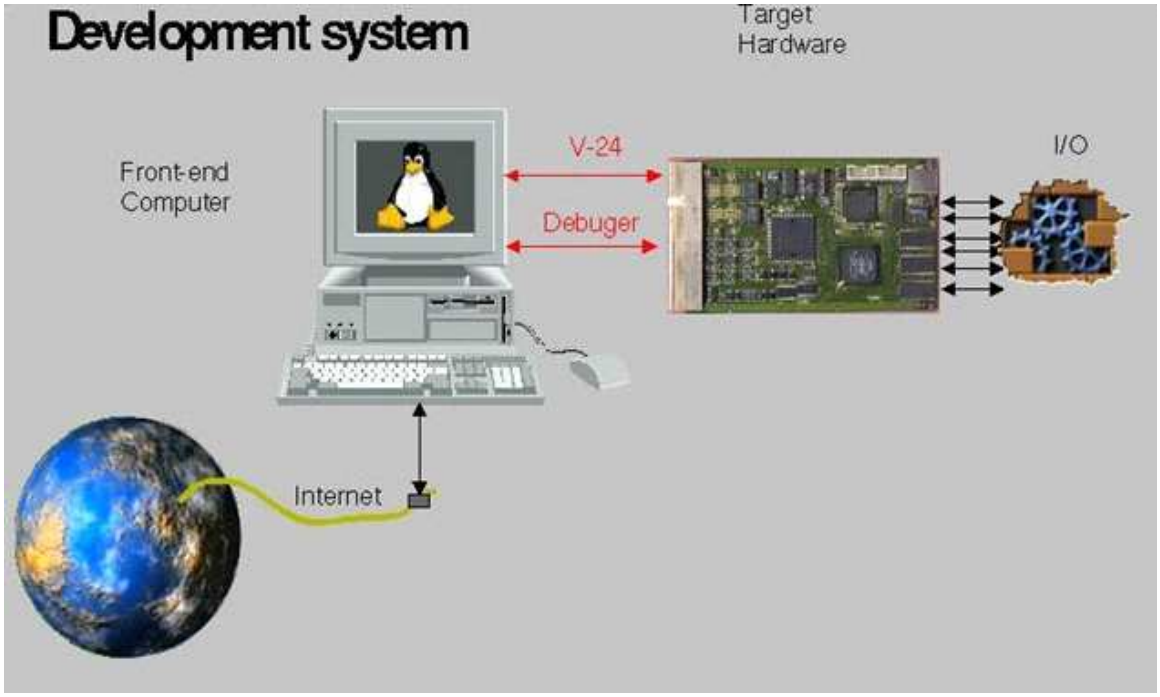

Abbildung 3: BOSS Entwicklungsumgebung

5. Makrostrukturen

BOSS ist in verschiede Schichten (Layer) gegliedert, von der Hardware bis zur finalen Applikation. Jede dieser Schichten stellt ein virtuelles Interface bereit. Diese virtuelle Sicht ist immer gleich, auch wenn die unterste Schicht ersetzt wird.

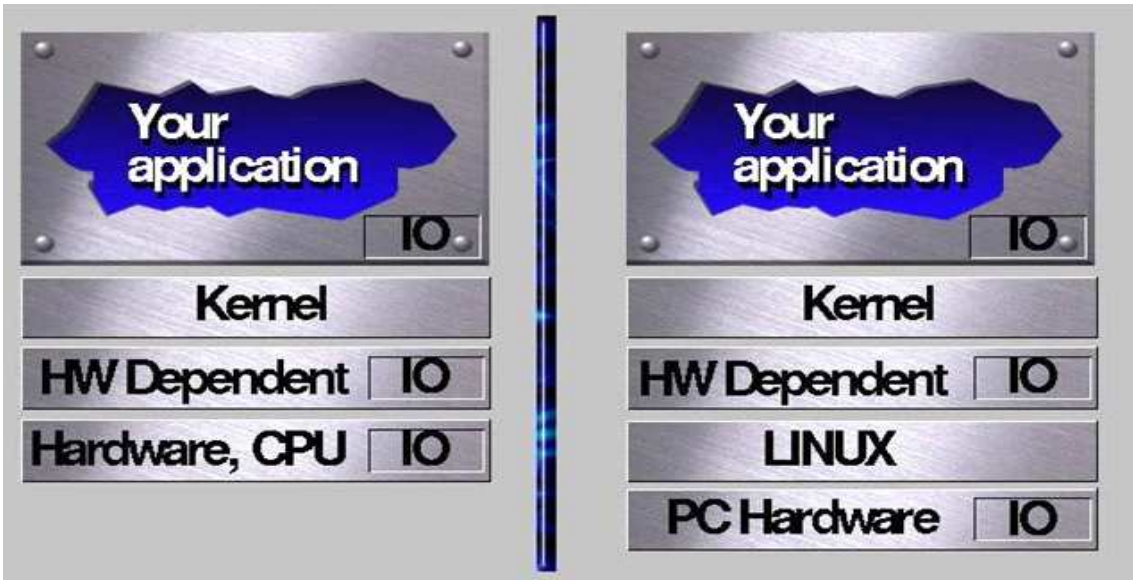

Abbildung 4: BOSS Schichten

Die unterste Schicht besteht aus der Zielhardware, wie CPU, E/A-Geräte usw. oder einer Simulation dieser Geräte unter LINUX. Diese Schicht ist für den Kernel

und die Applikationen (ausser natürlich Geschwindigkeit und Zeitauflösung) transparent und kann sehr einfach gewechselt werden.

