EIN SOFTWARE-FRAMEWORK FÜR MODULARE, REKONFIGURIERBARE SATELLITEN

K. Uhl, M. Göller, J. Oberländer, L. Pfotzer, A. Rönnau, R. Dillmann
FZI Forschungszentrum Informatik, Interaktive Diagnose- und Servicesysteme
Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe

Zusammenfassung

Durch einen modularen Aufbau aus standardisierten Bausteinen mit identischer Grundstruktur wird der Bau von Satelliten vereinfacht, die Reparatur erleichtert und der automatisierte Umbau im Orbit ermöglicht. Dieser Beitrag stellt ein verteiltes, dynamisches Software-Framework für solche modularen, rekonfigurierbaren Satelliten vor. Es basiert auf dem modularen, verteilten Framework MCA3, welches Mechanismen zur Entwicklung von wiederverwendbaren Modulen und Komponenten zur Verfügung stellt, die Ausführung der Datenverarbeitung steuert und die Netzwerkkommunikation übernimmt. Auf MCA3 aufbauend werden drei Basis-Softwaremodule und Richtlinien für die Umsetzung von weiteren Software-Modulen für modulare, rekonfigurierbare Satelliten definiert.

1. EINLEITUNG

Der modulare Aufbau eines Satelliten aus standardisierten Bausteinen (BS) mit identischer Struktur bietet Vorteile gegenüber der gängigen Praxis eines monolithischen, abgeschlossenen Designs. Der Bau von Satelliten wird vereinfacht, da man auf einen Katalog von Standardbausteinen zurückgreifen kann und da die standardisierte Bausteinstruktur die softwaregestützte Planung des Satellitenaufbaus sowie die Analyse der mechanischen, elektrischen und thermischen Eigenschaften des Satelliten erleichtert. Die Lebensdauer der Satelliten wird verlängert, da defekte Bausteine und Bausteine, die eine kürzere Lebensdauer haben als der restliche Satellit, einzeln ausgetauscht werden können, ohne den gesamten Satelliten ersetzen zu müssen. Außerdem können Satelliten durch den modularen Aufbau im Orbit umgebaut bzw. erweitert werden, um sie so an neue Missionsanforderungen anzupassen.

Gleichzeitig stellt der modulare Aufbau eines Satelliten aber deutlich höhere Anforderungen an die Satelliten-Software. Durch den Aufbau aus Einzelbausteinen ergibt sich eine verteilte Softwarearchitektur, welche eine aufwändigere Kommunikation und Koordination erfordert als bei einem monolithischen System. Darüber hinaus setzt die Möglichkeit der Rekonfiguration im Orbit voraus, dass die Satelliten-Software in der Lage ist, die Komponenten und die Struktur des Satelliten automatisch zu ermitteln.

1.1. Bausteinbasierte Satellitensysteme

Im Projekt *iBOSS* (*Intelligent Building Blocks for On-Orbit Satellite Servicing*) werden Konzepte für modulare, einfach wartbare und im Orbit rekonfigurierbare Satelliten entwickelt. Das Konzept sieht vor, dass ein Satellit in würfelförmige Bausteine identischer Größe aufgeteilt wird. Größere Bausteine sind ebenfalls möglich, solange sie ein vielfaches der Standard-Würfelgröße einnehmen. Die Bausteine sind an möglichst vielen Seiten mit Schnittstellen zur mechanischen Kopplung und zur Energie-, Wärme- und Datenübertragung ausgestattet. Die Schnittstellen sind dabei sowohl 90° rotations-, als auch spiegelsymmetrisch ausgelegt, so dass zwei Bausteine in beliebiger Orientierung zueinander verbunden werden können. Aufbauend auf dieser standardisierten Bausteinform wird ein Katalog von Standardbausteinen erstellt, der die gängigen Satellitenfunktionen abdeckt. Spezielle Funktionen können als Sonderbausteine individuell umgesetzt werden [1].

Die Software-Infrastruktur, die im Rahmen des iBOSS-Projekts definiert wird, ist in der Lage, die in einem Satelliten vorhandenen Bausteine und deren Kopplungen zu anderen Bausteinen im laufenden Betrieb zu erkennen und daraus den Aufbau des Satelliten und die darin enthaltenen Komponenten zu rekonstruieren. Außerdem stellt die Software-Infrastruktur Mechanismen zur Verfügung, um transparent und effizient auf die Komponenten eines Satelliten zuzugreifen, ohne sich um deren Position im Satelliten oder um Netzwerkkommunikation Gedanken machen zu müssen.

Die allgemeinen Eigenschaften von Bausteinen sowie die speziellen Eigenschaften der Standardbausteine aus dem Bausteinkatalog und ggf. der Sonderbausteine werden in einer Ontologie modelliert. Neben der Größe der Bausteine, der vorhandenen Schnittstellen und der benötigten Ressourcen und enthaltenen Fähigkeiten, werden insbesondere auch globale, baustein-relative und bausteinspezifische Regeln modelliert, welche die Positionierung von Bausteinen in einem Satelliten beeinflussen. Die Ontologie wird von einem Planungswerkzeug verwendet, um aus einer Definition von Missionsanforderungen halb- oder vollautomatisch Vorschläge für die benötigten Bausteine und deren Platzierung im Satelliten zu berechnen [2].

1.2. Aufbau des Beitrags

In diesem Beitrag wird ein Software-Framework für modulare, rekonfigurierbare Satelliten vorgestellt, welches die Software-Infrastruktur bereit stellt, die im iBOSS-Projekt definiert wurde. Zunächst wird das modulare, verteilte Software-Framework MCA3 beschrieben. Im Anschluss daran werden Basis-Software-Module für modulare Satellitensysteme eingeführt und Entwicklungsrichtlinien für weitere baustein- bzw. funktionsspezifische Software-Module skizziert. Danach wird der Prozess erläutert, über den der Aufbau eines Satelliten im laufenden Betrieb rekonstruiert werden kann. Der Beitrag

schließt mit einer Zusammenfassung der vorgestellten Arbeiten.

2. DAS MODULARE, VERTEILTE SOFTWARE-FRAMEWORK MCA3

Das modulare, verteilte Software-Framework *MCA3* (*Modular Controller Architecture Version 3*) stellt Mechanismen zur Entwicklung von wiederverwendbaren Modulen und Komponenten zur Verfügung. Es legt die Grundstruktur des Software-Systems fest und kümmert sich – für den Programmierer transparent – um die zeit- oder ereignisgesteuerte Ausführung der Datenverarbeitung sowie die Netzwerkkommunikation. Darüber hinaus bietet MCA3 Funktionen zur Verwaltung von verteilten Software-Systemen und zur Diagnose.

MCA3 basiert auf den bewährten Konzepten des Software-Frameworks MCA2, das seine Wurzeln in der Robotik-Forschung hat [3].

