KONFIGURATIONSMANAGEMENT FÜR INTEGRIERTE MODULARE AVIONIK

M. Halle, F. Thielecke,
TU Hamburg-Harburg, Institut für Flugzeug-Systemtechnik,
Nesspriel 5, 21129 Hamburg, Deutschland

ZUSAMMENFASSUNG

Integrierter Modulare Avionik (IMA) als Nachfolger herkömmlicher Avionik bietet diverse Vorteile in Bezug auf Flexibilität, Wiederverwertbarkeit und Standardisierung. Systeme auf der Basis von IMA bestehen aus einer generischen Hardware Plattform mit einer standardisierten Hardware-/Softwareschnittstelle und Funktionen zum Modulmanagement. Eine Konfiguration definiert, wie Systeme und Systemapplikationen an die Hardware-Module angebunden werden und ist eine wesentliche Komponente im IMA Entwicklungsprozess. Aufgrund der Möglichkeit, mehrere unabhängige Systemapplikationen auf IMA zu implementieren, ist der Entwicklungsprozess im Vergleich zu herkömmlicher Avionik jedoch nicht auf ein System konzentriert, sondern verteilt. Damit erhöht sich der Integrationsaufwand, was eine neue Herausforderung darstellt.
Um die Komplexität des verteilten Konfigurationsprozesses zu minimieren, wird im Rahmen dieses Artikels ein flexibles und erweiterbares Softwareframework für ein Konfigurationswerkzeug vorgestellt. Neben den Standardaufgaben wie Erstellen, Modifizieren, Verifizieren und Verwalten von Konfigurationsdaten erlaubt es ebenso eine grafische Repräsentation, sowie geeignete Import- und Export-Funktionen von bestehenden Konfigurationsdaten. Eine zentrale Aufgabe ist weiterhin eine robuste Konsistenzprüfung der Konfigurationsdaten, um den Integrationsprozess zu vereinfachen.

SCHLAGWORTE

Integrierte Modulare Avionik; IMA; Konfiguration; Datenmodell; Software; Framework

ABKÜRZUNGEN UND SYMBOLE

ADCN	**A**vionics **D**ata **C**ommunication **N**etwork
AFDX	**A**vionics **F**ull **D**uple**X** Switched Ethernet
API	**A**nwendungs**p**rogrammierschnn**i**ttstelle
ATA	**A**ir **T**ransport **A**ssociation
CPIOM	**C**ore **P**rocessing **I**nput **O**utput **M**odule
I/O	**I**nput **O**utput
IMA	**I**ntegrierte **M**odulare **A**vionik
LRU	**L**ine **R**eplacable **U**nit

1 EINFÜHRUNG

Elektronische Komponenten in Flugzeugen zur Steuerung oder Überwachung von Flugzeugsystemen werden als Avionik[1] bezeichnet. Konventionelle Avionik gibt es seit vielen Jahren in Flugzeugen, z.B. für Aufgaben im Bereich der Kommunikation. Üblicherweise war eine Avionik-Komponente exakt einer Funktion (z.B. Kommunikation) zugeordnet. Auf Flugzeugebene bedeutet dies dedizierte und oft lokale Systeme mit optimaler Ausstattung, die auch als „Line Replaceble Unit“(LRU) bezeichnet werden, aber in der Regel einen geringen Wiederverwendungswert besitzen. Trotzdem gibt es sehr viele Gemeinsamkeiten in ihrem Aufbau und den erforderlichen Ressourcen, z.B. im Hinblick auf Akquisition von Daten, Bearbeitung und Kommandos, siehe Abbildung 1.

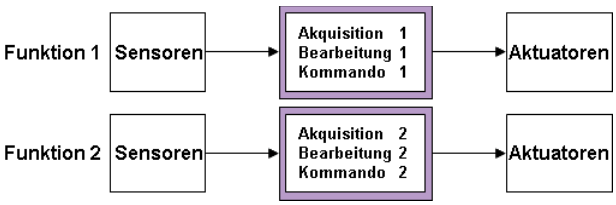

BILD 1: Funktionsweise einer LRU

Das Konzept Integrierter Modularer Avionik (IMA) beschreibt einen Aufbau, in dem die Hardware generalisiert und modular aufgebaut ist. Die Funktionen laufen als spezielle Softwarebausteine auf der Hardware (siehe im Vergleich Abbildung 2).

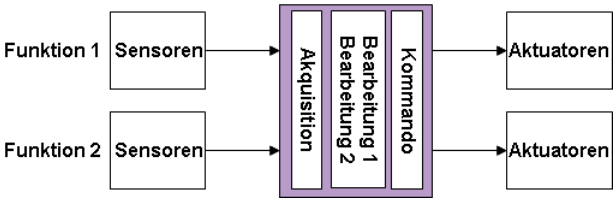

BILD 2: Funktionsweise von IMA

Die Hardwareressourcen werden durch eine Segregation auf die Funktionen einzelner Flugzeugsysteme aufgeteilt. Dadurch wird ein hoher Wiederverwendungswert und optimale Aufteilung der zur Verfügung stehenden Ressourcen erreicht. In modernen Flugzeugen wird konventionelle Avionik zunehmend von IMA Komponenten abgelöst, nicht zuletzt getrieben von einer star-

[1] Der Begriff Avionik ist abgeleitet von Aviation Electronic

ken Zunahme an Funktionen und Schnittstellen. Systeme auf der Basis von IMA sind an das Avionics Data Communication Network (ADCN) mit Hilfe eines auf Ethernet basierendes Netzwerks, dem Avionics Full Duplex Switched Ethernet (AFDX) angebunden (siehe Abbildung 3).

