Mathias Schraps

(mathias.schraps@audi.de)
ist Doktorand bei der Audi Electronics Venture GmbH und promoviert auf dem Gebiet „Ontologiegestütztes Anforderungsmanagement in der Fahrzeugfunktionsentwicklung" an der Leibniz Universität Hannover.

Maximilian Peters

studiert Informatik im Master-Studiengang an der Leibniz Universität Hannover.

Spezifikation und Verifikation von Anforderungen mit dem GO-Editor

Die Behandlung von Fehlern und Inkonsistenzen in Anforderungsdokumenten stellt in fortgeschrittenen Phasen von Systementwicklungsprojekten einen erheblichen Kostenfaktor dar. Je früher eine fehlerfreie und vollständige Anforderungsspezifikation vorliegt, desto weniger Kosten entstehen im Nachhinein für die Beseitigung der Mängel. Um diese Probleme zu vermeiden, wurde eine Spezifikationssprache auf Basis einer Grammatik entwickelt, die sicherstellen soll, dass Anforderungen bei deren Formulierung eine bestimmte Struktur einhalten. Durch die Verwendung dieser Grammatik und der damit verbundenen Formalisierung von Anforderungen können zum einen Spezifikationsfehler bereits in frühen Phasen der Anforderungserhebung vermieden werden und zum anderen können Ontologien dabei helfen, die Anforderungen zu speichern und im weiteren Verlauf der Systementwicklung maschinell weiterzuverarbeiten und auszuwerten. Zur toolseitigen Unterstützung dieses Ansatzes und zur Erhebung von Anforderungen wurde der GO-Editor entwickelt, der den Einsatz einer Grammatik in Verbindung mit Ontologien ermöglicht und die Spezifikateure beim Schreiben von projektkonformen Anforderungen unterstützt.

Motivation und Problemstellung

Die Behandlung von Fehlern und Inkonsistenzen in Anforderungsdokumenten stellt in fortgeschrittenen Phasen von Systementwicklungsprojekten einen erheblichen Kostenfaktor dar [BoBa05]. Je früher eine fehlerfreie und vollständige Anforderungsspezifikation vorliegt, desto weniger Kosten entstehen im Nachhinein für die Beseitigung der Mängel.

Besonders bei interdisziplinären Stakeholdern aus verschiedenen Domänen und der Verwendung von natürlicher Sprache zur Spezifikation von Anforderungen ist es in der Praxis eher schwierig sicherzustellen, dass jeder Beteiligte die Anforderungen gleich interpretiert und versteht, z. B. aufgrund des persönlichen Wissenshintergrundes. Hier kann es beispielsweise zu einer nicht einheitlichen Verwendung von Bezeichnungen oder (Fach-) Begriffen zwischen den beteiligen Entwicklern kommen. Auch bei Abkürzungen oder Akronymen ist dieser Effekt zu beobachten, wenn sie nicht vorher explizit definiert worden sind. Bei der Kommunikation über Anforderungsdokumente können auch sprachliche Defekte auftreten (vgl. [RuSO09] [Poh08]) wie z. B.

- **Tilgung:** Weglassen von wichtigen Informationen oder implizite Annahmen,
- **Generalisierung:** Verallgemeinerung von speziellen Eigenschaften,
- **Verzerrung:** Änderung von Fakten durch Nominalisierung o. Ä.

Diese nicht einheitliche Verwendung einer gemeinsamen Spezifikationssprache betrifft nicht nur die frühen Phasen des Entwicklungsprozesses (z. B. die Anforderungserhebung), sondern hat auch maßgeblichen Einfluss auf die darauf folgenden Modellierungs-, Implementierungs- und Testphasen eines Softwaresystems.

Aus den gestellten Anforderungen werden beispielsweise Design-Artefakte, wie System- und Verhaltensmodelle, entwickelt, die dann in den eigentlichen Programmcode abgeleitet werden. Während dieser Entwicklungsschritte müssen die aktuellen Entwicklungsartefakte fortwährend gegen die Ergebnisse der vorangegangenen Phasen verifiziert werden, um sicherzustellen, dass die Anforderungen umgesetzt wurden.

