

Entwicklung eines Software-Leitstands zur Einhaltung von Modellierungsrichtlinien

Elodie Legros, Tobias Rötschke, Andy Schürr
Fachgebiet Echtzeitsysteme
Technische Universität Darmstadt
D-64283 Darmstadt

Abstract:

Angesichts zunehmend modellbasierter Entwicklung komplexer Software-Systeme, werden heutzutage passende Modellierungsrichtlinien und dazugehörige Werkzeuge unerlässlich, welche die Einhaltung derselben kontinuierlich überwachen. Derartige Regeln können zahlreiche verschiedene Formen annehmen, so dass die Entwicklung eines effizienten Ansatzes eine systematische Klassifizierung dieser Regeln und die Definition eines passenden Prozesses erforderlich macht. Dieses Papier stellt einen Klassifikationsvorschlag vor, welcher anhand von konkreten Modellierungsrichtlinien für MATLAB / Simulink + Stateflow erläutert wird, und zeigt weitere Fragestellungen als nächste Schritte zur Entwicklung eines Software-Leitstands zur Einhaltung von Modellierungsrichtlinien auf.

1. Einleitung

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

In Analogie zu Prozessleitständen, wie z.B. in Kraftwerken, gibt es bereits verschiedene Arten von Software-Leitständen. Ein Leitstand kann allgemein als Mittel zur Kopplung und Visualisierung von unterschiedlichen Informationsquellen definiert werden. Software-Leitstände ermöglichen das Sammeln von verschiedenen Informationsarten über ein Software-System und die Darstellung derselben in einer zentralisierten Sicht, so dass der tatsächliche Zustand und aktuelle Trends des Systems überwacht und beurteilt werden können.

Nach unseren Erkenntnissen sind bislang nur isolierte Ansätze zu Metriken auf Modellen, Refactoring von Modellen oder Überwachung von Konsistenzbedingungen auf Modellen vorhanden. Obwohl Software-Leitstände für weit verbreitete Modellierungssprachen wie UML sinnvoll wären oder sogar im Fall von Matlab / Simulink + Stateflow schon von der Industrie gewünscht werden [MAAB01], sind vollständige Ansätze zur Richtlinieneinhaltung noch weitgehend unbekannt.

Das Ziel des hier vorgestellten Forschungsvorhabens soll es sein, 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.

2. Verwandte Ansätze

Auch wenn die Idee von Leitständen für die Softwareentwicklung noch relativ neu ist, ist sie schon heute in anderen Bereichen verbreitet, wie im Management von Unternehmen oder in der Produktenentwicklung. Ein Management-Cockpit [RMMB00] bezeichnet ein Informationszentrum zur Optimierung der Entscheidungsfindung im Team, indem es die Visualisierung von verschiedenen entscheidungsrelevanten Daten ermöglicht. Mit einem Produktdaten-Cockpit [PLM07] können auf dieselbe Weise Produktdaten visualisiert werden.

Existierende Software-Leitstände stehen zur Überwachung von Quelltext [Bisc05], Software-Architektur [RoKr02], Programmierrichtlinien [TIOBE07], Projekt-Management [MuJe04], und Software-Betrieb [GiRH06] zur Verfügung. Das ConQAT-Projekt strebt nach einem systematischen Ansatz, um verschiedene Qualitätsmonitore in einem vereinheitlichten Leitstand zu integrieren [DeSe06].

Die Universitäten Darmstadt, Paderborn, Kassel und Siegen arbeiten zurzeit mit der Firma DaimlerChrysler an einem Projekt namens „MATE“ (MATLAB Simulink/Stateflow Analysis and Transformation Environment) [MATE06]. Ziel dieses Projekts ist es, die bislang manuelle Prüfung von Modellierungsrichtlinien zu automatisieren, sowie Modelltransformationen, die Visualisierung verschiedener Entwicklungsstände und die Integration der Modelle mit anderen Entwicklungsartefakten zu untersuchen. Dieses Projekt wird als Ausgangspunkt für den hier betrachteten Modellierungs-Leitstand genommen.