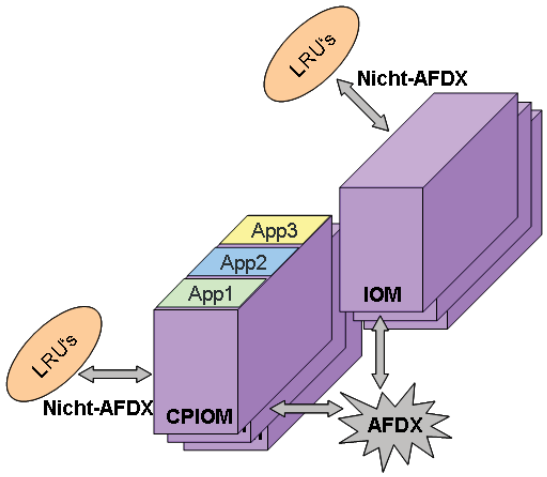

BILD 3: IMA Netzwerk

Das Netzwerk von IMA Komponenten umfasst eine Anzahl von Recheneinheiten, den „Core Processing Input Output Modules“(CPIOM), die im Avionic Bay eines Flugzeugs angesiedelt sind. Auf der Hardware können verschiedene Systemapplikationen mit unterschiedlichen Kritikalitäten ausgeführt werden. Dazu gehören beispielsweise Applikationen aus den Bereichen Cockpit, Energieversorgung, Pneumatik und Kabinensysteme.
Da auf jedem IMA Modul eine, bzw. mehrere Systemapplikationen laufen, die durch eine Partitionierung segregiert sind, definiert eine Konfiguration, wie die Systemapplikationen an die IMA Module angebunden werden. Dabei werden z.B. Hardware-Ressourcen wie CPU-Zeit, Speicher oder I/O für das Nominalverhalten, sowie Rekonfigurationsparameter für vorab identifiziertes Fehlverhalten zugewiesen. Im Gegensatz zur Entwicklung konventioneller Avionik bedeutet dies neue Anforderungen, da der Konfigurationsprozess von einer stark verteilten Struktur geprägt ist.
Der Konfigurationsprozess, an dem mehrere Akteure beteiligt sind, bedient sich verschiedener Softwarewerkzeuge zum Erstellen der Konfiguration. Inkompatibilitäten und fehlende Flexibilität der Werkzeuge, sowie unterschiedliche Formate und Datenmodelle machen diesen Prozess komplex und somit schwer handhabbar. Dieser Artikel präsentiert Beiträge für geeignete Datenmodelle und ein Softwareframework für eine Konfigurationssoftware. Einige wesentliche Aspekte werden exemplarisch beleuchtet und entsprechende Lösungen unter Berücksichtigung auftretender Randbedingungen vorgeschlagen.
Nach einer Einführung in den Konfigurationsprozess in Abschnitt 2 wird das Konzept der Konfigurationssoftware im Detail beschrieben. Den Kern der Software bildet ein modulares Datenmodell, welches in Abschnitt 3 erläutert wird. Das Datenmodell erlaubt es, auch unterschiedliche IMA Komponenten gleichzeitig abzubilden und berücksichtigt Zugriffsrechte und spezialisierte Anforderungen der Akteure im Konfigurationsprozess. Der Aspekt einer verteilten Konfigurationserstellung definiert die grundlegende Struktur des Datenmodells. Anschließend wird das Softwareframework auf der Basis des Datenmodells in Abschnitt 4 erläutert. Ein spezielles Plugin zur Validierung von Integrationsregeln zur Sicherstellung der Konsistenz der Konfigurationsdaten wird in Abschnitt 5 vorgestellt. Abschließend werden die Ergebnisse und Erfahrungen zusammengefasst und ein Ausblick auf kommende Entwicklungen gegeben.

2 ENTWICKLUNGS- UND KONFIGURATIONSPROZESS

Entgegen konventioneller Avionik, bei dem durch die 1:1 Zuordnung von Funktionen und Hardware der Entwicklungsprozess stark lokalisiert war, ist dies bei der Entwicklung von IMA Komponenten nicht der Fall. Die Entwicklung und Konfiguration ist nun gekennzeichnet durch einen iterativ verlaufenden Prozess, der eine verteilte Struktur aufgrund mehrerer Akteure mit unterschiedlichen Rollen aufweist. Im Bereich des Flugzeugherstellers gibt es die Systemabteilungen und den Modulintegrator, daneben Modulhersteller und schließlich die Zulieferer, oder Systemhersteller (siehe Abbildung 4).

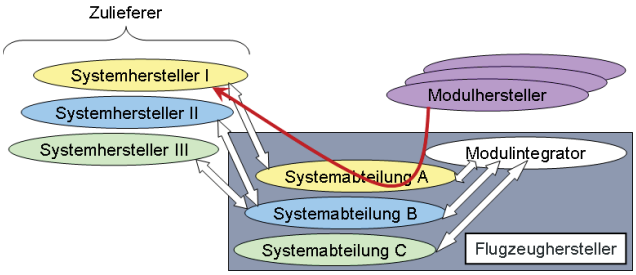

BILD 4: Akteure im Konfigurationsprozess

Der Entwicklungsprozess erfolgt zyklisch in Iterationen [5]. Zunächst werden durch die Systemabteilungen (ATA XX) in einer Spezifikationsphase grundlegende Eigenschaften der IMA Module anhand der Systemanforderungen (wie Konnektoren, ARINC 600 [2]) und Modulanforderungen (wie Partitionierung, ARINC 653 [1]) definiert. Anschließend werden anhand der Systeme, welche auf der IMA Hardware Anwendung finden, Belange wie etwa Signaltypen, Prozesszyklen und Speicheranforderungen festgelegt. Diese Spezifikationen bestimmten die Auslegung der IMA Module durch den Modulhersteller nach dem Prinzip einer generalisierten Hardware, z.B. eines Core Processing Input-Output Modules (CPIOM). Dies beinhaltet bspw. auch das Betriebssystem des CPIOM und legt somit auch die

Anwendungsprogrammierschnittstelle (API) fest. Dazu gehören auch die Parameter, welche die Partitionierung eines IMA Moduls definieren.
Den Systemherstellern wird die Systemspezifikation und die erforderlichen Spezifikation der Schnittstellen (API) zur Entwicklung ihrer Systeme zur Verfügung gestellt. Diese Spezifikation kann sich im Verlauf der Entwicklung noch ändern, explizit sind entsprechende Korrektur-Iterationen vorgesehen.
Um den Systemherstellern die Entwicklung zu ermöglichen, werden ihnen entsprechende Entwicklungswerkzeuge zur Verfügung gestellt. Diese ermöglichen z.B. das Debugging der entwickelten Systemapplikationen für das IMA Modul bereits in einem sehr frühen Entwicklungsstadium, zu dem noch nicht notwendigerweise das IMA Modul physikalisch zur Verfügung steht. Ist das IMA Modul verfügbar, wird es entsprechend der verschiedenen Systemapplikationen der unterschiedlichen Systemhersteller im Hinblick auf die Systemintegration vorkonfiguriert. Die einzelnen Systemapplikationen sind hierbei „entkoppelt“von denen anderer Systemhersteller und werden nacheinander inkrementell integriert. Dies ist aufgrund der Partitionierung von IMA Modulen möglich. Integrationstest anhand bestimmter Regeln gewährleisten, dass die sich Systemapplikationen nicht gegenseitig beeinflussen und stellen somit eine korrekte Partitionierung sicher.
Schließlich werden alle Systeme und Systemapplikationen auf der IMA Hardware in das Flugzeug durch den Modulintegrator (ATA 42) zusammengeführt und im Verbund auf der originalen Hardware getestet. Dabei müssen alle IMA Module entsprechend konfiguriert werden. Die Konfiguration setzt sich aus Teilkonfigurationen zusammen, wird kompiliert und auf die IMA Module geladen, um einen Betrieb zu ermöglichen. Zusammenfassend ist der Entwicklungsprozess schematisch in Abbildung 5 dargestellt.

