MODICA Tutorial

MODICA Version 5.4.0

Einführung

Das MODICA Tutorial beschreibt den Prozess der Modellerstellung und Testfallgenerierung anhand eines einfachen Beispiels. Ziel ist es, ein Modell anhand der Anforderungen für einen Abstandsregeltempomat zu erstellen und für dieses Modell Testfälle zu generieren, welche die Anforderungen abdecken.

Voraussetzungen:

- Installiertes MODICA
- DOORS Client inkl. Zugangsdaten

Verwendetes Beispiel

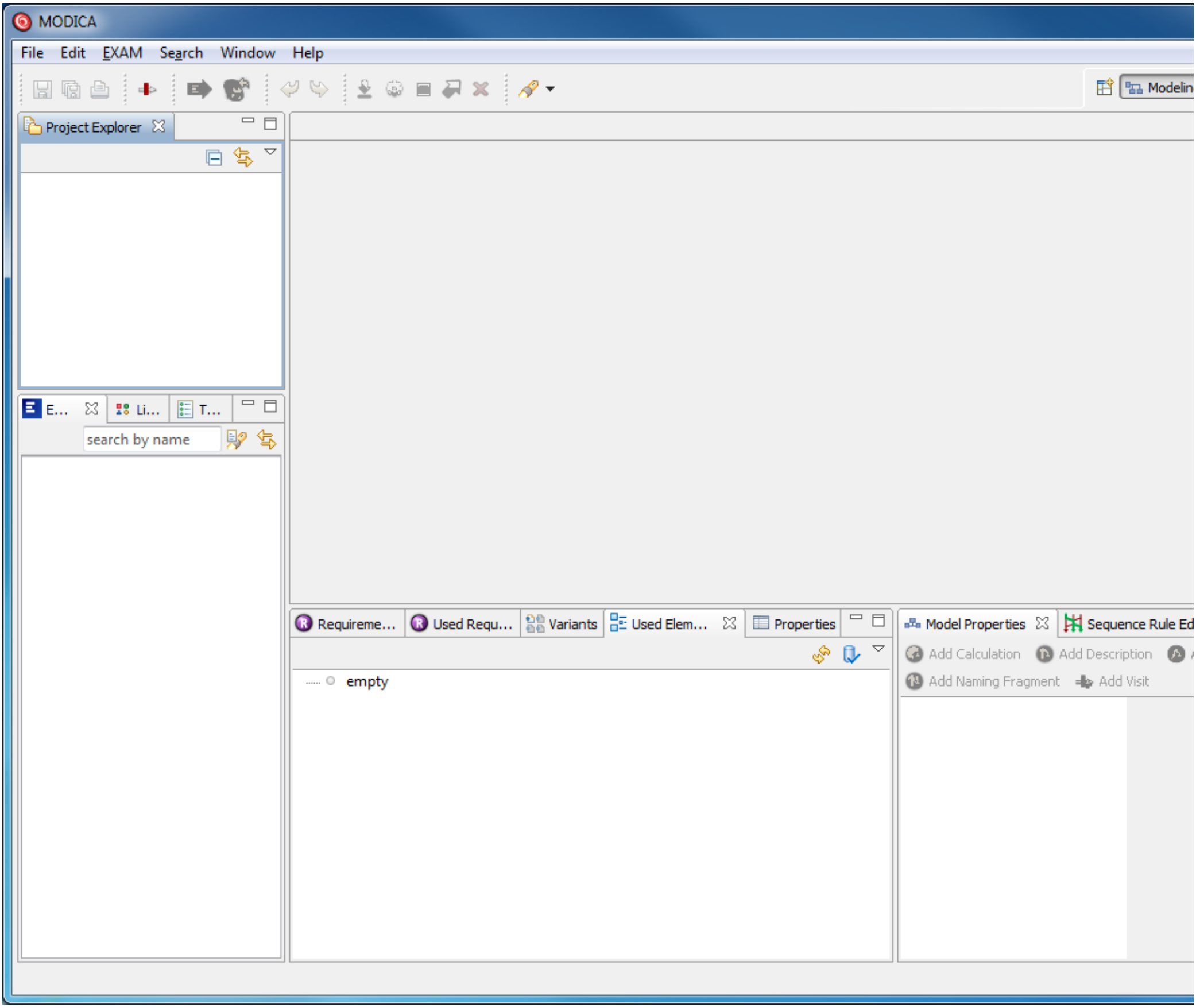
ACC oder auch Adaptive Cruise Control ist ein Abstandsregeltempomat, der automatisch die Fahrzeuggeschwindigkeit anpasst, in Abhängigkeit vom vorausfahrenden Fahrzeug. Die Geschwindigkeit und der Abstand werden mit Hilfe von Sensoren gemessen. Wenn aktiviert, stellt der Fahrer die gewünschte Geschwindigkeit ein, ähnlich wie bei einem Tempomat. Die eingebauten Sensoren erfassen das Verkehrsaufkommen und sorgen dafür, dass der Wagen den definierten Abstand zum vorausfahrenden Fahrzeug einhält. Die maximale Geschwindigkeit des eigenen Fahrzeugs wird vom Fahrer festgelegt. Im vorliegenden Beispiel liegen die Anforderungen in Form einer Systemspezifikation in DOORS vor. Ziel ist, diese Anforderungen z.B. in einer Simulation am HiL-Prüfstand (kurz für Hardware in the Loop, eine Methode zum Testen und Absichern von eingebetteten Systemen) mit Hilfe einer Testautomatisierung zu testen und das korrekte Verhalten nachzuweisen.

ACC17	**5 Break-down**
ACC29	**5.1 When turned off**
ACC18	ACC can be turned on (PowerOn).
ACC28	**5.2 When turned on**
ACC19	ACC can be turned off (PowerOff).
ACC30	**5.2.1 When inactive**
ACC20	ACC can be activiated.
ACC23	On activation, the currently active speed of the vehicle is set as desired speed for the system.
ACC31	**5.2.2 When active**
ACC21	ACC can be deactivated.
ACC32	**5.2.2.1 Speed variation by driver**
ACC22	Desired speed can be increased by 1 kmh pressing the Increase1-Button.
ACC47	The desired speed is increased by 1 kmh.
ACC25	Desired speed can be decreased by 1 kmh pressing the Decrease1-Button.
ACC48	The desired speed is decreased by 1 kmh.
ACC49	Desired speed can remain unchanged.
ACC50	The desired speed remains unchanged.
ACC34	**5.2.2.2 Other vehicles**
ACC35	Vehicle is recognized.
ACC39	When the actual distance between vehicles is lower than the security gap.
ACC51	The actual speed of the vehicle is reduced.
ACC36	Recognized vehicle leaves range.
ACC40	If actual speed is below desired speed, the original desired speed is restored.
ACC52	The car accelerates.
ACC37	Recognized vehicle increases speed (accelerates).
ACC41	When the actual speed is below the desired speed.
ACC53	The vehicle accelerates while maintaining the security gab between vehicles
ACC55	When the actual speed is the desired speed.
ACC56	Do not change speed.
ACC38	Recognized vehicle decreases speed (brakes).
ACC42	When the actual distance between vehicles is lower than the security gap.
ACC54	The actual speed of the vehicle is reduced

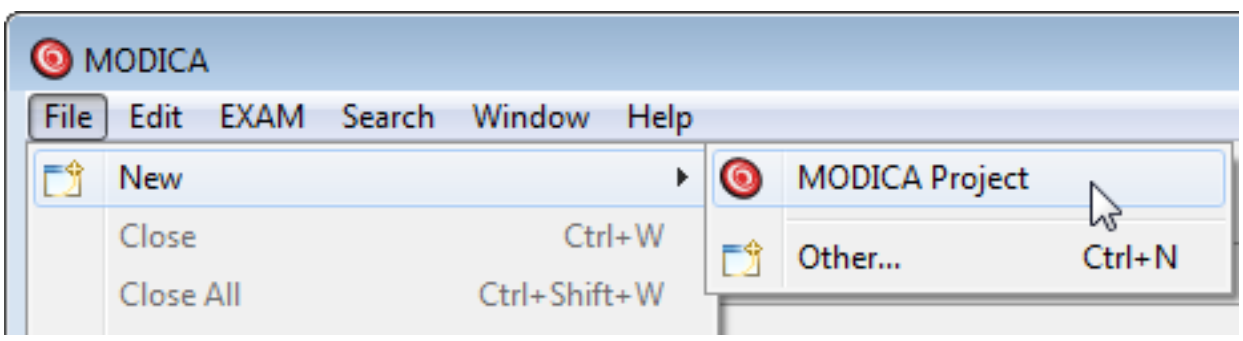
Beispiel-Anforderungen in DOORS

Schritt 1 - Projekt erstellen

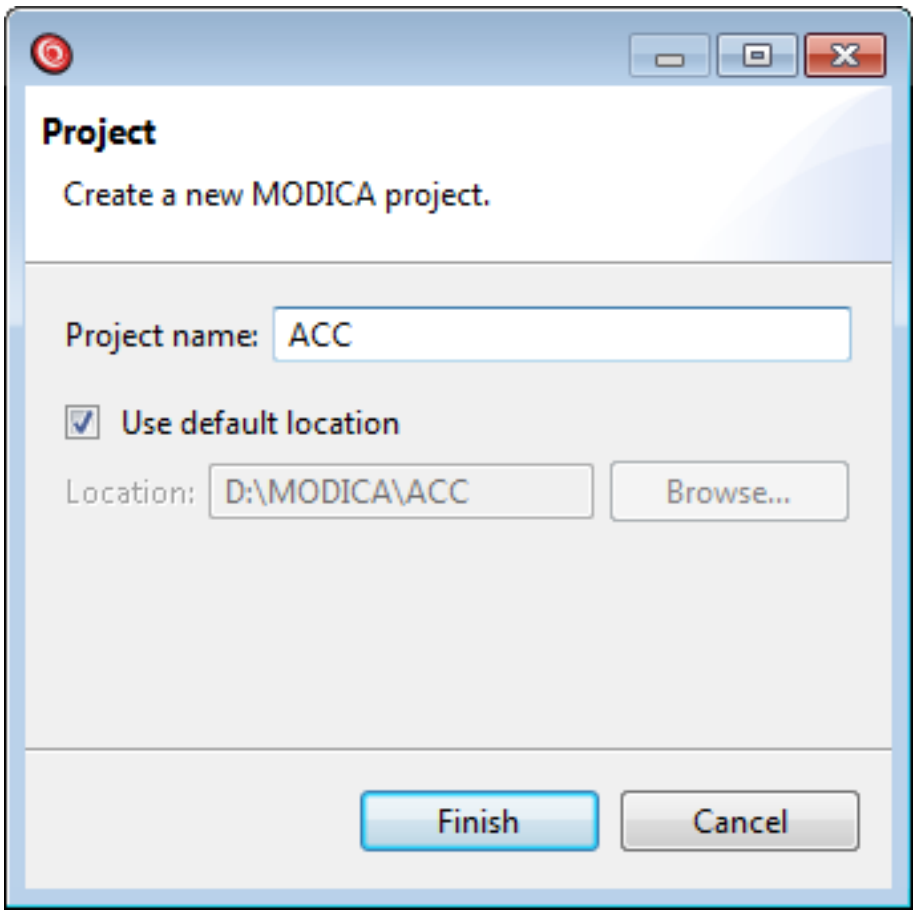
Wenn man **MODICA** zum ersten Mal startet, befindet man sich in der *Modellierungsperspektive*. Diese wird verwendet, um Benutzungsmodelle zu erstellen und zu bearbeiten.



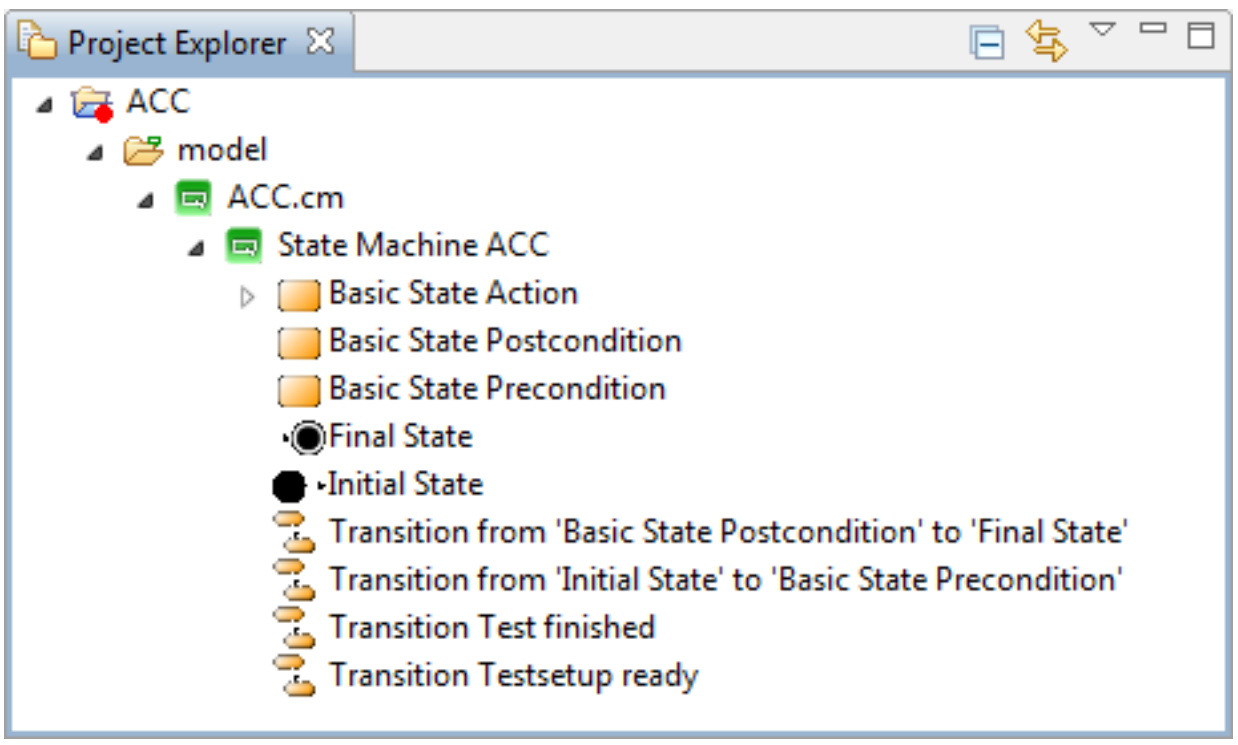
Um mit der Modellerstellung zu beginnen, muss zuerst ein Projekt angelegt werden. Dazu wählt man in der Menüleiste *File → New → MODICA Project*.