Die zweite Schicht ist die einzige hardwareabhängige Schicht. Sie implementiert die Funktionen, die sich auf den verschiedenen Plattformen unterscheiden, z.B. Laden und Sichern der CPU-Register, systemnahe Hardwaretreiber und rudimentäres Interruptmanagement. Um von einer Plattform zu einer neuen Plattform zu wechseln, muss nur diese Schicht neu geschrieben werden. Ein kleines Problem ist die Ein- und Ausgabe. Die grundlegende E/A-Behandlung ist in der zweiten Schicht implementiert, aber es gibt Unmengen verschiedener Bausteine mit unterschiedlichen und zum Teil sehr komplexen Protokollen. Diese komplexen Protokolle sollten daher in der Applikationsschicht implementiert werden. Eine Abstraktion, die alle möglichen Bausteine abdeckt, würde sehr große Komplexität und sehr geringe Flexibität hervorbringen. BOSS bietet nur grundlegende Unterstützung der E/A-Behandlung wie zum Beispiel Ereignisweiterleitung.

Die dritte Schicht bildet der Kernel, hier wird das Interface für die Applikationen implementiert. Es verwaltet alle Threads und Ressourcen.

6. BOSS Kernel

Der Kernel ist so einfach, dass er komplett auf einer Seite erklärt werden kann:

Die Basisklasse ist *Thread*: Sie erzeugt ausführbare Objekte mit Kontext, Stack und eigenen Daten. Threads können laufen (run), unterbrochen und reaktiviert werden, und zu einem bestimmten Zeitpunkt oder internen oder externen Ereigniss reaktiviert werden.

Die Basisoperation ist die normale Listenverwaltung. Alle Ressourcen werden in (sortierten) verketteten Listen verwaltet; aufgrund der Verkettung sind diese Listen in ihrer Größe nicht begrenzt. Kein Element kann gleichzeitig in zwei verschiedenen Listen sein. Bevor ein Element in eine Liste eingefügt wird, wird es aus jeder anderen Liste gelöscht.

Für jede Liste gibt es zwei Basisoperationen:

insert: einfügen eines Eintrags an den korrekten Platz (abhängig von einem Sortierfeld). Dieses Sortierfeld wird verglichen und der neue Eintrag wird vor dem ersten mit einem höheren Wert eingetragen. Wenn z.B. andere Einträge mit demselben Wert gefunden werden (z.B. dieselbe Prioritität), wird der neue Eintrag hinter diesen eingeordnet (FIFO-Prinzip bei gleicher Priorität).

remove: entfernen eines Eintrags aus einer Liste (wenn vorhanden). Am häufigsten wird die remove-Operation auf das erste Listenelement angewendet.

Die Kernellisten:

Ready list: eine Liste der Threads, die bereit sind, die CPU zu benutzen, sortiert nach (absteigender) Priorität. Der erste Thread der Liste benutzt gegenwärtig die

CPU, d.h er wird ausgeführt. Alle anderen warten. Die Priorität ist intern invertiert, so dass die höchste Priorität am Anfang und die niedrigste am Ende der Liste steht. Wenn die Liste leer ist, wird der Idle-Thread geweckt und somit die ungenutzte CPU-Zeit verbraucht.

Timer list: eine Liste der Threads, die auf einen bestimmten Zeitpunkt warten, sortiert nach der Zeit. Der nächste zu weckende Thread steht am Anfang der Liste.

Semaphore: eine Liste von Threads, sortiert nach Priorität. Semaphoren werden für die Implementierung von Überwachern(monitors), zum Schutz von exklusiven Sektoren und für Synchronisation benutzt. Alle Threads, die darauf warten, exklusive Bereiche zu betreten, werden in einer Semaphore verwaltet. Der Thread mit der höchsten Priorität bekommt die Ressource zuerst.

Messages and communication lists: Nachrichten von einem Thread für andere Threads werden in Listen – sortiert nach Priorität – geschrieben. Üblicherweise haben alle Nachrichten dieselbe Priorität, so dass die Liste wie ein FIFO arbeitet. Ein Thread, der versucht von einer leeren MessageBox zu lesen, wird suspendiert/schlafen gelegt.

User lists: Applikationen können Listen-Klassen benutzten, um jede Art von Listen für die eigene Nutzung aufzubauen.