Letztendlich wird das entstandene System dann in den späteren Testphasen gegen die gestellten Anforderungen validiert. Dabei kann es vorkommen (also erst recht spät!), dass Testspezifikateure die Anforderung anders interpretieren als Modellierer, z. B. aufgrund der o. g. Defekte natürlicher Sprache. Hier soll gezeigt werden, wie die Qualität von Anforderungen mithilfe von Grammatik und Ontologien verbessert werden kann, da durch diesen Ansatz wichtige Eigenschaften von Softwareanforderungen unterstützt werden, wie Eindeutigkeit, Konsistenz und Verifizierbarkeit [IEEE98].

Gerade bei der Entwicklung von Anforderungsspezifikationen im Automotive-

Bereich besteht in diesem Ansatz ein erheblicher Mehrwert, da hier viele Stakeholder und Entwickler unterschiedlicher Fachdomänen miteinander kommunizieren und an Standards und Normen gebunden sind [ScAl12]. Zudem kommen hier Prozesse und Tools nach dem Automotive SPICE® Process Reference Model zum Einsatz [Aut10], wobei Anforderungen transparent über den gesamten Entwicklungsprozess nachvollzogen werden müssen.

Ansatz über Grammatik und Ontologien

Um diesen Problemen bei der Anforderungsspezifikation in frühen Phasen der Entwicklung entgegenzuwirken, wurde eine eingeschränkte natürliche Sprache oder Normsprache entwickelt (englisch: Controlled Natural Language – CNL). Diese basiert auf einer Grammatik und Regeln, mit deren Hilfe alle Anforderungen ausgedrückt und formuliert werden können. Formulierungen, die nicht in Anforderungsspezifikationen auftreten sollten, wie z. B. Fragesätze oder vage bzw. unvollständige Aussagen (f_i), werden damit explizit nicht zugelassen $(f_i \in NL \mid i \in [1..3])$.

Es werden also nur wohlgeformte Satzstrukturen durch die Grammatik der Anforderungssprache *RL* akzeptiert. Abbildung 1 verdeutlicht diese Überlegung, da die Anforderungssprache *RL* als Teilmenge in der natürlichen Sprache *NL* enthalten ist $(RL \subseteq NL)$. Im Gegensatz zu textuellen Satzmustern oder Textbausteinen $(t_i \in RL \mid i \in [1..4])$, die im Requirements Engineering recht weit verbreitet sind (vgl. [RuSO09] [Poh08] [HJD11]), bietet der Einsatz von Grammatiken einen weitaus größeren Sprachraum und damit auch mehr Möglichkeiten zur Formulierung.

Dies wird verdeutlicht durch die einzelnen t_i im Vergleich zur Fläche der Anforderungssprache *RL*. Die t_i als einzelne Elemente sind in *RL* enthalten, sofern die Satzmuster in der Grammatik definiert sind (vgl. Abbildung 1).

Ein weiterer Aspekt dieses Ansatzes beinhaltet die Kombination dieser Anforderungssprache mit der Möglichkeit, Anforderungen mithilfe von Ontologien rechnerintern speichern und mit dem GO-Editor weiterverarbeiten zu können. Sind die Anforderungen einmal über die Anforderungssprache (Grammatik) formalisiert und in der Ontologie gespeichert, können sie in späteren Entwicklungsphasen (z. B. Design / Modellierung oder Test) dazu verwendet werden, um daraus Systemmodelle zu erstellen bzw. Testfälle abzuleiten.

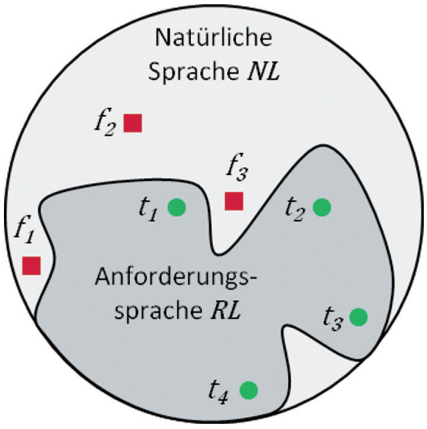

Abb. 1: Abgedeckter Sprachraum einer Anforderungssprache

Diese Systemmodelle können sogar mithilfe des GO-Editors gegen die erhobenen Anforderungen verifiziert werden.