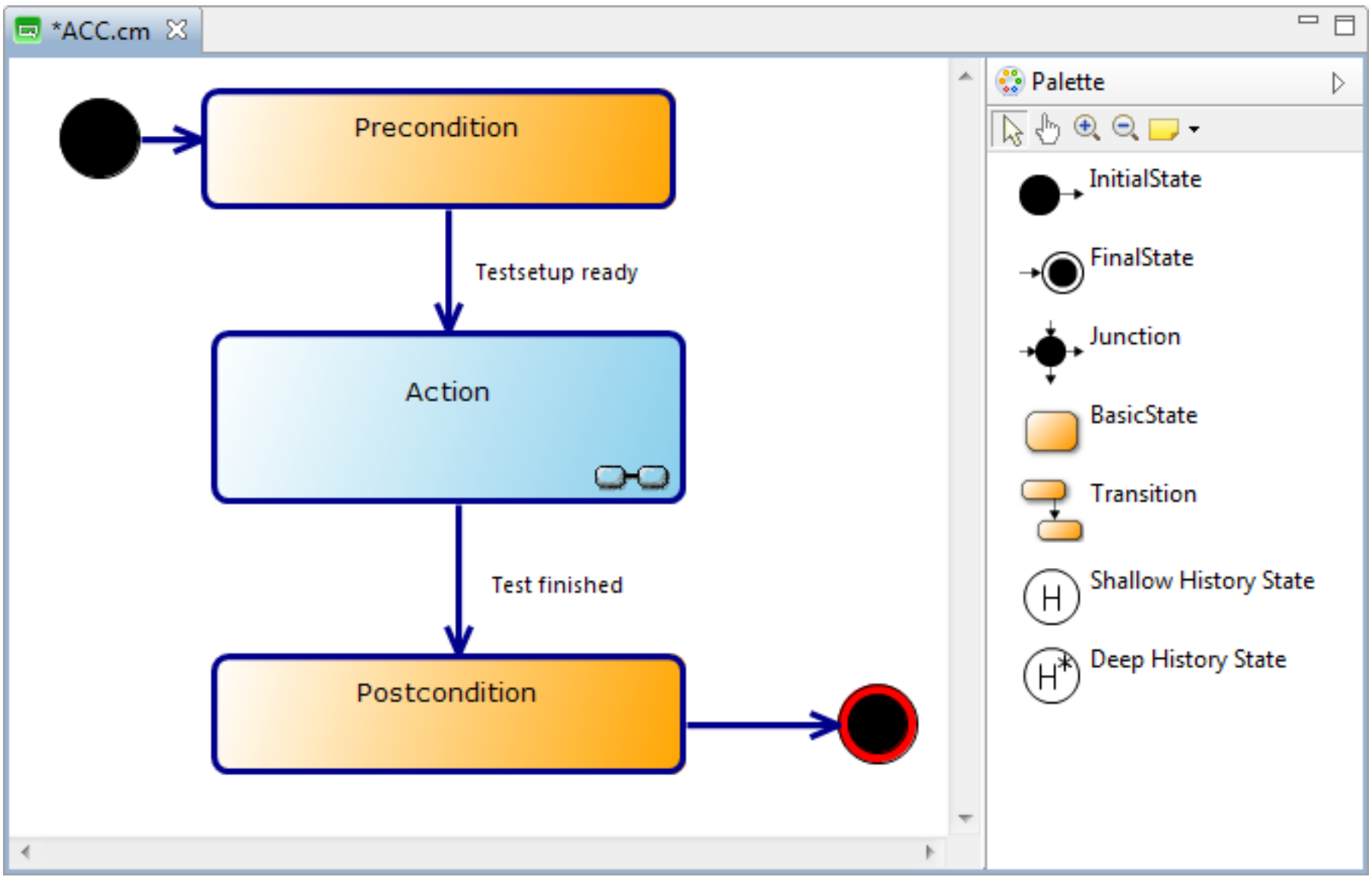
In dem erscheinenden Dialog wird ein Name für das neue Projekt angegeben. Für das Tutorial wird entsprechend des Beispiels ACC verwendet. Mit einem Klick auf *Finish* wird das Projekt erstellt.



Das neue Projekt erscheint im *Project Explorer*. Klappt man die einzelnen Elemente auf, kann man alle enthaltenen Objekte betrachten. Jedes neu erstellte Projekt besteht aus einem Grundmodell, welches den Anfangszustand InitialState, die Standardzustände Precondition, Action und Postcondition und den Endzustand FinalState beinhaltet.



Um das Modell zu bearbeiten, muss es im *Editor* geöffnet werden. Dazu wählt man den Unterpunkt *ACC.cm* aus. Diesen kann man entweder doppelklicken oder über das Kontextmenü den Befehl *Open* verwenden. In dem bisher noch grauen Feld erscheint der Editor, mit Benutzungsmodell und den Modellierungswerkzeugen.



Jedes neu erstellte Projekt beinhaltet ein Grundmodell auf oberster Ebene. Dieses besteht immer aus einem InitialState, der über Precondition, Action und Postcondition zum FinalState führt. Die Precondition dient zur Herstellung und Überprüfung definierter Voraussetzungenfür die Testdurchführung. Der BasicState Action beinhaltet das eigentliche Benutzungsmodell des **SuT**. Als SuT, also "System under Test" bezeichnet man das zu testende System, in diesem Fall das ACC. Die Postcondition enthält die Nachbereitung, sei es das Herunterfahren des Testsystems oder eine Auswertung der Testergebnisse.

Die Strukturelemente in MODICA haben entsprechend der Modellierungsrichtlinien feste Namen, die englisch sind. Dadurch ergeben sich für das Modell englische Bezeichner.

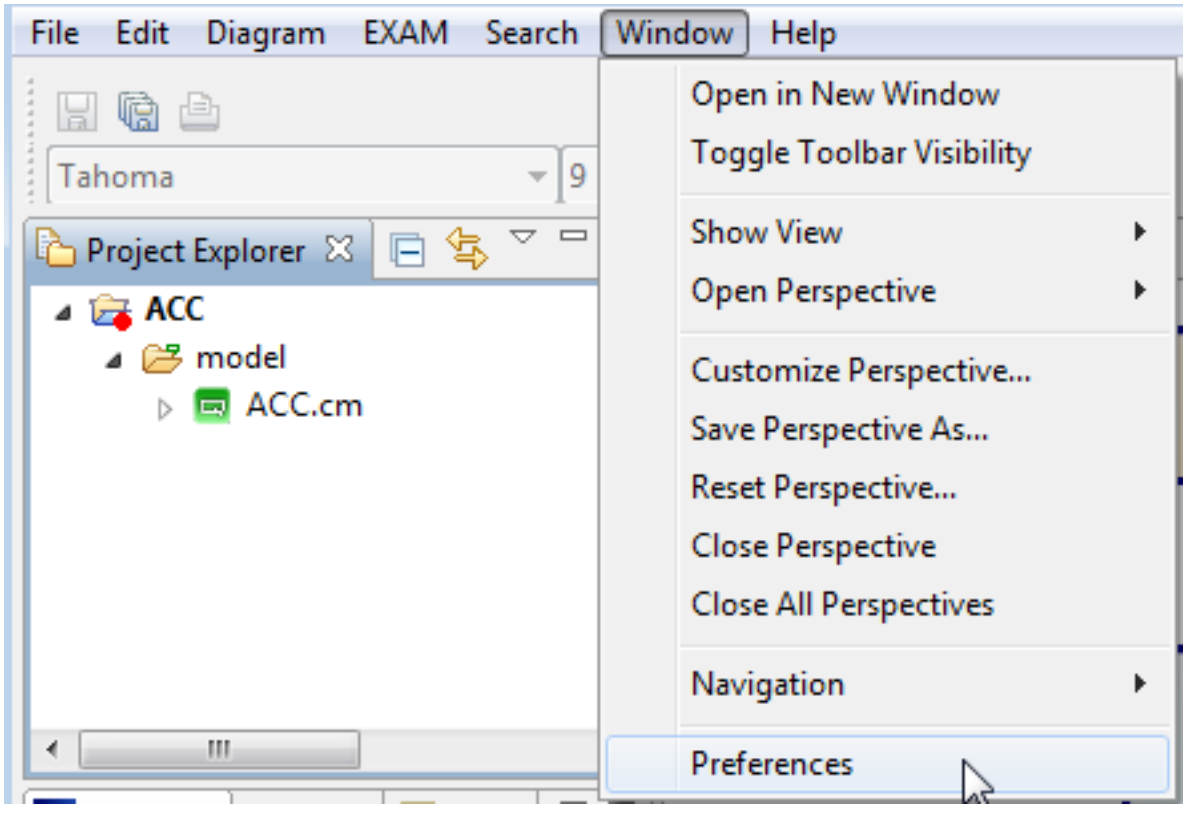
Schritt 2 - Anbindung Requirements (DOORS)

Das folgende Kapitel ist am Beispiel von DOORS geschrieben, kann aber auch analog mit einer Excel Datei durchgeführt werden wenn diese mit ähnlichen Inhalten befüllt ist.

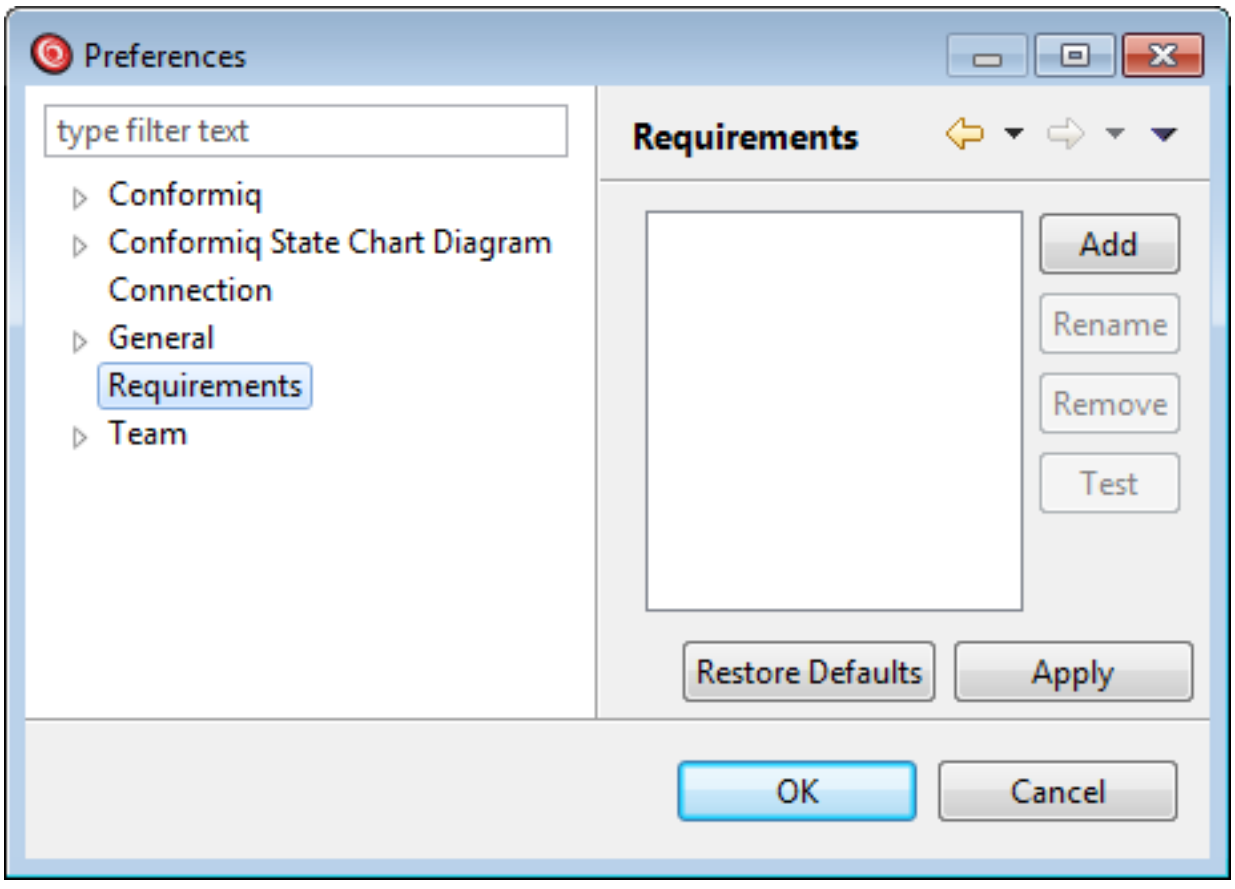
Anforderungen oder auch Requirements bilden die Basis für die Modellierung mit MODICA. In den Anforderungen wird das erwartete Systemverhalten des SuT beschrieben, das später in den Tests geprüft werden soll. Für das Beispiel wird eine Verbindung mit einem DOORS Server hergestellt, die Anforderungen in das Projekt importiert und im nächsten Schritt ein Modell daraus erstellt.

Um eine Verbindung mit DOORS herstellen zu können, muss der DOORS Client auf dem eigenen Rechner installiert sein.

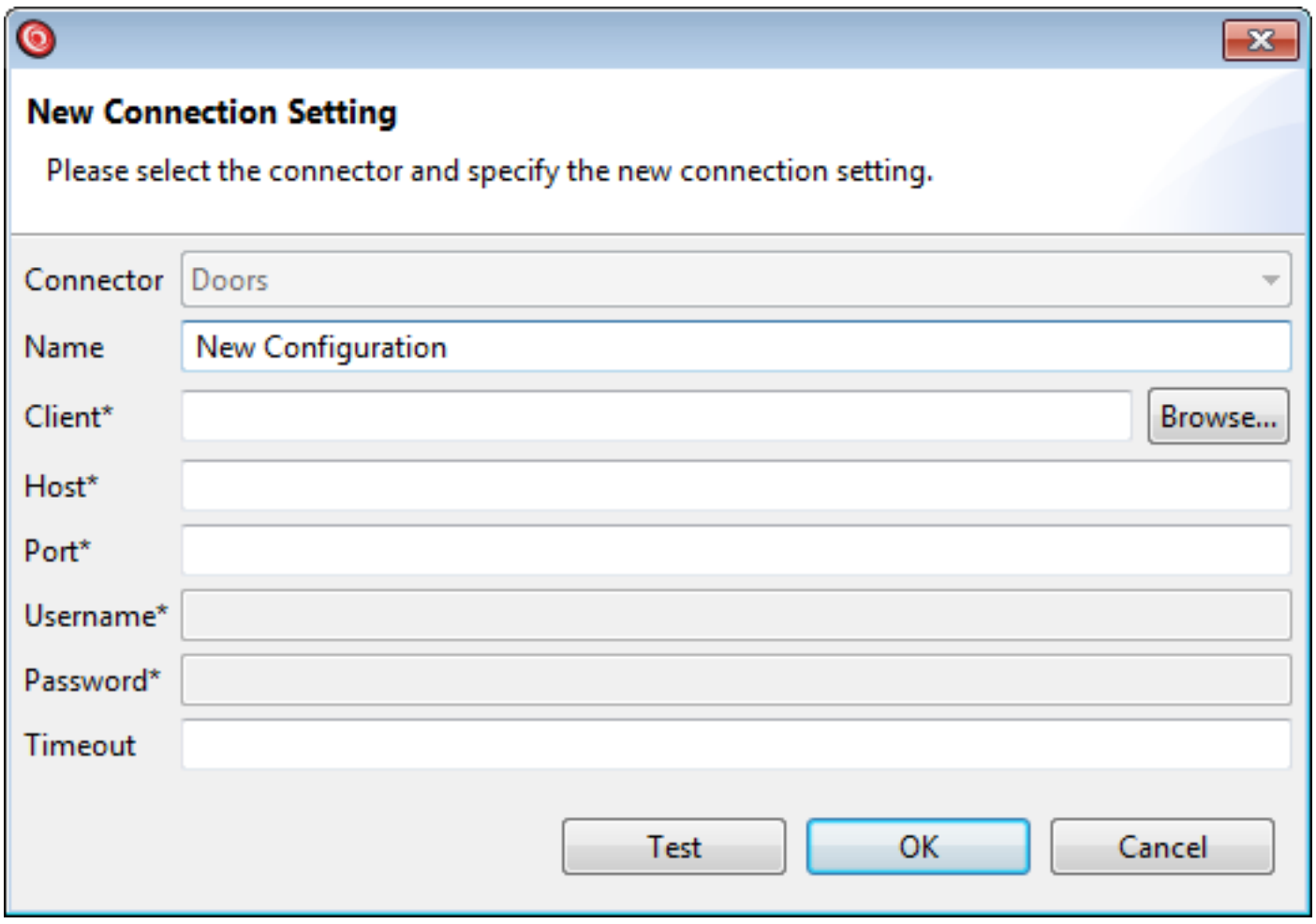
Die Grundeinstellungen zur DOORS Verbindung werden anschließend in den Einstellungen definiert. Dazu ruft man *Window → Preferences* auf.



In dem Dialog sind Einstellungsmöglichkeiten für verschiedene Features zu sehen. Um eine Verbindung zu DOORS herzustellen, klickt man auf *Requirements*. Auf der rechten Seite werden die verfügbaren Verbindungen aufgelistet. Mit einem Klick auf *den Add* Button wird eine neue Verbindung hinzugefügt.