2.1. Komponentenstruktur

Die kleinste Einheit in MCA3 stellt das *Modul* dar. Ein Modul kapselt eine in sich abgeschlossene Funktion. Um die Wiederverwendbarkeit von Modulen zu erhöhen, implementieren Module i.d.R. möglichst kleine funktionale Einheiten. Jedes Modul verfügt über vier unterschiedliche Arten von Schnittstellen (siehe BILD 1):

- Über *Eingangsschnittstellen* empfängt ein Modul Daten von anderen Modulen.
- Über *Ausgangsschnittstellen* sendet ein Modul Daten zu anderen Modulen.
- *Wartungsschnittstellen* veröffentlichen den internen Zustand eines Moduls.
- Über die *Parameterschnittstelle* wird ein Modul mit Konfigurationsparametern versorgt.

Die Datentypen, die über die einzelnen Modul-Schnittstellen übertragen werden, werden in einer IDL (Interface Definition Language) spezifiziert [4].

Die Datenverarbeitung in einem Modul, d.h. die Ausführung der implementierten Funktion, kann auf zwei unterschiedliche Arten erfolgen. Bei der zeitgesteuerten Verarbeitung wird vom MCA3-Framework eine Verarbeitungs-Funktion des Moduls zyklisch aufgerufen. Bei der ereignisgesteuerten Verarbeitung wird eine Verarbeitungs-Funktion des Moduls aufgerufen, wenn neue Daten an einer bestimmten Eingangsschnittstelle empfangen wurden. Beide Varianten der Datenverarbeitung können gleichzeitig verwendet werden.

Komplexe Funktionen werden umgesetzt, indem mehrere Module, die jeweils eine einfache Funktion implementieren, mittels *Kanten* zu einem Modul-Netzwerk verknüpft werden. Eine Kante transportiert dabei Daten von einer Ausgangsschnittstelle des Quellmoduls zu einer Eingangsschnittstelle des Zielmoduls. Dies kann auf drei Arten geschehen:

- Bei *referenzierenden Kanten* wird der Speicherbereich der Ausgangsschnittstelle direkt auf die Eingangsschnittstelle abgebildet. Dadurch müssen die Daten nicht kopiert werden.
- Bei *kopierenden Kanten* werden die Daten von der Ausgangsschnittstelle zur verbundenen Eingangsschnittstelle kopiert.
- Bei *gepufferten Kanten* werden die Daten von der Ausgangsschnittstelle in einen FIFO-Puffer kopiert.

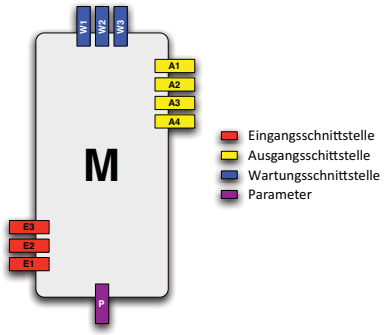

BILD 1. Ein *Modul* mit Eingang-, Ausgangs-, Wartungs- und Parameter-Schnittstellen.

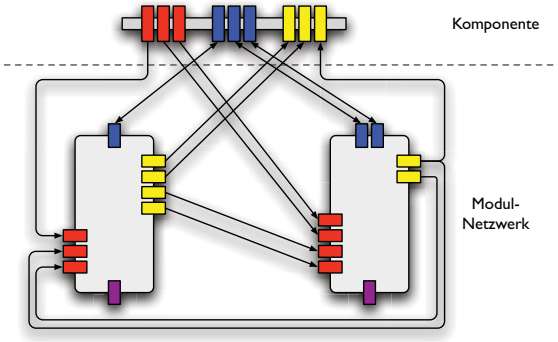

BILD 2. Eine *Komponente*, die mit einem Modul-Netzwerk verknüpft ist.

An der Eingangsschnittstelle können die gepufferten Daten einzeln in der Reihenfolge abgerufen werden, in der sie gesendet worden sind.

Falls das empfangende Modul langsamer arbeitet als das sendende Modul, steht bei referenzierenden und kopierenden Kanten immer nur das zuletzt gesendete Datum an der Eingangsschnittstelle zur Verfügung. Bei gepufferten Kanten werden alle gesendeten Daten vorgehalten.

Die komplexe Funktion, die das Modul-Netzwerk implementiert, wird in Form einer *Komponente* gekapselt. Ähnlich wie Module enthalten Komponenten Eingangsschnittstellen, Ausgangsschnittstellen und Wartungsschnittstellen. Die Komponentenschnittstellen werden über Kanten mit Schnittstellen von Modulen des gekapselten Modul-Netzwerks verbunden (siehe BILD 2).

Komponenten können ebenfalls untereinander mittels Kanten zu einem Komponenten-Netzwerk verknüpft werden. Die einzelnen Komponenten des Netzwerks können dabei transparent auf mehrere Rechner verteilt werden und bilden auf diese Art und Weise ein komplexes, verteiltes Software-System.

Die Definition einer Komponente enthält keine eigene Funktionalität, sie stellt lediglich eine Schnittstelle dar. Erst durch Verknüpfungen mit einem Modul-Netzwerk wird die Komponente mit Funktionalität gefüllt. Diese Trennung von Komponenten-Schnittstelle und Implementierung ermöglicht es, die Implementierung einer Komponente zu verändern oder auszutauschen, ohne dass die Klienten der Komponente angepasst werden müssen.

2.2. Kommunikationsstruktur

Die Kanten zwischen Komponenten-Schnittstellen sind

netzwerktransparent. Sie werden in der gleichen Art und Weise erzeugt, egal ob die Komponenten, die verknüpft werden sollen, im selben Prozess, auf demselben Rechner oder auf verschiedenen Rechnern laufen. MCA3 bedient sich dazu des *Data Distribution Service* (*DDS*) [5].

Bei DDS handelt es sich um eine Middleware zur datenzentrierten Kommunikation in hochdynamischen, verteilten Systemen, die auf dem *Publisher-Subscriber*-Prinzip basiert. DDS wird von der Object Management Group (OMG) standardisiert und wurde in mehreren Implementierungen umgesetzt. Die Anwendungen reichen von automatisierten Handelssystemen über die Flugverkehrskontrolle und Systeme im Verteidigungssektor bis hin zu verschiedenen robotischen Systemen der NASA.

DDS enthält folgende Hauptkomponenten:

- Ein *Topic* ist ein benannter, typisierter Datenstrom, über den Informationen übertragen werden können.
- Eine *Domäne* dient zur logischen Gliederung und zur Abschottung von unterschiedlichen DDS-Anwendungen, die im selben Netzwerk betrieben werden. Eine Domäne enthält eine Menge von Topics.
- Ein *Publisher* veröffentlicht Daten auf einem oder mehreren Topics.
- Ein *Subscriber* empfängt Daten von einem oder mehreren Topics.
- Die exakte Art und Weise der Datenübertragung kann über eine Vielzahl von *Quality-of-Service-Parametern* (QoS) konfiguriert werden.

