



```

/**[0..2]*/ 10, 15, 20
/* 10., 15., 20. */

/**[0..2]*/
/* 1.. 6.. */

GLOBAL Int16 YAx

/**[0..2]*/ -6
/* -2.. 0.. 2 */
/* LSB: 2^5 0FF: 00000000 */
GLOBAL Int16 YAx

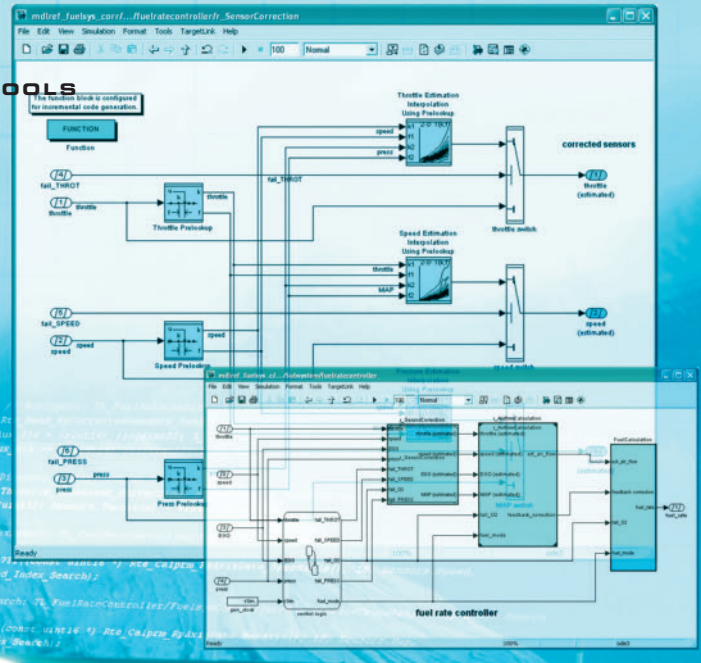
/**[0..2]*/ -6
/* -2.. 0.. 2 */
/* LSB: 2^5 0FF: 00000000 */
GLOBAL Int16 YAx

GLOBAL: global variables [RAM | Watch | 16

GLOBAL Int16 Output /* LSB:

```

The screenshot shows a software development environment. On the left, the 'Data Dictionary Navigator' displays a tree structure of modules and their properties. On the right, the 'Property Value List' shows details for a selected property. At the bottom, the 'Message Browser' displays a list of messages with columns for severity, time, title, origin, and message content.



UNTERSTÜTZUNG EINER MODULAREN VORGEHENSWEISE BEI DER MODELLBASIERTEN SOFTWARE-ENTWICKLUNG



Von einzelnen Modulen zur Applikationssoftware

Dieser Beitrag soll aufzeigen, welche Möglichkeiten moderne Entwicklungswerkzeuge heute bereithalten, um eine modulare, komponentenbasierte Entwicklung zu unterstützen. Eine auf Simulink/TargetLink-basierende Werkzeugkette wird hierbei als Beispiel genommen, da diese im Automobilbereich weit verbreitet ist.

Modellbasierte Entwicklung und automatische Seriene-code-Generierung sind in der Automobilindustrie mittlerweile weit verbreitet und erfreuen sich steigender Beliebtheit. Da auch die Anwendung dieser Techniken in großen Teams zur Entwicklung immer umfangreicherer Funktionalitäten weiter voranschreitet, rücken Anforderungen zur modularen Entwicklung immer stärker in den Fokus, um die erzielten Effizienz- und Qualitätsgewinne auch im großen Stil nutzen zu können. Insgesamt lässt sich das Vorgehen im weitesten Sinne auf einzelne Entwicklungsschritte und Rollen herunterbrechen, wie sie in Bild 1 skizziert sind. Ein Administrator oder Software-Architekt definiert eine Menge von Modulen bzw. Komponenten mit ihren Schnittstellen, um die Gesamtfunktionalität in kleinere, modulare Einheiten aufzuteilen, die später

auch in einem anderen Kontext wiederverwendet werden können. Diese Module müssen anschließend von einer Zahl von Entwicklern modellbasiert entworfen, durch automatische Code-Generierung implementiert und in Form eines Modultests verifiziert werden. Auch die Software-Integration kann anteilig modellbasiert erfolgen, etwa in Form eines übergeordneten Integrationsmodells, welches mit ähnlichen Methoden wie die einzelnen Module entworfen, implementiert und anschließend verifiziert wird. Insgesamt sind in einem Entwicklungsprozess – entsprechend Bild 1 – Aspekte der konsistenten Datenhaltung, der Trennung von Datenständen und Algorithmen, der modellübergreifenden und inkrementellen Code-Generierung und des effizienten Tests von besonderer Bedeutung, wie nachfolgend aufgezeigt werden soll.

