Fliegen mit dem SysML-Bus

Dokumentation und Verifikation von Feldbus-Systemen mit SysML

Axel Scheithauer, oose Innovative Informatik eG
Malte Rahm, Deutsches Zentrum für Luft- und Raumfahrt, Institut f. Flugsystemtechnik

Die Autoren haben ein SysML-Modell für die High Altitude Platform des Deutschen Zentrums für Luft- und Raumfahrt erstellt. Für die Modellierung von Bussen wurde dabei ein SysML-Profil entwickelt, das es ermöglicht, die Busnachrichten, deren Quellen und Senken und die Bus-Verbindungen über beliebige Verschachtelungstiefen hinweg zu beschreiben. Zusätzlich wurden Regeln beschrieben, die die Gültigkeit der Busse prüfen. Unser verwendetes Tool markiert dann die fehlerhaften Elemente und gibt eine Fehlermeldung aus. Aus dem Modell lässt sich schließlich jederzeit eine Dokumentation der Busse erzeugen.

Das Deutsche Zentrum für Luft- und Raumfahrt entwickelt aktuell eine hochfliegende Solarplattform. Für die Anforderungen und die Systemarchitektur dieser Plattform wurde ein SysML Modell [3] erstellt, mit dem die Komplexität des Gesamtsystems kommuniziert werden kann und Inkonsistenzen vermieden werden sollen. Auch die genutzten Datenbusse sollen in diesem Modell dargestellt und verwaltet werden. Von diesem modellbasierten Vorgehen erwarteten wir Vorteile gegenüber dem dokumentenbasierten Vorgehen, insbesondere die Validierbarkeit und die Möglichkeit zum Datenexport. Gleichzeitig sollen das Eintragen und Verändern von Informationen im Modell nicht zu aufwendig sein. Schnell wurde uns klar, dass wir dafür eine eigene Erweiterung der SysML benötigen würden. Diese Erweiterung in Form einer Sammlung von Stereotypen (ein SysML-Profil) soll im Folgenden vorgestellt werden. Dazu werden zuerst die wichtigsten Stereotypen des Profils eingeführt und die Nutzung der Stereotypen im Modellierungskonzept erläutert. Abschließend wird die Nutzung des Profils in Modellabfragen an Hand der Validierungen und der Exporte gezeigt.

Stereotypen des Profils

Die drei zentralen Begriffe sind Bus, Parameter und Message. Dafür haben wir je einen Stereotyp definiert.

Bus

Ein *Bus* ist die Repräsentation des Feldbusses im SysML Modell (Abbildung 1). Als Basiselement haben wir die Metaclass Port gewählt. Betrachtete Alternativen für die Modellierung während der Entwicklung des Profils sind die Nutzung eines «Block» als generalisierten Stereotypen oder der Metaclass Connector.
Den Bus als Generalisierung des Stereotyps «Block» zu modellieren, wie es zum Beispiel in [1] gezeigt wird, hat verschiedene Nachteile. Einerseits müsste ein Bus, der Systemelemente verschiedener Subsysteme verbindet, auf der gleichen Ebene modelliert

und im Block Definition Diagram gezeigt werden, wie die Subsysteme. Dabei ist dieser auf den ersten flüchtigen Blick gleich einem Subsystem, nur der Stereotyp unterscheidet sich. Andererseits muss dann sehr stark auf eine konsistente Modellierung entweder mit Ports für jeden angeschlossenen Connector oder immer ohne Ports geachtet werden, sonst leidet die Verständlichkeit bei Modelllesern und automatische Abfragen im Modell zum Beispiel für derived Properties oder Validierungen werden komplex.
Die Verwendung von Connectoren läge eigentlich nahe, da ein Bus ja eine Verbindung zwischen n Busteilnehmern darstellt. N-äre Connectoren unterstützt unser Tool allerdings nicht, obwohl der Standard es zulässt. Stattdessen mehrere Connectoren für einen Bus zu verwenden ist auch nicht praktikabel, weil hierbei die Modellabfragen sehr komplex werden. Connectoren für die Modellierung des Busses werden in [2] verwendet, wobei hier keine Validierungen oder Exporte angestrebt werden.
Die Verwendung der Metaclass Port ist dagegen vorteilhaft. Es gibt damit ein zentrales Element, das den Bus repräsentiert. Das kommt den Erwartungen der Leserinnen des Diagramms entgegen und ist ein guter Startpunkt für Modellabfragen.
Der Bus wird immer am Block definiert, der alle Busteilnehmer direkt oder indirekt besitzt. Da er nur intern verwendet wird, wird er als privat gekennzeichnet, wodurch das Portsymbol nicht mehr überlappend, sondern bündig mit dem Rahmen des Blocks gezeigt wird (siehe Abbildung 4).