3. Klassifikation von Richtlinien

In diesem Abschnitt werden verschiedene Aspekte von Modellierungsrichtlinien betrachtet. Diese sollen zu einem Klassifikationsschema zusammengefasst werden.

Richtlinien unterscheiden sich in ihrer *Wichtigkeit* und *Dringlichkeit*. Die Wichtigkeit gibt an, wie schwerwiegend die Auswirkungen von Verletzungen sind. So sind Verstöße gegen Richtlinien, welche lediglich die Lesbarkeit eines Modells verbessern, offensichtlich weniger wichtig als solche, die bei einer nachgeschalteten Modelltransformation zu Fehlern führen. Durch Vergabe unterschiedlicher Dringlichkeiten kann der zeitliche Ablauf von Korrekturmaßnahmen detailliert geplant und überwacht werden.

Um die Einhaltung von Richtlinien zu gewährleisten, reicht eine reine Überprüfung der Modellierungsregeln häufig nicht, der Leitstand muss zusätzlich auf eine Regelverletzung *reagieren*. Am besten wird eine Verletzung von vorne herein durch geeignete Maßnahmen *verhindert*. Alternativ sollte eine (idealerweise automatische) *Reparatur* des Modells möglich sein. Da sich einige Richtlinien widersprechen können, benötigt man gegebenenfalls *Prioritäten*, um eine eindeutige Wahl von Reparaturaktionen zu erzielen. Auch wenn eine Reparatur nicht möglich ist, sollen Verstöße dem Entwickler zumindest *angezeigt* werden.

Ein nächstes Klassifizierungskriterium ist der *Zeitpunkt* der Überprüfung: Im Idealfall erfolgt die Überprüfung *inkrementell* während der Bearbeitung des Modells oder zumindest *auf Anfrage* des Entwicklers. Sofern aus Modellen Code generiert wird, bietet sich eine Überprüfung *während der Übersetzung* an. Wenn Modelle von mehreren Entwicklern bearbeitet werden, empfiehlt sich eine Überprüfung *während des Eincheckens* in ein gemeinsames Repository. Sofern die Überprüfung der Richtlinien sehr aufwändig ist, kann auch regelmäßig eine *globale Überprüfung* in Stapelverarbeitung nötig sein.

Wenn Verstöße nicht automatisch behoben werden können, wird zudem eine zusammenfassende *Darstellung* benötigt. Dabei kann die *absolute Anzahl* der Verstöße berücksichtigt werden, um den aktuellen Zustand festzustellen. Im Sinne einer *Regressionsanalyse* können 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 dieser Überprüfung

werden mit derjenigen der vorigen Version des Modells verglichen. Dabei werden die Ergebnisse aufeinander folgender Regressionsanalyse ausgewertet, um die Evolution des Modells über Versionen hinweg beobachten zu können. Für die längerfristige Fortschrittskontrolle bieten sich *Trendanalysen* an, welche die Anzahl von Verstößen über einen längeren Zeitraum darstellen.

Letztlich stellt sich auch die Frage bezüglich der Integration in den *Modellierungsprozess*: Sollen festgestellte Verstöße den nächsten Prozessschritt (z. B. Einchecken) *blockieren*? Müssen automatisch durchgeführte Reparaturaktionen dennoch manuell *autorisiert* werden? Werden Verstöße gegen Richtlinien als *Defekte* im Sinne der Änderungsverwaltung überwacht und beeinflussen damit die *Freigabe* des Produkts?

4. Anwendung auf ein Beispiel

In [MAAB01] wurden Modellierungsrichtlinien in der Form eines Katalogs definiert, in dem die verschiedenen Richtlinien in natürlicher Sprache beschrieben sind. Es fehlt aber noch ein Werkzeug, welches die Einhaltung dieser Richtlinien ermöglichen würde. Ein erster Schritt zur Entwicklung eines solchen Ansatzes besteht in der Klassifizierung der Richtlinien gemäß den im vorigen Abschnitt vorgestellten Kriterien.