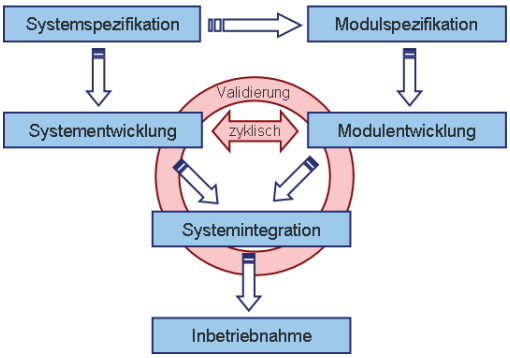

BILD 5: Entwicklungsprozess

Es ist ersichtlich, dass im Konfigurationsprozess zunächst in den Konfigurationsdaten lediglich allgemeine Schnittstellen spezifiziert werden. Im weiteren Verlaufe des Entwicklungsprozesses werden diese Daten dann iterativ immer weiter detailliert und spezifiziert. Sie bestehen aus einen globalen- und einem lokalen Teil. Der globale Teil spezifiziert dabei die vom Modul vorgegebenen Randbedingungen, der lokale Teil die Anbindung der Systemapplikation an die vom Betriebssystem des IMA Moduls vorgegebene API.
Im Konfigurationsprozess haben alle Akteure unterschiedliche Sichten auf die Module, benötigen aber (Teile der) Konfigurationsdaten, bzw. spezifizieren diese. In diesem Entwicklungsprozess entstehen daher diverse „Konfigurationsdatenelemente“mit unterschiedlichen Informationen, die jedoch miteinander verknüpft sind. Wie beschrieben, müssen in der Integrationsphase Regeln beachtet werden, die durch das Modul, die Partitionierung, aber auch durch die Spezifikation der einzelnen Systemapplikationen und deren Interaktion miteinander vorgegeben sind. Der Integrationsprozess scheitert, wenn diese Regeln verletzt werden. Aufgrund der verteilten Konfigurationsentwicklung ist es jedoch häufig nicht möglich, frühzeitig Inkonsistenzen zu erkennen. Dies resultiert in eine neue Konfigurationsiteration, was zusätzlichen Zeitaufwand - und damit Kosten - bedeutet.
Ein weiterer Aspekt tritt gerade in der frühen Entwicklungsphase neuer IMA Komponenten, bzw. Systemapplikationen auf. Hierbei ändert sich die Spezifikation und damit auch das Konfigurationsschema häufig. Dies bedeutet unter Umständen eine Änderung des Formats der Konfigurationsdaten, oder des Umfangs durch obsolete oder zusätzlich erforderliche Konfigurationsparameter. Eine flexible Konfigurationssoftware im Hinblick auf die Konfigurationsparameter ist also wünschenswert.
Um mit der zunehmenden Anzahl der auf IMA basierenden Systeme und -applikationen und der damit immer größer werdenden Komplexität des Konfigurationsprozesses gerecht zu werden, sollte die Erstellung der Konfiguration eng an den Gesamtprozess gekoppelt sein und (soweit möglich) automatisiert werden.

3 MODELL DER KONFIGURATIONSDATEN

Wie im Abschnitt 2 verdeutlicht, sind verschiedene Akteure am Konfigurationsprozess beteiligt. Die erstellten Konfigurationsdaten setzen sich aus mehreren Teilen iterativ zusammen. Beim Design einer Konfigurationssoftware, welche den Konfigurationsprozess geeignet unterstützen soll, müssen solche Aspekte beachtet werden. Die zu erfassenden Konfigurationsdaten durch die einzelnen Akteure beschreiben ein Datenmodell, welches sich am Konfigurationsprozess orientiert. Exemplarisch für IMA Komponenten im Allgemeinen wurde am Beispiel eines Airbus CPIOM ein solches Datenmodell („Domain Model“) entwickelt, welches wesentliche Aspekte des Konfigurationsprozesses, z.B. Akteure und eingeschränkte Zugriffsrechte, berücksichtigt und geeignet abbildet.
Ein Datenmodell ist noch keine Software, sondern lediglich eine Beschreibung der Datenstrukturen, deren Eigenschaften und ggf. Einschränkungen. Um ein Da-

tenmodell zu erstellen, gibt es aus dem Bereich der Informatik verschiedene Werkzeuge zur Modellierung von Datenstrukturen. Beispiele dafür sind die Unified Modelling Language (UML), die Systems Modeling Language (SysML) oder XML Schema (XSD). Im Rahmen dieser Arbeit wurde zunächst ein Abbild der Konfigurationsdaten auf der Basis von XML Schema [16],[4] modelliert.

Angestrebt ist eine „Plug&Play“Struktur der Konfigurationsdaten, in der sich Konfigurationsmodule in einer Baumstruktur aus einzelnen Bestandteilen zusammensetzen. Die Struktur der Konfigurationsdaten des Airbus CPIOM berücksichtigt nur eingeschränkt einen modularen Aufbau. Daher wurde die Struktur modifiziert, zerlegt und erweitert, dass sie einen modularen und objekt-orientierten Charakter aufweist, der es erlaubt, die einzelnen Sektionen der Konfigurationsdaten entsprechend der Akteure und Rollen zusammenzuführen. Das resultierende Datenmodell besteht aus diversen Modulen, welche durch entsprechende Verknüpfungen ein Gesamtmodell abbilden.

Aufgrund des Umfangs des Datenmodells (mehrere 1000 Konfigurationsparameter) ist es nicht möglich, es im Rahmen dieses Artikels im Detail darzustellen. Stattdessen soll der prinzipielle Aufbau im Folgenden anhand einiger wesentlicher Beispiele erläutert werden.

Letztlich werden in den Konfigurationsdaten ganz allgemein ausgedrückt als numerische Werte und Zeichenketten abgelegt. Jeder Konfigurationsparameter hat gemäß seiner Spezifikation einen bestimmten Datentyp und Wertebereich. Die Basis des Datenmodells bildet daher eine Bibliothek, in der diese Datentypen abgebildet sind. Diese beinhalten neben der Beschreibung eines Datentyps (z.B. „Float“oder „String“) auch Einschränkungen des Wertebereiches (z.B. nur positiv, oder in einem Bereich von 1 bis 1024) und Syntaxvorschriften.

Aufbauend auf dieser Bibliothek wurden die spezialisierten Teilmodelle der Konfiguration modelliert (siehe Abbildung 6). Die Wurzel des resultierenden Konfigurationsbaums bezeichnet ein „Aircraft“, welches aus 1..n „Equipment“bestehen kann. Dies gewährleistet das Bearbeiten mehrerer Konfigurationen eines Flugzeugs (z.B. Ausstattungsvarianten), aber auch das Einbinden verschiedener Avionik-Hardware als Spezialisierungen von „Equipment“, z.B. eines CPIOM.

Das Konfigurationsmodul „Connectors“beschreibt die physikalischen Ports (Pins) an einem IMA Modul. Dazu gehören z.B. Namen von Ports und deren Adressen. Es ist neben einem gemeinsamen Teil untergliedert in Abschnitte, welche vom jeweiligen Bussystem (z.B. CAN oder AFDX) abhängig sind.

Der globale, modulabhängigen Teil der Konfiguration („GlobalModule“) beschreibt Parameter der Dienste des Moduls, wie z.B. Watchdogs oder das Health Monitoring. Außerdem gehören hardwarespezifische Parameter, wie etwa Größe des Speichers oder Taktzyklen dazu, welche einen dimensionierenden Charakter haben, z.B. indem sie der den Systemapplikationen zu Verfügung stehenden Ressourcen festlegen.

