

Entwicklung eines Software-Leitstands zur Einhaltung von Modellierungsrichtlinien

Elodie Legros, Tobias Rötschke, Andy Schürr
Fachgebiet Echtzeitsysteme
Technische Universität Darmstadt

Abstract

Angesichts zunehmend modellbasierter Entwicklung komplexer Software-Systeme werden heutzutage passende Modellierungsrichtlinien und dazugehörige Werkzeuge zur Einhaltung derselben unerlässlich. Zur Entwicklung eines effizienten Ansatzes sind eine systematische Klassifizierung dieser Regeln und die Definition eines passenden Prozesses erforderlich. Dieses Papier stellt einen Klassifikationsvorschlag vor, welcher anhand von konkreten Modellierungsrichtlinien erläutert wird, und zeigt weitere Fragestellungen zur Entwicklung eines Software-Leitstands für die Einhaltung von Modellierungsrichtlinien auf.

1. Einleitung und verwandte Ansätze

Zur Softwareentwicklung werden immer häufiger modellbasierte Methoden verwendet. Um die Wirksamkeit und Effizienz der modellbasierten Entwicklung zu erhöhen, sind Modellierungsrichtlinien und dazugehörige Werkzeuge zur Erstellung von verständlichen und eindeutigen Modellen unerlässlich geworden.

In Analogie zu Prozessleitständen gibt es bereits verschiedene Arten von Software-Leitständen, die das Sammeln von verschiedenen Informationsarten über ein Software-System und die Darstellung derselben in einer zentralisierten Sicht ermöglichen. Diese Software-Leitstände stehen zur Überwachung von Quelltext [1], und Software-Architekturen [2] zur Verfügung. Obwohl Software-Leitstände für weit verbreitete Modellierungssprachen wie UML oder Matlab/Simulink/Stateflow von der Industrie dringend gefordert werden [3], sind vollständige Ansätze zur Richtlinieneinhaltung noch weitgehend unbekannt.

Ziel des hier vorgestellten Forschungsvorhabens ist es, einen generativen Ansatz zur Erzeugung von Leitständen für Modellierungsrichtlinien zu entwerfen und umzusetzen. Als erster Schritt wird dazu in diesem Papier ein Schema zur systematischen Klassifikation von Modellierungsrichtlinien erstellt, welches an Hand von konkreten Richtlinien aus einem bestehenden Industrieprojekt erläutert wird. Die Universitäten Darmstadt, Paderborn, Kassel und Siegen arbeiten mit der Firma DaimlerChrysler an einem Projekt namens „MATE“ (MATLAB Simulink/Stateflow Analysis and Transformation Environment) [4], dessen Ziel u.a. ist,

die bislang manuelle Prüfung von Modellierungsrichtlinien und Korrektur von -fehlern zu automatisieren. Dieses Projekt wird als Ausgangspunkt für den hier betrachteten Modellierungsleitstand genommen.

2. Klassifikation von Richtlinien

In diesem Abschnitt werden verschiedene Aspekte von Modellierungsrichtlinien zu einem Klassifikationschema zusammengefasst werden.

Richtlinien unterscheiden sich in ihrer *Wichtigkeit* und *Dringlichkeit*. Die Wichtigkeit gibt an, wie schwerwiegend eine Verletzung ist. Mit unterschiedlichen Dringlichkeiten kann der zeitliche Ablauf von Korrekturmaßnahmen geplant werden.

Um die Einhaltung von Richtlinien zu gewährleisten, muss der Leitstand nicht nur Regeln überprüfen, sondern auch auf eine Regelverletzung *reagieren*. Eine Verletzung kann von vorne herein *verhindert* oder *repariert* werden. Falls Richtlinien sich widersprechen, benötigt man *Prioritäten* zu einer eindeutigen Wahl von Aktionen. Ist eine Reparatur unmöglich, sollen Verstöße zumindest *angezeigt* werden.

Ein weiteres Klassifizierungskriterium ist der *Zeitpunkt* der Überprüfung. Diese kann *inkrementell* während der Modellbearbeitung oder *auf Anfrage* des Entwicklers erfolgen. Wird Code generiert, bietet sich eine Überprüfung *vor der Übersetzung* an. Werden Modelle von mehreren Entwicklern bearbeitet, empfiehlt sich eine Überprüfung *vor dem Einchecken* in das Repository. Ist die Überprüfung der Richtlinien sehr aufwändig, kann regelmäßig eine *globale Überprüfung* in Stapelverarbeitung nötig sein.

3. Software-Leitstand und Versionierung

Wenn Verstöße nicht automatisch behoben werden können, wird zudem eine zusammenfassende *Darstellung* benötigt. Dabei kann einerseits die *absolute Anzahl* der Verstöße berücksichtigt werden. Im Sinne einer *Regressionsanalyse* können andererseits aber auch nur neu eingeführte Verstöße zeitnah identifiziert werden. Das Konzept der *Regressionsanalyse* umfasst die Begriffe *Regressionstest* und *Analyse*. In Analogie mit Regressionstests werden alle Richtlinien überprüft und die Ergebnisse werden mit derjenigen der vorigen Version des Modells verglichen. Dabei werden

Ergebnisse aufeinander folgender Analysen verglichen, um die Evolution des Modells über Versionen hinweg beobachten zu können. Außerdem bieten sich *Trendanalysen* an, welche die Anzahl von Verstößen über einen längeren Zeitraum darstellen.