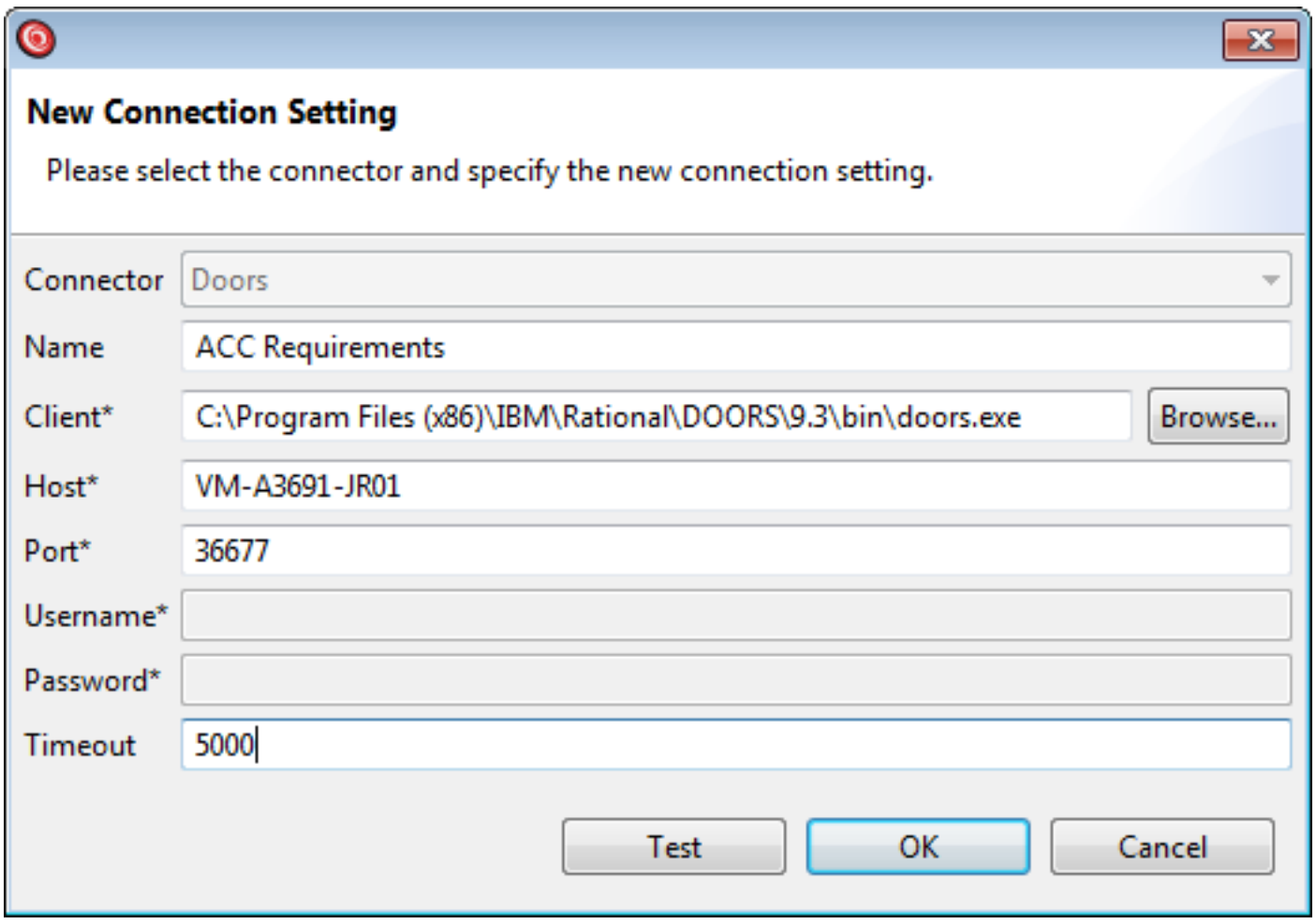


Der neue Dialog enthält noch keinerlei Informationen.



- Name: Man kann den Defaultnamen verwenden. Ansonsten eignet sich immer eine aussagekräftige Bezeichnung wie zum Beispiel ACC_Requirements.
- Client: Hier klickt man auf Browse und wählt den zu verwendenden DOORS Client aus.
- Host: Als Host trägt man den Server ein auf dem sich die DOORS Datenbank befindet.
- Port: Hier trägt man den Port ein, über den auf die DOORS Datenbank zugegriffen wird.
- Username & Passwort: Aus Sicherheitsgründen ist es nicht möglich diese Einstellungen hier schon vorzunehmen. Beides wird erst bei der eigentlichen Verbindung zu DOORS eingegeben.
- Timeout: Hier kann ein Wert in Millisekunden festlegt werden.

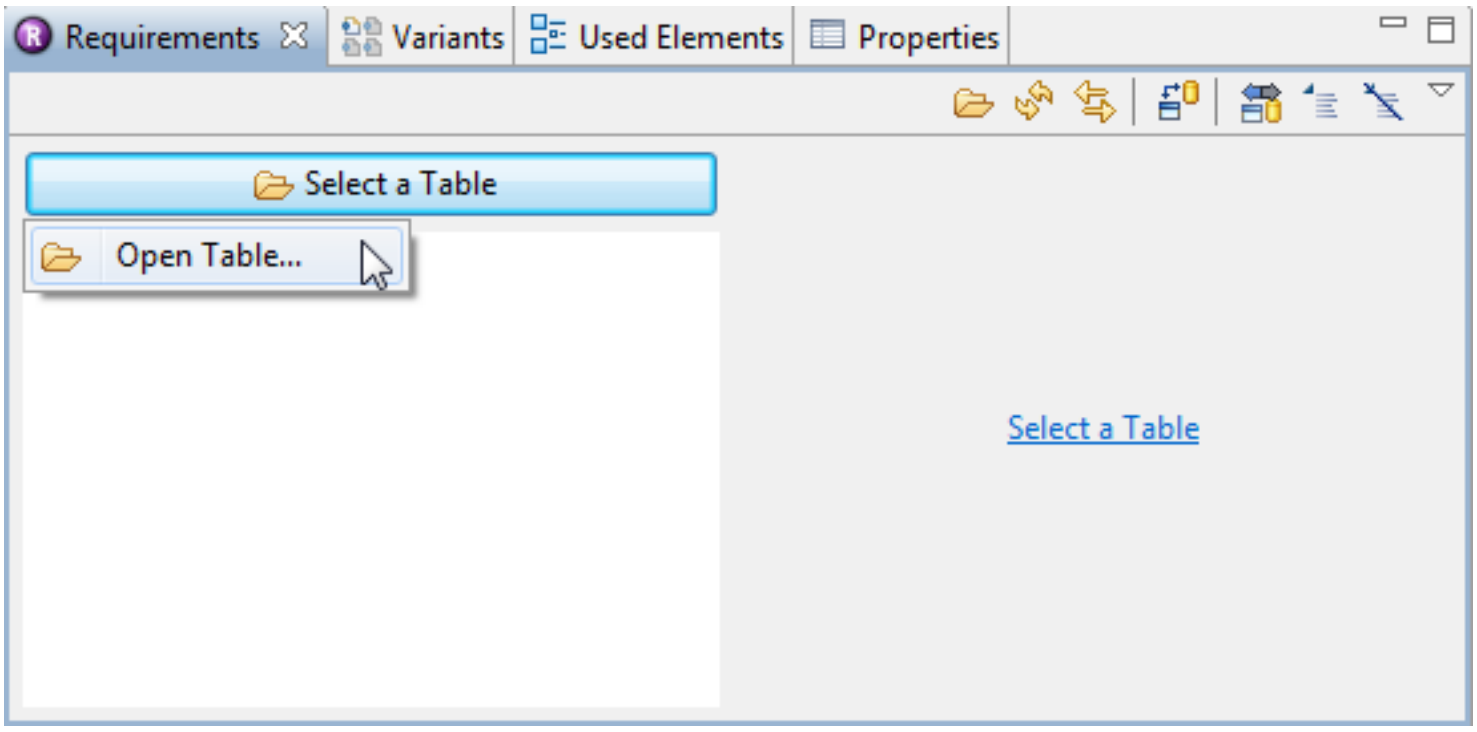
Fertig befüllt sieht es dann so aus:



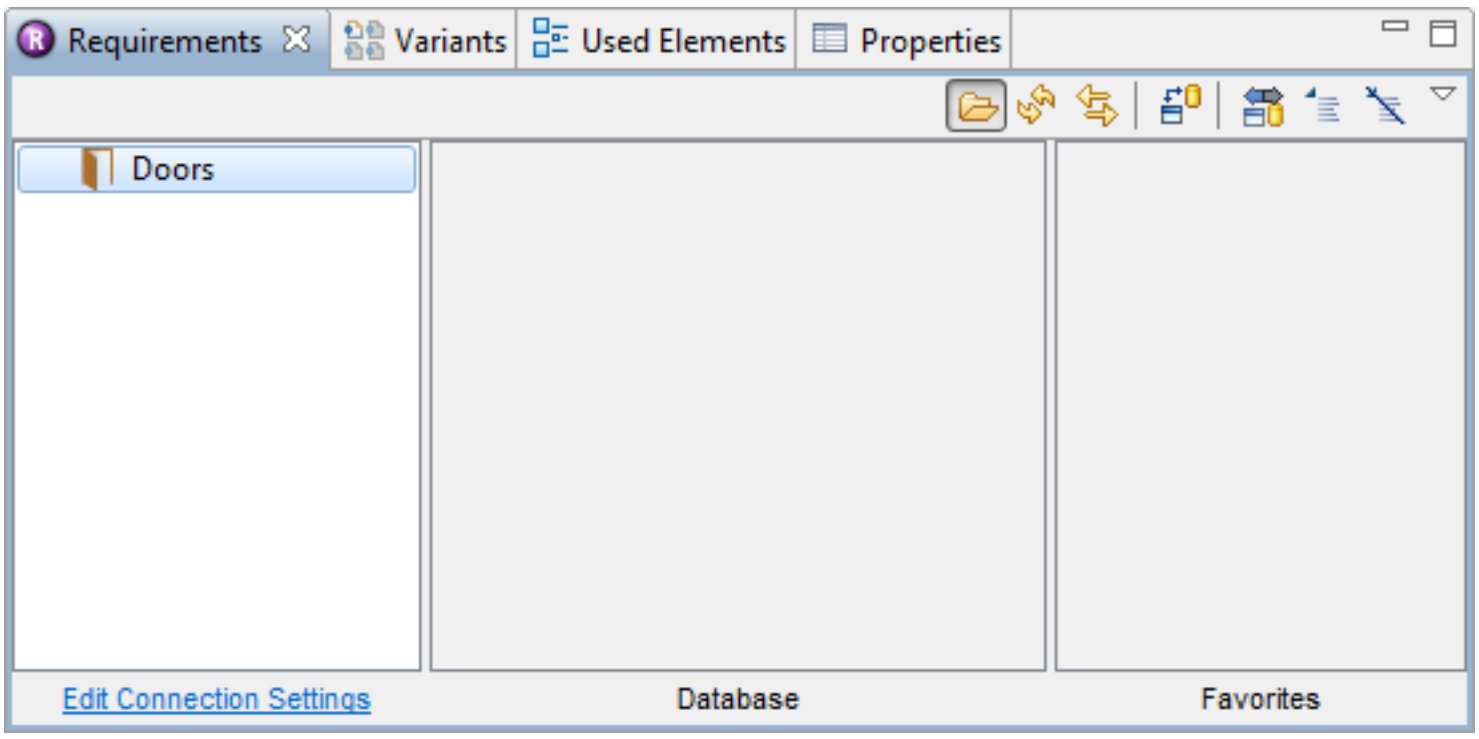
Mit dem Button *Test* kann überprüft werden, ob die Verbindung gültig ist, ansonsten schließt man die Dialoge jeweils

mit *OK*.

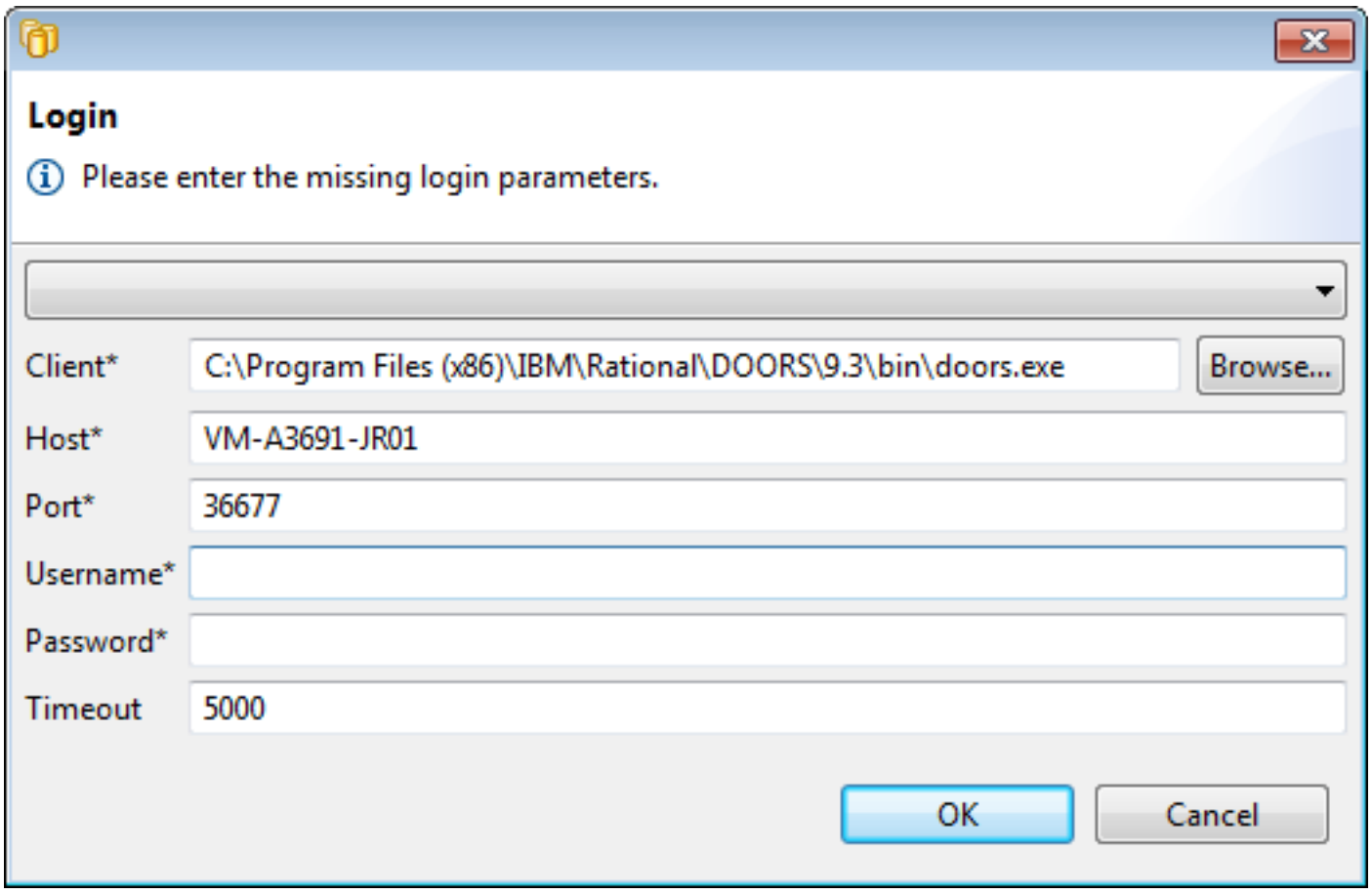
Als Nächstes wechselt man zur *Requirements View*. Dieser befindet sich ebenfalls in der Perspektive *Modeling* unten in der Mitte. Iin der *Requirements View* wird eine Verbindung zur DOORS Datenbank hergestellt und die Anforderungen angezeigt. Um die Verbindung herzustellen, klickt man auf *Open Table*.



Als nächstes kann die Datenbank ausgewählt werden. Sind mehrere Datenbanken vorhanden, erweitert sich die Liste. Mit einem Doppelklick auf Doors öffnet sich die Auswahl bzw. der Login-Dialog.

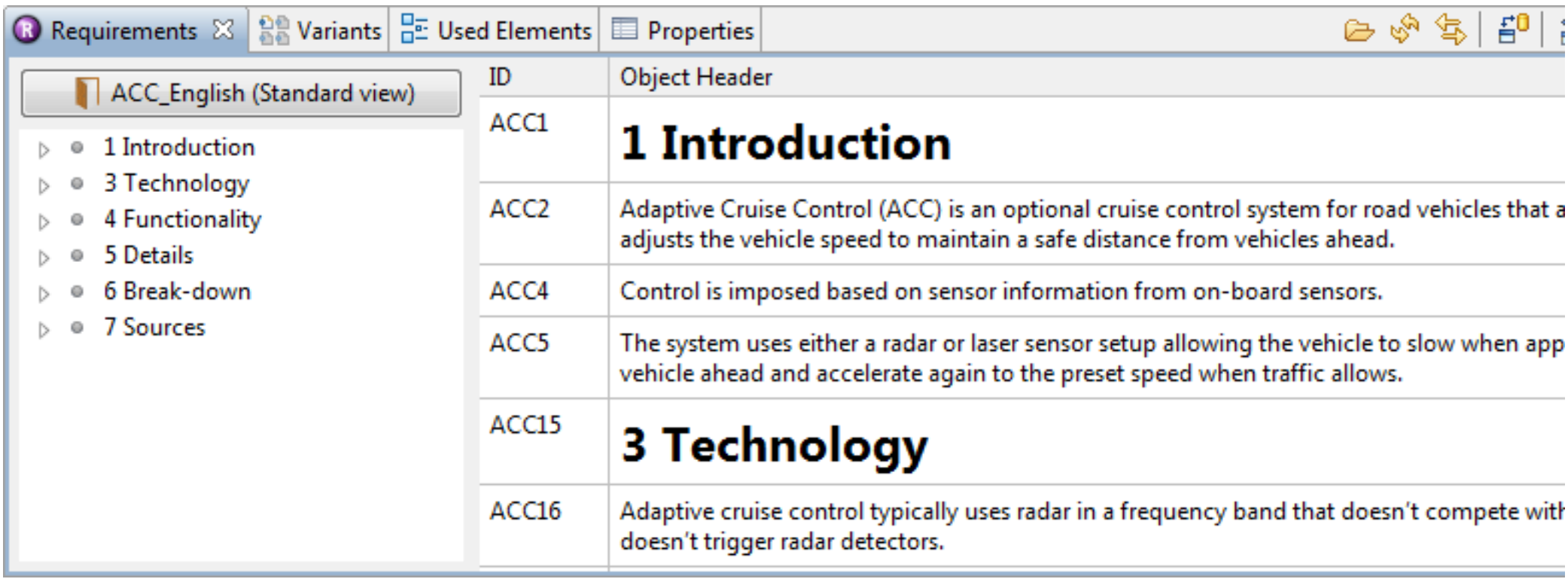


Im nächsten Schritt werden Username und Password befüllt. Mit einem Klick auf *OK* wird die Verbindung hergestellt.