Verwendung einer Grammatik

Um Anforderungen automatisiert prüfen zu können, ist es nicht ausreichend, dass sie nach einem bestimmten Muster (Syntax) formuliert sind. Es müssen zusätzlich Regeln (Semantik) festgelegt sein, wie ein gültiger Satz nach einer Grammatik aufgebaut werden kann, d. h. an welcher Stelle im Satz Variablen oder Verben vorkommen dürfen und welche Platzhalter (Nicht-Terminalsymbole) in andere Wörter oder Satzteile der Grammatik aufgelöst werden dürfen.

Diese Vorschriften werden Produktionsregeln genannt [Cho56]. Damit lassen sich wohlgeformte Anforderungen nach einer festgelegten Grammatik bzw. Anforderungssprache formulieren, sodass am Ende keine Platzhalter mehr abgeleitet werden können und damit ausschließlich Terminalsymbole in der Anforderung vorkommen. Auf die Definition der gesamten Grammatik wird hier verzichtet und auf [Pet13] verwiesen.

Mittels der folgenden Ableitungsschritte wird beispielsweise eine Anforderung an eine Eigenschaft <p:prop> eines Individuums <i:a> mit dem Wert <#:num> spezifiziert.

1. /start()
2. </dataRestriction():1>.
3. The property <p:prop> of <i:a> must be </restr(prop):1> <#:num>.
4. The property <p:prop> of <i:a> must be at most <#:num>.
5. The property weight of CarX must be at most 1800kg.

Hier werden in jedem Ableitungsschritt von 1 bis 4 alle Nicht-Terminalsymbole in Terminalsymbole aufgelöst und damit wird die Struktur der späteren Anforderung vorgegeben. Konkret wird im Schritt 2 eine Anforderung vom Typ einer Rahmenbedingung bzw. Beschränkung gewählt. Die resultierende Satzstruktur wird im Schritt 4 weiter verfeinert, indem die Restriktion im hinteren Teil des Satzes auf „at most“ gesetzt wird. Im letzten Schritt 5 werden dann die Platzhalter mit den spezifischen Werten einer Anforderung ersetzt und die Anforderung ist fertig abgeleitet.

Die Verwendung einer solchen Grammatik zur Anforderungsdefinition bietet gegenüber statischen Satzmustern einige Vorteile:

- **Änderbarkeit:** Die Produktionen einer Grammatik können so definiert werden, dass Änderungen sehr einfach vorgenommen werden können, z. B. wenn im Projektverlauf weitere Anforderungstypen hinzukommen.
- **Effizienz:** Der Speicherplatz einer Grammatik ist deutlich geringer als der für eine Liste aller möglichen Satzmuster (bspw. Endlose Sprache, „If ... and … and ... then…else…“)
- **Wiederverwendbarkeit:** Teile einer Grammatik können in mehreren Produktionsregeln verwendet und kombiniert werden. Zum Beispiel kann ein Subjekt-Prädikat-Objekt-Satz in mehreren Anforderungstypen hilfreich sein.
- **Mächtigkeit:** Eine Anforderungssprache ist ausdrucksstärker als eine Liste von Schablonen. Dies wurde bereits in Abbildung 1 verdeutlicht.

Abbildung der Anforderungen in eine Ontologie

Um jedoch die Anforderungen in eine Ontologie abbilden zu können, reicht diese Art von Semantik noch nicht aus. Die einzelnen Produktionsregeln der Grammatik wurden deshalb noch mit semantischen Annotationen erweitert (siehe Abbildung 2).

Jede dieser Annotationen bewirkt, dass die entsprechende Produktionsregel in bestimmte Konstrukte der Ontologie transformiert wird. Die Annotation einer Produktionsregel erfolgt direkt in der Grammatikdefinition über sogenannte