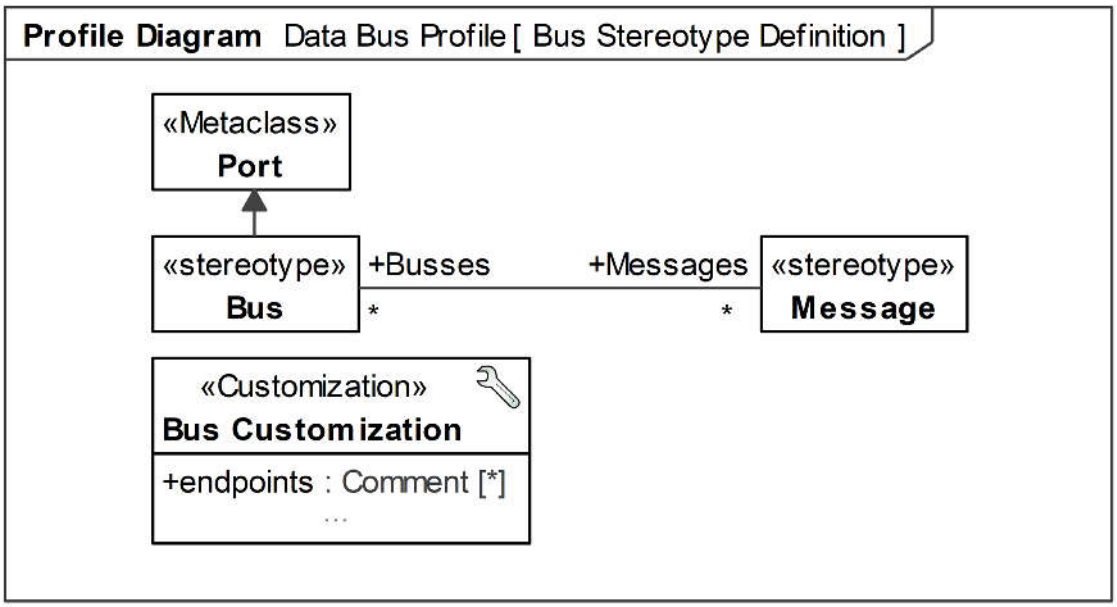

Abbildung 1: Definition des Bus Stereotypen

Ein Bus hat einen Namen und als zusätzliches Attribut die auf diesem Bus zu übermittelnden Messages (Definition s. u.). Dabei wird durch die Multiplizität * dargestellt, dass auf einem Bus verschiedene Messages übermittelt werden können. Unser Tool erlaubt es zudem, mittels einer Customization derived Properties zu definieren, die automatisch berechnet werden. Die endpoints enthalten alle über den Bus erreichbaren Systemelemente über alle Dekompositionsebenen.

Parameter

Ein *Parameter* repräsentiert die Daten, die über den Bus versendet werden sollen. Diese werden meist mit einer gewissen Regelmäßigkeit aktualisiert und neu versendet. Parameter werden im Profil mit einem eigenen Stereotypen modelliert, wie in Abbildung 2 gezeigt wird. Dieser ist eine Spezialisierung des Stereotypen „ValueType“ und hat deswegen die Metaclass DataType.