Nach dem Login kann man das Modul auswählen. Man wählt ACC aus und öffnet es durch Doppelklick.

Der Inhalt wird geladen und erscheint in einer Tabelle in der *Requirements* View. Links ist eine Baumstruktur zu sehen, mit der man innerhalb der Anforderungen navigieren kann.



Mit Hilfe der Anforderungen wird nun das Modell entwickelt.

Schritt 3 - Benutzungsmodell erstellen

Anhand der Anforderungen wird ein Benutzungsmodell erstellt, mit dem später Testfälle generiert werden, die auch in eine automatisierte Testausführungsumgebung importiert werden können. Dazu werden die funktionalen Anforderungen nochmal näher betrachtet.

ACC17	**5 Break-down**
ACC29	**5.1 When turned off**
ACC18	ACC can be turned on (PowerOn).
ACC28	**5.2 When turned on**
ACC19	ACC can be turned off (PowerOff).
ACC30	**5.2.1 When inactive**
ACC20	ACC can be activiated.
ACC23	On activation, the currently active speed of the vehicle is set as desired speed for the system.
ACC31	**5.2.2 When active**
ACC21	ACC can be deactivated.
ACC32	**5.2.2.1 Speed variation by driver**
ACC22	Desired speed can be increased by 1 kmh pressing the Increase1-Button.
ACC47	The desired speed is increased by 1 kmh.
ACC25	Desired speed can be decreased by 1 kmh pressing the Decrease1-Button.
ACC48	The desired speed is decreased by 1 kmh.
ACC49	Desired speed can remain unchanged.
ACC50	The desired speed remains unchanged.
ACC34	**5.2.2.2 Other vehicles**
ACC35	Vehicle is recognized.
ACC39	When the actual distance between vehicles is lower than the security gap.
ACC51	The actual speed of the vehicle is reduced.
ACC36	Recognized vehicle leaves range.
ACC40	If actual speed is below desired speed, the original desired speed is restored.
ACC52	The car accelerates.
ACC37	Recognized vehicle increases speed (accelerates).
ACC41	When the actual speed is below the desired speed.
ACC53	The vehicle accelerates while maintaining the security gab between vehicles
ACC55	When the actual speed is the desired speed.
ACC56	Do not change speed.
ACC38	Recognized vehicle decreases speed (brakes).
ACC42	When the actual distance between vehicles is lower than the security gap.
ACC54	The actual speed of the vehicle is reduced

Die Anforderungen, die aus DOORS nach MODICA importiert werden können, müssen ins Modell überführt werden (Verknüpfung an Transitionen und Aktionen). D.h., dass Systemzustände daraus abgeleitet werden müssen, die das gewünschte Verhalten und die Übergänge in einem Zustandsautomaten beschreiben. Oftmals liegen die Anforderungen in DOORS nicht der Struktur entsprechend vor, die für die Modellierung erforderlich ist. Bei der Überführung ins Modell muss diese Strukturierung berücksichtigt werden. Beim abgebildeten Anforderungsmodul sind die Anforderungen bereits sinnvoll in Ebenen eingeteilt: Auf der obersten Ebene (gelb markiert) kann das ACC entweder ein- oder ausgeschaltet werden. In der darunterliegenden Ebene (grün markiert) kann es, wenn es eingeschaltet wurde, auch aktiviert werden. In der untersten Ebene (türkis markiert), wenn es aktiviert ist, kann das System auf Änderungen durch den Fahrer oder das Vorhandensein anderer Fahrzeuge reagieren. Insgesamt muss aus den Anforderungen hervorgehen, wie das System unter welchen Umständen stimuliert werden kann und wie die erwartete Reaktion darauf ist (Test Orakel). Ebenso sollten die im Test zu kombinierenden Abläufe und Werte ersichtlich sein.

In der Praxis kann es vorkommen, dass die Anforderungen Unklarheiten (z.B. Widersprüche) enthalten, die beim

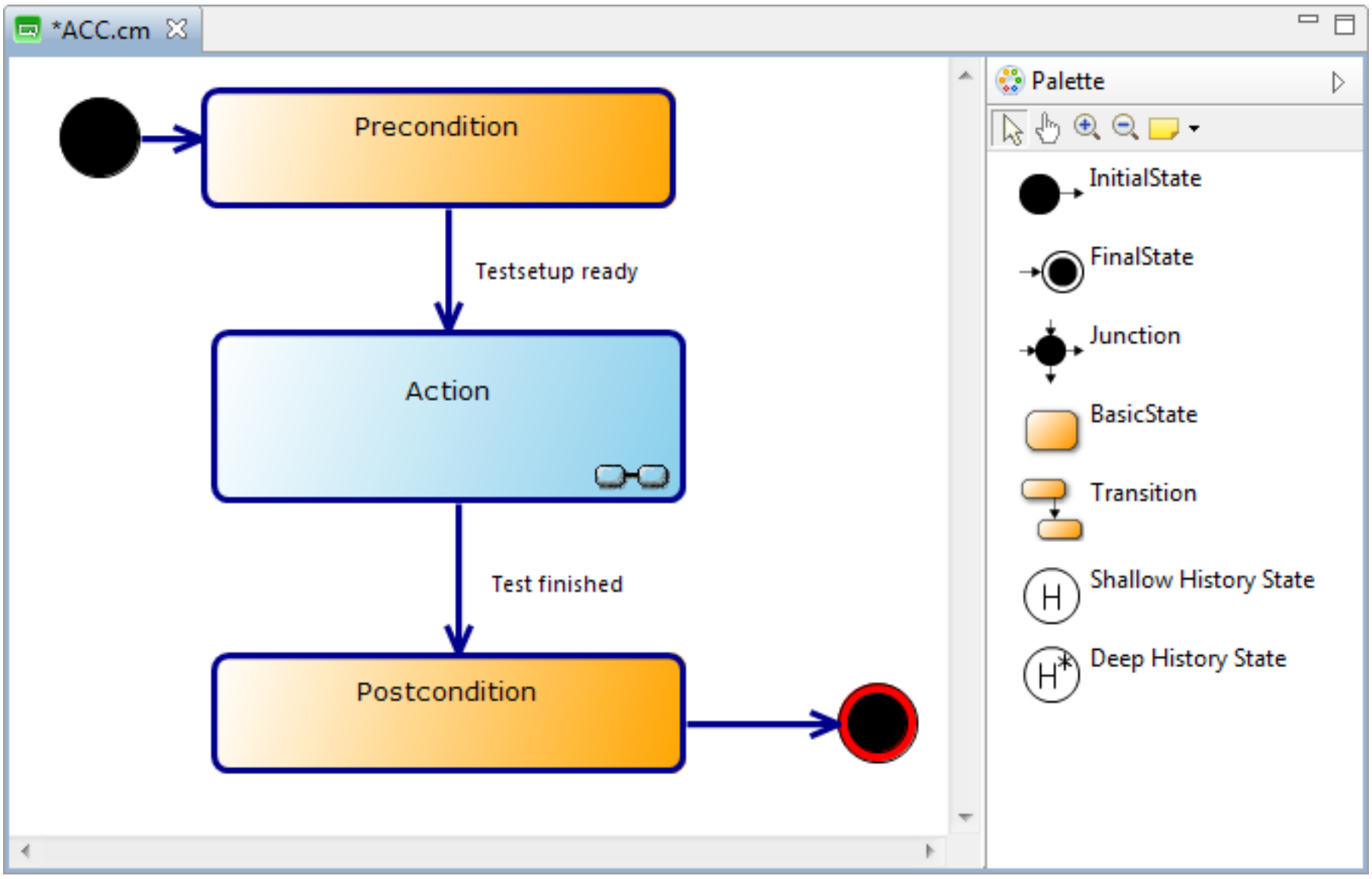
Modellieren aufgedeckt werden. In diesem Fall müssen diese mit den Erstellern der Anforderungen geklärt werden.

Modellierung des Substates "Action"

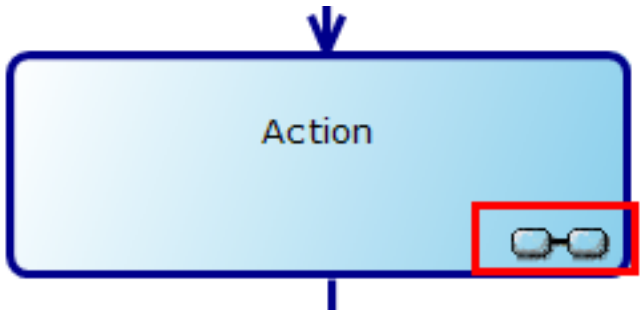
Bei der Erstellung des Benutzungsmodells sind die nach außen unterscheidbaren Zustände des Systems zu modellieren. Grundsätzlich gilt: die möglichen Aktionen in einem Zustand ergeben sich durch seine ausgehenden Transitionen (Bsp: Zunächst ist ein System ausgeschaltet (Zustand aus) und kann eingeschaltet werden). Wenn nach der Ausführung der Aktion andere Aktionen möglich sind, wird dies mit einem neuen Zustand modelliert. Diese Aktion (z.B. Einschalten) wird als Transition zu dem neuen Zustand modelliert.

ACC17	**5 Break-down**
ACC29	**5.1 When turned off**
ACC18	ACC can be turned on (PowerOn).
ACC28	**5.2 When turned on**
ACC19	ACC can be turned off (PowerOff).

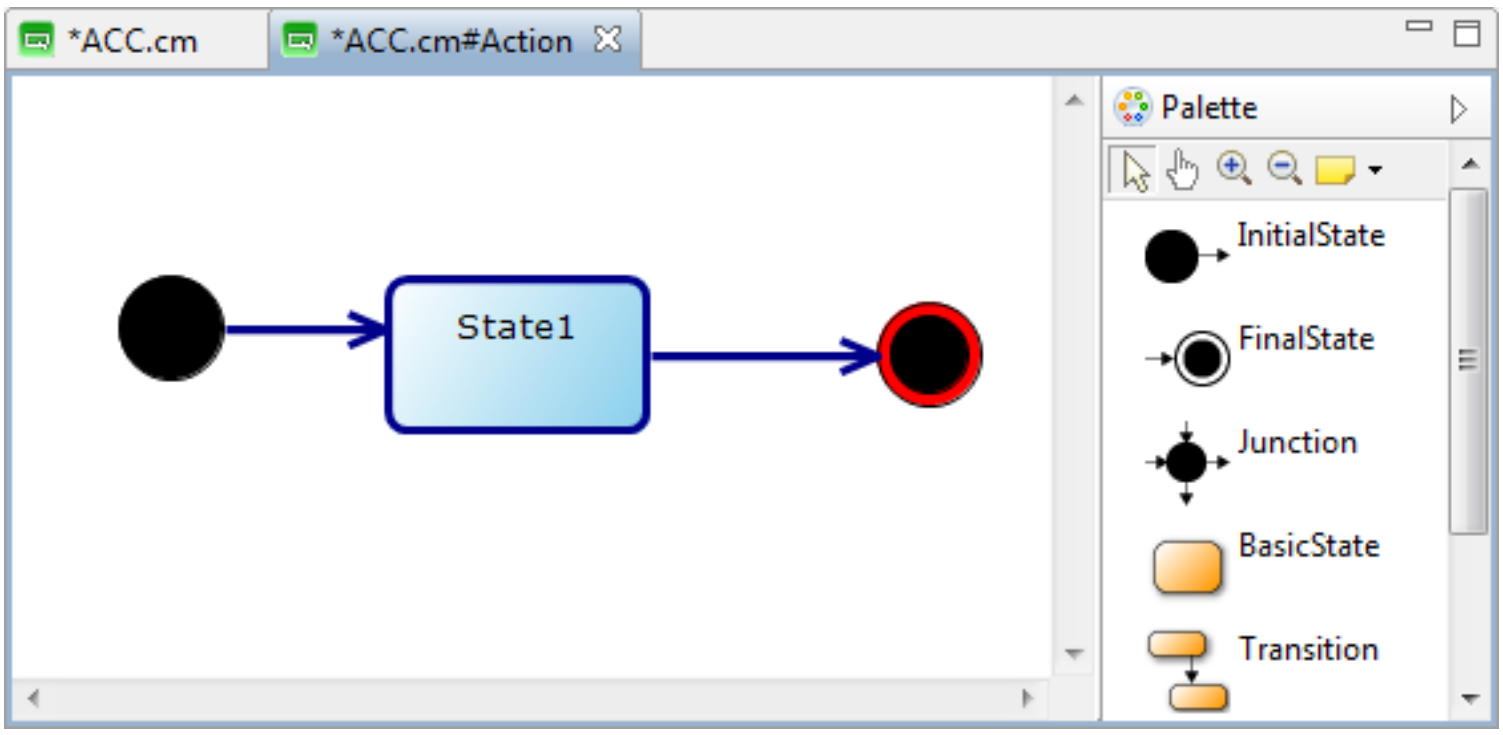
Die abgebildeten Anforderungen beschreiben zwei grundlegende Zustände bzw. Zustandsänderungen. Zum einen, dass das ACC, wenn es ausgeschaltet ist, eingeschaltet werden kann. Und wenn es eingeschaltet ist, auch wieder ausgeschaltet werden kann. Daraus ergeben sich die Zustände Turned Off und Turned On, die innerhalb des States Action modelliert werden.



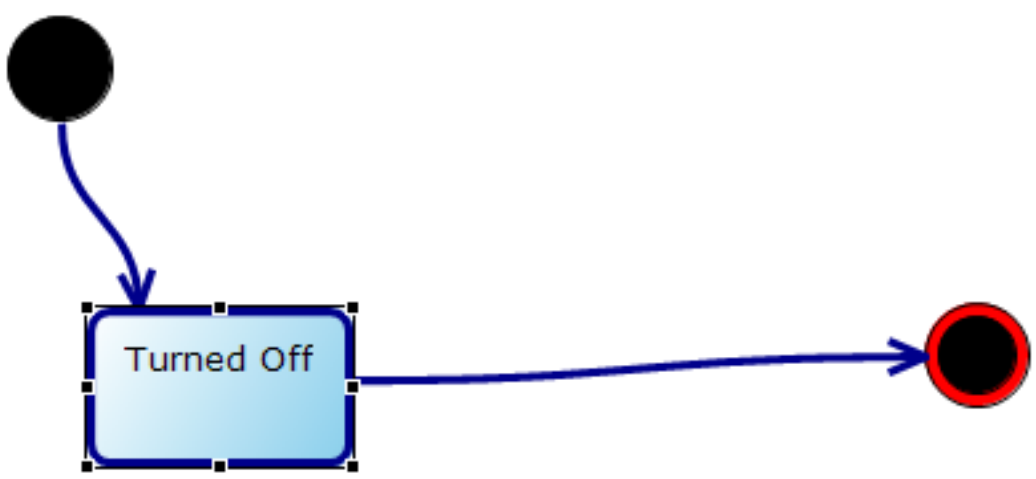
Um Zustände zu modellieren, die innerhalb des States Action liegen, muss dieser geöffnet werden. Dazu wird das Symbol in der unteren, rechten Ecke des States mit Doppelklick ausgewählt.