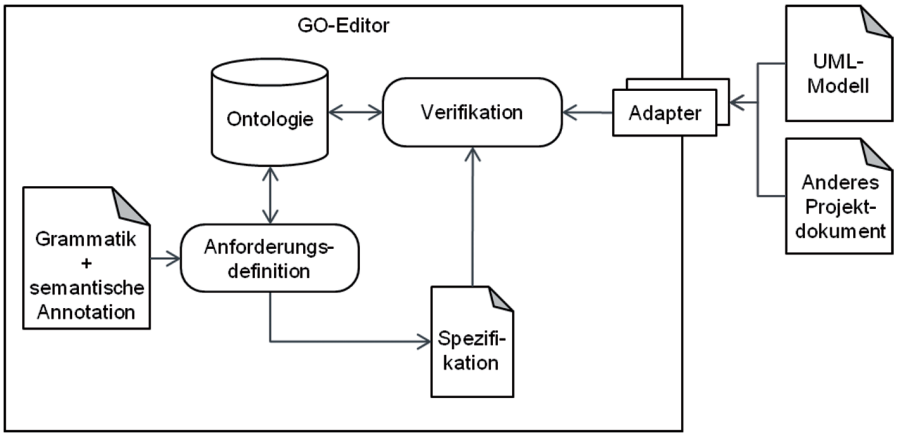

Abb. 2: Architektur und Schnittstellen des GO-Editors

Semantic Patterns. Diese sind durch eine spezielle Meta-Grammatik erlaubt (eine Grammatik für die Grammatik), sodass sie ein entsprechender Parser einlesen kann.

Während der Formulierung einer Anforderung lösen die Produktionsregeln alle Nicht-Terminalsymbole in Terminalsymbole auf. Wenn die Anforderung fertig formuliert ist, wird – verborgen vor dem Benutzer – jedes Terminalsymbol, da es nicht weiter aufgelöst werden kann, mit einem Semantic Pattern gekennzeichnet.

Die Produktionsregel einer Restriktion, wie sie im oberen Beispiel bereits verwendet wurde, wird annotiert, indem die Regel durch das Pfeilsymbol ==> mit restrMax(1) gekennzeichnet wird:

/restr(1)->at most==>restrMax(1)

Die (1) stellt lediglich einen Platzhalter für Parameter dar, womit im o. g. Beispiel der Wert 1800 übergeben wird. Die sich daraus ergebenen Ontologiekonstrukte stellt Abbildung 3 stark vereinfacht dar.

Die Restriktion für das Individuum CarX der Klasse Car wird in ein DataPropertyAssertion-Konstrukt mit der Bezeichnung „dataRestriction_CarX_weight_property“ umgesetzt, die durch die DataPropertyRange einen Maximalwert kleiner gleich 1800 zulässt.

Die Abbildung der Grammatik in eine Ontologie ist die Grundlage dafür, dass die Informationen aus den formulierten Anforderungen (konkrete Objekte, Verben, Eigenschaften) in der Ontologie gespeichert werden können. Da verschiedene Semantic Patterns in unterschiedliche Ontologiekonstrukte abgebildet werden, wird hier auf eine weitere Detaillierung der Abbildungsvorschriften verzichtet und auf [Pet13] verwiesen.

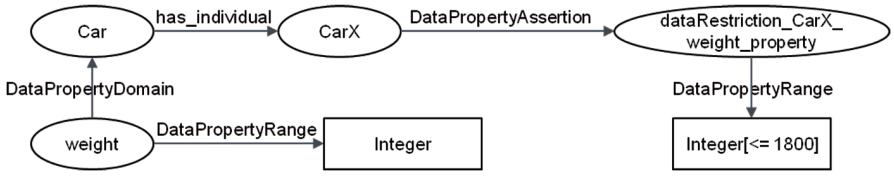

Abb. 3: Ontologiekonstrukte einer annotierten Produktionsregel

Der GO-Editor

Der GO-Editor ermöglicht die Erstellung einer Ontologie, in der das Wissen über das zu spezifizierende bzw. zu entwickelnde System enthalten ist. In einem ersten Schritt können die einzelnen Komponenten, deren Eigenschaften und Signale als Individuen oder Klassen sowie ggf. Prozessworte des Systemverhaltens in die Ontologie aufgenommen werden. Basierend auf dieser Ontologie kann der Benutzer Anforderungen entsprechend der o. g. Grammatik erheben.

Erhebung der Anforderung mit Grammatikunterstützung

Zur Erhebung einer Anforderung mit Grammatikunterstützung werden die Produktionsregeln sukzessive aufgelöst bis ein vollständiger Satz abgeleitet wurde. Abbildung 4 zeigt diesen Vorgang anhand einer Schrittfolge im GO-Editor mithilfe von Drop-Down-Menüs.

Der Benutzer wählt die Produktionsregel „/start“ (in Abbildung 4 Schritt 1) zur Ableitung aus und selektiert konkrete Werte für Platzhalter aus der vorher erstellten Ontologie. Von Schritt 2 auf 3 ist zu sehen, dass der Platzhalter „<individual:a>“ in ein Individuum „ControllerX“ aus der Ontologie aufgelöst wird.