Die Erzeugung eines Topics erfolgt unabhängig von einem Publisher oder Subscriber, sondern findet innerhalb der Domäne statt. Ein Topic kann sogar mehrfach an unterschiedlichen Stellen erzeugt werden. In diesem Fall wird das Topic beim ersten Erzeugen in der Domäne registriert. Alle weiteren Versuche, dasselbe Topic zu erzeugen, referenzieren lediglich das bereits registrierte Topic. Durch diese Entkopplung ist es möglich, Publisher und Subscriber in einer beliebigen Reihenfolge zu erzeugen.

MCA3 bildet jede Komponenten-Schnittstelle auf ein DDS Topic ab. Zusätzlich werden Ausgangsschnittstellen zu Publishern und Eingangsschnittstellen zu Subscribern. Wartungsschnittstellen sind bidirektional und sind daher gleichzeitig Publisher und Subscriber.

Die Ausgangsschnittstellen von Komponenten veröffentlichen veränderte Daten laufend auf den zugehörigen DDS Topics, unabhängig davon, ob sie über Kanten mit Eingangsschnittstellen von anderen Komponenten verknüpft sind. Beim Erzeugen einer Kante zwischen zwei Komponenten abonniert der Subscriber der Ziel-Eingangsschnittstelle das Topic der Quell-Ausgangsschnittstelle und ist somit sofort in der Lage, die gesendeten Daten zu empfangen (siehe BILD 3).

2.3. Prozesstruktur

Die Komponenten- und Kommunikationsstruktur von MCA3 ist darauf ausgelegt, ein Software-System ohne Berücksichtigung von Daten-Netzwerken, Rechnern und Prozessen in ein Netzwerk von kleinen, handhabbaren Einheiten zu zerlegen. Jedoch muss zu einem Zeitpunkt im Software-Entwicklungsprozess festgelegt werden, wie das Software-System ausgeführt werden soll.

Hierfür stellt MCA3 sog. *Parts* zur Verfügung. Ein Part stellt ein ausführbares Programm dar, das beliebig viele

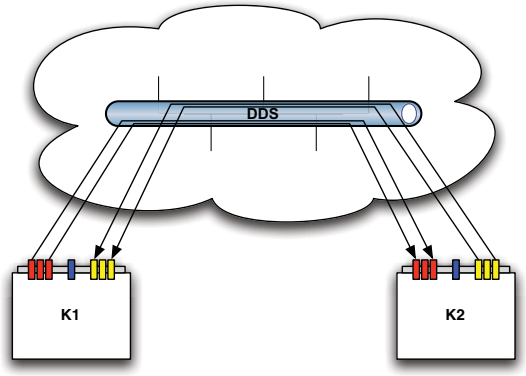

BILD 3. Zwei Komponenten tauschen netzwerktransparent Daten über DDS aus.

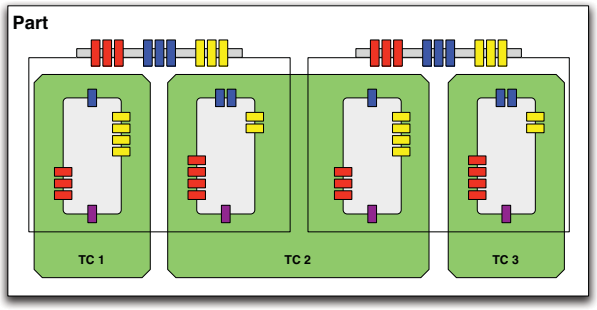

BILD 4. Ein *Part* mit Komponenten, Modulen und ThreadContainern. Im ThreadContainer TC2 laufen Module aus zwei unterschiedlichen Komponenten.

Komponenten und Module enthalten kann. Gleichzeitig können auf demselben Rechner mehrere Parts ausgeführt werden. Ein Part kümmert sich außerdem um die Verwaltung der DDS-Domäne, die für die Netzwerkkommunikation verwendet wird.

Innerhalb eines Parts kann die Ausführung der Datenverarbeitung in den Modulen auf mehrere Threads aufgeteilt werden. Hierfür werden die Module, die in einem Part enthalten sind, bei sog. *ThreadContainern* registriert. Die Zugehörigkeit von Modulen zu ThreadContainern ist unabhängig von deren Komponenten-Zugehörigkeit. Unterschiedliche Module, die zu derselben Komponente gehören, können in unterschiedlichen Threads ablaufen. Umgekehrt können aber auch Module, die zu unterschiedlichen Komponenten gehören, in demselben Thread ablaufen (siehe BILD 4).

2.4. MCA-Daemon

Ein statisches MCA3-System, das aus einem einzelnen lokalen oder einer festen Anzahl an verteilten Parts besteht, kann problemlos manuell gestartet und verwaltet werden. Bei einem dynamischen, verteilten System, in dem Parts, je nach aktueller Anforderung, automatisch gestartet, beendet oder umgezogen werden müssen, ist dies nicht mehr möglich. Aus diesem Grund enthält das MCA3 Framework den sog. *MCA-Daemon*. Dieser läuft auf jedem Rechner und stellt Funktionen zum Übertragen von Software und zum Verwalten von Parts zur Verfügung. Zur Datenübertragung verwendet er das TFTP Protokoll [6]. Zur Bereitstellung von Statusinformationen und zum Empfangen und Bestätigen von Kommandos wird DDS verwendet.

Die Übertragung von Software läuft wie folgt ab. Im ersten Schritt werden alle Dateien, die für die Ausführung der Software benötigt werden, per TFTP übertragen. Aus Sicherheitsgründen legt der TFTP-Server, der im MCA-Daemon integriert ist, diese Dateien zunächst in einem temporären Verzeichnis ab. Nachdem alle Dateien erfolgreich übertragen wurden, wird ein Kommando zur Installation der Software an den MCA-Daemon geschickt. Dieses Kommando enthält die Liste der zu installierenden Dateien incl. Prüfsummen sowie den Namen, die Version und die Architektur (d.h. Betriebssystem und Prozessorarchitektur) der Software. Der MCA-Daemon überprüft nun die Vollständigkeit und Korrektheit der Dateien und verschiebt sie anschließend in ein Installationsverzeichnis. Nach erfolgreicher Installation trägt er die Software in seine Liste von installierten Software-Versionen ein und veröffentlicht diese per DDS.