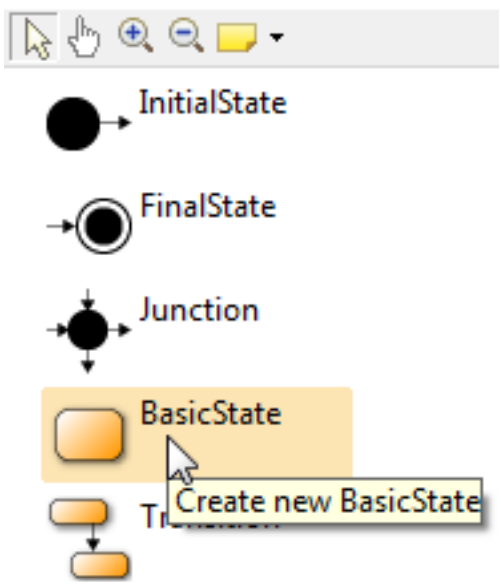
Im Grundmodell ist der *Substate* für den State Action per Default enthalten. Möchte man anderen States weitere Substates hinzufügen, muss man die Modellelemente manuell hinzufügen. Mehr Informationen dazu finden sich im Handbuch unter Model Diagram Editor - Substates.



Entsprechend der Anforderungen werden hier zwei Zustände benötigt. Das heißt, der vorhandene wird umbenannt und ein zweiter wird hinzugefügt. Die Namen der States im Benutzungsmodell sollten den wahrnehmbaren Zustand des SuT möglichst prägnant beschreiben. Das Umbenennen funktioniert entweder durch einen Doppelklick auf das entsprechende Element oder durch Anklicken und `F2` drücken. Der vorhandene State wird Turned Off genannt, weil der Standardzustand immer **Aus** ist (dies ist eine nicht näher spezifizierte Anforderung).

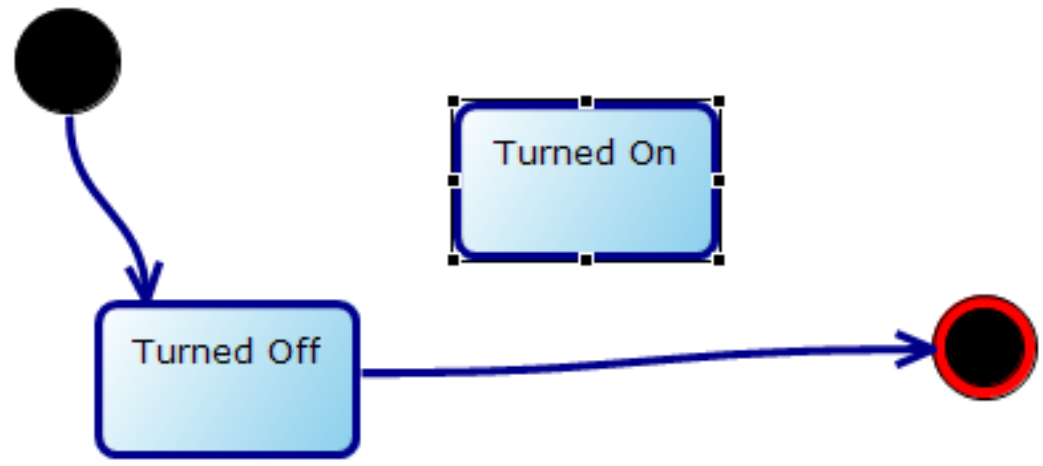


Als nächstes wird ein zweiter State eingefügt. Dazu wird das *Werkzeugmenü* rechts verwendet und *BasicState* ausgewählt.



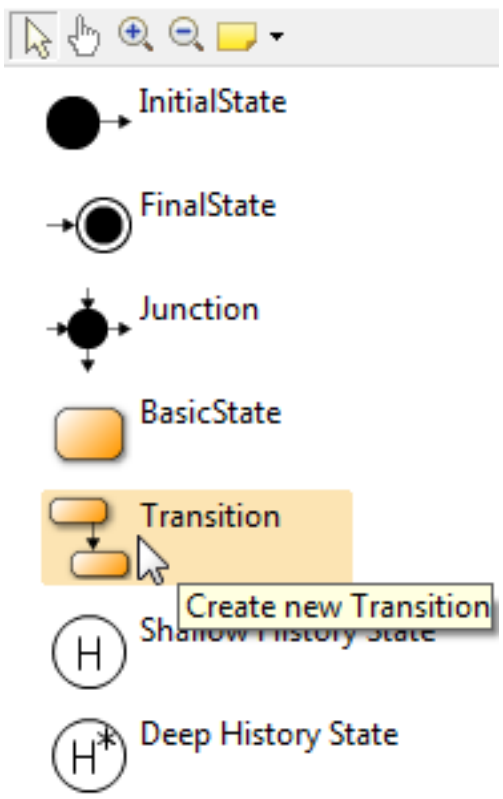
Mit dem ausgewählten Werkzeug wird in den *Editor* geklickt, um ein neuen *State* zu erstellen. Dieser wird in Turned On

umbenannt.

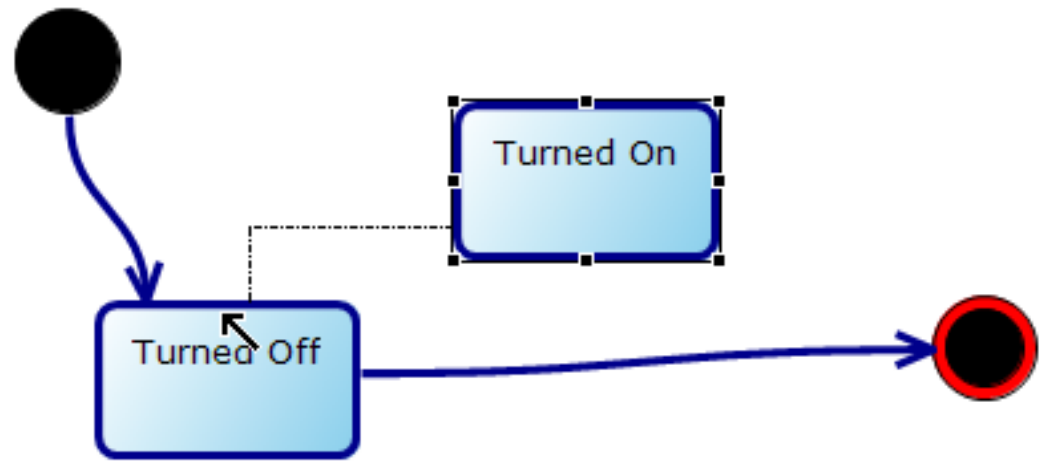


Danach müssen die States miteinander verbunden werden. Aus den Anforderungen ist bekannt, dass das ACC, wenn es ausgeschaltet ist, eingeschaltet werden kann und umgekehrt. Das bedeutet, dass eine Verbindung von Turned Off nach Turned On und von dort wieder zurück hergestellt werden muss. Dafür verwendet man Transitionen. Diese beschreiben Zustandsübergänge.

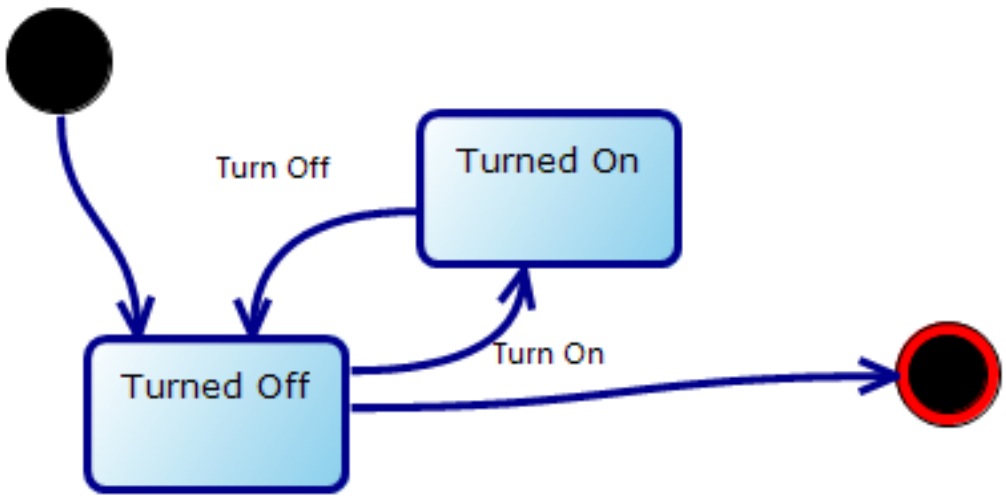
Um eine Transition hinzuzufügen, wird in dem *Werkzeugmenü* die *Transition* ausgewählt.



Um zwei Elemente zu verbinden, wird zuerst das Startelement und danach das Zielelement ausgewählt. Es funktioniert wie Drag&Drop. Das heißt, man klickt den Startzustand an, in diesem Fall Turned On und hält die Maustaste gedrückt und zieht sie zu Turned Off. Die entstehende Verbindung lässt sich gut an der dünn gestrichelten Linie erkennen.



Wenn man die Maustaste dann loslässt, entsteht die *Transition*, die sofort benannt werden kann. Sie wird Turn Off genannt. Dann kommt noch eine zweite *Transition* von Turned Off zu Turned On hinzu. Dafür muss das Werkzeug erneut ausgewählt werden. Die zweite Transition heißt Turn On. Ähnlich wie bei den States sollte die Bennenung der Transitionen ihren Zweck (meist eine Aktion) kurz und prägnant beschreiben.



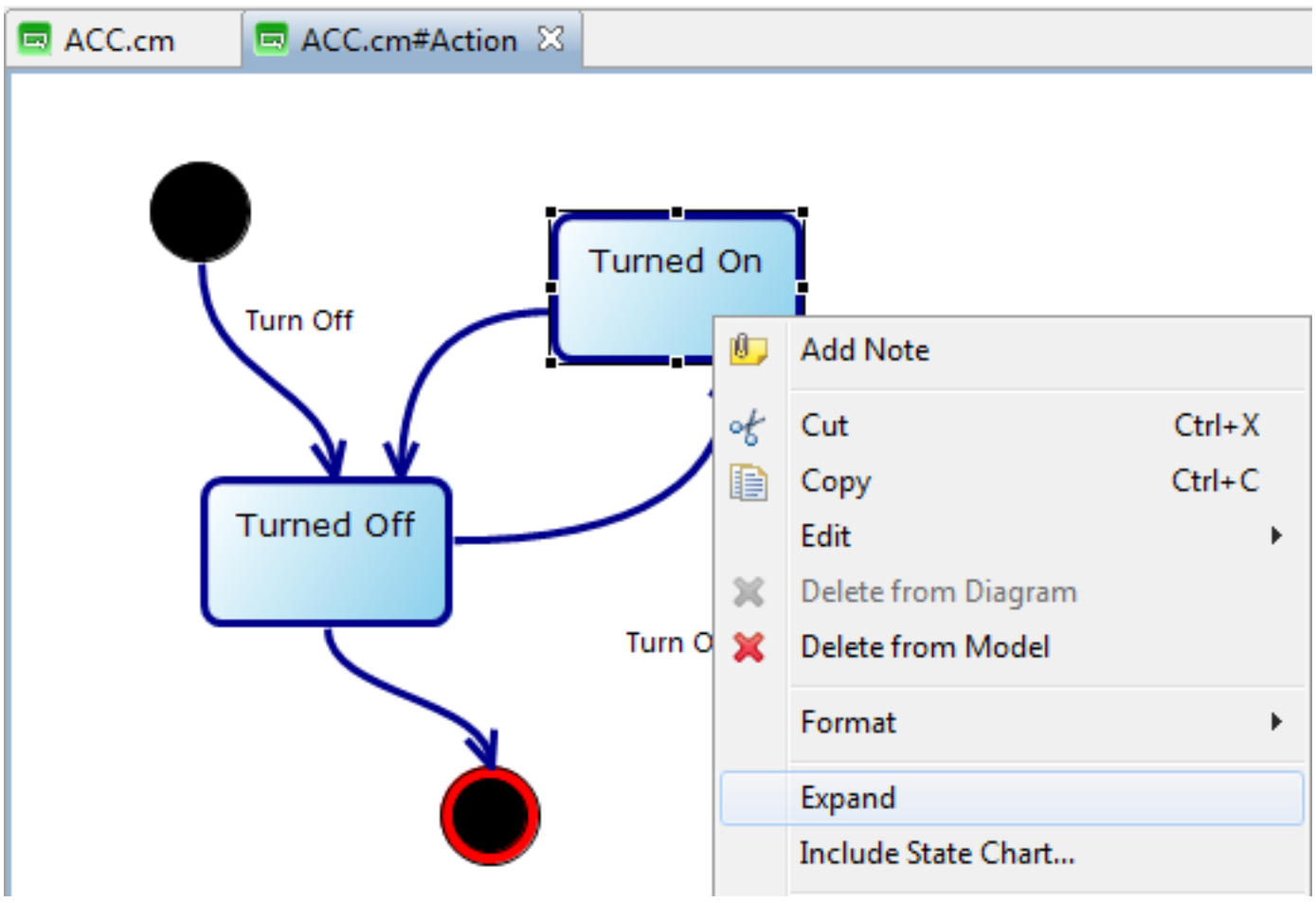
Um das Tutorial überschaubar zu halten, enden alle Testfälle im ausgeschalteten Zustand. Daher führt hier eine *Transition* von Turned Off zum *FinalState*. Damit ist die erste Ebene fertig modelliert. Weiter geht es mit der nächsten Ebene der Anforderungen.

ACC30	**5.2.1 When inactive**
ACC20	ACC can be activiated.
ACC23	On activation, the currently active speed of the vehicle is set as desired speed for the system.
ACC31	**5.2.2 When active**
ACC21	ACC can be deactivated.

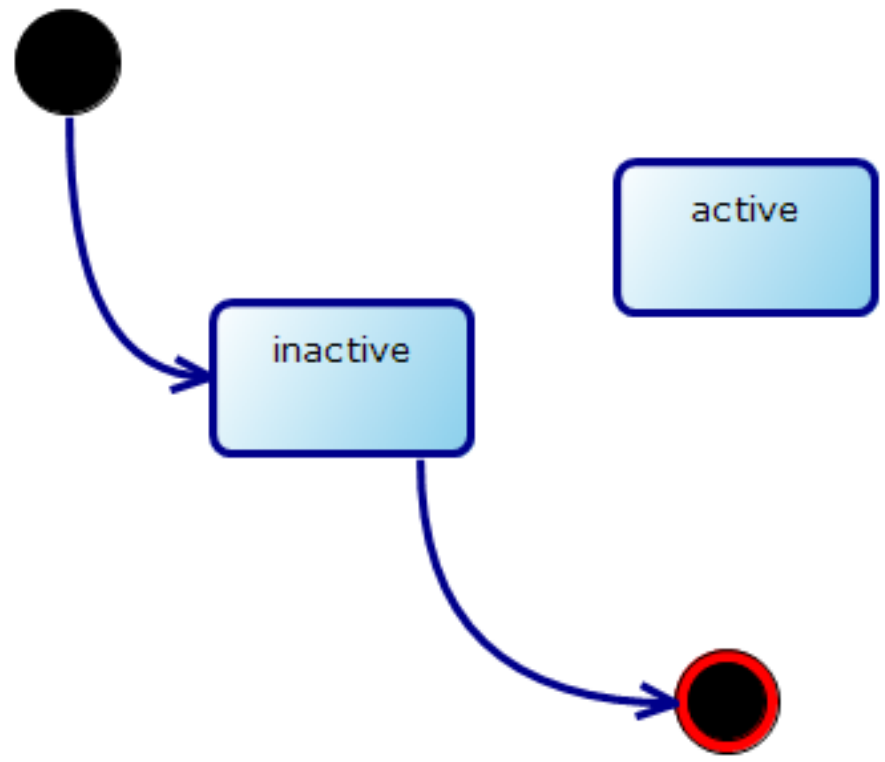
e

Wird das System eingeschaltet, ist es erst einmal inaktiv. Man kann es aktivieren. Wenn es aktiv ist, kann man es deaktivieren.