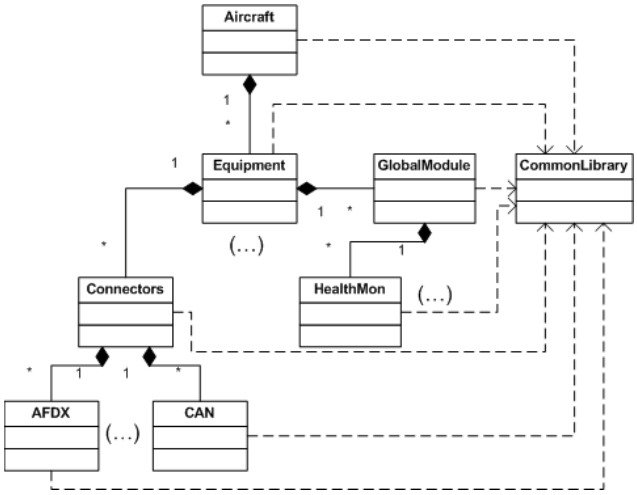

BILD 6: Datenmodell (Auszug 1)

Die Ressourcen eines Moduls werden durch die Partitionierung (siehe Abschnitt 2) den Systemapplikationen zur Verfügung gestellt. Die Konfigurationsparameter in Bezug auf die Partitionierung teilen sich in einen globalen Teil („GlobalPartition“) und einen lokalen Teil („LocalPartition“) auf. Im globalen Teil werden durch den Modulintegrator (siehe 2) die Partitionen zugewiesen und im lokalen Teil das Routing der Informationen an die Systemapplikation unter Verwendung der API vorgenommen (dies nicht in der Abbildung aufgeführt).

Den größten Teil der Konfigurationsdaten bilden die protokollspezifischen Kommunikationsparameter (I/O, siehe Abbildung 7), wie z.B. die Definition der in einem CAN- oder AFDX Protokoll gekapselten Nachrichten (Parameter). Diese Klasse ist unterteilt in einen allgemeinen I/O Teil, der sich anschließend je nach Protokolltyp wiederum in einen datenflussorientierten Teil (z.B. mit Quell- und Zieladresse, wie etwa die Klasse „Bus“in der Abbildung), sowie den „Payload“-Teil aufteilt. Der Payload-Teil (Klasse „Message“in der Abbildung) wird schließlich auf einen Parametertyp spezialisiert, um den konkreten Datentypen und ihren Einstellungen gerecht zu werden.

Bei der Auslegung des Datenmodells wurde versucht, auch kommende IMA Architekturen mit anderer Parametrierung und Funktionalität einzubeziehen, um eine Wiederverwendung zu ermöglichen. Es ist anzumerken, dass die Struktur des Datenmodells an das bestehende Format angelehnt ist. Dies lässt Raum für zusätzliche Optimierungen, allerdings erleichtert es eine Konvertierung bestehender Konfigurationsdaten. Soll die Konfigurationssoftware in den bestehenden Prozess integriert werden, so es nicht ratsam, bestehende Datenstrukturen und Benennungen zu ignorieren.

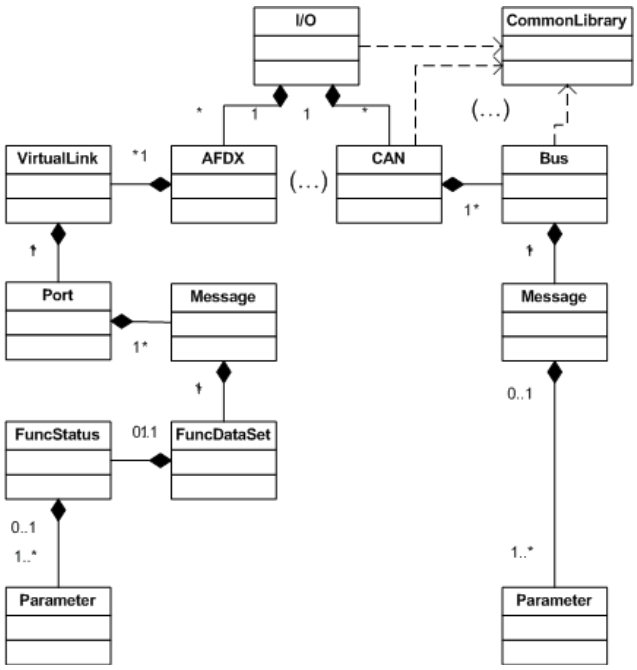

BILD 7: Datenmodell (Auszug 2)

4 SOFTWAREFRAMEWORK

Die bestehende Software-Landschaft zur Konfiguration von IMA Komponenten ist sehr spezialisiert an die Belange der einzelnen Akteure angepasst. Das führt durchaus zu Problemen [12]. Im Rahmen des zugrunde liegenden Projekts wurde eine übergreifende Konfigurationssoftware auf der Basis eines Demonstrators entwickelt. Diese erlaubt neben den Standardaufgaben wie Erstellen, Modifizieren, Verwalten und Austauschen von Konfigurationsdaten ebenso eine grafische Repräsentation, sowie geeignete Import- und Export-Funktionen von bestehenden Konfigurationsdatensätzen (siehe Abbildung 8).

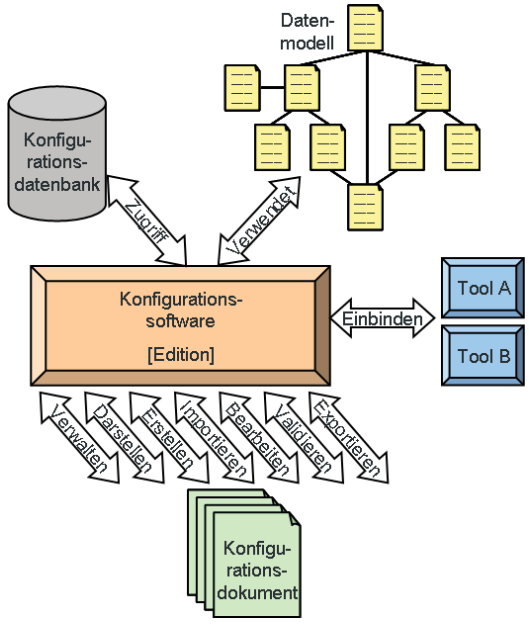

BILD 8: Konfigurationssoftware

Beim Design der Konfigurationssoftware müssen unter anderem folgende Aspekte berücksichtigt werden:

- Unterstützung verschiedener Akteure im Konfigurationsprozess durch Softwarevarianten („Editionen“)
- Validierungsmechanismen zur Konsistenzprüfung der Konfigurationsdaten im verteilten Entwicklungprozess
- Unterstützung des Konfigurationsprozesses für die Lebensdauer eines Flugzeuges
- Erweiterbarkeit im Hinblick auf Unterstützung verschiedener (Generationen von) IMA Komponenten
- Einfache Anpassbarkeit bestehender Datenmodelle
- Bereitstellung von Schnittstellen zur Integration rollenspezifischer Anforderungen und Anwendungen

Dies stellt eine Herausforderung im Hinblick auf die Implementierung der Konfigurationssoftware dar. Nur ein flexibles und erweiterbares Softwareframework kann diesen Belangen gerecht werden. Flexibilität spielt eine entscheidende Rolle, da das Datenmodell nicht als unveränderlich angenommen werden kann, sondern (im Gegenteil) häufigen Änderungen gerade in der frühen Entwicklungsphase, z.B. beim Design neuer IMA Komponenten, unterliegt. Zur Bei der Entwicklung des Softwareframeworks werden daher moderne Softwareentwicklungsmethoden auf der Basis automatischer Quellcodegenerierung verwendet, um diesen Aspekt zu unterstützen. Dies erlaubt sehr effizient und fehlerunanfällig die Anpassung der Konfigurationssoftware an neue Belange.