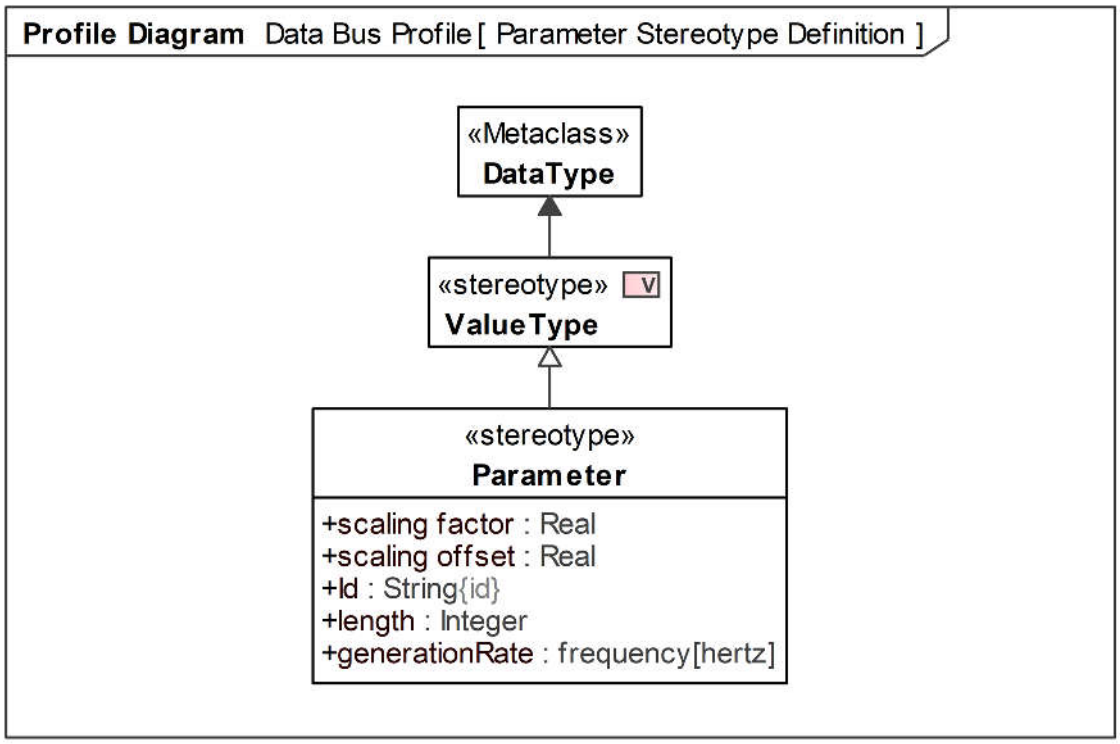

Abbildung 2: Definition des Parameter Stereotypen

Damit können die Parameter grundsätzlich eine QuantityKind und eine Einheit haben. Darüber hinaus soll die Skalierung der Daten auf dem Feldbus modelliert werden. Dies erfolgt einerseits mit den beiden Attributen „scaling factor“ und „scaling offset“, mit denen eine lineare Skalierung abgebildet wird, und einer Generalisierungsbeziehung zu einem Datentyp, wie zum Beispiel Unsigned Integer. Diese Datentypen werden in einer separaten Library gepflegt, die für die meisten Anwendungsfälle mindestens die Datentypen Signed und Unsigned Integer, Floating Point und Boolean beinhalten sollte. Die Regelmäßigkeit, mit der die Daten aktualisiert werden, wird im Attribut „generationRate“ erfasst. Um eine bessere Nachverfolgbarkeit bei Namensänderungen zu gewährleisten, wird eine „Id“ genutzt. Das Attribut „length“ gibt die Länge des Parameters auf dem Feldbus in Bit an, wobei hier nur die Datenbits gezählt werden, etwaige Stuffbits oder Start/Stopbits sollen nicht mitgezählt werden.

Message

Eine *Message* stellt den Zusammenschluss von mehreren Parametern zu einer Nachricht auf dem Bus dar. Als Basiselement haben wir die Metaclass Signal gewählt, weil dieses Modellelement den Datenfluss zwischen zwei Systemelementen darstellt. Die Definition des Stereotypen ist in Abbildung 3 dargestellt.