Die Installation von Software kann manuell „von außen“ angestoßen werden, z.B. wenn eine noch nicht vorhandene Software oder eine neue Software-Version installiert werden soll. Ein MCA-Daemon kann aber auch per Kommando angewiesen werden, eine bei ihm installierte Software automatisch zu einem anderen MCA-Daemon zu übertragen. Diese Funktion wird beim automatischen Start von MCA-Parts und beim Umzug eines Parts auf einen anderen Rechner genutzt (s.u.).

Neben der Übertragung von Software ist die Hauptaufgabe eines MCA-Daemon, Parts zu starten und zu beenden. Dies kann auf unterschiedliche Art und Weise geschehen. Im einfachsten Fall erhält ein MCA-Daemon ein direktes Kommando, einen Part lokal zu starten.

Es besteht aber auch die Möglichkeit, ein Kommando an das gesamte verteilte MCA3-System zu senden, um einen Part auf einem beliebigen Rechner im System zu starten und dafür zu sorgen, dass er neu gestartet wird, falls dieser Rechner ausfällt oder der Part aus anderen Gründen beendet wird.

Da alle MCA-Daemon-Instanzen zunächst gleichberechtigt sind, muss automatisch ein *Koordinator* bestimmt werden, der entscheidet, wo der Part gestartet wird und diesen anschließend überwacht. Ohne Koordinator kann kein Konsens garantiert werden. Die MCA-Daemon-Instanzen würden möglicherweise endlos darüber „diskutieren“, wo der Part ausgeführt werden soll, oder gar mehrere Instanzen des Parts starten.

Zur Bestimmung des Koordinators kommt der sog. *Bully-Algorithmus* [7] zum Einsatz. Dazu erhält jeder MCA-Daemon eine eindeutige ID. Solange kein Koordinator bestimmt ist, sendet jeder MCA-Daemon (über DDS) eine Koordinator-Ankündigung mit seiner eigenen ID an alle anderen Instanzen. Empfängt ein MCA-Daemon eine solche Ankündigung, vergleicht er die empfangene ID mit der eigenen. War die empfangene ID größer, sendet der MCA-Daemon keine weiteren Koordinator-Ankündigungen, sondern begibt sich in einen passiven Zustand. War die empfangene ID kleiner, so sendet er sofort eine neue Koordinator-Ankündigung mit der eigenen ID. Empfängt ein MCA-Daemon nach einem Timeout keine Antwort auf seine Koordinator-Ankündigung, dann wird er zum Koordinator. Um den passiven Teilnehmern mitzuteilen, dass er noch funktionsfähig ist, sendet der Koordinator periodisch eine neue Koordinator-Ankündigung. Wird in einem gewissen Zeitraum keine Koordinator-Ankündigung empfangen, so gehen die passiven MCA-Daemon-Instanzen

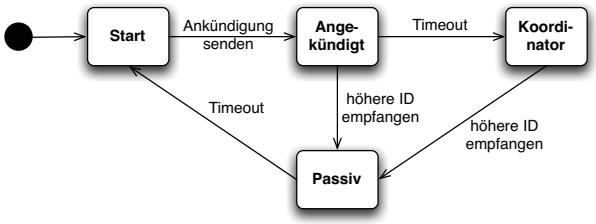

BILD 5. Zustandsübergänge beim Bully-Algorithmus.

davon aus, dass der bisherige Koordinator ausgefallen ist und der Prozess beginnt von vorne (siehe BILD 5).

Um nun einen Part auf einem beliebigen Rechner im MCA3-System zu starten, sammelt der Koordinator die Listen der installierten Parts von allen MCA-Daemon-Instanzen und bestimmt daraus die Architekturen (Betriebssystem und Prozessorarchitektur), für welche der Part zur Verfügung steht. Anschließend sucht er aus den MCA-Daemon Instanzen mit passender Architektur einen Rechner mit ausreichend freier Rechenleistung als Zielrechner für die Ausführung des Parts aus. Alle MCA-Daemon-Instanzen ermitteln und veröffentlichen dazu die verfügbare Rechenleistung auf ihren Rechnern. Falls der Part auf diesem Rechner noch nicht installiert ist, sorgt der Koordinator dafür, dass der Part dorthin übertragen und installiert wird. Schließlich weist er den MCA-Daemon auf dem Zielrechner an, den Part zu starten.

Ein MCA-Daemon ist auch in der Lage, mit minimaler Umschaltzeit eine neue Software-Version eines Parts zu starten bzw. einen Part auf einen anderen Rechner umzuziehen. Falls der Part auf einen anderen Rechner umgezogen wird, wird zunächst überprüft, ob die Software auf dem Zielrechner verfügbar ist und ggf. dorthin übertragen. Ansonsten wird die aktuell laufende Version des Parts zunächst gestoppt, d.h. die Ausführung der Datenverarbeitung der enthaltenen Module wird ausgesetzt. Anschließend werden die Komponenten des Parts angewiesen, den internen Zustand über die Wartungsschnittstellen zu veröffentlichen. Nun wird der gestoppte Part beendet und die neue Software-Version bzw. der Part auf einem anderen Rechner gestartet. Bei der Initialisierung erkennen die Komponenten des neuen Parts, dass auf den Wartungsschnittstellen Daten zur Verfügung stehen und initialisieren ihren internen Zustand mit diesen Daten. Danach wird die Datenverarbeitung im neuen Part mit dem wiederhergestellten internen Zustand fortgesetzt.

2.5. Diagnose

Umfangreiche Diagnosemöglichkeiten sind – sowohl bei der Softwareentwicklung als auch im Betrieb – ein wichtiges Instrument, um Fehler effizient finden und beheben zu können. Aus diesem Grund verfügt MCA3 über eine Reihe von Diagnoseschnittstellen:

- MCA3 stellt Listen aller Module, Schnittstellen, Komponenten, Kanten, Thread-Container und Parts sowie deren Eigenschaften zur Verfügung. Dadurch kann die Komponenten-, Kommunikations- und Prozessstruktur eines MCA3-Systems zur Laufzeit analysiert werden.
- Module erzeugen für ihre Schnittstellen DDS-seitig Diagnose-Topics. Auf Anfrage werden die Daten, die über die Schnittstellen kommuniziert werden, nicht nur über die verbundenen Kanten transportiert, sondern auch auf dem Diagnose-Topics veröffentlicht.

Dadurch kann bei Bedarf sowohl auf die Komponenten- als auch auf die Modulschnittstellen über das Netzwerk zugegriffen werden.

- Die Diagnose-Schnittstellen sind bidirektional ausgelegt, d.h. die Modul- und Komponentenschnittstellen können darüber auch beschrieben werden, z.B. um zu Testzwecken bestimmte Werte vorzugeben. Die beschriebenen Schnittstellen können dabei auch eingefroren werden, damit die vorgegebenen Werte vom restlichen System, das parallel weiterläuft, nicht wieder überschrieben werden.