Mit der Festlegung eines Datenmodells aus Abschnitt 3 ist bereits das Fundament für die Implementierung eines geeigneten Softwareframeworks geschaffen, welches im Folgenden vorgestellt wird. Als Basis wurde das frei verfügbare Eclipse Framework gewählt, da diese Umgebung eine eine ausgereifte Plattform und vielfältige Bibliotheken bereitstellt [14],[7]. Die Konfigurationssoftware basiert auf Java und nutzt das Eclipse Modelling Framework (EMF) und das Graphical Editing Framework (GEF) [15],[11]. Es ist anzumerken, dass EMF ein Framework zur Erstellung sogenannter „Meta-Modelle“ist. Dies dient z.B. zur Beschreibung von Strukturen und Elementen wie UML oder SysML. EMF wird im Rahmen dieser Arbeit jedoch verwendet, um die Konfigurationsparameter abzubilden, da es für diese Aufgabe sehr interessante Aspekte aufweist.

Um das Datenmodell, welches in XML Schema vorliegt, im Softwareframework abzubilden, wurde zunächst eine „Model2Model“(M2M) Transformation [9] nach Ecore [15] vorgenommen. Ecore ist eine weitere Beschreibungssprache für Datenmodelle, jedoch explizit für die Verwendung mit dem Eclipse Framework vorgesehen. Mit Hilfe der Abbildung in Ecore ist es möglich, den Quellcode für wesentliche Teile des Softwareframeworks automatisch generieren zu lassen. Der entscheidende Vorteil dieser automatisierten gegenüber einer herkömmlichen Entwicklungsmethode liegt darin, dass schnell auf Änderungen am Datenmodell reagiert werden kann. Der generierte Quellcode legt eine stan-

dardisierte und generalisierte API fest und ermöglicht es, höhere Anwendungsschichten sehr gut vom Datenmodell zu entkoppeln. Dies entspricht einem „Model-View-Controller“Prinzip [8].
Die automatische Generierung des Quellcodes funktioniert nach folgendem Prinzip: In EMF sind zwei sehr mächtige Werkzeuge zur automatischen Quellcodegenerierung integriert: Java Emitter Templates (JET) und Java Merge (JMerge). Mit Hilfe von JET können auf der Basis einer Java Server Pages (JSP)-ähnlichen Syntax Templates programmiert werden [13], die eine zu generierenden Anwendungsprogrammierschnittstelle (API) beschreiben. JET ist eine generische „Template-Engine“, die verwendet werden kann, um mit Hilfe dieser Templates z.B. SQL, XML oder Java Quellcode zu erstellen. Der Quellcode wird mit Hilfe eines Generatormodells, welches das Datenmodell einbezieht, erstellt. Abbildung 9 verdeutlicht den Prozess.

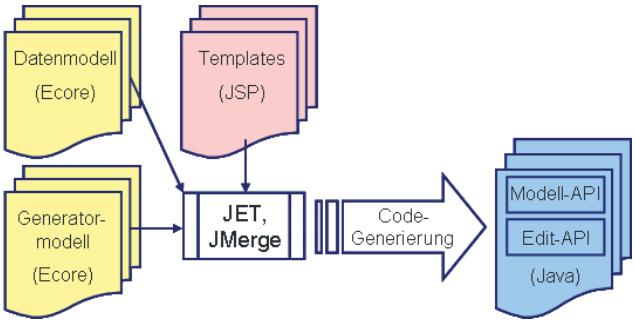

BILD 9: Quellcodegenerierung

Dabei werden aus einem Set von Dateien, die das Datenmodell in Ecore abbilden und einem Set von Java Emitter Templates primär zwei Anwendungsprogrammierschnittstellen erstellt: Ein Abbild des Datenmodells mit all seinen strukturellen Abhängigkeiten als „Modell-API“und eine Bearbeitungsschicht („Edit-API“), mit deren Hilfe man geeignet auf das Datenmodell zugreifen kann. Die Modell-API kapselt alle Datentypen in Objekte, die eine gemeinsames Interface besitzen, um auf ihre Eigenschaften (z.B. Name, Beschreibung, Wert, Meta-Informationen etc.) zugreifen zu können. Dies beinhaltet z.B. auch eine einfache Validierung der zulässigen Werte, wie sie im Datenmodell festgelegt wurden. Die Edit-API beinhaltet Methoden zum Instantiieren der Objekte und einen Ereignis-gesteuerten Zugriff („Command-Response“Prinzip). Dies ermöglicht bspw. eigenständige komplexe Validierungsmechanismen, auf die in Abschnitt 5 noch genauer eingegangen wird.
Die Generation des Quellcodes wird mit Hilfe der Templates an die Belange der Konfigurationssoftware angepasst. Dadurch werden Aspekte wie etwa Zugriffsrechte und Sichtbarkeiten von Parametern geeignet und ebenso automatisch generiert. Abbildung 10 zeigt neben dem beschriebenen Kern den weiteren Aufbau des Softwareframeworks. Dieses basiert auf verschiedenen Schichten und einigen Werkzeugklassen (Utilities).

