Werkzeugunterstützung bei kooperativer Modellierung und Variantenmanagement von Motorsteuergeräte-Funktionen

Marc Janßen, Christian Bartelt, Andreas Rausch

Software Systems Engineering
TU Clausthal
Julius-Albert-Str. 4
38678 Clausthal
vorname.nachname@tu-clausthal.de

Andreas Schulze, Stefanie Jauns-Seyfried, Hanno Jelden

Antriebselektronik - EAE
Volkswagen AG
Brieffach 16870
38436 Wolfsburg
vorname.nachname@volkswagen.de

Abstract: In vielen industriellen Anwendungsbereichen des System- und Softwareengineering ist eine hochgradig parallele Entwicklungsarbeit notwendig. Zudem spielen formale, grafische Modellierungssprachen eine immer größere Rolle. Ein Beispiel dafür ist die ASCET-Modellierung von Motorsteuergeräte-Funktionen bei der Volkswagen AG.
Hierbei stellt sich die manuelle Integration von parallel erzeugter Arbeitsartefakten von Funktions- und Softwareentwicklern, als auch von parallel entwickelten Funktionsfeatures zu neuen Varianten als sehr aufwendig und fehleranfällig heraus. Aus diesem Grund sind viele Entwicklungsprozesse heutzutage immer noch durch eine streng sequentielle Arbeitsweise geprägt. Im Folgenden wird eine Technologie zur automatischen Integration von parallel erstellten ASCET-Modellen vorgestellt. Ein sinnvoller Einsatz dieser Technologie wird zudem durch zwei repräsentative Anwendungsfälle zur Effizienzsteigerung in der kooperativen Entwicklung bzw. in der Variantenverwaltung erläutert.

1 Einleitung

1.1 Ausgangssituation

Softwaresysteme werden in immer mehr sicherheitskritischen Bereichen eingesetzt, sodass die Softwareingenieure trotz anspruchvoller Zeitvorgaben keine Abstriche bei der Verlässlichkeit und Qualität des Softwaresystems machen können. Stattdessen wird immer mehr Wert auf eine detaillierte Modellierung gelegt. Hierfür existieren standardisierte und grafische Modellierungsprachen zur Erzeugung von intuitiven und eindeutig verständlichen Modellen. Diese Modelle spielen eine wichtige Rolle bei der Dokumentation des Systems und Kommunikation zwischen den Entwicklern. Auf der anderen Seite

wird Software heute oft aus Softwaremodellen generiert. Dies führt, in Verbindung mit der parallelen Arbeit der Entwickler, zu einer kollaborativen Entwicklung von Modellen.

Ein Beispiel für generierte Software aus Modellen sind die Softwaremodule für Motorsteuergeräte, die mit dem Modellierungswerkzeug *ASCET* [ETA11] erstellt werden. Bei der Entwicklung der Softwaremodule sind dabei zwei unterschiedliche Gruppen von Entwicklern involviert, die als Funktionsentwickler (FE) und Softwareentwickler (SE) bezeichnet werden. Die FE modellieren die Konzepte und Funktionen der Softwaremodelle und die SE implementieren diese. Verteilte Modellierung gewinnt dadurch immer mehr an Stellenwert und die Anforderungen an Softwarewerkzeuge wachsen.

Softwarewerkzeuge die diesen Prozess unterstützen können, sind Versionierungstools wie *Subversion* (SVN), *Concurrent Versions System* (CVS) und *MKS Integrity* [Coma]. Mit ihnen werden Dateien verwaltet und Änderungen automatisch dokumentiert.

Besonders nützlich sind Tools zur Versionierung von Quellcode-Dateien. Mit ihnen können parallele Weiterentwicklungen automatisch zusammengefasst werden. Für Modellierungssprachen wie die *Unified Modeling Language* (UML [Groa]) gibt es zurzeit kaum Konzepte, die zu einer qualitativ hochwertigen Zusammenführung von Modellen genutzt werden können.