Ist die Ableitung der Anforderung abgeschlossen (Schritt 5), kann diese Anforderung in die Spezifikation eingefügt werden. Dazu werden im Hintergrund die Annotationen an den verwendeten Produktionsregeln und die gewählten Einträge in den Drop-Down-Menüs in die Ontologie abgebildet und stellen damit die ausge-

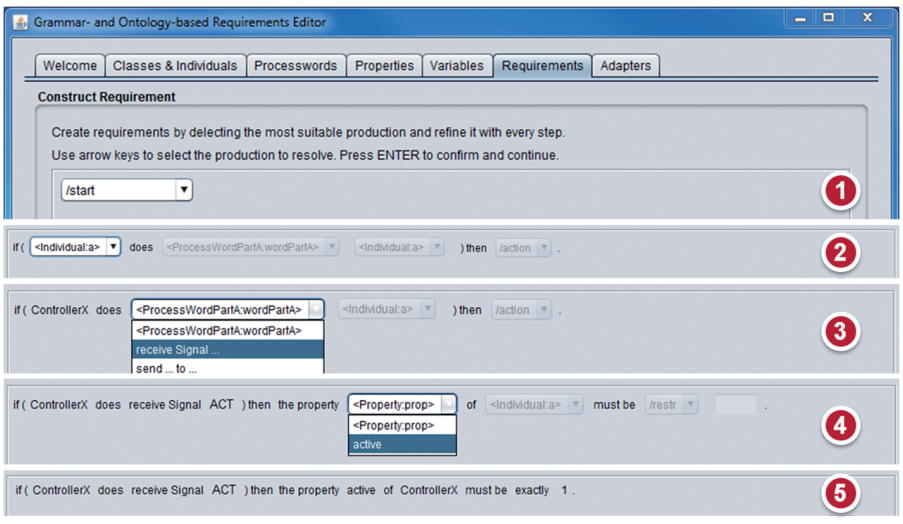

Abb. 4: Unterstützung bei der Formulierung von Anforderungen

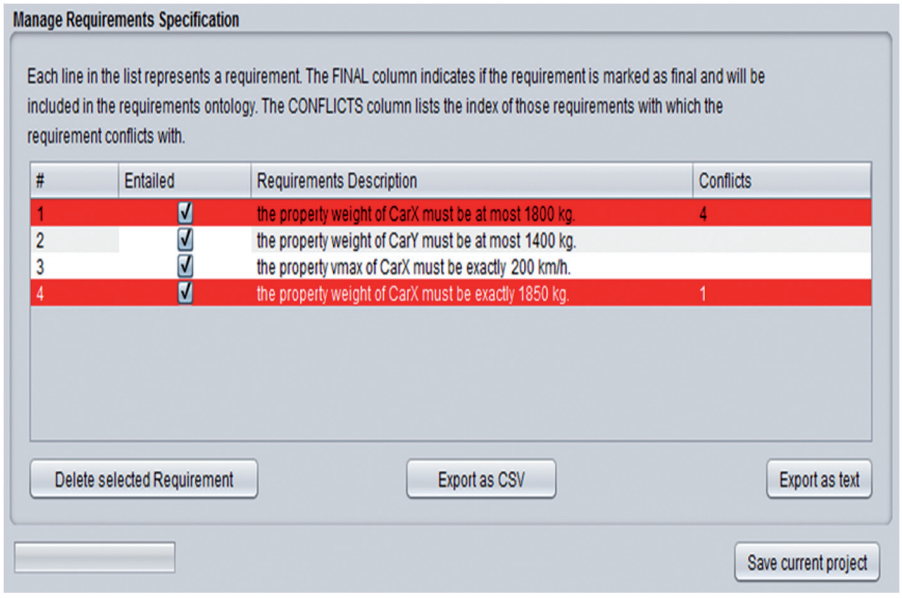

Abb. 5: Interne Konsistenzprüfung

wählten Individuen in einen Zusammenhang.