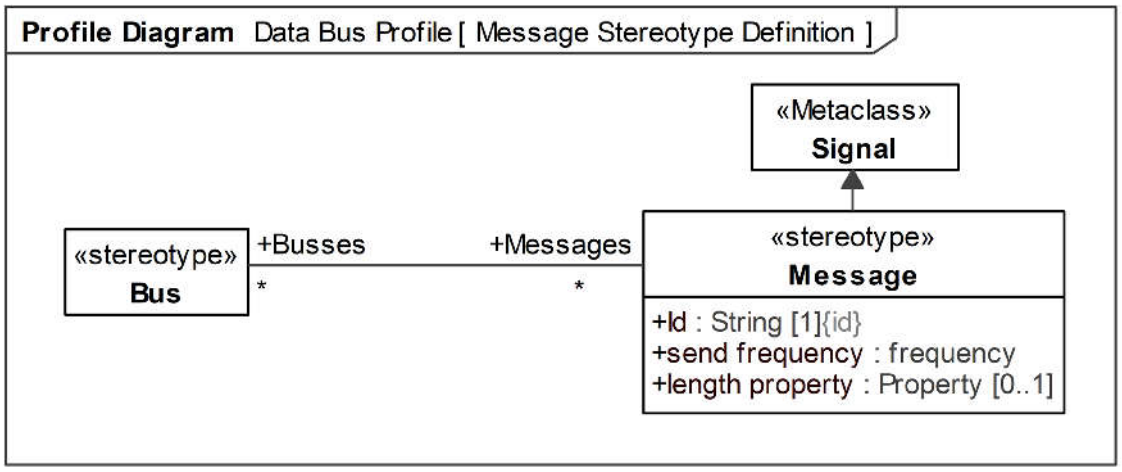

Abbildung 3: Definition des Message Stereotypen

Wie auch die Parameter werden auch die Messages in vielen Fällen mit einer festen Rate gesendet. Diese wird mit dem Attribut „send frequency“ modelliert. Auch eine Id soll wie bei den Parametern vergeben werden. Für den Stereotypen ist nur wichtig, dass eine Länge der Nachricht angegeben wird, dabei kann es sich bei der Länge um die reine Payloadlänge handeln oder um die gesamte Länge der Nachricht mit Protokolloverhead. Deswegen wird hier das Attribut „length property“ genutzt, das vom Typ Property ist, damit auf eine Value Property gezeigt werden kann. Damit kann auch die Länge entweder in Bit oder Byte gezählt werden. Außerdem hat die Message ein Attribut, in dem die zugehörigen Busse gezeigt werden. Die Multiplizität dieses Attributes ist *, da eine Message auch auf mehreren Bussen gesendet werden kann, zum Beispiel auf Grund von Gateways oder redundanten Bussen.

Modellierungskonzept

Der grundsätzliche Top-Down-Ansatz des Systems Engineerings findet sich auch in der Modellierung der Feldbusse wieder. So werden im Systemdesign zuerst die Busse in den Internal Block Diagrams modelliert, wobei noch nicht wichtig ist, welche Informationen wie übertragen werden. Einen ersten Hinweis geben die Namen und Typen der Busse. Im weiteren Verlauf des Systemdesigns werden dann die Parameter erfasst, die von den Geräten generiert oder benötigt werden. Zur Modellierung eignen sich Tabellen für die Attribute und Matrizen für die Generalisierung auf die Datentypen und die Zuordnung zu den Geräten.
Wenn die Busse und die Parameter modelliert sind, kann die Modellierung der Messages anfangen. Hierzu können auch Tabellen und Matrizen sowie Block Definition Diagrams genutzt werden. Außerdem werden weitere Stereotypen des Profils benötigt. Für jeden Parameter, der mit einer Message übertragen werden soll, bekommt diese Message eine Property, die als Typ den Parameter besitzt. Für den Aufbau der Nachricht müssen noch weitere Informationen modelliert werden. Dies ist einmal die Byte-Reihenfolge, mit der die Daten übertragen werden, und andererseits die Positionierung der Daten in der Message, die mit der Angabe des Startbits modelliert wird. Für diese Informationen wird der Stereotyp «payload property», der auf der Metaclass Property basiert, genutzt. Darüber hinaus sollen die Messages nicht nur den Bussen zugeordnet werden,

sondern auch den Sendern und Empfängern. Diese Zuordnung muss kontextbezogen sein, weswegen auf die PartProperty referenziert werden muss und nicht auf den Block. Auch muss die Referenz über mehrere System Breakdown Ebenen genau sein, so dass die Referenz den kompletten Pfad enthalten muss. Die SysML unterstützt das (allerdings leider längst nicht alle Tools). Darüber hinaus muss definiert sein, ob dieses Systemelement der Sender oder der Empfänger der Message ist. Deswegen werden zwei Stereotypen genutzt: «MessageReceiver» und «MessageSender». Diese sind Spezialisierungen vom Stereotyp «DirectedRelationshipPropertyPath», sodass der Kontext und der Property Path für Empfänger bzw. Sender modelliert werden können.

Validierungen

Nachdem alle Informationen über Quellen, Senken und Verbindungen im Modell erfasst sind, kann nun formal geprüft werden, ob das Bussystem korrekt ist.
Einige Prüfungen sind da sehr einfach: Passen zum Beispiel alle benötigten Parameter überlappungsfrei in den durch die Message definierten Platz? Da bei jedem Parameter der Platzbedarf angegeben ist (Parameter.length) und die Message eine Gesamtlänge (Message.length property.default value) enthält, kann das direkt geprüft werden.
Andere Prüfungen sind komplizierter: Ist über beliebig tiefe Verschachtelungsebenen eine Quelle für jeden benötigten Parameter erreichbar? Das wird richtig kompliziert, wenn die gleiche Komponente mehrfach eingesetzt wird (ECU1 in Abbildung 4). Es kann dann sein, dass sie an einer Stelle alle Anforderungen erfüllt, an anderer Stelle aber ein Parameter fehlt. Dies manuell zu prüfen ist sehr fehleranfällig. Durch unsere automatische Validierung können verschiedene Varianten des Bussystems schnell durchgespielt werden, wobei sichergestellt ist, dass jede Variante gültig ist.