1.2 Problemstellung

Wie oben bereits erörtert wurde, arbeiten in der Automotive-Domäne verschiedene Teams von Entwicklern kollaborativ und verteilt an Modellen von Softwaremodulen. Die Modelle werden dabei von den FE und SE nicht parallel bearbeitet. Stattdessen bearbeiten sie ein Modell alternierend. Das FE-Team erstellt die Modelle und modelliert die Funktionalität. Danach erhält das SE Team eine Freigabe zur Weiterberarbeitung. Dazu kommt, dass die Softwaremodelle iterativ mehrere Modellversionen durchlaufen, in denen die FE Funktionen erweitern. Erweiterungen können jedoch erst begonnen werden, wenn die SE die Modelle ausreichend implementiert haben.

Bei diesem Entwicklungsverfahren kommt es häufig zu Wartezeiten bei den Entwicklern. Die SE können erst implementieren, wenn die Funktionalität bereits vorhanden ist. Die FE müssen allerdings nicht auf die Implementierung warten. Die funktionale und konzeptionelle Erweiterung kann theoretisch parallel zu der Implementierung erfolgen.

Das Modellierungstool *ASCET* bietet den Entwicklern derzeit keine Möglichkeit, parallel entwickelte Softwareversionen zusammenzuführen.

Eine Integration von parallel bearbeiteten Implementierungen und Funktionserweiterungen ist somit nicht ohne weiteres möglich. Dies ist das Hauptproblem, dass in dieser Publikation behandelt wird.

Eine erfolgversprechende Lösung bietet das Konzept in [Bar11]. Darin ist unter anderem beschrieben, wie eine Integration von asynchron erstellten Modellen realisiert werden kann. Aktuell wird das Konzept in *„Team.Mode“* an der TU Clausthal in Form eines Plugins für die Rich Client Plattform *Eclipse* [Foua] realisiert. Es kann UML Diagramme versionieren, die auf dem *Eclipse Modeling Framework* (EMF [Foub]) und dem *XML Metadata Interchange* (XMI [Grob]) basieren. Das Dateiformat, das zur Kommunikation der Modelle von ASCET genutzt wird, basiert hingegen auf dem *Extensible Markup*

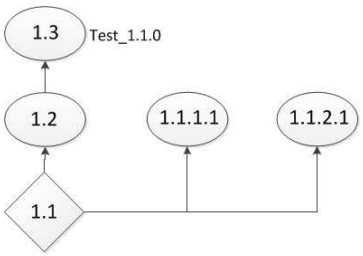

Abbildung 1: Ein beispielhafter Versionsgraph aus Integrity. Die Modellversionen 1.1-1.3 bilden die Mainline des Versionsgraphen, wobei die Modelle 1.1.1.1 und 1.1.2.1 jeweils Varianten des Modells sind, die parallel zur Mainline entwickelt werden.

Language Standard (XML [Con98]) mit einem Metamodell in einer *XML Schema Definition* (XSD [Con]). XMI ist jedoch ein spezieller XML Standard und das Merge-Verfahren umfasst gemäß dem Konzept alle Modellierungssprachen.

Die Team.Mode Software ermöglicht nicht nur die Zusammenfassung von verschiedenen Modellständen, sondern auch eine automatisierte Erkennung von inkonsistenten Modelländerungen, mit der zukünftig an einer Konfliktbehebung von inkonsistenten Modellen geforscht werden kann.

2 Kollaboratives modellbasiertes Entwicklungsverfahren

In diesem Kapitel wird die **Modellverwaltung mit Integrity** und den daraus resultierenden Anwendungsszenarien einer ASCET-Team.Mode-Integration erörtert. Die **kollaborative Modellverwaltung mit Team.Mode** zeigt, wie eine solche Integration die Entwickler unterstützen kann.

Dieses Kapitel erläutert die Motivation der Arbeit und einer ASCET-Team.Mode-Integration.