Abbildung 5: BOSS Listenverwaltung

7. Benutzung der Listen

Alle Thread-operationen sind nur durch Einfügen und Entfernen der Threads in die bzw. aus den Listen implementiert.

start: Erzeugen des Thread-Kontextes und Einfügen des Threads in die *ready*-Liste.

suspend: Entfernen des Threads aus der *ready*-Liste

resume: Einfügen des Threads in die *ready*-Liste.

wakeAt: Entfernen des Threads aus der *ready*-Liste und Einfügen in die *timer*-Liste.

Der Scheduler verwaltet die *ready*-Liste und aktiviert den ersten gefunden Thread in der Liste; es wird jedoch eine kleine Hilfe von der hardwareabhängigen Schicht benötigt:

transfer: Sicherung der Kontextes des laufenden Threads, Laden des Kontextes des anderen Threads und Übergabe der CPU an diesen. Der Scheduler ist präemptiv. Zu jeder Zeit, wenn ein Thread (B) mit höherer Priorität als der laufende Thread (A) wiederhergestellt werden soll, wird die CPU von Thread A an Thread B übergeben (Transfer A B).

Der TimerDriver kümmert sich um die Zeitverwaltung, hierfür wird entweder die direkte Zeitmessung der Hardware verwendet oder mithilfe von periodischen Interrupts eine Zeitmessung aufgebaut, und reaktiviert die Threads, die auf bestimmte Zeitpunkte warten.

IO-Drivers können ebenfalls Threads reaktivieren, hier jedoch aufgrund von E/A-Ereignissen/Interrupts.

8. Weitere Referenzen

1999: Sergio Montenegro. Buch Entwicklung sicherheitsrelevanter Systeme, Hanser Verlag, ISBN: 3-446-21235-3

2003: Sergio Montenegro, P. Behr, Igor Rodionov, Alexey Rodionov, Evgeniy Fedounin
Hyperspectral monitoring data processing
4th IAA Symposium on Small Satellites for earth observation, April 7 -11, 2003
ISBN 3-89685-569-7

2003: Briess, K., Baerwald, W., Gill, E., Halle, W., Kayal, H., Montenbruck, O., Montenegro, S. Skrbek, W., Studemund, H., Terzibaschian, T., Venus, H.
Technology demonstration by the bird-mission
4th IAA Symposium on Small Satellites for earth observation, April 7 -11, 2003
ISBN 3-89685-569-7

2002: Brieß, K., Bärwald, W., Gill, E., Kayal, H, Lorenz, E., Montenbruck, O., Montenegro, S., Halle, W. Schlicker, M., Skrbek, W., Studemund, H., Terzibaschian
Technologiedemonstration mit dem dlr-Kleinsatelliten BIRD
DGLR Tagung, münchen Nov 2002

2002: Briess, K., Bärwald, W., Hartmann, M., Kayal, F., Krug, H.3, Lorenz, E., Lura, F., Maibaum, O., Montenegro, S., Oertel, D., Röser, H.P., Schlotzhauer, G., Schwarz, J., Studemund, H., Turner, P., Zhukov, B.
Orbit experience and first results of the bird-mission
53rd International Astronautical Congress The World Space Congress -2002, 10-19 October 2002 / Houston, Texas

2002: Sunny LEUNG, Eberhard GILL, Oliver MONTENBRUCK, Sergio MONTENEGRO
A navigation processor for flexible real-time formation flying applications
International Symposium formation flying Missions and technologies
29, 30 and 31 October 2002 Centre National d'Etudes Spatiales Toulouse Space Centre - France

2002: K. Briess, S. Montenegro, W. Bärwald, W. Halle, H. Kayal, E. Lorenz, W. Skrbek, H. Studemund, T. Terzibaschian, I. Walter
Demonstration of Small Satellite Technologies by the BIRD Mission
16th Annual AIAAA/USU COnference on small satellites, Logan,Utah, USA 2002

2002: Eberhard Gill, Oliver Montenbruck, Sergio Montenegro
Flight Results from the BIRD Onboard Navigation System
5th International ESA Conference on Guidance Navigation and Control Systems and Actuator and Sensor Products, 22-25 October 2002 Frascati (Rome), Italy

2002: K. Briess, W. Baerwald, F. Lura, S. Montenegro, D. Oertel, H. Studemund, G. Schlotzhauer
The BIRD MISSION is completed for Launch with the PSLC-C3
Photogrammetrie Fernerkundung Geoinformationen 1/2002, S.55-58

2002: Sergio Montenegro, Volkert Barr
BOSS/Ada: An Open Source Ada 95 Safety Kit
Ada Deutschland Tagung 2002 6. 8. März 2002, Jena

2001: Sergio Montenegro
Bird-Spacecraft bus controller
Small satellites Workshop 2001 IAA 2001 2. / 5. April 2001 in Berlin