SiDiff: generische, auf Ähnlichkeiten basierende Berechnung von Modelldifferenzen

Maik Schmidt

Universität Siegen, Praktische Informatik
Hölderlinstr. 3
57068 Siegen
mschmidt@informatik.uni-siegen.de

1 Motivation

In den letzten Jahren hat die von der OMG proklamierte, modellgetriebene Softwareentwicklung *Model Driven Architecture* (MDA) bzw. das *Model Driven Engineering* (MDE) in vielen Bereichen der Informatik an Bedeutung gewonnen. Bei diesem Vorgehen steht das Modell des zu realisierenden Systems im Vordergrund, welches iterativ mittels geeigneter Werkzeuge und Sprachen erstellt und präzisiert wird. Auch im Bereich der Elektrotechnik ist ein ähnliches Vorgehen zur Spezifikation von Regelkreisen und Controllern durch Modelle bereits seit Jahren gängige Praxis.
Der hohen Durchgängigkeit dieses Vorgehens stehen gegenwärtig Einschränkungen im Bereich des Konfigurations- und Versionsmanagements gegenüber, da klassische Versions- und Konfigurationsmanagementwerkzeuge zur Verwaltung von Modelldaten ungeeignet sind und versagen bzw. unbrauchbare Ergebnisse liefern. Vor allem wenn mehrere Entwickler (parallel) an der Modellierung beteiligt sind, wird dies zum Problem. Zur weiteren Steigerung der Entwicklungseffizienz wurde daher im Rahmen des MATE Projektes[1] (MATLAB Simulink/Stateflow Analysis and Transformation Environment) eine Werkzeugfamilie zur automatischen und interaktiven Korrektur von Entwurfsfehlern sowie für den Vergleich unterschiedlicher Diagrammversionen für die MATLAB/Simulink Modellierungsplattform[2] entwickelt. Die Vergleichsfunktion wurde auf der Grundlage des in [1] und [2] beschriebenen SiDiff-Algorithmus entwickelt.

2 Adaption von SiDiff

Der SiDiff-Algorithmus ermöglicht es, Dokumente in Form streng getypter, attributierter Graphen zu vergleichen. Zur Adaption von SiDiff für den Vergleich von Simulink-Diagrammen wurde zunächst eine geeignete, graphartige Darstellungsform für diesen Diagrammtyp bestimmt.

Jedes sichtbare Diagrammelement wurde dazu auf einen Knoten bzw. Knotentyp abgebildet. Mittels zusätzlicher Knoten, die keinen sichtbaren Elementen entsprechen, wurden semantische Aspekte des Diagramms modelliert. Beziehungen zwischen Diagrammelementen wurden durch Kanten, die entsprechend der Beziehung getypt sind, modelliert. Daneben wurden allgemeine Teil-von-Beziehungen durch spezielle Kanten modelliert. Diese überlagern den Graphen mit einer Baumstruktur. Neben den derart gebildeten Graphen benötigt der SiDiff-Algorithmus noch knotentyp-spezifische Vergleichsfunktionen. Diese bestimmen ein Ähnlichkeitsmaß zwischen jeweils zwei Knoten eines Typs und berücksichtigen lokale sowie strukturelle Eigenschaften.
Der SiDiff-Algorithmus vergleicht nun alle Knoten eines Typs paarweise und definiert die Knoten mit maximaler Ähnlichkeit als korrespondierend, sofern die festgestellte Ähnlichkeit eine untere Schranke überschreitet. Dabei werden die Typen i.d.R. derart durchlaufen, dass die überlagerte Baumstruktur bottom-up traversiert wird. Falls Knoten als korrespondierend erkannt werden, so wird diese Information anschließend top-down propagiert. Auf Basis dieser neuen Information können nun Knoten als korrespondierend erkannt werden, die zuvor eine zu geringe Ähnlichkeit aufwiesen oder deren Ähnlichkeit mehrdeutig war. Anschließend wird der bottom-up-Durchlauf fortgesetzt und wiederholt, bis keine neuen Korrespondenzen mehr gefunden werden.
Auf Grundlage der berechneten Korrespondenzen kann nun eine symmetrische Differenz berechnet werden. Diese bezieht sich jedoch zunächst auf die Graphen und enthält daher kaum semantische Differenzinformation. Diese muss in einem anschließenden Schritt diagrammtypspezifisch durch Interpretation gewonnen werden.
Abbildung 1 zeigt das auf den aufbereitenden Differenzinformationen basierende MATE-GUI. Das Ergebnis der Differenzberechnung wird in Form einer hierarchischen Liste von textuellen Differenz-Beschreibungen angezeigt. Durch Selektion einer einzelnen Differenzbeschreibung werden die entsprechenden Dokumentteile in zwei Fenstern geöffnet und die an der Differenz beteiligten Modellelemente farblich markiert.