2.1 Modellverwaltung mit Integrity

In diesem Abschnitt wird die modellbasierte Softwareentwicklung in der Automotive-Domäne betrachtet. Dabei wird der Workflow der unterschiedlichen Entwicklergruppen geschildert, aus dem Konflikte bei der kollaborative Entwicklungsarbeit von Software- und Funktionsentwickler hervorgehen. Zusätzlich wird die Synchronisation parallel entwickelter Modellvarianten betrachtet, welche ebenfalls problematisch ist.

Wie in der Ausgangssituation bereits erwähnt wurde, arbeiten SE und FE nie parallel an dem selben Softwaremodul. Stattdessen werden die Arbeitsschritte sequentiell in ein Modell eingepflegt, sodass entweder die SE oder die FE ein Modell bearbeiten. Die gemeinsame Arbeit wird verwirklicht, indem die Modelle mit Integrity[1] verwaltet werden, wodurch ein Modell von einer Gruppe gesperrt und exklusiv bearbeitet werden kann.

Ein Problem, dass aus dem Prozess hervorgeht, betrifft die **kollaborative Entwicklungsarbeit von Software- und Funktionsentwicklern**. Diese Arbeitsweise kann nur reibungslos funktionieren, solange die Modellsperrungen keine Wartezeiten bei den Entwicklern verursachen. Insbesondere der schnelle Fortschritt eines einzelnen Entwicklungszweiges im Versionsgraphen, wie die Mainline selbst, wird auf diese Weise durch Wartezeiten unterbunden.

[1] Integrity ist ein Software System Lebenszyklus Management (SSLM [Comb]) mit dem es unter anderem möglich ist, Dateien zu versionieren. Es ist für die Nutzung der beiden Entwicklergruppen SE und FE geeignet.

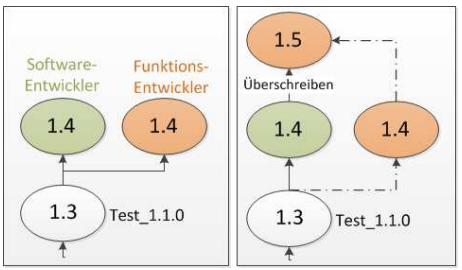

Abbildung 2: Auf der linken Seite ist ein inkonsistenter Versionsgraph zu sehen und auf der rechten die „Überspeicherung“ einer Modellversion zwecks Versionsgraph-Reperatur.

Diese Sperrungen sind notwendig, um eine kontrollierte und fehlerfreie Entwicklungsarbeit zu ermöglichen.

Ein Entwicklungsszenario ohne Sperrungen zeigt welche Störungen und Fehler auftreten können. In diesem Entwicklungsszenario arbeiten FE trotz einer Übergabe an einem Modul weiter. Durch die parallele Arbeit von SE und FE kommt es dazu, dass von einer Modellversion, zwei Entwicklungspfade abgeleitet werden. Auf diese Weise wird ein eindeutiger Versionsgraph inkonsistent (vgl. Abbildung 2). Die Teilung des bisherigen Enticklungspfades ist hier nicht beabsichtigt. Dieser *unbeabsichtigte Branch* muss aufgelöst werden, indem die beiden Ableitungen in einem Pfad zusammengefasst werden. Dies ist technisch bisher nicht automatisiert möglich. Eine manuelle Integration der beiden weiterentwickelten Modulversionen ist wiederum zeitaufwendig und fehleranfällig.

Eine herkömmliche Dateiversionierung bietet an dieser Stelle keine Abhilfe. Mit ihr kann ein unbeabsichtigter Branch nur aufgelöst werden, indem eines der Modelle als Nachfolger der anderen Modellversion abgelegt wird (vgl. Abbildung 2). Dies ist semantisch nicht korrekt und führt zur Vernachlässigung von Modelländerungen.

Dieses Szenario beschreibt somit eine ineffiziente Entwicklungsarbeit, die kollaborative Arbeit nicht unterstützt. Stattdessen arbeiten FE und SE im Szenario gegeneinander. In der Praxis werden solche Entwicklungsszenarien mit den Sperrungen von Modulen in Integrity verhindert. Ein Branch muss mit dem Versionierungstool Integrity aktiv erzeugt, wodurch inkonsistente Versiongraphen vermieden werden.