4. Anwendung auf ein Beispiel

In [3] wurden in natürlicher Sprache beschriebene Modellierungsrichtlinien in der Form eines Katalogs definiert. Exemplarisch betrachten wir drei ausgewählte der insgesamt 58 Regeln und klassifizieren sie (Abb 1):

- *jm_0001 (Prohibited Simulink standard blocks inside controllers)*: Bestimmte zur Verfügung stehende Simulink-Blöcke dürfen nicht im Inneren eines Controllers vorkommen.
- *db_0032 (Simulink signal appearance)*: Signalinien dürfen sich, wenn möglich, nicht kreuzen, werden rechtwinklig, nicht übereinander und nicht durch einen Block gezeichnet.
- *db_0081 (Unconnected signals)*: Ein- bzw. Ausgangsblöcke und Signallinie dürfen nicht unverbunden bleiben.

Die Wichtigkeit der Regeln wird bereits in [3] definiert. So sind *jm_0001* und *db_0081* „verbindlich“, und *db_0032* „dringend empfohlen“. Wann eine Richtlinie überprüft und wie sie repariert werden soll, lässt sich aus ihrer Beschreibung ableiten.

jm_0001 verbietet die Verwendung von bestimmten Komponenten und sollte daher überprüft werden, wenn der Entwickler einen neuen Block auswählt. Die gewünschte Reaktion ist ein Verbot der Aktion, wenn der gewählte Block in einem Controller platziert wird.

Die Richtlinie *db_0032* besteht bei genauerer Betrachtung aus vier Regeln: Dass Signallinien rechtwinklig und nicht übereinander gezogen werden, kann überprüft und automatisch repariert werden, während der Entwickler Signallinien zieht. Dass die Linien sich überkreuzen bzw. über Blöcke gezogen werden, passiert so oft während der Bearbeitung eines Modells, dass es viel sinnvoller ist, diese Regeln auf Anfrage zu überprüfen. Wegen den möglicherweise unerwünschten Änderungen des Layouts, ist es besser, die Reparatur vorzuschlagen statt automatisch auszuführen.

Regel *db_0081* sollte ebenso lieber auf Anfrage des Entwicklers getestet werden. Weil nur der Entwickler wissen kann, wie die Signallinien verbunden werden sollen, kann die Verletzung der Richtlinie nicht repariert, sondern nur gemeldet werden. Im Gegensatz zu *db_0032* ist *db_0081* nicht für das Layout, sondern für das Modell selbst wichtig. Daher muss diese Richtlinie auch vor der Codegenerierung sowie vor dem Einchecken überprüft werden, und diese Aktionen gegebenenfalls verboten werden.

Richtlinie	jm_0001	db_0032	db_0081
Wichtigkeit	verbindlich	dringend empfohlen	verbindlich
Bei Modellbearbeitung	X (ar)	X (ar)	
Auf Anfrage		X (vr)	X (m)
Vor Codegenerierung			X (m) (v)
Vor dem Einchecken			X (m) (v)

Aktion

- (m) = Meldung

- (v) = Verbot

- (ar) = Automatische Reparatur

-(vr) = Vorgeschlagene Reparatur

Abb.1: Richtlinienklassifizierung für MAAB-Richtlinien

5. Zusammenfassung und Ausblick

In diesem Positionspapier wird die Idee für einen Software-Leitstand zur Überprüfung von Modellierungsrichtlinien formuliert. Als erster Schritt werden Möglichkeiten zur Klassifikation von Modellierungsrichtlinien diskutiert. Diese werden anhand von Matlab/Simulink/Stateflow-Beispielen aus einem realen Projekt erläutert. Die Ideen lassen sich aber problemlos auf andere Modellierungssprachen wie z.B. UML übertragen.

Als nächster Schritt sollen geeignete Spezifikations-sprachen, wie OCL, Graphtransformationen oder reguläre Ausdrücke zur formalen Richtliniendefinition ausgewählt werden. Es ist absehbar, dass die jeweils optimal geeignete Spezifikations-sprache sich als weiteres Klassifikationsmerkmal von Richtlinien herausstellen wird. Langfristig ist die Implementierung eines Generators für Modellierungsleitstände mit dem an unserem Fachgebiet entwickelten Meta-CASE-Werkzeug MOFLON [5] und die Evaluierung des vorgestellten Ansatzes in Industrieprojekten geplant.

Literatur

- [1] Walter Bischofberger. *Werkzeugunterstütztes Architektur- und Qualitätsmonitoring – von in-house bis offshore*. In Software Engineering Today (SET 2005), Mai 2005.
- [2] Tobias Röttschke and René Krikhaar. *Architecture Analysis Tools to Support Evolution of Large Industrial Systems*. In IEEE International Conference on Software Maintenance (ICSM 2002), Seiten 182–193, Oktober 2002.
- [3] Mathworks Automotive Advisory Board, *Controller Style Guidelines For Production Intent Using Matlab®, Simulink® And Stateflow®*, www.mathworks.com/industries/auto/maab.html
- [4] Stürmer, I. et al.: *Das MATE Projekt - Visuelle Spezifikation von MATLAB/Simulink/Stateflow Spezifikation und Transformationen*. Dagstuhl Seminar "Modellbasierte Entwicklung eingebetteter Systeme", Januar 2006.
- [5] Technische Universität Darmstadt, *MOFLON*, <http://www.moflon.org> . 2007