Mit dem *Inspector* wird MCA3 ein Werkzeug enthalten, das die Diagnoseschnittstellen aus einem laufenden MCA3-System ausliest, grafisch ansprechend darstellt und Eingabemöglichkeiten zur Verfügung stellt. Der Inspector kann auch genutzt werden, um neue Software zu installieren und um Kommandos an die MCA-Daemon Instanzen zu schicken (siehe BILD 6).[1]

3. SOFTWARE-MODULE FÜR MODULARE SATELLITENSYSTEME

Aufbauend auf den Mechanismen von MCA3 wurden drei Basis-Software-Module für modulare, rekonfigurierbare Satelliten entworfen. Außerdem wurden Entwicklungs-Richtlinien definiert, wie weitere SW-Module für die verschiedenen Funktionen des Satelliten umzusetzen sind. Die SW-Module werden dabei als MCA3-Parts implementiert und stellen ihre Funktionalität über MCA3-Komponentenschnittstellen zur Verfügung.

3.1. ID-Modul

Das *Identifikations-Modul* (*ID-Modul*) ist ein zentrales Element für die Ermittlung der Satellitenstruktur und der im Satelliten vorhandenen Einzelkomponenten im laufenden Betrieb. Jeder Baustein verfügt über genau ein ID-Modul und stellt sicher, dass es immer läuft, solange der Baustein mit Strom versorgt wird.

Das ID-Modul hat die Aufgabe, den Baustein im Satelliten bekannt zu machen. Es veröffentlicht verschiedenartige Informationen über den Baustein. Dazu zählt zunächst die Baustein-ID, die den Baustein eindeutig im Satelliten identifiziert. Außerdem veröffentlicht das ID-Modul eine Sub-Ontologie, welche die statischen Eigenschaften des Bausteins beschreibt. Dazu gehören die Komponenten, die im Baustein enthalten sind, und allgemeine Kenngrößen, wie z.B. die maximale elektrische Leistung, die der Baustein durchleiten kann, die maximale elektrische Leistungsaufnahme und die maximal verfügbaren Rechner-Ressourcen. Darüber hinaus ermittelt und veröffentlicht das ID-Modul dynamische Informationen über den Baustein, wie z.B. die aktuelle Leistungsaufnahme des Bausteins und die aktuell verfügbaren und belegten Rechnerressourcen.

Zuletzt veröffentlicht das ID-Modul auch Informationen über die unmittelbaren Nachbarn des Bausteins. Hierzu erhält es zum einen von den Docking-Schnittstellen des Bausteins Informationen darüber, an welchen Baustein-Seiten benachbarte Bausteine angedockt sind. Diese Information ist insbesondere auch dann verfügbar, wenn der benachbarte Baustein defekt ist und deshalb weder dessen ID-Modul Informationen sendet, noch dessen Switch im Netzwerk erkannt werden kann. Zum anderen

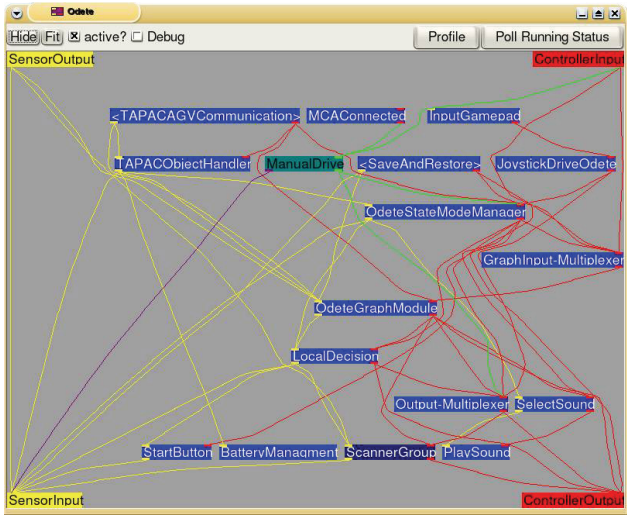

BILD 6. Ansicht eines Modul-Netzwerks im MCA2-Browser. Der MCA3-Inspector wird vergleichbare Funktionen unterstützen.

ermittelt das ID-Modul über Netzwerk-Management-Protokolle die benachbarten Netzwerkteilnehmer.

3.2. RFSM-Modul

Das *Raumfahrtsystem-Management-Modul* (*RFSM-Modul*) dient der Überwachung und Verwaltung des Satelliten. Es aggregiert die Informationen der ID-Module aller Bausteine, rekonstruiert daraus die aktuelle Satelliten-konfiguration (siehe Abschnitt 4) und kann ausgefallene Bausteine erkennen.

Das RFSM-Modul enthält die in Abschnitt 1.1 angesprochene Basis-Ontologie mit dem Bausteinkatalog sowie eine Sub-Ontologie, welche die Missionsanforderungen enthält. Diese beiden Ontologien fügt es mit den Sub-Ontologien der einzelnen Bausteine zu einer Satelliten-Ontologie zusammen. Damit kennt das RFSM-Modul nicht nur den Aufbau des Satelliten, sondern auch alle im Satelliten enthaltenen Komponenten und deren Position.

Über die Satelliten-Ontologie ist das RFSM-Modul auch in der Lage, abzuleiten, welche allgemeinen SW-Module, die nicht an einen bestimmten Baustein bzw. eine bestimmte Komponente gebunden sind, für die Mission benötigt werden und welche Ressourcenanforderungen diese SW-Module haben. Das RFSM-Modul verteilt die benötigten allgemeinen SW-Module gemäß ihren Anforderungen auf die verfügbaren Rechner, startet sie dort und überwacht deren Ausführung. Falls ein SW-Modul ausfällt, z.B. wegen eines Baustein-Defekts, ermittelt das RFSM-Modul automatisch eine neue Verteilung der SW-Module, startet das ausgefallene SW-Modul neu und zieht dabei ggf. weitere SW-Module auf andere Rechner um.

Es kann vorkommen, dass nicht genügend Ressourcen auf den aktiven Rechnern zur Verfügung steht. In diesem Fall aktiviert das RFSM-Modul einen Rechner, der normalerweise ausgeschaltet ist und im Regelfall nur eingeschaltet wird, wenn eine bestimmte Komponente, z.B. ein Sensor, benötigt wird. Falls auch kein ausgeschalteter Rechner mehr zur Verfügung steht, können nicht alle geforderten SW-Module ausgeführt werden. Für diesen Fall liefert die Satelliten-Ontologie eine Priorisierung der SW-Module, so dass das RFSM-Modul entscheiden kann, welche SW-Module für den Betrieb des Satelliten und für

[1] Derzeit ist die Implementierung des MCA3-Inspector noch in Arbeit.

die Mission am wichtigsten sind und nur diese ausführt.