Diese Arbeitsweise ist bisher notwendig, da eine automatisierte Integration parallel bearbeiteter Modellvarianten mit Integrity nicht möglich ist. Integrity ist nicht in der Lage ASCET Modellinformationen zu versionieren.

Ein weitere Herausforderung der modellbasierten Softwareentwicklung ist die **Synchronisation paralleler Entwicklungspfade**.

In Integrity wird ein Branch meist dazu genutzt, eine neu entworfene Funktionalität als Prototyp und parallel zur Mainline zu entwickeln. Auf diese Weise kann an der Mainline weiterentwickelt werden und der Prototyp parallel dazu verbessert werden. Integrity kann dabei nur echte Bäume als Versionsgraphen erzeugen, sodass jede Modellversion nur einen Vorgänger hat.

Ein Beispiel dafür ist ein Modul zur Steuerung einer Start-Stop Automatik. Ein solches Modul kann verschiedene Varianten haben, zum Beispiel für Hybridfahrzeuge. Auf der Mainline wird dagegen lediglich eine Basis-Funktion für alle Fahrzeugtypen entworfen.

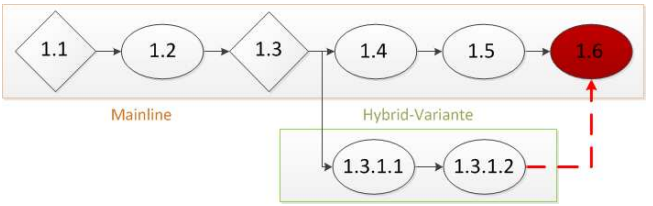

Abbildung 3: Versionsgraph mit zwei Varianten eines Softwaremoduls.

Soll die Start-Stop Automatik der Hybridfahrzeuge in allen Modellen Verwendung finden, muss diese Variante in die Mainline integriert werden.

In der Abbildung 3 ist die Entwicklung einer Hybridvariante parallel zur Mainline dargestellt. Eine automatisierte Integration der Hybridvariante in die Mainline (siehe roter Pfeil) wird derzeit nicht von Integrity unterstützt, da eine Zusammenführung von zwei Modellen bisher technisch nicht möglich ist. In der Praxis muss die Version 1.6 wie im vorigen Fall manuell erzeugt werden.

Die Dateiversionierung von Integrity bietet keine Lösung zur automatisierten Integration von ASCET Modellen und kann somit die Anwendungsfälle „kollaborative Entwicklungsarbeit von SE und FE“ und „Synchronisation paralleler Entwicklungspfade“ nicht ausreichend unterstützen.

2.2 Kollaborative Modellverwaltung mit Team.Mode

Das Hauptproblem der **kollaborative Entwicklungsarbeit von Software- und Funktionsentwicklern** ist, dass es für die Entwickler mit ASCET und Integrity nur eingeschränkt möglich ist, simultan an einem Modell zu arbeiten. Das optimistische Versionsmanagement mit Team.Mode erlaubt es dagegen, parallel gefertigte Modelle zu integrieren.

Ein Vorteil von Team.Mode gegenüber Integrity besteht darin, dass die FE nach ihrer Modell-Übergabe an dem Modell weiterarbeiten können. SE sind durch Team.Mode nicht gezwungen auf die Freigabe der FE zu warten und können direkt das Modul implementieren. Wird Team.Mode als Versionierungstool verwendet, entfallen die Sperrungen und ein paralleler Zugriff auf die Modelle wird unterstützt.