Konsistentes Datenmanagement

Für die effiziente modulare Entwicklung in großen Teams sind Datenkonsistenz und Unterstützung des Single-Source-Prinzips von großer Bedeutung. Daten von projektglobaler Relevanz wie etwa Datentypen, Skalierungsformeln, aber auch Interfaces, Schnittstellenvariablen und Applikationsparameter müssen mehreren Entwicklern zugänglich gemacht und die Konsistenz der Daten sichergestellt werden (siehe Bild 1). Gleichzeitig soll jeder Modul-Entwickler naturgemäß nur mit den Größen befasst sein, die Relevanz für das von ihm zu entwickelnde Modul haben. In Simulink/TargetLink-Werkzeugketten werden Modelle grundsätzlich in Verbindung mit dem dSPACE Data Dictionary eingesetzt, welches alle für den TargetLink-Anwender relevanten Daten aufnimmt und auf unterschiedlichste Art und Weise befüllt werden kann. Stellt das in Bild 1 skizzierte Globale Data Dictionary etwa eine abteilungsweite Datenbank dar, so kann der XML-Import oder das API des dSPACE Data Dictionary genutzt werden, um

ware-Integration und dem Software-Integrationstest zu vermeiden. Das dSPACE Data Dictionary bietet hierzu die Möglichkeit, Code- und A2L-Files unabhängig von einzelnen Modellen separat zu erzeugen und somit dem projektglobalen Charakter der Daten Rechnung zu tragen. In **Bild 2** ist exemplarisch dargestellt, wie Applikationsparameter inklusive ihrer Datentypen und Skalierungsformeln im Data Dictionary spezifiziert und dort individuellen Modulen zugeordnet sind. Diese Module werden dann vom Integrator, siehe Bild 1, direkt aus dem Data Dictionary generiert, zusammen mit den zur Kalibrierung erforderlichen A2L-Files. Hierdurch gelingt es zum Beispiel, die Gesamtheit aller Parameter eines Steuergeräteprojektes in ein einziges Code- und A2L-File zu partitionieren und so das Handling zu vereinfachen. Dementsprechend eignet sich das dSPACE Data Dictionary auch dazu, nicht nur die Applikationsparameter der modellbasierten Anteile aufzunehmen, sondern die Legacy-Parameter gleich mit zu verwalten und ebenfalls aus dem Data Dictionary zu generieren.

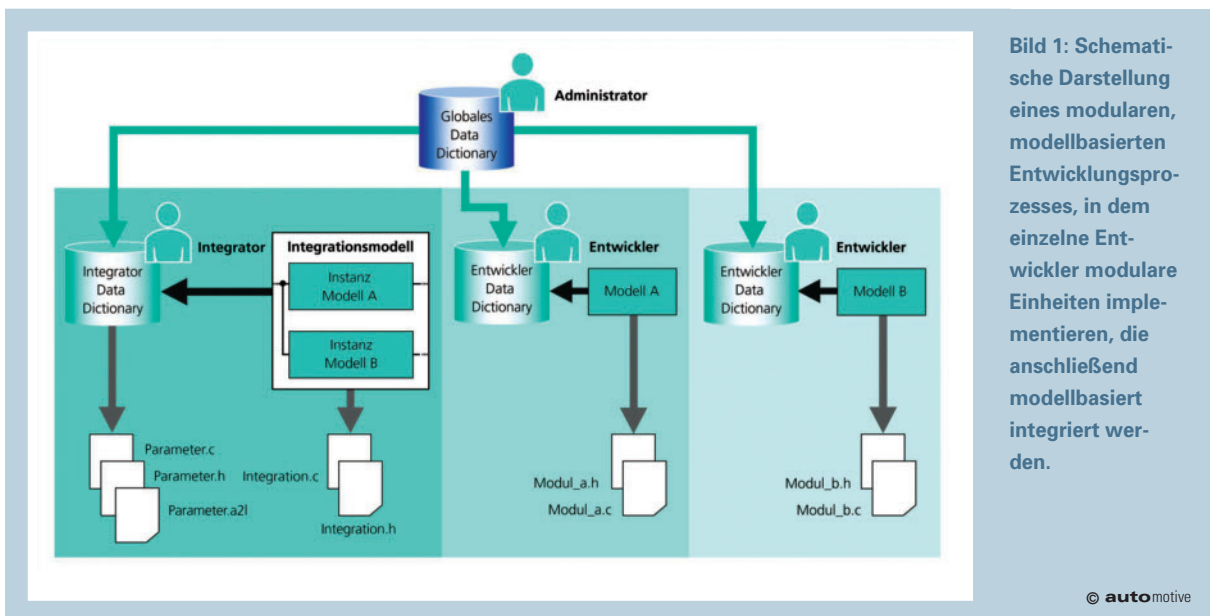


Bild 1: Schematische Darstellung eines modularen, modellbasierten Entwicklungsprozesses, in dem einzelne Entwickler modulare Einheiten implementieren, die anschließend modellbasiert integriert werden.