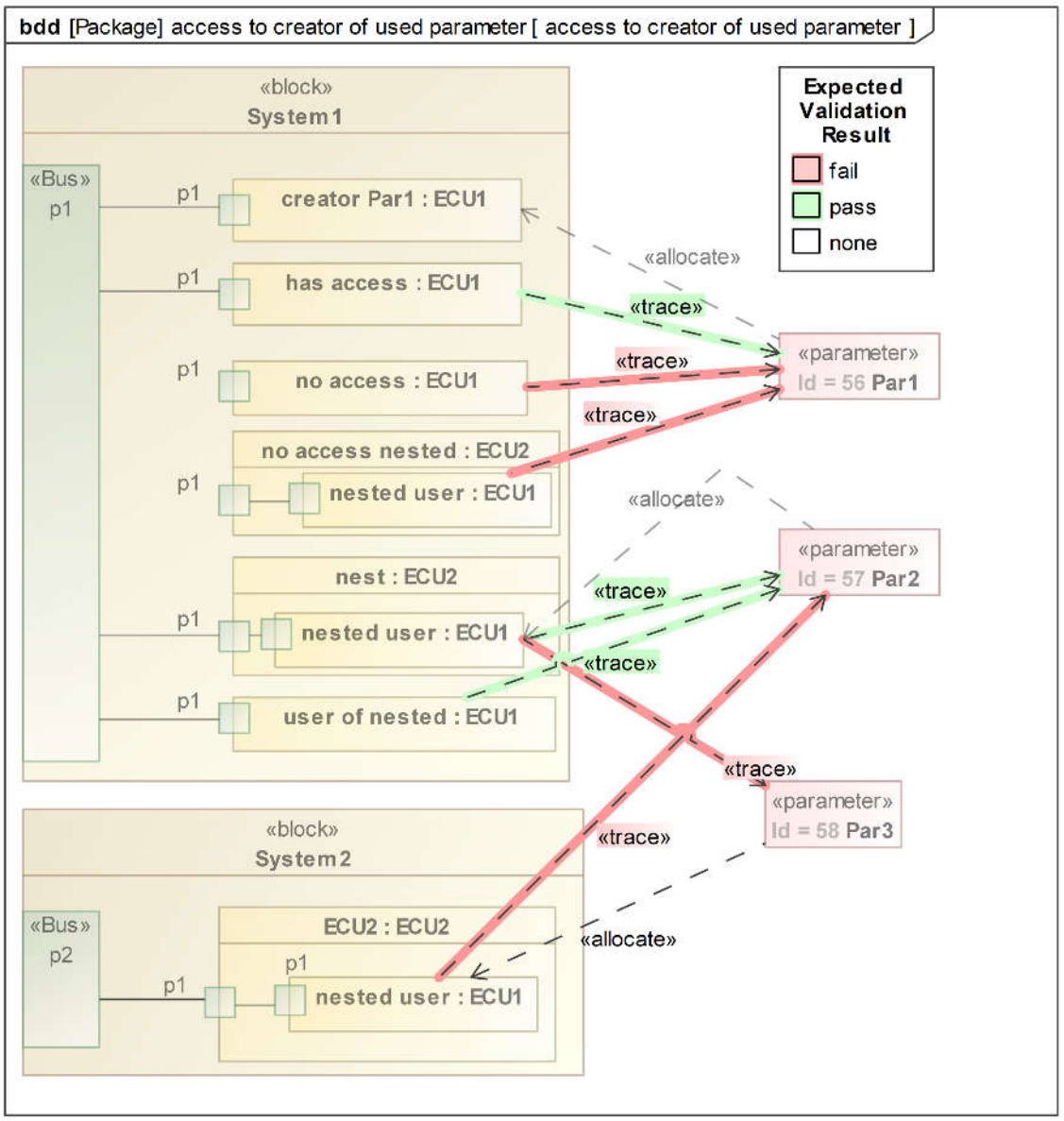

Abbildung 4: Validierung der Verfügbarkeit benötigter Parameter

Weitere Prüfungen betreffen die Korrektheit der Modellierung. Wurden die Stereotypen wie vorgesehen eingesetzt? Wurden die Regeln für Bus-Konnektoren eingehalten? Dies zu prüfen ist sehr empfehlenswert, denn Modellierungskonventionen werden schnell mal vergessen und führen dann zu wenig aussagekräftigen Modellen.
All diese Prüfungen haben wir mit der Object Constraint Language [4] definiert, so dass sie unabhängig vom Tool funktionieren sollten (zum Beispiel wird bei einem Parameter durch `self.owner.oclIsTypeOf(_'Data Bus Profile'::Message)` geprüft ob er Teil einer Message ist).

Export der Modelldaten

Das Gesamtmodell des Systems, in dem auch die Informationen für die Feldbusse integriert sind, ist für die Domänenexperten, die auch nicht immer mit SysML vertraut sind, leider nicht immer direkt verständlich. Hierzu müssen entsprechende Views des Modells erstellt werden. Dies kann einerseits durch spezielle Diagramme im Modell erreicht werden, meist bieten sich aber Exporte in Formate an, die die Domänenexperten

gewöhnt sind. Darüber hinaus können auch Exporte erstellt werden, die direkt in die Software für die Geräte am Feldbus eingebunden werden können, wodurch Übertragungsfehler bei der Programmierung wie Zahlendreher vermieden werden können. Im Rahmen des Projekts ist bisher ein Export in das Vector CAN Database Format [5][6] erfolgreich umgesetzt wurden.

Zusammenfassung und Ausblick

Wir haben einen neuen Ansatz gezeigt, wie Feldbusse modelliert werden können. Er berücksichtigt die Tatsache, dass der Feldbus kein eigenständiges Systemelement ist, sondern nur aus der Zusammenschaltung der Teilnehmer besteht. Durch Ausnutzung der kontextspezifischen Connectoren der SysML konnten beliebig verschachtelte Strukturen modelliert werden. Dass der Ansatz tragfähig ist, wurde durch die Erstellung einer umfassenden Validierungssuite, die im Projekt regelmäßig genutzt wird, und durch den Export in das CAN-Database-Format bewiesen.