Das RFSM-Modul wird beim MCA3-Framework registriert, um automatisch auf einem beliebigen Rechner gestartet bzw. neu gestartet zu werden. Um darüber hinaus sicher zu stellen, dass das RFSM-Modul auch wirklich gestartet werden kann, ist die Software des RFSM-Moduls auf jedem Rechner vorinstalliert, auf dem sie ausgeführt werden könnte.

Aufgrund der Flexibilität, die der modulare Satellitenaufbau bietet, kann nicht ausgeschlossen werden, dass – insbesondere bei der Rekonfiguration eines Satelliten im Orbit – zwei aktive Satellitenteile gekoppelt werden. In einem solchen Fall ist in beiden Satellitenteilen ein eigenes RFSM-Modul aktiv. Nach der Kopplung detektieren beide RFSM-Module diese Situation und vereinbaren über den in Abschnitt 2.4 beschriebenen Bully-Algorithmus, wer weiter läuft und wer sich beendet.

3.3. BSC-Modul

Genau wie das ID-Modul ist das *Baustein-Kontrollmodul* (*BSC-Modul*) auf jedem Baustein im Satelliten vorhanden und immer aktiv. Es hat zwei Aufgaben:

- Das BSC-Modul stellt die Software-Schnittstelle zur Verfügung, um den Kopplungs- und Entkopplungsvorgang der mechanischen und elektrischen Bausteinschnittstellen zu steuern.
- Falls ein Baustein Komponenten enthält, die von der Energieversorgung getrennt werden können, stellt das BSC-Modul die SW-Schnittstelle für das Ein- und Ausschalten dieser Komponenten zur Verfügung.

In beiden Fällen transportiert die SW-Schnittstelle sowohl Kommandos, als auch Statusinformationen.

3.4. Weitere SW-Module

Jede Komponente und jede Funktion eines Satelliten, wie z.B. Sensoren, Kommunikation oder Lageregelung, wird über ein dediziertes SW-Modul gesteuert. Die SW-Module können dabei Daten (z.B. Sensordaten) von anderen SW-Modulen abrufen und Kommandos an andere SW-Module senden. Neben den drei Basis-SW-Modulen gibt es in jedem Satelliten daher noch eine Reihe von zusätzlichen SW-Modulen.

Je nach Funktion werden die SW-Module in zwei Klassen eingeordnet:

- *Bausteingebundene SW-Module* steuern eine Hardware-Komponente, die in einem bestimmten Baustein eingebaut ist. Dementsprechend muss ein solches SW-Modul auf genau dem Rechner laufen, an dem die zugeordnete Hardware-Komponente angeschlossen ist. Bausteingebundene SW-Module können nicht vom RFSM-Modul umgezogen werden.
- *Freie SW-Module* sind nicht mit einer Hardware-Komponente verknüpft. Sie greifen nur indirekt über die Schnittstellen anderer SW-Module auf Hardware-Komponenten des Satelliten zu. Deshalb können freie SW-Module auf einem beliebigen Rechner im Satelliten, der über genügend freie Ressourcen verfügt, ausgeführt werden. Außerdem können sie, bei Bedarf, vom RFSM-Modul auf einen anderen Rechner umgezogen werden.

MCA3-Komponenten bilden die Schnittstellen zwischen den SW-Modulen eines Satelliten. Die SW-Module müssen aber nicht notwendigerweise mit dem MCA3-Framework implementiert werden. Sie müssen lediglich DDS-Schnittstellen implementieren, die mit den MCA3-Schnittstellen kompatibel sind.

3.5. Stellvertreter-Module

Um Energie zu sparen, können Komponenten eines Satelliten-Bausteins zeitweise abgeschaltet werden. Hierzu gehören natürlich auch zusätzliche Rechner, die für die Verarbeitung der Daten benötigt werden und auf denen die zugehörigen SW-Module laufen. Um eine ausgeschaltete Komponente, z.B. einen Sensor, zu aktivieren, müsste ein externes SW-Modul sowohl den Baustein kennen, in dem die Komponente verbaut ist, als auch den Rechner, auf dem das zugehörige SW-Modul läuft. Dann könnte das externe SW-Modul über das BSC-Moduls des passenden Bausteins sowohl die Komponente, als auch ggf. den zugehörigen Rechner aktivieren. Dieser Prozess würde aber komplexe Anfragen an das RFSM-Modul erfordern.

Um das Aktivieren und Deaktivieren von Satelliten-Komponenten zu vereinfachen, können *Stellvertreter-Module* eingesetzt werden. Ein Stellvertreter-Modul stellt exakt dieselben Schnittstellen zur Verfügung wie das vertretene Modul. Es läuft auf einem Rechner im Baustein, der immer eingeschaltet ist (i.d.R. der Rechner, auf dem das ID-Modul und das BSC-Modul laufen, siehe BILD 7). Im Gegensatz zum vertretenen Modul verarbeitet das Stellvertreter-Modul nur die Kommandos „Aktivieren“ und „Deaktivieren“ und setzt den Zustand „inaktiv“, solange die Komponente ausgeschaltet ist.

Auch bei der Verwendung eines Stellvertreter-Moduls senden Klienten einer Komponente das „Aktivieren“- bzw. „Deaktivieren“-Kommando direkt an das SW-Modul der Komponente. Das zugehörige Stellvertreter-Modul empfängt diese Kommandos ebenfalls und kümmert sich ggf. um das Ein- bzw. Ausschalten des passenden Rechners. Da das Stellvertreter-Modul den Aufbau des Satelliten-Bausteins kennt, sind hierfür keine komplexen Anfragen an das RFSM-Modul notwendig (siehe BILD 8).

4. ERMITTLUNG DER SATELLITEN-KONFIGURATION

Zur Ermittlung der Satellitenkonfiguration arbeiten das RFSM-Modul und die ID-Module der einzelnen Bausteine zusammen. Zusätzlich ergeben sich Anforderungen an den Aufbau von Satelliten-Bausteinen.

Zunächst muss jeder Baustein über einen Netzwerk-Switch verfügen, der ein Netzwerkmanagement-Protokoll unterstützt, über das Informationen über die Netzwerktopologie abgerufen werden können. Der Switch muss außerdem in der Lage sein, mit einem feinmaschig verknüpften Netzwerk umzugehen, das Mehrfach-Zyklen enthält. Für die Evaluation der Konzepte im iBOSS-Projekt kommt ein Ethernet-Netzwerk zum Einsatz. Die verwendeten Switches können über das *Simple Network Management Protokoll* (*SNMP*) [8] abgefragt werden und lösen Zyklen über das *Rapid Spanning Tree Protokoll* (*RSTP*) [9] auf. Die Mechanismen können aber problemlos auf andere Netzwerktechnologien, wie z.B. *SpaceWire* [10], oder andere Protokolle zur Ermittlung der Netzwerktopologie, wie z.B. *LLDP* (*Link Layer Discovery Protocol*) [11], übertragen werden.