Team.Mode ist in der Lage inkonsistente Versionsgraphen zu reparieren. Die Reparatur besteht aus einem lokalem Update. Dadurch wird die parallel erzeugte Modellversion geladen und die eigene Version als Nachfolger dieser geladenen Version interpretiert. Mit Integrity müssen dadurch die Änderungen in dem geladenen Modell überschrieben werden. Team.Mode hingegen identifiziert die Änderungen, die seit der gemeinsamen Vorgänger-Modellversion modelliert wurden und fasst sie in dem lokalem Modell zusammen. Auf diese Weise wird das eigene Modell zu einem echten Nachfolger des parallel entwickelten Modells (vgl. Abbildung 4).

Durch das Wegfallen der Sperrungen können die Entwickler simultan an der Mainline arbeiten, ohne dass Modellinformationen verloren gehen, überschrieben werden oder zeitaufwendig nachmodelliert werden müssen. Damit kann der gesamte Entwicklungsprozess flexibler gestaltet werden.

Team.Mode unterstützt die kollaborative Entwicklungsarbeit von Software- und Funktionsentwicklern, indem eine Parallelisierung der Entwicklungsschritte ermöglicht wird.

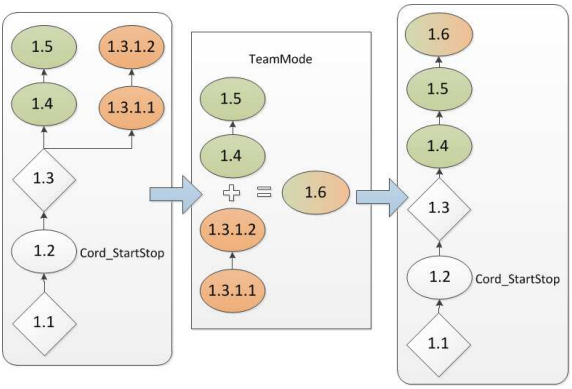

Abbildung 4: Integration parallel entwickelter Modellstände mit Team.Mode

Die manuelle Arbeit einer **Synchronisation von parallelen Entwicklungspfaden** kann dem Entwickler mit Team.Mode abgenommen werden.

In diesem Fall existieren durch den Branch wieder parallele Entwicklungspfade im Versionsgraphen. Team.Mode kann diese beiden Entwicklungspfade zusammenführen, indem die Änderungen seit der gemeinsamen Modellversion automatisiert erfasst und in einer Kopie des gemeinsamen Modells integriert werden. Die Lösung des Problems ist analog zu der Lösung des Problems der kollaborativen Entwicklungsarbeit von SE und FE. Eine Integration von zwei Modellen ist in Abbildung 4 skizziert.

Mit Team.Mode können parallel entwickelte Modellversionen automatisiert in eine Modellversion integriert werden.

3 Konzeption und Implementierung einer ASCET-Team.Mode-Integration

In diesem Kapitel wird erklärt, wie eine **Zerlegung von XML-Daten in versionierbare Einheiten** aussehen kann, um daraufhin zu zeigen, wie eine **Abbildung von ASCET-Informationen ins Team.Mode-Repository** realisiert werden kann. Zuletzt wird ein **ASCET-Connector in der Team.Mode-Architektur** vorgestellt der diese Abbildung implementiert.

3.1 Zerlegung von Modell-Daten in versionierbare Einheiten

Die morphologische Modell Repräsentation, im folgenden MMR genannt, ist die Modellrepräsentation innerhalb von Team.Mode. Sie stellt atomare Modellinformationen in Form von maschinenverarbeitbaren Datensätzen dar. Es gibt vier verschiedene MMR Typen, die jeweils unterschiedliche Aspekte eines Modells abbilden.