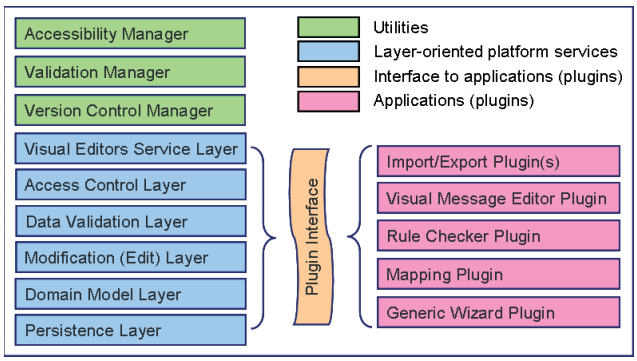

BILD 10: Softwareframework

Zunächst existiert eine Validierungsschicht, welche es erlaubt, unterschiedliche Konzepte zur Validierung der Daten anzubinden, inklusive parameterübergreifende Konsistenzregeln. Eine Zugriffsschicht steuert Lese- und Schreibrechte auf die Daten und kann somit verschiedenen Akteure abbilden. Diese können durch Konfigurationsdateien vorgegeben werden. Eine weitere Schicht stellt spezielle Funktionen bereit, die UI-orientierte Aufgaben vereinfachen, indem entsprechende Methoden zur Verfügung gestellt werden. Dies ist z.B. für visuelle (UI-basierten) Komponenten hilfreich. Alle Schichten sind durch eine Plugin-Schnittstelle auch für externe Anwendungen verfügbar. Generell ist die Konfigurationssoftware daher durch Plugins erweiterbar, um leicht an spezielle Bedürfnisse angepasst werden zu können.
Eine weitere wichtige Schicht ist die Persistenzschicht, welche die Konfigurationsdaten in eine XML basierte Datei ablegt. Andere Formate, etwa der Export in andere Formate, werden über Plugins realisiert. Der Aufbau der XML Datei ist dabei anlog dem unter Abschnitt 3 beschriebenen Datenmodell. Die einzelnen Module sind mittels Namensräumen voneinander getrennt, was das Zusammenführen unterschiedlicher Konfigurationsdaten vereinfacht. Weiterhin ist diese Schicht jedoch auch austauschbar, wenn andere Formate zu bevorzugen sind, etwa um firmenspezifische Prozesse und Tools abzubilden. Da XML Dateien quasi beliebig erweiterbar sind, können auch zusätzliche Attribute an einzelne Parameter oder -gruppen angefügt werden, welche z.B. durch Plugins vorgegeben werden. Eine Anwendung hierfür sind spezialisierte Versionen der Konfigurationssoftware, etwa für unterschiedliche Akteure im Konfigurationsprozess.
Durch die Wahl von XML als Austauschformat ergibt sich in der Anwendung der Konfigurationssoftware ein interessanter Aspekt zur Versionierung: Es ist unvermeidbar, dass sich die zu speichernden Informationen mit einer neuen Version des Datenmodells ändern. Um trotzdem eine Kompatibilität mit einer alten Version herzustellen und diese geeignet importieren zu können, kann eine geeignete XSL Transformation (XSLT) definiert werden [3],[6]. Dies ist im Sinne einer Nachverfolgbarkeit über den Lebenszyklus einer IMA Komponente durchaus relevant und unterstützt eine Wiederverwert-

barkeit der Konfigurationsdaten. Ein Update einer alten Konfiguration entspricht dann quasi dem „Abspielen“aller Transformationsvorschriften.

5 RULE CHECKER

Nachdem das Datenmodell und die Softwarearchitektur vorgestellt wurden, soll nun exemplarisch eine Komponente (das „Rule Checker Plugin“) im Detail beschrieben werden. Wie in Abschnitt 2 bereits angedeutet, werden die Teile der Konfiguration durch den Modulintegrator zusammengeführt. Im Anschluss wird aus den Konfigurationsdaten Binärcode erzeugt, welcher auf die IMA Module geladen wird. Dadurch werden die IMA Module entsprechend konfiguriert, um die gewünschte Partitionierung und das Laufzeitverhalten aufzusetzen. Erst an dieser Stelle im Konfigurationsprozess können alle Teile der Konfiguration auch vollständig auf Konsistenz überprüft werden. Scheitert die Konsistenzprüfung ist der Fehler zwar leicht zu identifizieren, jedoch ist der zeitliche Aufwand, ihn zu beheben, sehr hoch (und ebenso mit Kosten verbunden), da eine neue Version der Konfiguration erstellt werden muss.

Sinnvoll ist daher eine frühzeitige Konsistenzprüfung unter Ausnutzung aller zur Verfügung stehenden relevanten Daten. Als Beispiel für eine Konsistenzregel sei hier eine Adressierung aufgeführt: Eine Parameter im Anhang einer I/O Nachricht (siehe auch 3) besteht unter anderem aus einer Startadresse und einer Länge. Diese Konfigurationsparameter müssen also so gesetzt sein, dass sich die Bereiche nicht überschreiten. Zusätzlich dürfen durch die Partitionierung vorgegebene Speicherbereiche auch über die Systemapplikationen hinweg nicht verletzt werden. Da in einem IMA Modul häufig mehrere Systemapplikationen von unterschiedlichen (durchaus verteilten) Systemhersteller implementiert sind, werden diese Teile der Konfiguration unabhängig voneinander erstellt, was eine weitere Randbedingung darstellt. Daher ergeben sich folgende wesentliche Fragestellungen für ein Softwaremodul, welches eine geeignete Konsistenzprüfung realisiert:

- Welche Informationen (Konfigurationsparameter) müssen bei welchem Akteur im Konfigurationsprozess vorliegen, um eine möglichst vollständige frühe Konsistenzprüfung zu erlauben?
- Wie sehen die Konsistenzregeln aus und lässt sich eine Verallgemeinerung (Kategorisierung) ableiten?
- Wie werden die durchaus komplexen Konsistenzregeln geeignet implementiert, so dass sie ggf. de-/aktiviert oder angepasst werden können?
- Wie werden die Konsistenzregeln zu Laufzeit der Konfigurationssoftware ausgewertet und behandelt?

In Bezug auf die Abbildung der Konsistenzregeln wurden alle Regeln, welche üblicherweise durch den Modulhersteller vorgegeben sind, analysiert und kategorisiert, um daraus ein Regelwerk für die Konfigurationssoftware abzuleiten. Besondere Beachtung fanden dabei Konsistenzregeln, die an mehrere Konfigurationsparameter gekoppelt sind, da diese nicht durch ein Datenmodell abgebildet werden können. Aufgrund des Datenmodells sind lediglich Einschränkungen vorgesehen, die sich auf einzelne Parameter beziehen, wie etwa Wertebereiche oder Syntaxvorschriften. Zusammenfassend ergeben sich die in der folgenden Aufzählung aufgeführten Regeltypen.

Bereich: Der numerischer Wert eines Konfigurationsparameters kann nur in einem bestimmten Intervall liegen oder nur bestimmte Werte annehmen (Enumeration) oder eine Zeichenkette darf eine gewisse Länge nicht überschreiten, teilweise in Abhängigkeit vom Zustand anderer Konfigurationsparameter.

Eindeutigkeit: Der Wert eines Konfigurationsparameters muss innerhalb einer Gruppe oder global eindeutig sein und darf nicht mehrfach verwendet werden, z.B. eine ID.

Konsistenz: Es existiert ein komplexer mathematischer Zusammenhang zwischen mehreren Konfigurationsparametern. Dies erfordert das Ermitteln einer bestimmten Anzahl von Konfigurationsparametern, eine Berechnungsvorschrift wie mit den ermittelten Konfigurationsparametern umgegangen werden soll und ein Konsistenzkriterium. Allerdings bezieht sich dies ausschließlich auf numerische Parameter.

Notwendigkeit: Ein Konfigurationsparameter muss angegeben werden und darf nicht fehlen, teilweise in Abhängigkeit vom Zustand anderer Konfigurationsparameter.

Option: Ein Konfigurationsparameter oder eine Gruppe von Konfigurationsparametern ist nur unter bestimmten Bedingungen optional, z.B. in Abhängigkeit vom Zustand eines anderen Konfigurationsparameters. Andernfalls muss er angegeben werden.