Abbildungsverzeichnis

Literatur- und Quellenverzeichnis

[1] Y. Guo, A. C. Rao und R. P. Jones: Architectural and Functional Modelling of an Automotive Driver Information System Using SysML. In: 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, pages 552-557. IEEE, 2008.

[2] M. Mittal: Use of SysML in a ‘Control Area Network’ Architecture Design. In: APCOSEC 2016 – 10th Asia Oceania Systems Engineering Conference. INCOSE, 2016.

[3] Object Management Group: Systems Modeling Language SysML 1.6, https://www.omg.org/spec/SysML/1.6/PDF, 2019

[4] Object Management Group: Object Constraint Language OCL 2.4, https://www.omg.org/spec/OCL/2.4/PDF, 2014

[5] Bryan Hennessy: An Introduction to J1939 and DBC files, https://www.kvaser.com/developer-blog/an-introduction-j1939-and-dbc-files,2019

[6] Socialledge: DBC Format, http://socialledge.com/sjsu/index.php/DBC_Format, 2017

Autoren

Axel Scheithauer, ursprünglich Physiker, begann in den 1990ern Softwaresysteme objektorientiert zu entwickeln. Später verlagerten sich seine Tätigkeiten mehr in die Anforderungsanalyse und zur Projektleitung. Seit 12 Jahren gibt er seine Erfahrungen als Trainer und Berater für Model Based Software und Systems Engineering weiter. Als Vertreter der oose Innovative Informatik eG ist er Mitglied in der UML-Arbeitsgruppe der Objekt Management Group und des Submission Teams für die neue SysML 2. Zurzeit überarbeitet er mal wieder den Klassiker „Analyse und Design mit der UML“.

Kontakt

Internet: www.oose.de
Email: axel.scheithauer@oose.de

Malte Rahm beschäftigt sich seit 8 Jahren mit der Entwicklung experimenteller Luft- und Raumfahrtsysteme. Der Einstieg ins Berufsleben am DLR Institut für Flugsystemtechnik in Braunschweig erfolgte parallel zum Studium an der DHBW Ravensburg und der TU Braunschweig. Zurzeit entwickelt er die Systemarchitektur für die hochfliegende Solarplattform des DLR.

Kontakt

Internet: www.dlr.de/ft
Email: malte.rahm@dlr.de