Die Vier verschiedenen Objekte der MMR:

ObjectPropertyAssertion: Modellverweise zwischen Modellelementen

DataPropertyAssertion: Atomare Dateneinheiten

TypeAssertion: Typverweise vom Modell hin zum Metamodell

Order: Ordnungsrelation zwischen Modellelementen

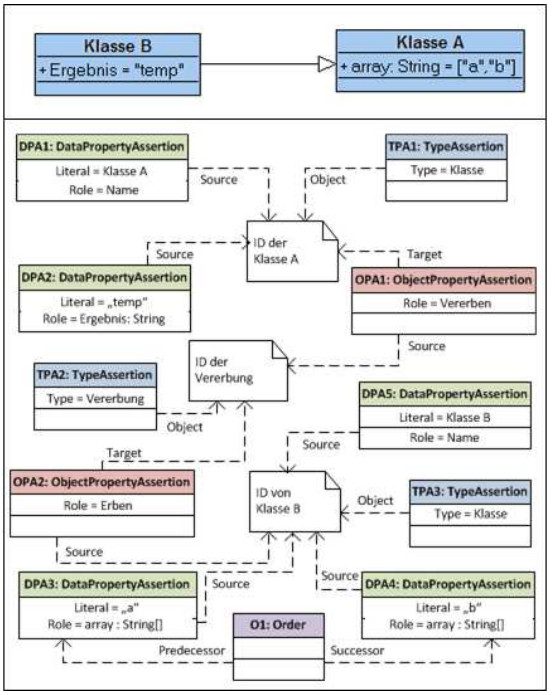

Abbildung 5: Abbildung eines kleinen UML Diagrams in die Morphologische Modell Repräsentation

MMR Objekte haben Zeiger auf andere MMR Objekte. Dabei zeigen sie auf Identifier (IDs), die aus natürlichen Eigenschaften von XML Informationen konstruiert werden, wie zum Beispiel Attributwerte und Typinformationen. Die Modellabbildung auf MMR sieht es vor, dass jedes Modellelement eine ID bekommt. Als Elemente können in diesem Zusammenhang UML Diagramm -Klassen, -Vererbungen, -Interfaces sowie deren Nutzung und Realisierung gesehen werden (vgl. Abbildung 5).

Die Abbildung zwischen einer Modellierungssprache und MMR kann beliebig feingranular gestaltet werden, um den Detailgrad der Verarbeitung von Team.Mode zu steigern. Je detaillierter die einzelnen Modellinformationen in MMR Objekte zerlegt werden, desto genauer kann mit Team.Mode eine Änderung identifiziert werden.

3.2 Abbildung von ASCET-Informationen ins Team.Mode-Repository

Das ASCET Dateiformat für Modell, ist das *.axl- Format*. Dieses Dateiformat besteht aus einem ZIP-Archiv mit mehreren Ordnern und XML Modelldateien. Daraus folgt, dass zur Abbildung eines ASCET Modells zuerst das Archiv abgebildet werden muss. Dabei erhält jeder Ordner und jede Datei eine ID und werden mittels ObjectPropertyAssertions in Beziehung zu einander gestellt.

Das weitere Vorgehen der Abbildung betrifft die XML Modelldateien. Diese können auf Basis der XML Informationen in MMR abgebildet werden. Dazu erhält jedes XML Tag eine ID. Dadurch können alle XML Tags mit ObjectPropertyAssertions in Beziehung zu einander gestellt werden. Des Weiteren werden die zugehörigen XML Attribute mit

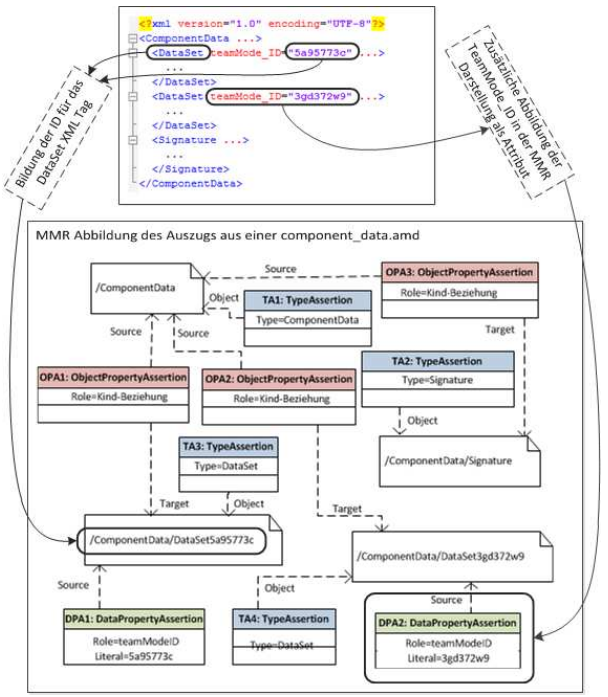

Abbildung 6: Abbildung einer AXL Modelldatei mit Nutzung von ID Attributen(TeamModeID)