Syntax: Die Syntax eines Konfigurationsparameters ist an eine Vorschrift geknüpft, z.B. durch eine RegEx [10]. Teilweise bezieht sich dies nur auf einen Teil des Konfigurationsparameters.

Nach der Analyse der Kategorien und der erforderlichen Informationen, bzw. Referenzparameter lassen sich alle Konsistenzregeln generell noch in zwei Klassen einteilen: Die erste Klasse beschreibt Konsistenzregeln, die schnell zu evaluieren sind und daher „on-the-fly“berechnet werden können, z.B. direkt nach der Eingabe in die Konfigurationssoftware. Die zweite Klasse beinhaltet Konsistenzregeln, die entweder nur aufwendig zu evaluieren sind, oder nur zu einem bestimmten Zeitpunkt (z.B. nach Abschluss der Eingabe einer Gruppe von Konfigurationsparametern).

Außerdem sind noch weitere Randbedingungen zu beachten: Die Konsistenzregeln sind nicht als statisch anzusehen. Das bedeutet, dass sie sich z.B. im Verlaufe der Entwicklung des IMA Moduls ändern, obsolet, oder gar durch neue Regeln ersetzt werden können. Ein dynamisches Regelwerk in Form von Definitionsdateien ist also wünschenswert. Nicht zuletzt soll die Möglich-

keit bestehen, alle Konsistenzregeln nacheinander abzuarbeiten und die Ergebnisse in einem Protokoll zusammengefasst zu bekommen.

Diese Anforderungen eines dynamischen Regelwerks geeignet in der Konfigurationssoftware umzusetzen ist nicht trivial. Da das Regelwerk parametrisierbar und austauschbar sein muss, kann es nicht direkt in die Konfigurationssoftware integriert werden, sondern wird in Definitionsdateien abgelegt. Daher wurde zunächst ein geeignetes Format für die Definitionsdateien festgelegt. Alle Konsistenzregeln bestehen aus einem gemeinsamen Teil, welcher z.B. eine ID, einen Namen, eine Version und eine Beschreibung beinhaltet. Ein Schlüsselparameter legt anschließend den Typ der Konsistenzregel fest. Letztlich folgen je nach Typ die zur Abarbeitung der Regel erforderlichen Parameter. Im Falle einer Bereichsregel wären dies also zunächst eine Kennung des Parameters und der Minimal- und Maximalwert. Die Kennung des Konfigurationsparameters erfolgt mit Hilfe einer eindeutigen URL, die Dank des objekt-orientieren Datenmodells gegeben ist (siehe 3). Die URL wird verwendet, um die Konsistenzregeln den Konfigurationsparametern zuzuordnen. Werden zur Berechnung der Konsistenzregel weitere Parameter benötigt, sind sie nach dem gleichen Schema abgelegt. An jeden Konfigurationsparameter können somit 1..n Konsistenzregeln angebunden werden (siehe Abbildung 11).

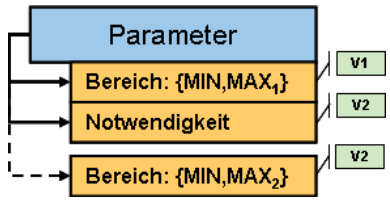

BILD 11: Verknüpfung der Konsistenzregeln

In der Anwendung werden die vorhandenen Regeln aus ihren Definitionsdateien kompiliert und angewandt. Das Rule-Checker Plugin der Konfigurationssoftware verfügt dazu analog der Kategorien über entsprechende Module zum Bearbeiten der Konsistenzregeln, die sich genau mit den Parametern aus der Definitionsdatei konfigurieren lassen. Konsistenzregeln werden zwei Modi bearbeitet: Zum Einen automatisch, um die einfachen (atomaren) Konsistenzregeln „on-the-fly“abzuarbeiten; zum Anderen gibt es einen „Batch“Bearbeitungsmodus, der manuell gestartet wird. In diesem Fall werden nacheinander alle Konsistenzregeln abgearbeitet und in einem Protokoll entsprechend zusammengefasst. Konsistenzregeln, die nicht anwendbar sind - z.B., weil die Referenzparameter in der (Teil-)Konfiguration nicht verfügbar sind - werden dabei ignoriert. Abbildung 12 zeigt schematisch das Ablaufdiagramm. In der Konfigurationssoftware erlaubt ein einfacher Editor sogar das Anpassen der Konsistenzregeln.

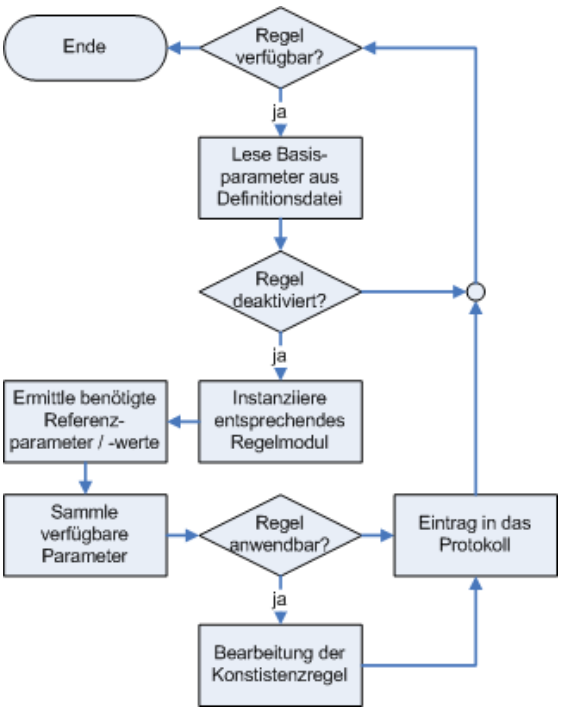

BILD 12: Abarbeitung der Konsistenzregeln

Das Rule Checker Plugin bietet zusammenfassend folgende Eigenschaften:

- Das Regelwerk besteht in Form von Definitionsdateien, welche evtl. sogar automatisch erstellt werden können (z.B. durch den Modulhersteller), da ihr Format eindeutig spezifiziert ist.
- Das Regelwerk kann an die Akteure angepasst werden. Dies geschieht einerseits, um Konsistenzregeln auszuschließen, die für bestimmte Akteure nicht anwendbar sind und andererseits, um sie ggf. außer Kraft zu setzen, bzw. zu modifizieren.
- Konsistenzregeln können möglichst früh angewandt werden, indem man verschiedene „Editionen“des Regelwerks (der Definitionsdateien) anlegt und z.B. direkt beim Systemhersteller installiert.
- Ein Austausch der Konsistenzregeln (z.B. für ein Update) ist einfach möglich, indem man das Set Definitionsdateien festlegt. Die Konfigurationssoftware braucht dafür nicht angepasst zu werden.
- Die Definitionsdateien der Konsistenzregeln können validiert werden, um auf korrekte Syntax zu überprüfen.