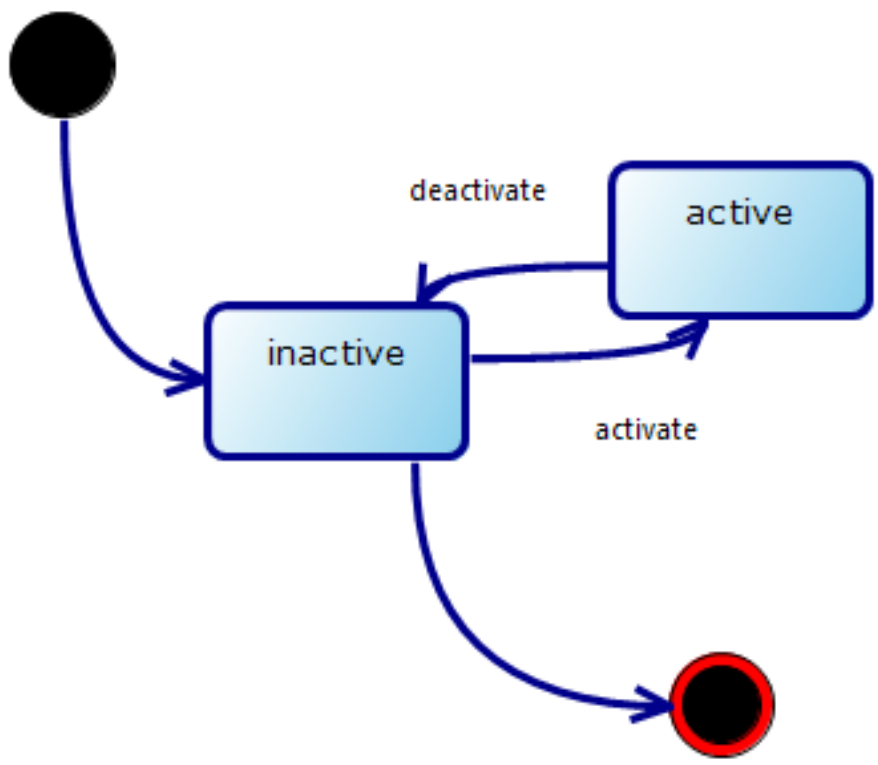
Dazu wird ein *Substate* in den State Turned On hinzugefügt. Dazu klickt man auf Turned On mit Rechtsklick, ruft das Kontextmenü auf und wählt *Expand*.



Der *Substate* öffnet sich gleich. Dieser ist leer und muss mit allen notwendigen States befüllt werden. Neben den beiden States für inactive und active müssen auch *InitialState* und *FinalState* hinzugefügt werden.



Dann werden die Transitionen hinzugefügt. Die erste *Transition* führt von inactive zu active und heißt activate. Die zweite *Transition* führt von active zurück zu inactive und heißt deactivate.



Um das Tutorial überschaubar zu halten, enden alle Testfälle im ausgeschalteten Zustand. Daher führt hier eine *Transition* von inactive zum *FinalState*. Damit ist dieser Teil abgeschlossen. Der dritte Anforderungsblock ist etwas größer und bedarf mehr Erläuterung.

ACC31	**5.2.2 When active**
ACC21	ACC can be deactivated.
ACC32	**5.2.2.1 Speed variation by driver**
ACC22	Desired speed can be increased by 1 kmh pressing the Increase1-Button.
ACC47	The desired speed is increased by 1 kmh.
ACC25	Desired speed can be decreased by 1 kmh pressing the Decrease1-Button.
ACC48	The desired speed is decreased by 1 kmh.
ACC49	Desired speed can remain unchanged.
ACC50	The desired speed remains unchanged.
ACC34	**5.2.2.2 Other vehicles**
ACC35	Vehicle is recognized.
ACC39	When the actual distance between vehicles is lower than the security gap.
ACC51	The actual speed of the vehicle is reduced.
ACC36	Recognized vehicle leaves range.
ACC40	If actual speed is below desired speed, the original desired speed is restored.
ACC52	The car accelerates.
ACC37	Recognized vehicle increases speed (accelerates).
ACC41	When the actual speed is below the desired speed.
ACC53	The vehicle accelerates while maintaining the security gab between vehicles
ACC55	When the actual speed is the desired speed.
ACC56	Do not change speed.
ACC38	Recognized vehicle decreases speed (brakes).
ACC42	When the actual distance between vehicles is lower than the security gap.
ACC54	The actual speed of the vehicle is reduced

Es gibt hier zwei große Blöcke. Der erste befasst sich mit den vom Fahrer gesetzten Geschwindigkeitseinstellungen. Der Zweite befasst sich mit der Erkennung anderer Fahrzeuge durch das ACC. Dabei gibt es zwei verschiedene Arten von Anforderungen. Die eine legt fest, was passieren kann, zum Beispiel kann die Geschwindigkeit um 1 km/h erhöht werden, indem der entsprechende Knopf gedrückt wird oder es wird ein vorausfahrendes Fahrzeug erkannt. Die andere definiert, wie das ACC mit diesem Input umgehen, also wie es darauf reagieren soll.

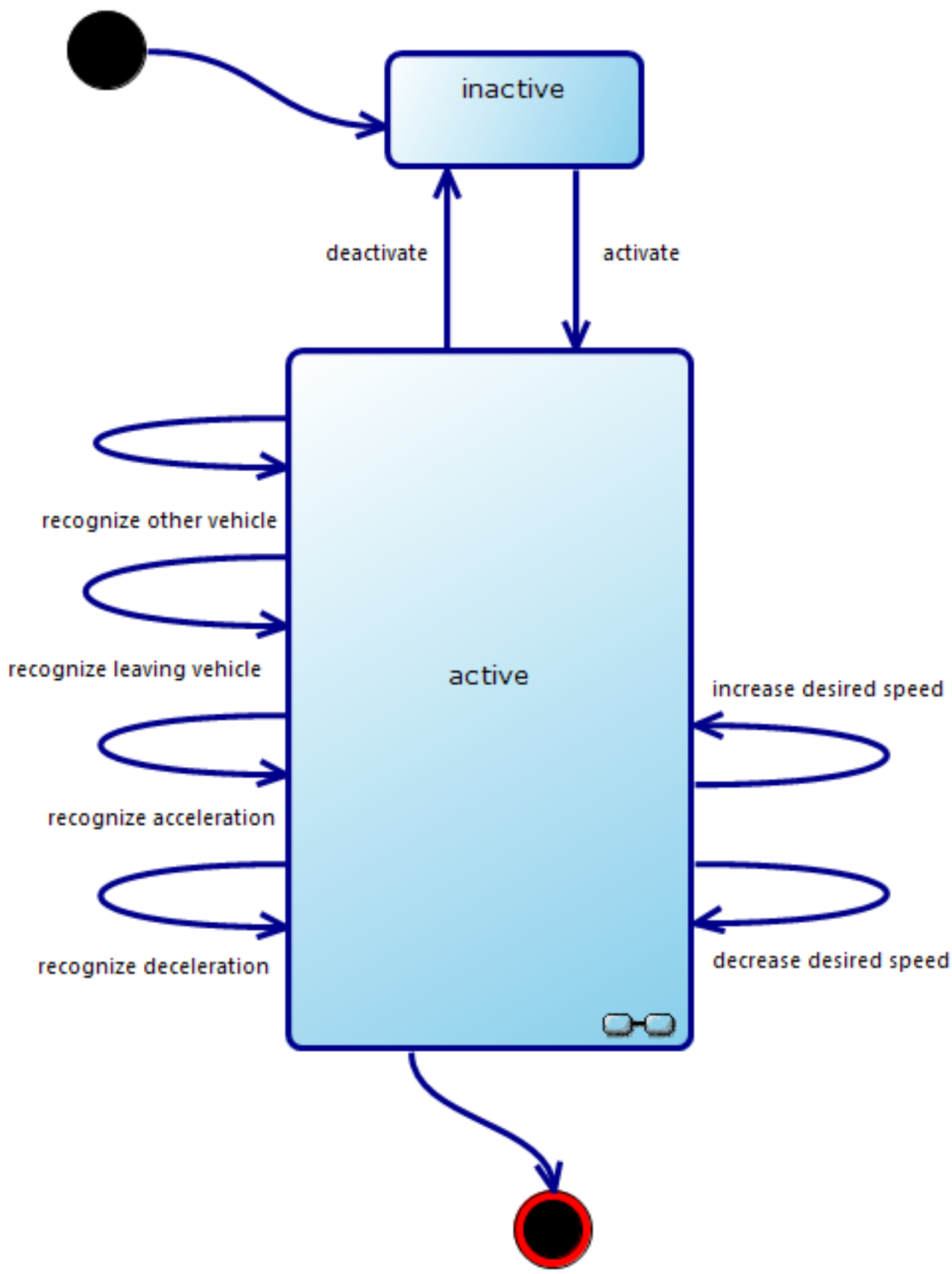
Da zuerst etwas passieren muss, zum Beispiel eine Geschwindigkeitsänderung oder das Erscheinen eines Fahrzeuges und erst dann das System darauf reagieren kann, werden diese Ereignisse mit in den Substate Turned On modelliert.

Zuerst werden die möglichen Zustandsübergänge betrachtet:

- Die gewünschte Geschwindigkeit kann manuell erhöht werden.
- Die gewünschte Geschwindigkeit kann manuell gesenkt werden.
- Der Sensor erkennt ein anderes Fahrzeug.
- Ein anderes Fahrzeug verlässt den Sensorbereich.
- Der Sensor bemerkt eine Geschwindigkeitserhöhung des vorausfahrenden Fahrzeugs.
- Der Sensor bemerkt, dass das vorausfahrende Fahrzeug langsamer wird.

Diese sechs Anforderungen werden nun als Transitionen mit in den aktuellen *Substate* eingetragen. Es ist möglich Schleifen zu generieren, bei denen Start- und Zielpunkt der *Transition* gleich ist. Sie verweisen dabei auf den State active, da während der Ereignisse, die vom ACC bemerkt werden, das ACC eingeschaltet ist und aktiv bleibt.

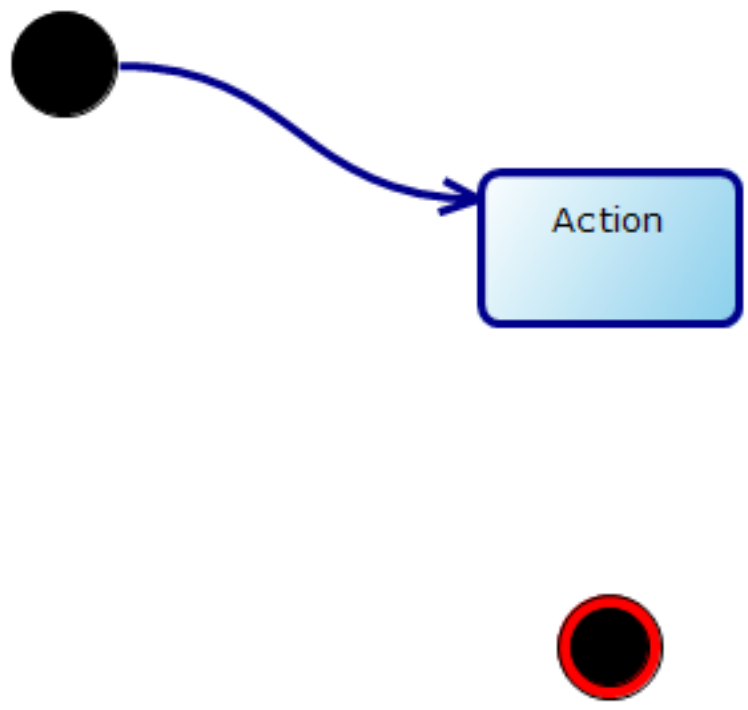
Mit den sechs zusätzlichen Transitionen sieht das Modell an dieser Stelle dann so aus:



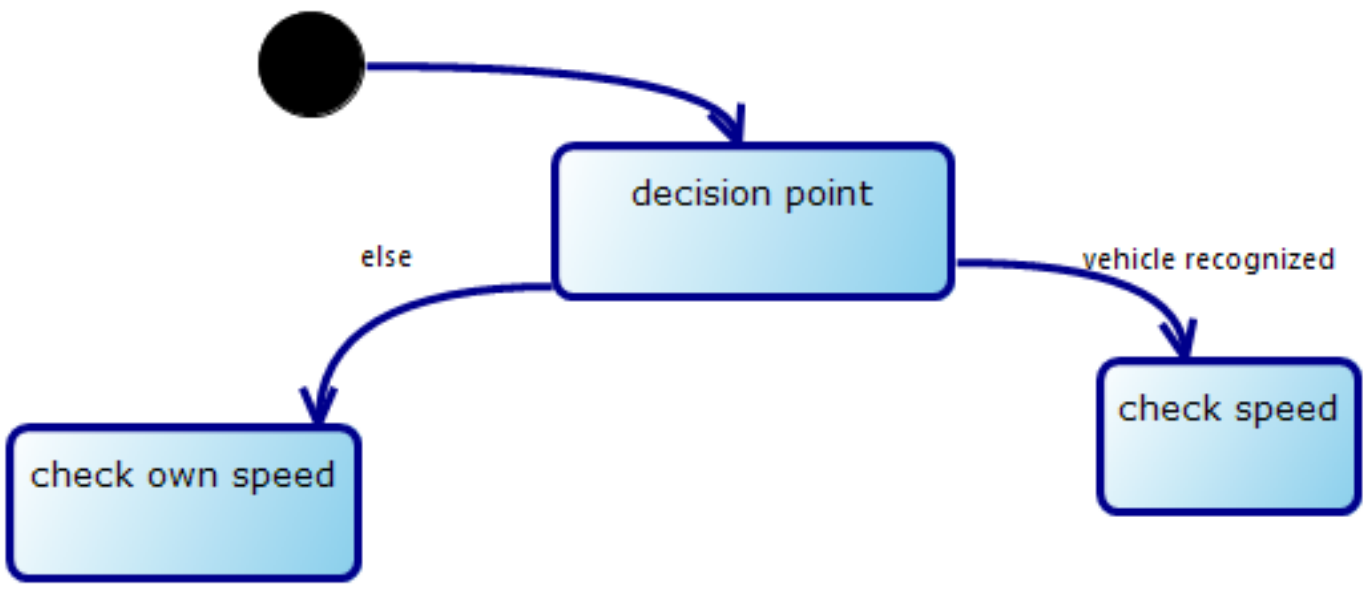
Wenn nach der Ausführung einer Aktion der Zustand des Systems sich nur geringfügig ändert und die möglichen Aktionen sich kaum ändern, ist dieses als Variable und nicht als Zustand modellierbar (z.B. Geschwindigkeitserhöhung), wenn es der Übersichtlichkeit und Wartbarkeit dient. Eine Variable ist insbesondere dann vorzuziehen, wenn es keine spezifischen Anforderungen für einzelne Werte gibt.

In diesem Fall werden daher die Anwesenheit eines anderen Fahrzeugs und die verschiedenen Sollgeschwindigkeiten nicht als separate Zustände, sondern im Nachgang als Variablen modelliert.

Die erwarteten Reaktionen des ACC, also das **Testorakel**, werden nun in einem weiteren *Substate* modelliert. Dazu fügt man einen *Substate* zum State active hinzu. Dazu wählt man den *State* aus, ruft das Kontextmenü mit Rechtsklick auf und wählt *Expand*. Der *Substate* ist leer. Man fügt InitialState, FinalState und einen BasicState aus dem *Werkzeugmenü* hinzu. Da vom Startzustand aus nur eine *Transition* ausgehen darf, verbindet man den Startzustand mit dem ersten *State*.