DataPropertyAssertions und der XML Tagtyp mit TypeAssertions verknüpft werden (vgl. Abbildung 6 zu sehen).

Die größte Herausforderung dabei ist die Abbildung von XML Tags mit „Geschwister-XML Tags“ vom selben Typ. Diese sind mit dem Pfad im XML-Baum nicht von einander unterscheidbar. Zur weiteren Differenzierung dieser XML Tags können neue XML Attribute genutzt werden, dessen Wert zur ID des XML Tags hinzugefügt wird (siehe Abbildung 6).

Daraus entsteht jedoch das Problem, dass das Schema der Modelldateien angepasst wird und die Modellierungssoftware ASCET um die Verarbeitung dieser ID Attribute erweitert werden muss.

3.3 Der ASCET-Connector in der Team.Mode-Architektur

Die Team.Mode Software besteht aus mehreren Eclipse-Plugins. Der Connector, der es Team.Mode ermöglichen soll auf ASCET-Modellinformationen zu zugreifen, wird ebenfalls als Eclipse-Plugin implementiert. In diesem Abschnitt wird die Architektur von einem Connector dargestellt.

Einem ASCET-Connector wird ein ASCET-Modell übergeben, welches er in MMR transformiert, um die Nutzung von Team.Mode zu ermöglichen. Des Weiteren transformiert der Connector MMR Modelle wieder zurück, damit sie mit ASCET weiterverarbeitet werden können.

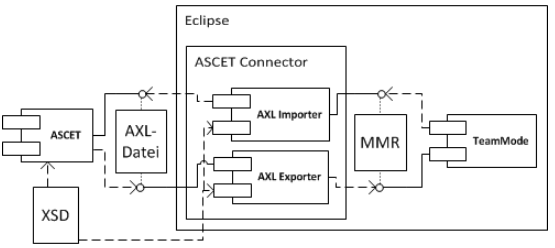

Abbildung 7: Verwendung des ASCET Connectors durch ASCET und Team.Mode

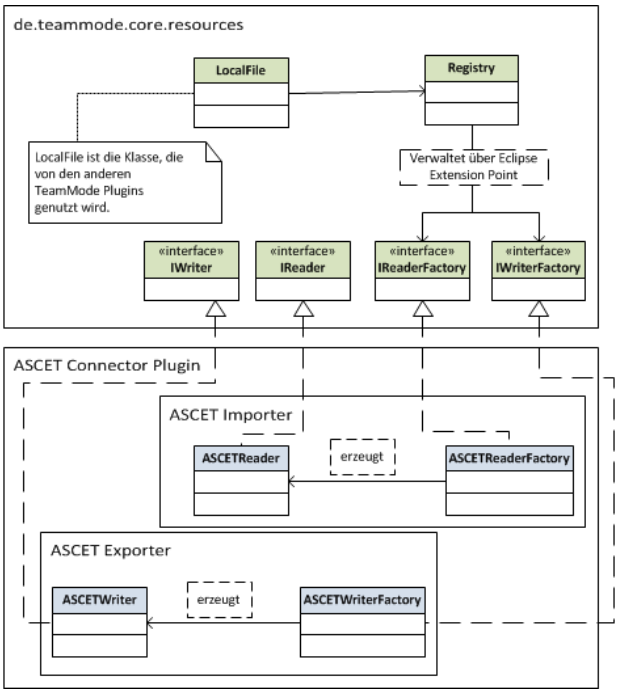

Abbildung 8: Schnittstelle zwischen Team.Mode und dem ASCET-Conector

Ein Connector für Team.Mode muss Reader- und Writer-Interfaces implementieren, sowie dazugehörigen Factorys. Diese sind in einem Package der Team.Mode-Software enthalten. Die Reader und Writer Objekte werden dabei zur Übersetzung der Modell Dateien genutzt. Durch die Factorys werden die Datei auf das Dateiformat geprüft und bei den geeigneten Dateien entsprechende Reader und Writer Objekte erstellt.