Interne Konsistenzprüfung

Jede in der Spezifikation enthaltene Anforderung wird, basierend auf der von ihr beschriebenen Semantik (also ihrer Bedeutung), mit den übrigen Anforderungen der Spezifikation auf Widerspruchsfreiheit geprüft. Jede Anforderung wird im GO-Editor durch Ontologiekonstrukte repräsentiert. Durch Reasoning-Mechanismen (Schlussfolgern auf Basis von Regeln) der verwendeten Ontologie kann bestimmt werden, ob zwei Anforderungen miteinander in Konflikt stehen.

Tritt zwischen zwei Anforderungen ein semantischer Widerspruch auf, so visualisiert der GO-Editor dies, indem zum einen die entsprechende Anforderung in der Spezifikation rot hinterlegt wird und zum anderem, indem die jeweilige Anforderung aufzeigt, zu welchen Anforderungen sie in Konflikt steht, siehe Abbildung 5.

Externe Konsistenzprüfung

Hat der Requirements Engineer eine Spezifikation erstellt, so kann er andere Projektdokumente (z. B. UML- oder MATLAB-Modelle, Tabellenkalkulationen, usw.) mithilfe sogenannter Adapter importieren und somit diese Dokumente gegen die Anforderungen verifizieren (siehe Abbildung 2). Dieser Vorgang wird hier als externe Konsistenzprüfung bezeichnet. Hierzu verfügt das Tool über eine Adapter-Schnittstelle. Ein Adapter besitzt Transformationsroutinen, die für beliebige Dokumentenformate entwickelt werden können. Sie beschreiben, wie das Wissen aus einem Dokumentenformat in der Ontologie repräsentiert werden soll.

In Abbildung 6 wurde zum Beispiel ein UML-Objekt-Diagramm aus dem ArgoUML-Editor [Arg] exportiert und mittels eines entwickelten UML-Adapters in den GO-Editor importiert. Das UML-Modell wird vom GO-Editor in eine Ontologie transformiert und durch Reasoning-Mechanismen kann auch hier verifiziert werden, ob die Anforderungen korrekt im Modell umgesetzt worden sind. Im GO-Editor wird dann aufgezeigt, welche Anforderungen der Spezifikation in diesem Objektdiagramm erfüllt sind.

Zusammenfassung

In diesem Artikel wurde eine Möglichkeit vorgestellt, wie mithilfe von Grammatiken eine eingeschränkte Anforderungssprache