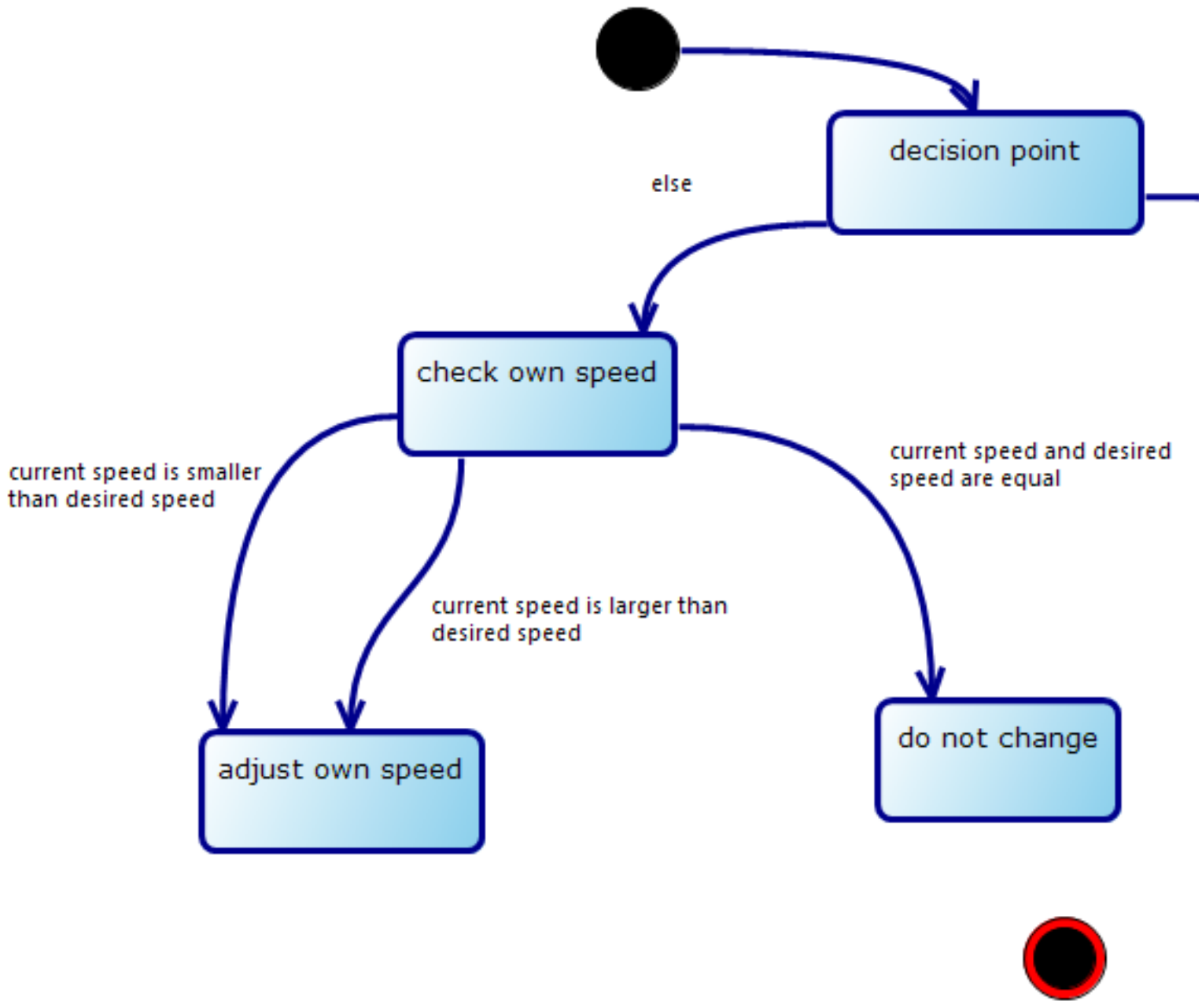


Dieser erste *State* ist ein Entscheidungspunkt, in diesen Zustand passiert selbst nichts, aber je nachdem, welche *Transition* in der Ebene darüber gewählt wurde, führt der Pfad entweder zur Behandlung der eigenen Geschwindigkeitseinstellungen oder zur Verarbeitung der Sensorinformationen bei vorausfahrenden Fahrzeugen. Damit werden zwei Transitionen benötigt und auch zwei States. Der State Action wird in decision point umbenannt und die Transitionen verbindet man mit den States wie im ersten Schritt. Eine für den Fall, dass ein Fahrzeug erkannt wurde und eine falls der Fahrer eine Geschwindigkeitsänderung eingestellt hat.



Im Pfad else werden die manuellen Änderungen an der gewünschten Geschwindigkeit geprüft. Es wird erwartet, dass das ACC je nachdem, ob die aktuelle Geschwindigkeit über oder unter der gewünschten Geschwindigkeit liegt, richtig reagiert und die Geschwindigkeit anpasst. Ist die Geschwindigkeit gleich, soll nichts passieren.

Somit gibt es drei Transitionen die von dem State check own speed ausgehen. Die zwei, welche die höhere oder niedrigere Geschwindigkeit beschreiben, führen zu einem weiteren *State*, der die Anpassung der Geschwindigkeit beschreibt. Die dritte Transition führt zu einem *State*, in dem keine Änderungen passieren.

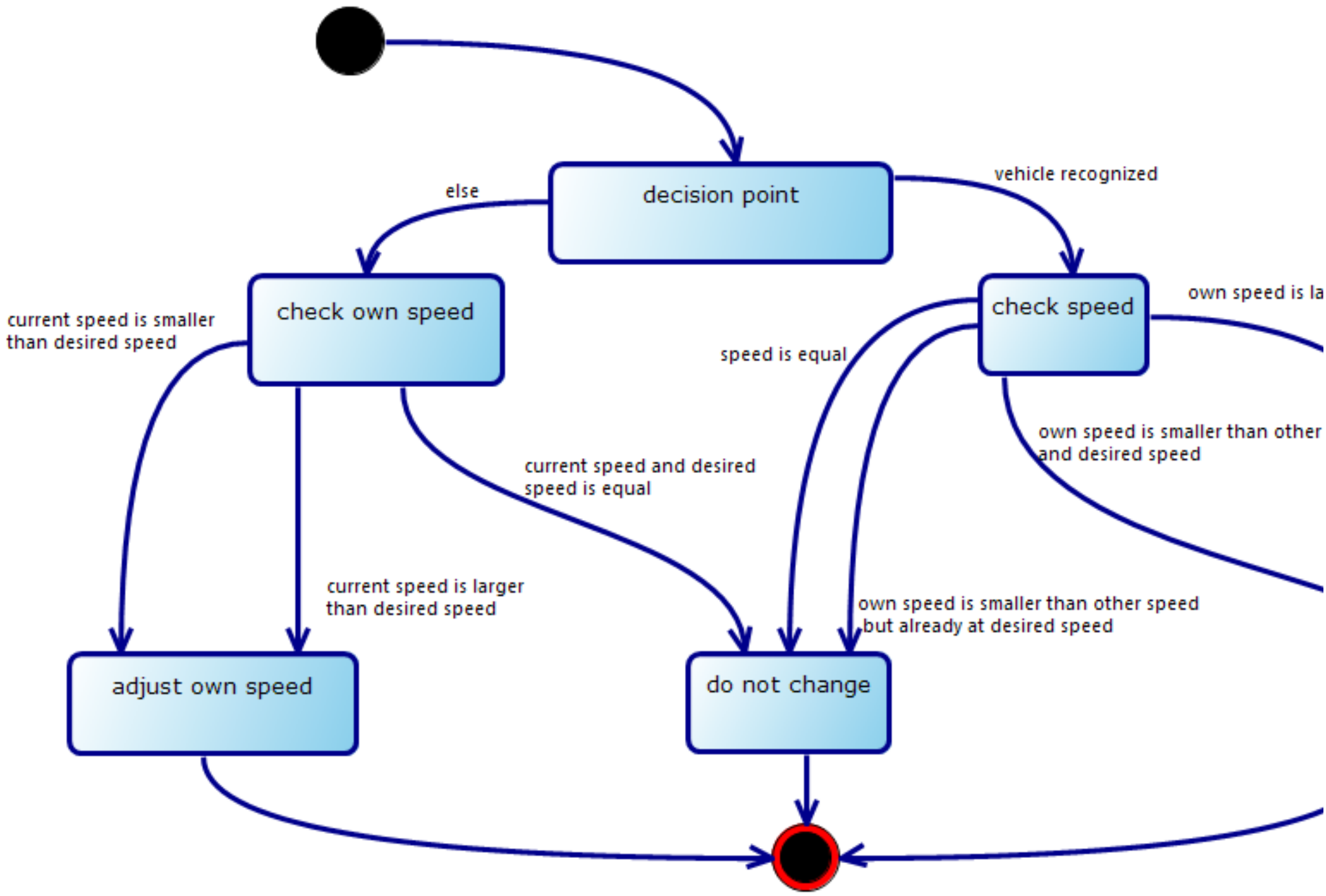


Im rechten Pfad wird die Geschwindigkeit des vorausfahrenden Fahrzeugs geprüft und mit der eigenen und der gewünschten Geschwindigkeit verglichen. Daraus ergeben sich vier Möglichkeiten:

- Die Geschwindigkeit des vorausfahrenden Fahrzeugs und die eigene sind gleich, dann soll die eigene Geschwindigkeit unverändert bleiben.
- Die Geschwindigkeit des vorausfahrenden Fahrzeugs ist größer als die eigene und man hat bereits die gewünschte Geschwindigkeit erreicht, dann soll die eigene Geschwindigkeit ebenfalls unverändert bleiben.
- Die Geschwindigkeit des vorausfahrenden Fahrzeugs ist größer als die eigene und man fährt langsamer als die gewünschte Geschwindigkeit, dann soll die eigene Geschwindigkeit angepasst werden.
- Die Geschwindigkeit des vorausfahrenden Fahrzeugs ist geringer als die eigene, dann soll die eigene Geschwindigkeit ebenfalls angepasst werden.

Die ersten zwei Transitionen führen also in den State do not change. Die letzten beiden in einen weiteren *State*, in dem die Geschwindigkeit an die des vorausfahrenden Fahrzeugs angepasst werden soll. Für alle drei States wird eine Transition zum Endzustand eingefügt.

Fertig sollte es dann so aussehen:

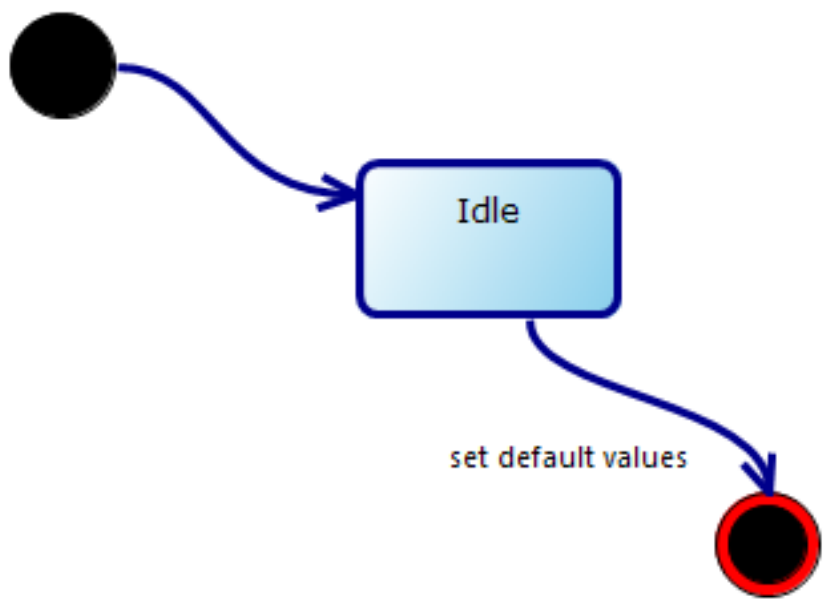


Vor- und Nachbedingungen festlegen

Wechselt man zurück zu ACC.cm, also auf die oberste Modellebene, existieren neben dem State Action zwei weitere States. Precondition und Postcondition. Diese enthalten in der Regel Vor- bzw. Nachbedingungen, die für das Modell relevant sind, d.h. in diesem Fall die Initialisierung der Standardwerte und die Auswertung der Durchläufe.

State Precondition

Um die Vorbedingungen festzulegen, erstellt man einen *Substate* in dem State Precondition. Dazu ruft man mit Rechtsklick das Kontextmenü auf und wählt *Expand*. Im *Substate* werden InitialState, FinalState und ein BasicState aus dem *Werkzeugmenü* hinzugefügt und mit Transitionen linear verbunden. Der *BasicState* wird umbenannt in Idle, die von Idle ausgehende *Transition* wird umbenannt in set default values.



Die *Transition* enthält die Voreinstellungen für das Benutzungsmodell.

Das Benutzungsmodell wird gespeichert und ist für den Augenblick fertig. Für die Testfallgenerierung muss das Benutzungsmodell noch mit Daten befüllt werden. Da die Anforderungsabdeckung getestet werden soll, werden zuerst die Anforderungen mit den States und Transitionen verknüpft. Neben den Anforderungen werden noch einige weitere Einstellungen gemacht, auf die im nächsten Abschnitt eingegangen wird.

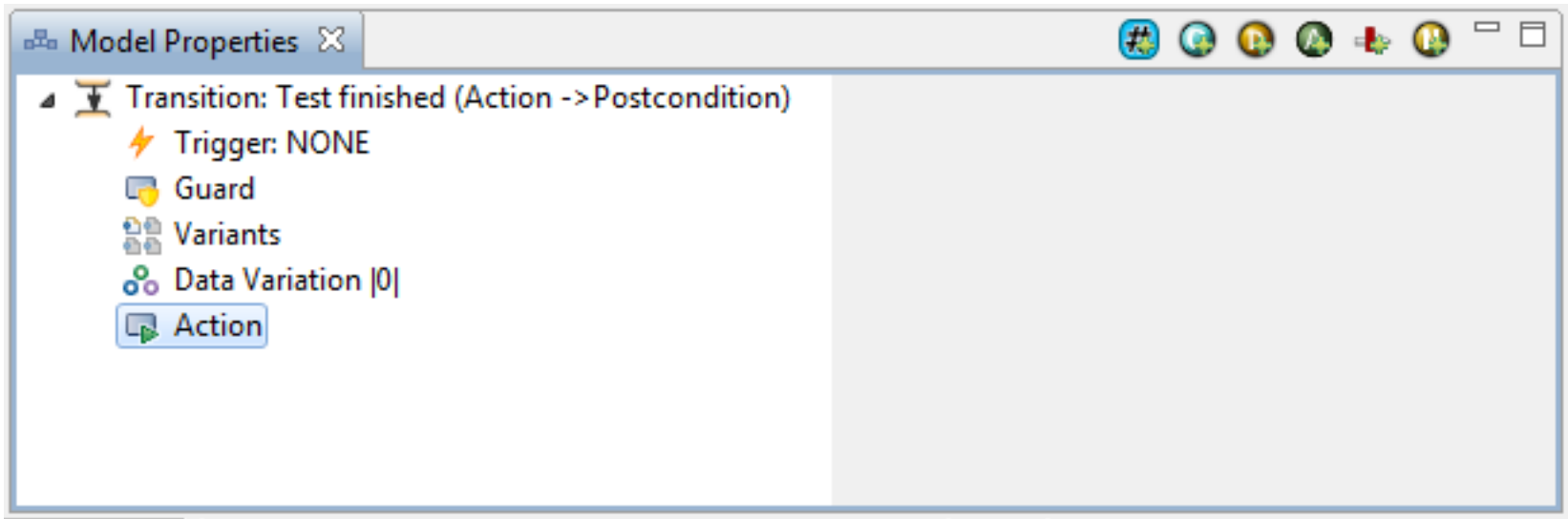
Schritt 3a - Modelleigenschaften befüllen

Wird ein Element im *Editor* selektiert, zum Beispiel ein *State* oder eine *Transition*, werden die Eigenschaften im *Model Properties View* angezeigt. Dieser befindet sich unten rechts in der Perspektive *Modeling*.

States, wie zum Beispiel der State Action enthalten *Entry Action* und *Exit Action*, die u.a. mit Objekten einer automatisierten Testausführungsumgebung befüllt werden können.

Modelleigenschaften befüllen, State

Transitionen hingegen enthalten mehrere Eigenschaften. Besonders wichtig sind hier *Trigger* und *Guard*. Damit der Testfallgenerator bei mehreren ausgehenden Transitionen aus einem Zustand die Auswahl der Transition treffen kann, müssen die Bedingungen der Ausführung der Transitionen eindeutig sein. Diese Eindeutigkeit kann entweder über sich unterscheidende *Guards* oder über *Trigger* (Ereignisse, die auf das System einwirken) erreicht werden.