Exemplarisch betrachten wir drei ausgewählte der insgesamt 58 Regeln. Modellierungsrichtlinien werden in [MAAB01] durch Bezeichner der Form „bb_nnnn“ identifiziert:

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

Die Wichtigkeit der Regeln wird bereits in [MAAB01] definiert. So gehören jm_0001 und db_0081 zur Kategorie „verbindlich“, und db_0032 zur Kategorie „dringend empfohlen“. Eine Dringlichkeit wurde den Regeln bisher nicht zugewiesen.

Wann eine Richtlinie überprüft und welche Aktion dementsprechend ausgeführt werden soll, lässt sich aus der Beschreibung der Richtlinien ableiten. In Übereinstimmung mit jm_0001 dürfen bestimmte Komponenten nicht zur Modellierung verwendet werden. Daher ist es sinnvoll, dass diese Richtlinie während der Bearbeitung des Modells überprüft wird, und zwar jedes Mal, wenn der Entwickler einen neuen Block auswählt. Die gewünschte Reaktion ist ein Verbot der Aktion, wenn der gewählte Block in einem Controller platziert wird. Da das Einfügen eines verbotenen Elements auf diese Weise vermieden wird, ist eine weitere Überprüfung, z.B. beim Einchecken, nicht nötig.

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. Die Reparatur kann automatisch stattfinden, weil sie nicht sehr aufwändig ist und den Entwickler nicht stören wird. Das ist nicht der Fall bei den beiden anderen Punkten von db_0032.

Dass die Linien sich nicht überkreuzen bzw. nicht über Blöcke gezogen werden, passiert so oft während der Bearbeitung eines Modells, dass diese Richtlinien nicht ständig prüfen werden sollten. Automatische Korrekturen würden wahrscheinlich zu Modellen mit vielen mehrfach abgewinkelten Linien führen, oder deren Elemente ständig neu anordnen. Es ist viel sinnvoller, diese Regeln auf Anfrage des Entwicklers zu überprüfen. Wegen den

möglicherweise unerwünschten Änderungen des Layouts, ist es zusätzlich besser, die Reparatur vorzuschlagen (beispielsweise mittels einer Vorschau) und nicht automatisch auszuführen. Weil db_0032 schließlich nur das Layout des Modells betrifft und nicht als „verbindlich“ klassifiziert wurde, ist keine Überprüfung vor der Codegenerierung bzw. vor dem Einschecken notwendig.

Regel db_0081 kann ebenso nicht ständig während der Bearbeitung des Modells überprüft werden. Es ist besser, diese Regel auf Anfrage des Entwicklers zu testen. Weil nur der Entwickler wissen kann, wie die Signallinien verbunden werden sollen, kann die Verletzung der Richtlinie nur gemeldet, jedoch keine Reparatur vorgeschlagen bzw. automatisch durchgeführt werden.

Im Gegensatz zu db_0032 ist db_0081 nicht für das Layout, sondern für das Modell selbst wichtig. Wird sie verletzt, wird das Modell nicht verwendbar. Aus diesem Grund muss diese Richtlinie auch vor der Codegenerierung sowie vor dem Einchecken überprüft werden. Wird Richtlinie db_0081 verletzt, dürfen die Codegenerierung bzw. das Einchecken nicht ausgeführt werden. Abbildung 1 fasst die Klassifizierung dieser Richtlinien zusammen.

| Richtlinie | jm_0001 | db_0032 | db_0081 |
|-------------------------------|-------------|--------------------|-------------|
| Priorität | verbindlich | dringend empfohlen | verbindlich |
| Während der Modellbearbeitung | X (v) | X (ar) | |
| Auf Anfrage | | X (vr) | X (m) |
| Vor der Codegenerierung | | | X (m) (v) |
| Vor dem Einchecken | | | X (m) (v) |

Aktion:

- (m) = Meldung
- (vr) = Vorgeschlagene Reparatur
- (ar) = Automatische Reparatur
- (v) = Verbot

Abb. 1 : Richtlinienklassifizierung – Anwendung auf drei 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 Beispielen aus einem realen Projekt erläutert, welches Matlab / Simulink + Stateflow verwendet. Die Ideen lassen sich aber problemlos auf andere Modellierungssprachen wie z.B. UML übertragen.