[1] Kooperation zwischen: DaimlerChrysler AG, Software Engineering Research Group - Universität Paderborn, Institute of Computer Engineering - TU Darmstadt, Software Engineering Research Group - Universität Kassel, Model Engineering Solutions, Software Engineering Group - Universität Siegen
[2] www.mathworks.com

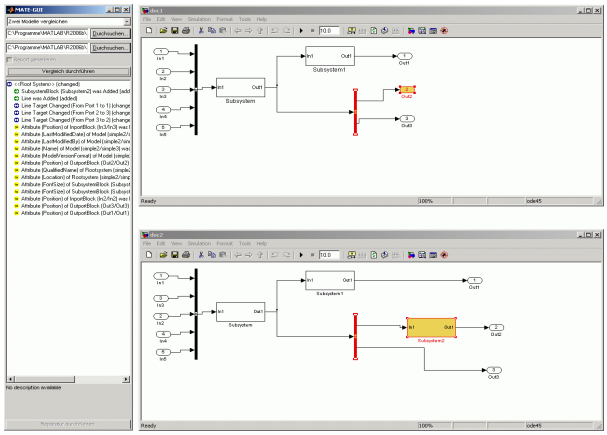

Abbildung 1: MATE -GUI des SiDiff für Simulink Prototyps

3. Problemdomänen

Zur Adaption von SiDiff für den Vergleich von Simulink-Diagrammen waren neben technischen Details auch konzeptuelle Probleme zu lösen. Im Folgenden werden die wesentlichen Problembereiche kurz skizziert.

3.1 Reverse Engineering und Modellierung

Wie in [2] beschrieben, basierte der ursprüngliche SiDiff-Ansatz auf der Eingabe von UML-Diagrammen im XMI-Format. Für Simulink sind mehrere XMI-Austauschformate verfügbar, die jeweils auf einer eigenen Meta-Modellierung basieren. Eine Analyse der physikalischen Repräsentation zeigt, dass die Modelle stark semi-strukturiert gespeichert sind und Teile der Modelle nur mit viel Aufwand aus den vorhandenen Daten inferiert werden können. Aus diesem Grund wurden mehrere vereinfachte Metamodellvarianten evaluiert. Dabei zeigte sich, dass die Konstruktion des Metamodells erheblichen Einfluss auf die Berechnungseffizienz und den Gehalt an semantischer Information in der symmetrischen Differenz hat. Die explizite Modellierung von Semantik führt zu einem starken Anstieg zu vergleichender Knoten und somit zu erhöhter Laufzeit. Der Verzicht auf derartige Modellelemente führt hingegen zu einem Verlust an Information. Es ist daher offen, auf welchem semantischen Niveau eine optimale Modellierung vorgenommen werden sollte.

3.2 Korrespondenzberechnung

Die typweise Korrespondenzbildung auf der Grundlage von Ähnlichkeiten kann zu semantisch unzulässigen Korrespondenzen zwischen Elementen führen (z.B. zwischen Blöcken unterschiedlicher Subsysteme). Um dies sicher zu verhindern darf eine Korrespondenz nur bei sehr hoher Ähnlichkeit gebildet werden, was eine sehr schlechte Qualität der Differenz zur Folge hat, da Korrespondenzen unerkannt bleiben. Dieses Problem wurde mittels kontextabhängiger Ähnlichkeitsmaße adressiert, wobei sich jedoch allgemein die Frage nach Methoden zur Einhaltung von Constraints stellt.

3.3 Interpretation und Weiterverarbeitung von Differenzen

Die auf Basis der Korrespondenzen gebildete Differenz beschreibt zunächst Unterschiede der Modelle auf Graph-Ebene. Auch wenn semantische Information in den Graphen kodiert wurde, muss die symmetrische Differenz interpretiert werden, um eine für den Nutzer aussagekräftige Differenz zu erhalten. Daher ist es zweckmäßig, aussagekräftige modellierungs- und dokumenttypabhängige Editieroperationen zu definieren und elementare Graphtransformations-operationen auf diese abzubilden. Darüber hinaus ist festzulegen, wie diese Differenz anschließend zu visualisieren ist.
Wurde zu wenig Information explizit modelliert, so führt dies u.U. zu nicht eindeutig interpretierbaren Differenzen. Liegt beispielsweise ein 1:n Beziehungstyp vor, so muss entscheiden werden, wie die Veränderung von Einzelreferenzen des Typs zu interpretieren ist. Betrachtet man alle Referenzen eines Typs als eine mengenwertige Referenz, so handelt es sich um eine ggf. komplexe Einzeldifferenz. Interpretiert man jede Referenz individuell als Menge von Einzelreferenzen, so erhält man viele Differenzen und steht u.U. vor unentscheidbaren Korrespondenz-problemen. Im einfachsten Fall verbietet man mehrwertige Referenzen zugunsten einer einfachen Handhabbarkeit, und erlaubt nur 1:1 Beziehungen, was aber zu explizierter Modellierung mittels Knoten und somit zu erhöhter Laufzeit führt.
Die Gewinnung einer eindeutigen semantischen Differenz wird zusätzlich durch eine mehrdeutige Interpretierbarkeit auf Modell-/Diagrammebene erschwert. So ist die Differenz bezüglich der zwischen den markierten Blöcken liegenden *Lines* in Abbildung 1 interpretierbar als *„eine Line gelöscht und zwei neue Lines hinzugefügt“* oder *„Line Target geändert und eine neue Line hinzugefügt“* oder als *„Line Source geändert und eine neue Line hinzugefügt“*. Eine Präferenz kann hier und in vielen anderen Fällen nur auf Grundlage der Benutzer festgelegt werden, und muss schon bei der Korrespondenzbildung berücksichtigt werden.

Literaturverzeichnis

[1] Jürgen Wehren. Ein XMI-basiertes Differenzwerkzeug für UML-Diagramme. Diplomarbeit, Universität Siegen, 2004.

[2] Udo Kelter, Jürgen Wehren, and Jörg Niere. A Generic Difference Algorithm for UML Models. In Proc. of the Software Engineering Conference 2005 (SE), Essen, Germany, March 2005.