© automotive

alle relevanten Daten auf konsistente Art und Weise zu importieren. Wird das dSPACE Data Dictionary selbst auch als globales Data Dictionary eingesetzt, so steht zusätzlich ein leistungsfähiger Include-Mechanismus zur Verfügung, um Daten auf Entwickler-/Integrator-Data-Dictionaries zu partitionieren. Das dSPACE Data Dictionary unterstützt zudem einen Diff&Merge-Mechanismus, um Änderungen an Datenständen transparent und abgesichert zu übernehmen, sowie die Vergabe von Schreib- und Leserechten, was zur Absicherung gegen unerwünschte Änderungen genutzt werden kann.

Data-Dictionary-basierte Code-Generierung

Für Objekte von projektglobaler Relevanz wie beispielsweise Applikationsparameter und Schnittstellenvariablen empfiehlt sich nicht nur eine gesonderte Administration, sondern auch eine übergeordnete, von konkreten Modellen losgelöste Generierung, um Probleme bei der Soft-

Design, Implementierung und Test von Modellen und Modulen

Die modellbasierte Entwicklung von einzelnen Modulen, wie sie in Bild 1 als Tätigkeit der individuellen Entwickler skizziert ist, ist mittlerweile vielfach etabliert. Dennoch sind auch hierbei einige Aspekte im Hinblick auf eine einfache Wiederverwendung der Module und die Integration in einen modularen Entwicklungsprozess zu beachten. So muss das Design naturgemäß auf den vorgegebenen Spezifikationen basieren, welche in das Entwickler-Data-Dictionary importiert wurden. Hierdurch ergibt sich insbesondere eine saubere Trennung der eigentlichen Algorithmik des TargetLink-Modells einerseits und der zugehörigen Bedatung im Entwickler-Data-Dictionary andererseits, die vom Modell aus referenziert wird. Zur Vereinfachung der späteren Software-Integration und der Wiederverwendung der Funktionalitäten empfiehlt sich zudem, Modellanteile entweder in Simulink-Libraries zu organisieren oder so zu gestalten, dass sie später aus übergeordneten Modellen

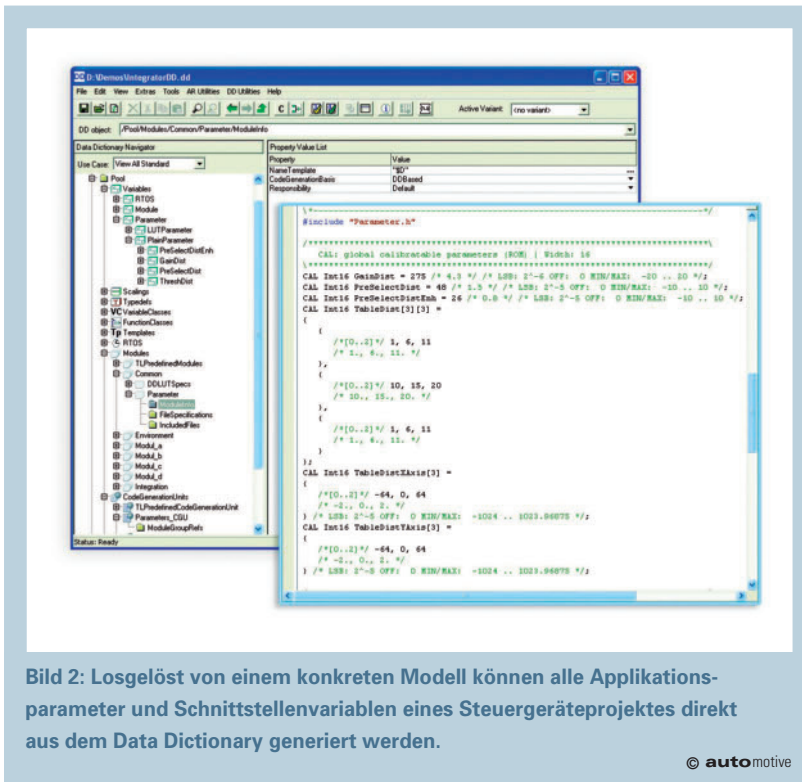


Bild 2: Losgelöst von einem konkreten Modell können alle Applikationsparameter und Schnittstellenvariablen eines Steuergeräteprojektes direkt aus dem Data Dictionary generiert werden.