Als nächster Schritt sollen geeignete Spezifikationsprachen zur formalen Definition von Richtlinien ausgewählt werden, damit eine automatische Überprüfung stattfinden kann. Unterschiedliche Möglichkeiten wie OCL, Graphtransformationen oder reguläre Ausdrücke bieten sich an, und es ist absehbar, dass die jeweils optimal geeignete Spezifikationsprache sich als weiteres Klassifikationsmerkmal von Richtlinien herausstellen wird. Anders herum könnte bei einer einheitlichen Formalisierung der Richtlinien die Klassifikation bezüglich der anderen Kriterien auch automatisch erfolgen.

Langfristig ist die Implementierung eines Generators für Modellierungsleitstände basierend auf dem an unserem Fachgebiet entwickelten Meta-CASE-Werkzeug MOFLON [TUD07] geplant. Es ist unser Ziel, den hier vorgestellten Ansatz auf diese Weise in realen Industrieprojekten zu evaluieren.

Literatur

- [Bisc05] Walter Bischofberger. *Werkzeugunterstütztes Architektur- und Qualitätsmonitoring – von in-house bis offshore*. In Software Engineering Today (SET 2005), Mai 2005.
- [DeSe06] Florian Deißböck and Tilman Seifert. *Kontinuierliche Qualitätsüberwachung mit ConQAT*. In Proc. Workshop “Softwareleitstände” in [HoLi06], Seiten 118–125. Springer, Oktober 2006.
- [GRH06] Simon Giesecke, Matthias Rohr, Wilhelm Hasselbring. *Software-Betriebs-Leitstände für Unternehmensanwendungslandschaften*. In Proc. Workshop “Softwareleitstände” in [HoLi06], Seiten 110–117, Oktober 2006.
- [HoLi06] Christian Hochberger, Rüdiger Liskowsky, Hrsg. *INFORMATIK 2006*. Band P-94 von „Lecture Notes in Informatics“, Springer, Oktober 2006.
- [MAAB01] Mathworks Automotive Advisory Board, *Controller Style Guidelines For Production Intent Using Matlab[®], Simulink[®] And Stateflow[®]*, Version 1.00, April 2001 - <http://www.mathworks.com/industries/auto/maab.html>
- [MATE06] Stürmer, I., Dörr, H., Giese, H., Kelter, U., Schürr, A., Zündorf, A.: *Das MATE Projekt - Visuelle Spezifikation von MATLAB/Simulink/Stateflow Analysen und Transformationen*. Dagstuhl Seminar "Modellbasierte Entwicklung eingebetteter Systeme", Januar 2006.
- [MuHe04], Jürgen Münch and Jens Heidrich. *Software project control centers: Concepts and approaches*. Journal of Systems and Software, 70(1):3–19, 2004.
- [PLM07] PartMaster GmbH. *"PLM.Cockpit – Alles in einer Oberfläche. Frei kombinierbar und erweiterbar"*
<http://www.partmaster.de/uploads/90/PLMCockpitProduktblatt.pdf>,
Februar 2007. Produktblatt.
- [RMMB00] Reiterer, H. ; Mann T.M. ; Mußler. G. ; Bleimann, U. : *Visualisierung von entscheidungsrelevanten Daten für das Management*. In: HMD, Praxis der Wirtschaftsinformatik, Heft 212 04/2000, S.71–83
- [RoKr02] 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.
- [TIOBE07] TIOBE Software BV. *TICS Quality Viewer*.
<http://www.tiobe.com/services/viewer.htm>, Februar 2007. Produkt-Webseite.
- [TUD07] Technische Universität Darmstadt, *MOFLON*, <http://www.moflon.org> . 2007