6 ZUSAMMENFASSUNG UND AUSBLICK

Da immer mehr Systeme auf IMA Module verlagert werden, nimmt die Komplexität und damit auch die Heterogenität der IMA Module und Prozesse zu. Umso entscheidender ist es, auch den begleitenden Prozess der Konfiguration durch eine Konfigurationssoftware geeignet zu unterstützen und auszubauen. Dieser Artikel stellt das Konzept einer Konfigurationssoftware vor, welches den steigenden Anforderungen des Prozes-

ses durch ein flexibles und einfach erweiterbares Softwareframework gerecht wird.

Zunächst wurde auf die Historie und die Eigenschaften der IMA Module eingegangen. In einem Konfigurationsprozess werden die IMA Module entsprechend der Systemapplikationen, welche IMA nutzen, im Hinblick auf Ressourcen und Datenverkehr (I/O) konfiguriert. Der Konfigurationsprozess ist durch eine stark verteilte Struktur gekennzeichnet und wurde zunächst insbesondere in Bezug auf die am Prozess beteiligten Akteure analysiert und vorgestellt.

Die daraus abgeleiteten Rahmenbedingungen definieren eine geeignete, den Konfigurationsprozess unterstützende, Konfigurationssoftware. Im Rahmen dieses Artikels wurde auf Methoden eingegangen, um ein geeignetes und flexibles Datenmodell der Konfigurationsdaten am Beispiel eines CPIOM zu entwerfen und zu implementieren. Das objekt-orientierte Datenmodell ist erweiterbar und die Entwicklungsmethode übertragbar und kompatibel zu anderen IMA Komponenten.

Ein Softwareframework für eine erweiterbare Konfigurationssoftware wurde vorgestellt, die mit Hilfe moderner Entwicklungsmethoden wie automatischer Quellcodegenerierung ebenso einfach wartbar, anpassbar, wie erweiterbar ist. Ein auf Eclipse basierender Softwarekern erlaubt über ein Plugin-Interface auch das Einbinden fremder (unbekannter) Softwaremodule im Hinblick auf spezielle Belange einzelner Akteure im Konfigurationsprozess. Das Format der Konfigurationsdateien ist XML basiert und austauschbar. Konfigurationsdateien können auch in andere Formate überführt werden, um eine Kompatibilität zu anderer Software herzustellen.

Am Beispiel des „Rule Checker“Plugins, welches es erlaubt, einfache und komplexe Konsistenzregeln zu prüfen, wurde eine Komponente der Konfigurationssoftware vorgestellt. Ziel des Plugins ist es, möglichst früh im Konfigurationsprozess die Integrität der Konfigurationsdaten sicherzustellen. Dazu wurde untersucht, welche Konsistenzregeln existieren und diese in ein geeignetes Regelwerk überführt. Die Konfigurationssoftware wertet diese Konsistenzregeln aus und protokolliert, wenn Fehler auftreten. Dies erfolgt in Abhängigkeit der Akteure möglichst vollständig und früh im Konfigurationsprozess.

Neben den vorgestellten Komponenten existieren auch noch weitere Plugins, um z.B. Konfigurationsparameter auf der Basis einer strukturierten Abfrage der Daten geeignet einzugeben. Außerdem erlaubt ein grafischer Editor auch die Visualisierung der Konfigurationsdaten, z.B. im Falle der Kommunikationsparameter (Kommunikationsprotokolle wir AFDX und CAN).

Nächste Schritte im sind das Modellieren anderer IMA Komponenten nach dem gleichen Prinzip, um z.B. eine teil-automatisierte Konfigurationserstellung zu ermöglichen. Denkbar wäre dies z.B. im Hinblick auf vergebene Ressourcen und Bandbreiten und schließlich der Verteilung der Systemapplikationen auf die einzelnen IMA Module.

DANKSAGUNG

Die Autoren danken der Airbus Deutschland GmbH für die Förderung und freundliche Unterstützung des Forschungsprojekts *Configuration Concurrent Engineering*.

LITERATUR

[1] Aeronautical Radio Incorporated (ARINC) (Hrsg.): *ARINC 653 Standard, Avionics Application Software Standard Interface*. Aeronautical Radio Incorporated (ARINC), 2006

[2] Aeronautical Radio Incorporated (ARINC) (Hrsg.): *ARINC 600 Standard, Air Transport Avionics Equipment Interfaces*. Aeronautical Radio Incorporated (ARINC), 2010

[3] Berglund, Anders (Hrsg.): *Extensible Stylesheet Language (XSL)*. W3C, 2006 (W3C Recommendation). http://www.w3.org/TR/xsl/

[4] Biron, Paul V. (Hrsg.) ; Malhotra, Ashok (Hrsg.): *XML Schema Part 2: Datatypes*. Second. W3C, 2004 (W3C Recommendation). http://www.w3.org/TR/xmlschema-2/

[5] Butz, H.: The Airbus approach to open integrated avionic modular avionics (IMA). In: *Workshop on Aircraft System Technologies (AST)*, 2007

[6] Clark, James (Hrsg.): *XSL Transformations (XSLT)*. W3C, 1999 (W3C Recommendation). http://www.w3.org/TR/xslt/

[7] Clayberg, Eric ; Rubel, Dan: *Eclipse Plugins (Third Edition)*. Addison-Wesley Professional, 2008 http://www.qualityeclipse.com/. – ISBN 0321553462

[8] Freeman, Elisabeth ; Freeman, Eric ; Bates, Bert ; Sierra, Kathy: *Head First Design Patterns*. O'Reilly Media, 2004 http://oreilly.com/catalog/9780596007126. – ISBN 0596007124

[9] Gaaevic, Dragan ; Djuric, Dragan ; Devedzic, Vladan ; Selic, Bran: *Model Driven Architecture and Ontology Development*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 3540321802

[10] Goyvaerts, Jan ; Levithan, Steven: *Regular Expressions Cookbook*. O'Reilly Media, 2009 http://oreilly.com/catalog/9780596520694

[11] Moore, Bill et a. (Hrsg.): *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM Redbooks, 2004 http://www.redbooks.ibm.com/abstracts/sg246302.html. – ISBN 0131425420

[12] Scheerer, F.: IMA Technologie aus der Sicht eines Systemherstellers. In: *Deutscher Luft- und Raumfahrtkongress*, 2004

[13] Seeboerger-Weichselbaum, Michael: *JavaServer Pages Kompendium. Mit CD-ROM.* Markt und Technik, 2004. – ISBN 3827264634

[14] Silva, Vladimi: *Practical Eclipse Rich Client Platform Projects.* Apress, 2009 http://apress.com/book/view/9781430218272. – ISBN 9781430218272

[15] Steinberg, Dave ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed: *EMF: Eclipse Modeling Framework (2nd Edition).* Second. Addison-Wesley Longman, Amsterdam, 2008 http://my.safaribooksonline.com/9780321331885. – ISBN 0321331885

[16] Thompson, Henry S. (Hrsg.) ; Beech, David (Hrsg.) ; Maloney, Murray (Hrsg.) ; Mendelsohn, Noah (Hrsg.): *XML Schema Part 1: Structures Second Edition.* Second. W3C, 2004 (W3C Recommendation). http://www.w3.org/TR/xmlschema-1/