© automotive

einfach referenziert werden können, was von TargetLink jeweils in Kombination mit inkrementeller Code-Generierung unterstützt wird. Des Weiteren ermöglicht TargetLink die Durchführung von Modultests mit unterschiedlichen Sätzen von Applikationsparametern ohne erneute Code-Generierung und Kompilierung. Hierdurch resultiert beim Test nicht nur eine substantielle Zeitersparnis, sondern auch ein erhöhtes Vertrauen in die Qualität des Moduls, weil alle Tests mit demselben Code durchgeführt wurden.

Modellbasierte Software-Integration

Obwohl in der Praxis derzeit noch recht häufig eine fehlerträchtige, rein manuelle Software-Integration angewandt wird, lässt sich dieser Schritt in einem modellbasierten Entwicklungsprozess in vielerlei Hinsicht automatisieren. Hierzu wird ein Integrationsmodell auf Basis der bereits entwickelten Teilfunktionalitäten erstellt, wobei zur Beachtung des Single-Source-Prinzips entweder Simulink-Modell-Bibliotheken oder sogenannte referenzierte Modelle für die einzelnen Teilfunktionalitäten eingesetzt werden. Die Software-Integration erfolgt dann anschließend durch inkrementelle Code-Generierung in Form von „Glue Code“, der die existierenden Funktionen für die Teilmodelle lediglich aufruft bzw. verbindet. Dadurch wird sichergestellt, dass der während der Modultestphase bereits ausführlich getestete Code wiederverwendet wird, was entweder auf Source-Code- oder Object-Code-Ebene erfolgen kann. Durch die übergeordnete Generierung von Schnittstellengrößen, Parametern und sonstigen gemeinsam genutzten Größen aus dem Data Dictionary lassen sich zudem Probleme beim Build-Prozess vermeiden, da die Allokierung und Zuordnung dieser Größen durch das Integrator-Data-Dictionary klar definiert ist.

Komponentenbasierte Entwicklung nach AUTOSAR

Die angesprochenen Prinzipien der modularen Entwicklung werden mit einem Standard wie AUTOSAR naturgemäß vollständig umgesetzt. In diesem Fall implementieren die einzelnen Entwickler AUTOSAR-Software-Komponenten, das heißt, wiederverwendbare, unabhängig voneinander entwickelbare Einheiten mit wohldefinierten Schnittstellen und Vorgaben zu deren Verschaltung. In der letzten Zeit haben sich die Techniken des modellbasierten Entwurfs zur Entwicklung von AUTOSAR-Software in der Praxis bereits sehr gut bewährt. In einem AUTOSAR-Workflow erstellt jeder einzelne Entwickler aus den modellbasierten Designs Software-Komponenten, die separat entworfen, durch automatische Code-Generierung implementiert und einem Kom-

ponenten-Test unterzogen werden. Das in Bild 1 dargestellte Globale Data Dictionary stellt in diesem Fall typischerweise ein AUTOSAR-Architektur-Werkzeug wie dSPACE SystemDesk dar, in dem initial eine Software-Architektur, bestehend aus einer Vielzahl von Komponenten mit wohldefinierten Schnittstellen erstellt wird. Das Architekturwerkzeug tauscht mit TargetLink Daten, basierend auf dem von AUTOSAR standardisierten XML-Format aus, wodurch die Kompatibilität der Schnittstellen der einzelnen Komponenten sichergestellt wird. Die Software-Integration erfolgt dann weitestgehend vollautomatisch durch den RTE-Generator, nachdem die erforderlichen Konfigurationen für die RTE wie beispielsweise die Festlegung der Tasks und die Abbildung von Software-Komponenten auf einzelne Steuergeräte erfolgt sind.

Unabhängig davon, ob konventionelle oder AUTOSAR-konforme Projekte bearbeitet werden sollen, lassen sich modellbasiertes Design und automatische Code-Generierung nicht nur auf Ebene einzelner Module und kleiner Funktionalitäten, sondern auch in großem Stil und in großen Entwicklungsteams nutzen. (oe)



Dr. Ulrich Eiseemann ist bei dSPACE als Produkt-Manager für den Seriencode Generator TargetLink beschäftigt und befasst sich mit Fragestellungen des modellbasierten Entwurfs, der automatischen Code-Generierung und AUTOSAR.