Beim RSTP wird ein vermaschtes Netzwerk zu einem

zyklenfreien Baum reduziert. Dazu wird zuerst ein Switch als Wurzel des Baums ausgewählt. Dabei handelt es sich i.d.R. um den Switch mit der niedrigsten MAC-Adresse. Die Topologie des Baums wird nun so organisiert, dass die Anzahl der Switch-Übergänge von jedem Switch zum Wurzel-Switch minimal ist. Falls es mehrere Pfade mit gleicher Länge gibt, wird der Pfad über den Switch mit der niedrigsten MAC-Adresse ausgewählt. Alle nicht im Baum enthaltenen Verbindungen werden deaktiviert und stehen bei einem eventuellen Verbindungsabbruch als Reserveverbindung zur Verfügung (siehe BILD 9). Bei einer Topologie Änderung baut RSTP den Baum innerhalb von 6s neu auf. Fällt eine bestehende Verbindung aus, so dauert dieser Prozess nur wenige Millisekunden.

Die ID-Module können über das SNMP-Protokoll aus dem Switch in ihrem Baustein sowohl für die aktive, als auch für die passiven Verbindungen in Richtung des Wurzel-Switches folgende Informationen auslesen:

- Die MAC-Adresse des verbundenen Switches.
- Die lokale Port-Nummer der Verbindung.
- Die Port-Nummer der Verbindung am entfernten Switch.

Die ID-Module können auch die MAC-Adresse des Switches in ihrem Baustein auslesen. Zusätzlich wird jedes ID-Modul mit einer Zuordnung von Switch-Ports zu Baustein-Seiten konfiguriert.

Defekte Bausteine nehmen i.d.R. nicht mehr am Datennetzwerk teil. Deshalb kann mit den bisher vorgestellten Mitteln weder die Existenz eines defekten Bausteins, noch die Tatsache, dass ein Baustein defekt ist, eindeutig und zuverlässig erkannt werden. Aus diesem Grund senden alle Kopplungs-Schnittstellen eines Bausteins ein digitales Signal an das ID-Modul, wenn sie mit einem anderen Baustein gekoppelt sind. Dieses Signal wird auch dann gesendet, wenn der Kopplungsvorgang ursprünglich vom anderen Baustein eingeleitet worden ist.

Mit diesen Mechanismen verfügt jedes ID-Modul über folgende Informationen:

- Über welche MAC-Adresse wird der Baustein im RSTP-Baum repräsentiert?
- Von welchen Ports des Baustein-Switches aus führen aktive oder passive Netzwerkverbindungen über welche Switches zur Wurzel des RSTP-Baums und mit welchen Ports sind diese auf der Gegenseite verbunden?
- Welche Ports des Baustein-Switches sind welchen Baustein-Seiten zugeordnet?
- Welche Bausteinseiten sind mit einem anderen Baustein gekoppelt? Dabei ist nur das Vorhandensein einer Kopplung bekannt, nicht aber die Identität des benachbarten Bausteins.

Jedes ID-Modul kennt dabei nur seine lokale Sicht und den lokalen Ausschnitt aus dem RSTP-Baum.

Das RFSM-Modul sammelt die Informationen aller ID-Module. Aus den Verbindungsinformationen der einzelnen Switches rekonstruiert es den gesamtem RSTP-Baum inkl. der redundanten Verbindungen und erhält so einen Graphen der gesamten Netzwerktopologie. Über die MAC-Adressen der Switches ordnet das RFSM-Modul die Knoten des Netzwerkgraphen den Bausteinen im Satelliten zu und übersetzt die Switch-Ports der Verbindungen anhand der Port-Seiten-Zuordnungen in Baustein-Seiten.

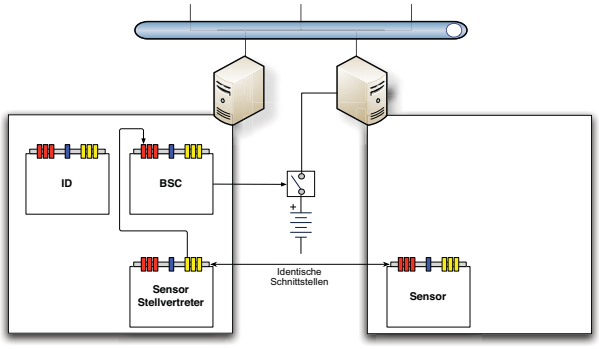

BILD 7. Aufbau bei der Verwendung eines Stellvertreter-Moduls. Der Sensor-Stellvertreter läuft auf demselben Rechner wie das ID-Modul und das BSC-Modul. Das Sensor-Modul läuft auf einem zweiten Rechner, der vom BSC-Modul ein- und ausgeschaltet werden kann.

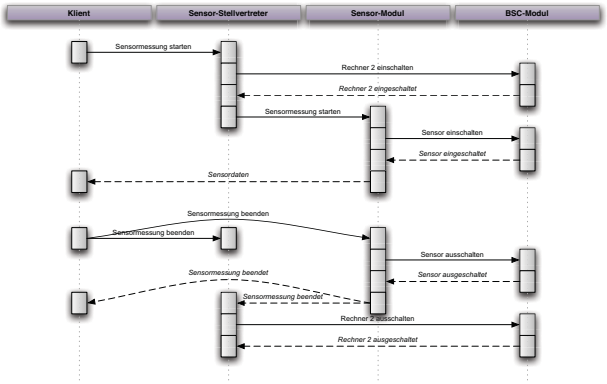

BILD 8. Aktivitätsdiagramm bei der Verwendung eines Stellvertreter-Moduls. Das „Starten“-Kommando des Klienten-SW-Moduls wird vom Stellvertreter empfangen. Dieser aktiviert den Rechner, auf dem das Sensor-Modul läuft, welches ab diesem Zeitpunkt die Kontroller übernimmt. Nach dem „Beenden“-Kommando des Klienten, schaltet der Stellvertreter den Rechner wieder aus, sobald das Sensor-Modul bestätigt hat, dass es selbst und der Sensor sich in einem sicheren Zustand befinden.

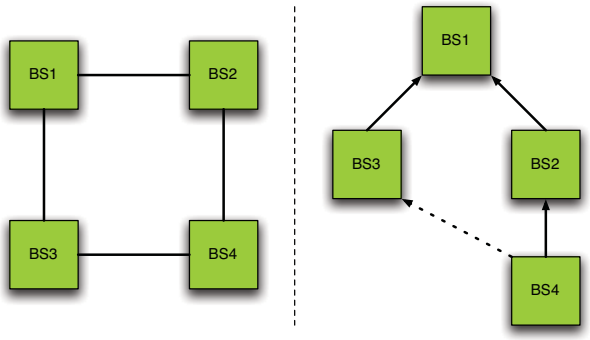

BILD 9. Durch das Rapid Spanning Tree Protokoll (RSTP) wird der Ring (links) in einen Baum umgewandelt. Die Verbindung zwischen BS3 und BS4 steht als Reserveverbindung zu Verfügung.