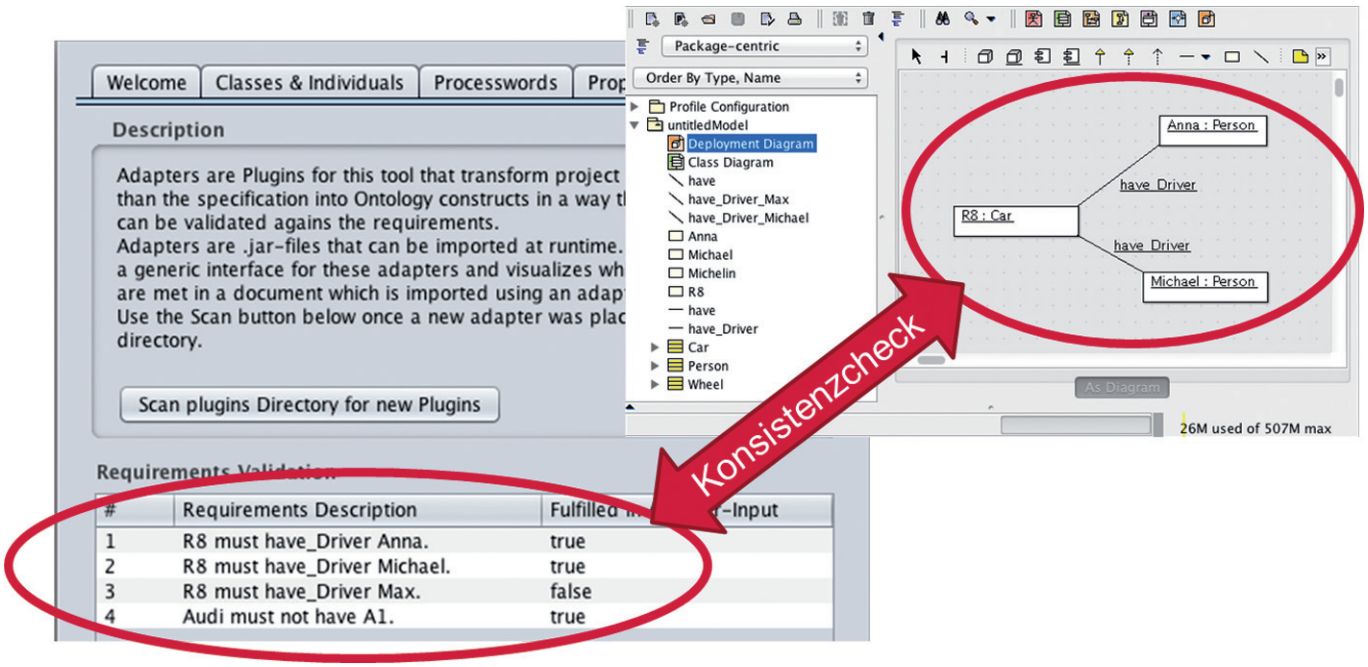

Abb. 6: Externe Konsistenzprüfung

definiert wird, die sowohl allgemein aus der Literatur bekannte als auch projektspezifische Satzmuster abdecken kann. Teile (Produktionsregeln) dieser Grammatik können semantisch annotiert werden, sodass es möglich ist, sie in bestimmte Konstrukte einer Ontologie abzubilden.

Der Requirements Engineer wird zum einen vom GO-Editor unterstützt, seine Anforderungen konform gegenüber einer vorgegebenen Anforderungssprache zu formulieren. Verwendete (Fach-) Begriffe werden dabei in einer Ontologie gespeichert und können in der gesamten Spezifikation wiederverwendet werden, wodurch auch Tippfehlern vorgebeugt wird.

Zum anderen ermöglicht die Ontologie, dass der Anwender die so erhobenen Anforderungen auf interne Konsistenz überprüfen kann. Die Verifikation der Anforderungen gegen andere Projektdokumente kann ebenfalls über Dokumenttyp-spezifische Adapter geprüft werden. Damit werden nicht nur der Requirements Engineer in seiner Arbeit unterstützt, sondern auch andere Entwickler, wie beispielsweise Modellierer bei der Erstellung von Design-Artefakten in späteren Entwicklungsschritten. ■

Referenzen

[Arg] http://www.argouml.tigris.org/

[Aut10] AUTOMOTIVE SPECIAL INTEREST GROUP (SIG): Automotive SPICE® Process Reference Model, 2010.

[BoBa05] BOEHM, Barry; BASILI, Victor R.: Software Defect Reduction Top-10 List. In: BOEHM, Barry; ROMBACH, Hans Dieter; ZELKOWITZ, Marvin V. (Hrsg.): Foundations of Empirical Software Engineering : The Legacy of Victor R. Basili : Springer-Verlag Berlin Heidelberg, 2005. – ISBN 978-3-540-24547-6, S. 426.

[Cho56] CHOMSKY, Noam: Three models for the description of language. In: Information Theory, IRE Transactions on 2 (1956), Nr. 3, S. 113–124.

[HJD11] HULL, Elizabeth; JACKSON, Ken; DICK, Jeremy: Requirements Engineering. 3. Aufl. London : Springer Verlag London Limited, 2011. – ISBN 978-1-84996-404-3.

[IEEE98] IEEE 830:1998: IEEE Recommended Practice for Software Requirements Specification, 25.06.1998.

[Pet13] PETERS, Maximilian: Concept to capture Relations between Project Documents using Ontologies. Masterarbeit. Hannover, Leibniz Universität Hannover, Institut für Praktische Informatik (FG Software Engineering), 15.05.2013. http://www.se.uni-hannover.de/pub/File/pdfpapers/Peters2013.pdf

[Poh08] POHL, Klaus: Requirements Engineering : Grundlagen, Prinzipien, Techniken. 2., korrigierte Aufl. Heidelberg : dpunkt-Verl, 2008. – ISBN 978-3-89864-550-8.

[RuSO09] RUPP, Chris; SOPHIST GROUP: Requirements-Engineering und -Management : Professionelle, iterative Anforderungsanalyse für die Praxis. 5., aktualisierte und erw. Aufl. München , Wien : Hanser, 2009. – ISBN 978-3-446-41841-7.

[ScAl12] SCHRAPS, Mathias; ALLMANN, Christian: Ontologiebasierte Entwicklung von Anforderungsspezifikationen im Automotive-Umfeld. In: Softwaretechnik-Trends 32, Nr. 4, 2012.