Die Schnittstelle zwischen einem Connector und Team.Mode ist der Extensionpoint de.teammode.resources.io des Team.Mode Plugins, über den Team.Mode die implementierten Reader- und Writer-Factorys nutzt.

4 Zusammenfassung und Ausblick

In den vorangegangenen Abschnitten wurde das Team.Mode-Konzept beispielhaft mit einer ASCET-Team.Mode-Integration evaluiert. Die beschriebene Implementierung einer ASCET-Team.Mode-Integration ist ein erster Schritt zu einem Versionsmanagement von

ASCET-Modellinformationen. Sie ermöglicht Nutzern von ASCET die Team.Mode- Technologie auf das ASCET-Exportdateiformat AXL anzuwenden. Mit Team.Mode können die ASCET-Entwickler parallel entwickelte Modellvarianten automatisiert zusammenzufassen. Die ASCET-Team.Mode-Integration stellt die Verwendung des optimistischen Versionsmanagements von Team.Mode mit ASCET-Modellen zur Verfügung.

Das Konzept ermöglicht die parallele, kollaborative Arbeit von Software- und Funktionsentwicklern, mit die Entwicklungsprozedur in der Automotive-Domäne flexibler gestaltet werden kann. Zusätzlich dazu kann die aufwendige Arbeit, die zur Synchronisation von parallelen Modellvarianten nötig ist, mit diesem Konzept automatisiert werden.

Durch die Erweiterung der ASCET-Software, bedingt durch erforderliche Metamodell-Anpassungen, könnte ASCET zukünftig in der Lage sein, AXL Archiv zu erzeugen, die problemlos vom Versionsmanagement von Team.Mode genutzt werden können. Des Weiteren kann mit der automatischen Konflikterkennung eine automatisierte Konfliktbehandlung für inkonsistente Modelländerungen entworfen werden. Zusätzlich ist es möglich das Team.Mode Konzept in Entwicklerwerkzeuge zu portieren, damit Team.Mode transparenter und komfortabler genutzt werden kann.

Literatur

[Bar11] Christian Bartelt. „Kollaborative Modellierung im Software Engineering“. Dissertation. Dr. Hut, 2011. München.

[Coma] The Product Development Company. Integrity. http://www.mks.com/. Abrufdatum: 03.02.2012, 11:00.

[Comb] The Product Development Company. Software System Lifecycle Management. http://www.mks.com/solutions/discipline/sslm. Abrufdatum: 03.02.2012, 11:00.

[Con] World Wide Web Consortium. XML Schema. http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/. Abrufdatum: 03.02.2012, 11:00.

[Con98] World Wide Web Consortium. Extensible Markup Language. http://www.w3.org/standards/xml/core, 1998. Abrufdatum: 03.02.2012, 11:00.

[ETA11] ETAS. ASCET. http://www.etas.com/en/products/ascet_software_products.php, 2011. Abrufdatum: 03.02.2012, 11:00.

[Foua] Eclipse Foundation. Eclipse. http://www.eclipse.org/. Abrufdatum: 03.02.2012, 11:00.

[Foub] The Eclipse Foundation. Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/. Abrufdatum: 03.02.2012, 11:00.

[Groa] Object Management Group. Unified Modeling Language. http://www.omg.org/spec/UML/. Abrufdatum: 03.02.2012, 11:00.

[Grob] Object Management Group. XML Metadata Interchange. http://www.omg.org/spec/XMI/2.1/. Abrufdatum: 03.02.2012, 11:00.