Mittels der Kopplungsinformation verifiziert es die Satellitenstruktur und fügt ggf. bisher nicht registrierte – und damit defekte – Bausteine hinzu.

Zuletzt ordnet das RFSM-Modul jedem Baustein eine geometrische Position und Orientierung im Satelliten zu. Dazu verwendet es auch die Größeninformation über die Bausteine aus der Satelliten-Ontologie. Als Ursprung für das Koordinatensystem wählt das RFSM-Modul einen beliebigen Baustein aus, z.B. den Baustein mit der höchsten Baustein-ID. SW-Module, welche die Satelliten-Geometrie benötigen, können diese anschließend in ein Koordinatensystem transformieren, das für ihre Zwecke am besten geeignet ist.

5. ZUSAMMENFASSUNG

In diesem Beitrag wurde ein Software-Framework für modulare, rekonfigurierbare Satelliten vorgestellt. Das modulare, verteilte Basis-Framework *MCA3* (*Modular Controller Architecture Version 3*) stellt Mechanismen zur Entwicklung von wiederverwendbaren Modulen und Komponenten zur Verfügung. Es legt die Grundstruktur des Software-Systems fest und kümmert sich – für den Programmierer transparent – um die zeit- oder ereignisgesteuerte Ausführung der Datenverarbeitung sowie die Netzwerkkommunikation. Darüber hinaus bietet es Funktionen zur Verwaltung von verteilten Software-Systemen und zur Diagnose.

Aufbauend auf den Mechanismen von MCA3 wurden Software-Module für modulare, rekonfigurierbare Satelliten definiert. Das ID-Modul identifiziert Satelliten-Bausteine und veröffentlicht Informationen über sie. Hierzu gehören statische Informationen über die Bausteineigenschaften, Verknüpfungen zu benachbarten Bausteinen und dynamische Informationen, wie z.B. die aktuelle Leistungsaufnahme des Bausteins und die aktuell verfügbaren und belegten Rechnerressourcen. Das RFSM-Modul rekonstruiert mithilfe der Informationen von den ID-Modulen den Satellitenaufbau und überwacht und verwaltet den Satelliten anhand einer Satelliten-Ontologie. Das BSC-Modul steuert den Kopplungsvorgang zwischen Bausteinen und kann Komponenten eines Bausteins ein- und ausschalten. Zusätzlich können weitere bausteingebundene und nicht bausteingebundenen SW-Module für die verschiedenen Funktionen eines Satelliten implementiert werden.

Durch die lose Kopplung des verteilten SW-Systems und die automatische Ermittlung der Systemkonfiguration zur Laufzeit ist das Framework in der Lage, System-Rekonfigurationen im laufenden Betrieb zu ermöglichen.

6. DANKSAGUNG

Gefördert durch die Raumfahrt-Agentur des Deutschen Zentrums für Luft- und Raumfahrt e.V. mit Mitteln des Bundesministeriums für Wirtschaft und Technologie aufgrund eines Beschlusses des Deutschen Bundestages unter dem Förderkennzeichen 50RA1007.

7. LITERATUR

[1] J. Weise, K. Brieß, A. Adomeit, H.-G. Reimerdes, M. Göller, R. Dillmann, D. Nölke: *Ein Bausteinkonzept für wartungsfreundliche und rekonfigurierbare Satelliten*. Deutscher Luft- und Raumfahrtkongress, 2011

[2] M. Göller, L. Pfotzer, J. Oberländer, K. Uhl, T. Büttner, A. Rönnau, R. Dillmann: *Modellierung und Konfigurationsgenerierung für bausteinbasierte Satellitensysteme*. Deutscher Luft- und Raumfahrtkongress, 2011

[3] K. Uhl, M. Ziegenmeyer: *MCA2 – An Extensible Modular Framework for Robot Control Applications*. In: *Advances in Climbing and Walking Robots – Proceedings of 10th International Conference (CLAWAR 2007)*, 2007, S. 680-689

[4] M. Henning, S. Vinoski: *Advanced CORBA Programming with C++*. Addison Wesley, 1999

[5] Object Management Group: *Data Distribution Service for Real-time Systems Version 1.2*. OMG Available Specification formal/07-01-01, Januar 2007

[6] K. Sollins: *The TFTP Protocol (Revision 2)*. STD 33, RFC 1350, MIT, 1992

[7] H. Garcia-Molina: *Elections in a Distributed Computing System*. In: *IEEE Transactions on Computers*, Vol. C-31, No. 1, Januar 1982, S. 48-59

[8] L. Walsh: *SNMP MIB Handbook*. Wyndham Press, 2008

[9] LAN MAN Standards Committee of the IEEE Computer Society: *IEEE Std. 802.1D-2004, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*. The Institute of Electrical and Electronics Engineers, Inc., 2004

[10] *SpaceWire – Links, nodes, routers and networks*. ECSS-E-ST-50-12C, European Cooperation for Space Standardization, 31. Juli 2008

[11] LAN MAN Standards Committee of the IEEE Computer Society: IEEE Std. 802.1AB-2005, IEEE Standard for Local and Metropolitan Area Networks: Station and Media Access Control Connectivity Discovery. The Institute of Electrical and Electronics Engineers, Inc., 2005

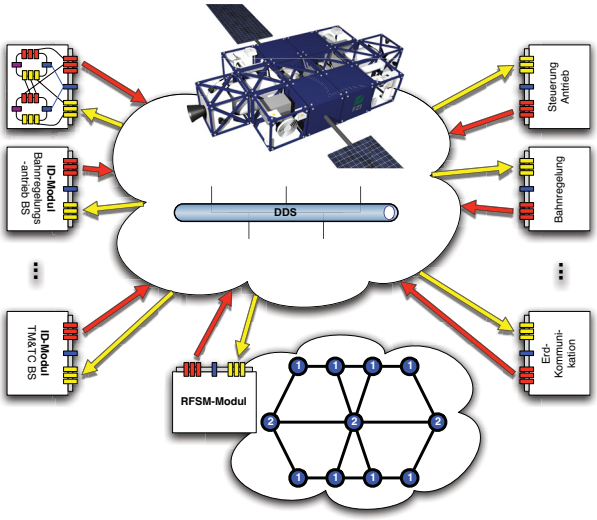

BILD 10. Software-Struktur eines Satelliten. Das KP-Modul hat die Konfiguration des Satelliten (drei Zweifach-BS in der Mitte und jeweils vier Einzel-BS an beiden Seiten) aus den Informationen der ID-Module rekonstruiert.