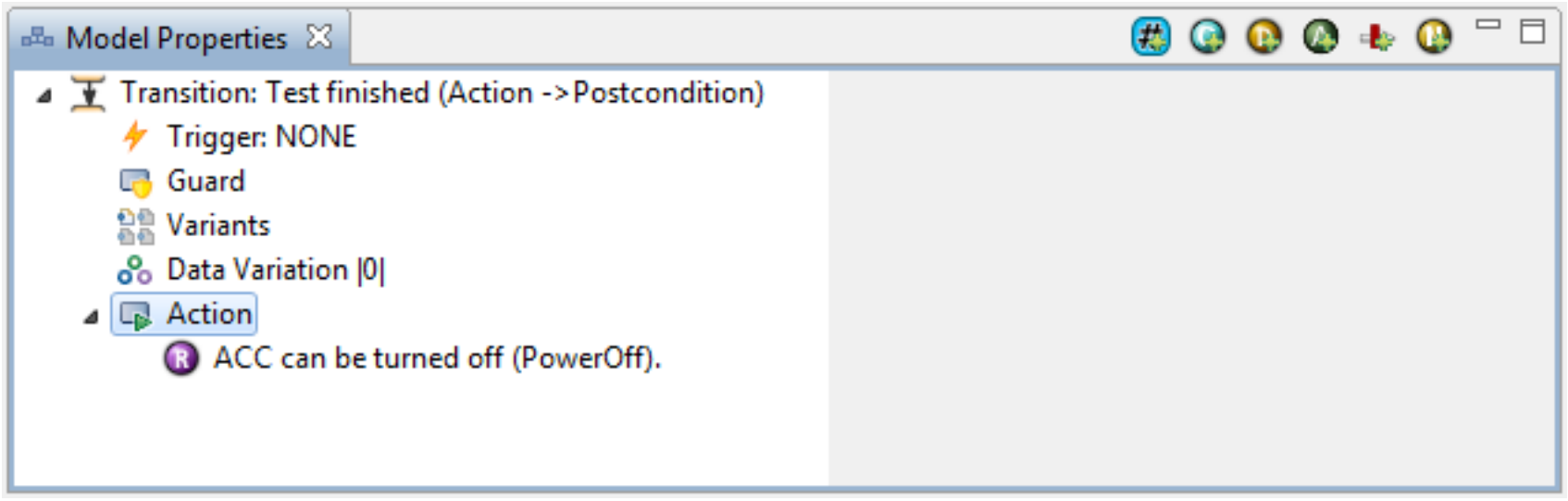


Modelleigenschaften befüllen, Transition

Ein weiterer wichtiger Punkt ist *Action*. Die *Action* einer *Transition* entspricht der *Entry* und *Exit Action* der *States*. Hier wird angegeben, was geschehen soll, wenn die Transition durchlaufen wird. An dieser Stelle werden Anforderungen, EXAM Objekte und andere Elemente verknüpft, auf die in dem folgenden Abschnitt näher eingegangen wird.

Anforderungen verknüpfen

Die Anforderungen, die aus DOORS nach MODICA importiert werden, müssen ins Modell überführt werden (Verknüpfung an Transitionen und Aktionen). D.h., dass Systemzustände daraus abgeleitet werden müssen, die das gewünschte Verhalten und die Übergänge in einem Zustandsautomaten beschreiben. Um eine Anforderung zu verknüpfen, wählt man die gewünschte Transition aus, sucht die entsprechende Anforderung in der *Requirements View* heraus und zieht sie per Drag&Drop in den *Model Properties View* in das *Action* Element.



Modelleigenschaften befüllen, Anforderung

Um die Zuweisung zu erleichtern, werden die zu verknüpfenden Anforderungen hier tabellarisch mit den dazugehörigen Elementen dargestellt.

Substate "Action"

Element	Anforderung
Transition turn on	ACC can be turned on (PowerOn).
Transition turn off	ACC can be turned off (PowerOff).

Substate "Turned On"

Element	Anforderung
Transition recognize other vehicle	Vehicle is recognized.
Transition recognize leaving vehicle	Recognized vehicle leaves range.
Transition recognize acceleration	Recognized vehicle increases speed (accelerates).
Transition recognize deceleration	Recognized vehicle decreases speed (brakes).
Transition deactivate	ACC can be deactivated.
Transition activate	ACC can be activiated.
Transition increase desired speed	Desired speed can be increased by 1 kmh pressing the Increase1-Button.
Transition decrease desired speed	Desired speed can be decreased by 1 kmh pressing the Decrease1-Button.

Substate "active"

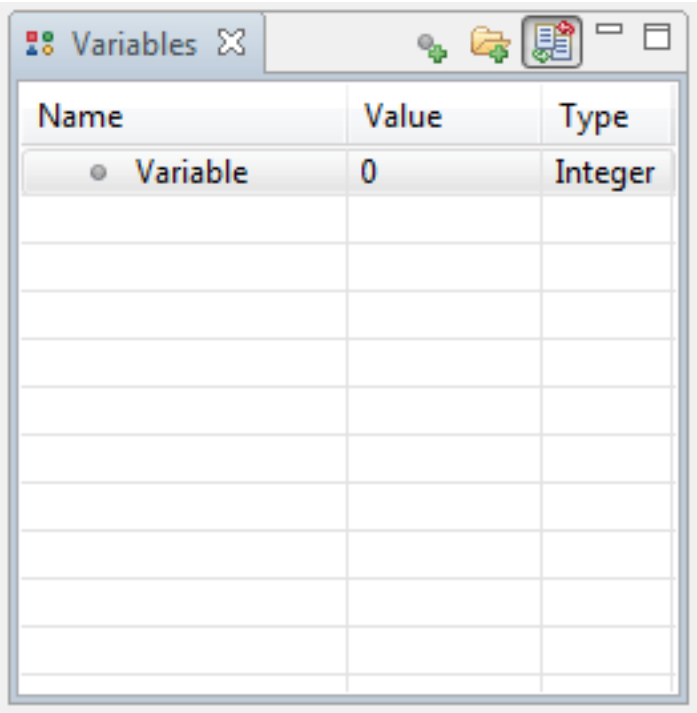
Element	Anforderung
Transition current speed is smaller than desired speed	The desired speed is increased by 1 kmh.
Transition current speed is larger than desired speed	The desired speed is decreased by 1 kmh.
Transition current speed and desired speed is equal	The desired speed remains unchanged.
Transition own speed is smaller than other speed but already at desired speed	Do not change speed.
Transition own speed is smaller than other speed and desired speed	The vehicle accelerates while maintaining the security gab between vehicles
Transition own speed is larger than other speed	The actual speed of the vehicle is reduced

Variablen erstellen

Variablen können in *States* und *Transitionen* verwendet werden, zum Beispiel zur Steuerung des Testfallgenerators oder als

Parameter für Funktionsaufrufe. In diesem Tutorial werden Variablen u.a. für die Geschwindigkeitseinstellungen verwendet.

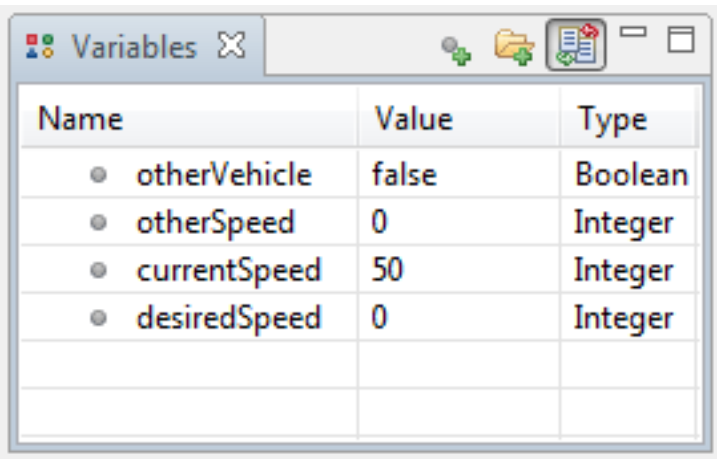
Der *Variables View (ehemals Library View)*, zur Verwaltung der Variablen, befindet sich unten links in der Perspektive *Modeling*.



Modelleigenschaften befüllen, Defaultvariable

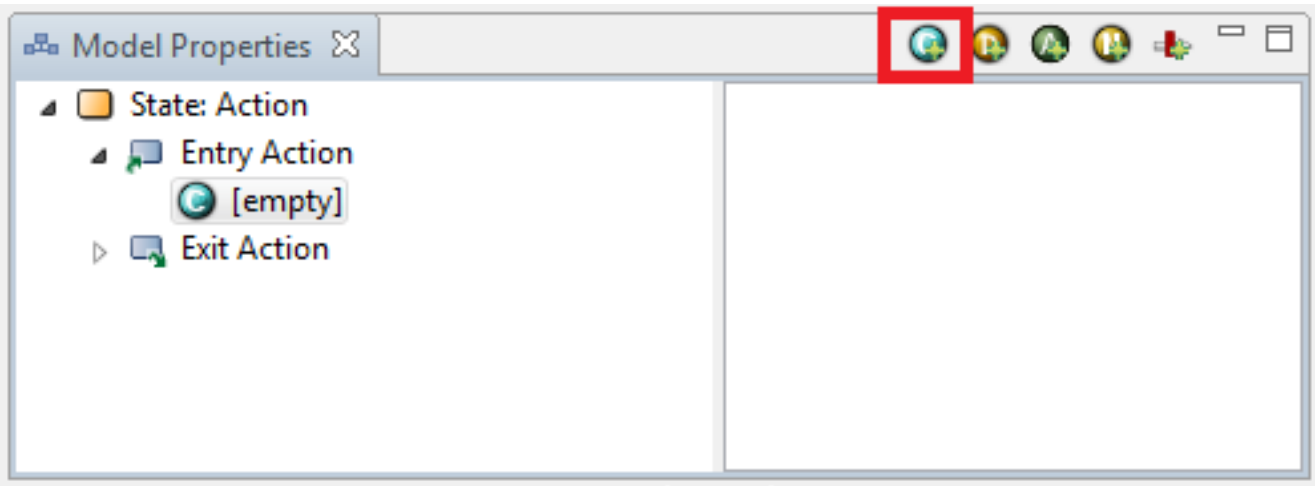
Eine Variable wird hinzugefügt, indem mit der rechten Maustaste in den *Variables View* geklickt wird und dem sich öffnenden Kontext-Menü *Add Variable* ausgewählt wird. Es wird eine Standard-Variable vom Typ Integer mit dem Default-Wert 0 und dem Namen Variable angelegt.

Es werden vier Variablen erstellt, welche die Sensorwerte enthalten sollen: otherVehicle, otherSpeed, currentSpeed und desiredSpeed. Die Variable otherVehicle wird verwendet, falls ein anderes Fahrzeug erkannt wird. Die anderen drei enthalten Geschwindigkeitswerte anderer Fahrzeuge, des eigenen Fahrzeugs und der Voreinstellung am ACC. Der Variable otherVehicle wird der Datentyp *Boolean* zugewiesen. Den Datentyp ändert man mit einem Klick auf den aktuellen Datentyp und wählt in dem Untermenü den gewünschten Datentyp aus.



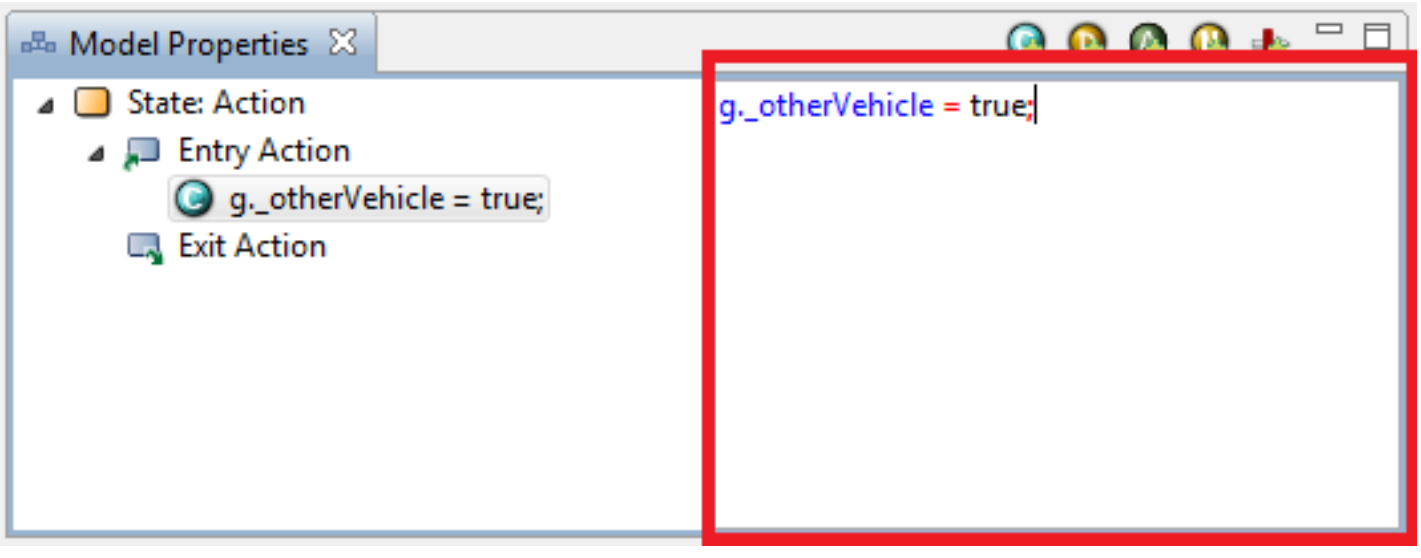
Calculations hinzufügen

In einer *Calculation* können einfache Berechnungen sowie Wertzuweisungen an MODICA-Variablen durchgeführt werden. In der *Model Properties View* findet man den Button zum Hinzufügen einer Berechnung *Add Calculation* im oberen Bereich der View, nachdem man ein Element ausgewählt hat.



Modelleigenschaften befüllen, Calculation hinzufügen

Mit Drag&Drop können Variablen aus der *Library View* in den Schreibbereich der *Calculation* (rechter Teil des Fensters) hinzugefügt und verwendet werden.



Modelleigenschaften befüllen, Calculation hinzufügen - Beispiel

In diesem Benutzungsmodell werden die *Calculations* dazu verwendet, um die Geschwindigkeit festzulegen und zu beschreiben, ob ein anderes Fahrzeug erkannt wird oder nicht. Als Beispiel sei hier die *Calculation* der Transition recognize other vehicle im Substate Turned On gezeigt:

```
g._otherVehicle = true;
g._otherSpeed = 49;
```

Sämtliche Berechnungsvorschriften für ein Element können in derselben *Calculation* angelegt und durch ein Semikolon voneinander getrennt werden.

Der Wert der Variable otherVehicle wird auf *true* gesetzt, d.h. ein anderes Fahrzeug wird erkannt. Das simulierte Fahrzeug hat eine Geschwindigkeit von 49 km/h, da diese geringer ist, als die aktuelle Fahrzeuggeschwindigkeit. Um das Tutorial hier übersichtlich zu halten, wird mit festen Werten gearbeitet. In realen Testszenarien können variable Geschwindigkeiten nötig sein, die auch über den Prüfstand ermittelt / definiert werden.

Hier eine Übersicht, welche *Calculations* in welchem Element gesetzt werden müssen. **Editor Element** beschreibt den *State* oder die *Transition*, in der die *Calculation* eingetragen wird. Um diesen zu sehen, muss das Element vorher ausgewählt sein. Unter **Calculation Text** ist der Inhalt der *Calculation* in Textform beschrieben. Er kann in die *Calculation* kopiert werden.

Elemente können auch mehrere *Calculations* enthalten.

ACC Statechart

Editor Element	Calculation Text	Erklärung

State Precondition / *Entry Action* `g._currentSpeed = 50;` Der Wert der Variable currentSpeed wird auf 50 gesetzt.

Substate "Turned On"

Editor Element	Calculation Text	Erklärung
Transition activate	`g._desiredSpeed = g._currentSpeed;`	Bei Aktivierung wird die aktuelle Geschwindigkeit als gewünschte Geschwindigkeit übernommen.
Transition activate	`g._otherVehicle = false;`	Der Wert der Variable otherVehicle wird auf false gesetzt.
Transition recognize other vehicle	`g._otherVehicle = true;` g._otherSpeed = 49;	Wenn ein anderes Fahrzeug erkannt wird, wird die Variable otherVehicle auf true gesetzt. Der Wert der Variable otherSpeed wird auf 49 gesetzt.
Transition recognize leaving vehicle	`g._otherVehicle = false;`	Verschwindet ein Fahrzeug aus dem Sensorbereich, wird die Variable otherVehicle auf false gesetzt.
Transition recognize accerleration	`g._otherSpeed++;`	Der aktuelle Wert der Variable wird erhöht.
Transition recognize deceleration	`g._otherSpeed--;`	Der aktuelle Wert der Variable wird verringert.
Transition decrease desired speed	`g._desiredSpeed--;`	Der aktuelle Wert der Variable wird verringert.
Transition increase desired speed	`g._desiredSpeed++;`	Der aktuelle Wert der Variable wird erhöht.

Substate "active"

Editor Element	Calculation Text	Erklärung
State adjust own speed / *Entry Action*	`g._currentSpeed = g._desiredSpeed;`	Die aktuelle Geschwindigkeit wird an die Wunschgeschwindigkeit angepasst.
State adjust speed / *Entry Action*	`g._currentSpeed = g._otherSpeed;`	Die aktuelle Geschwindigkeit wird an die Geschwindigkeit des vorausfahrenden Fahrzeugs angepasst.

Description hinzufügen

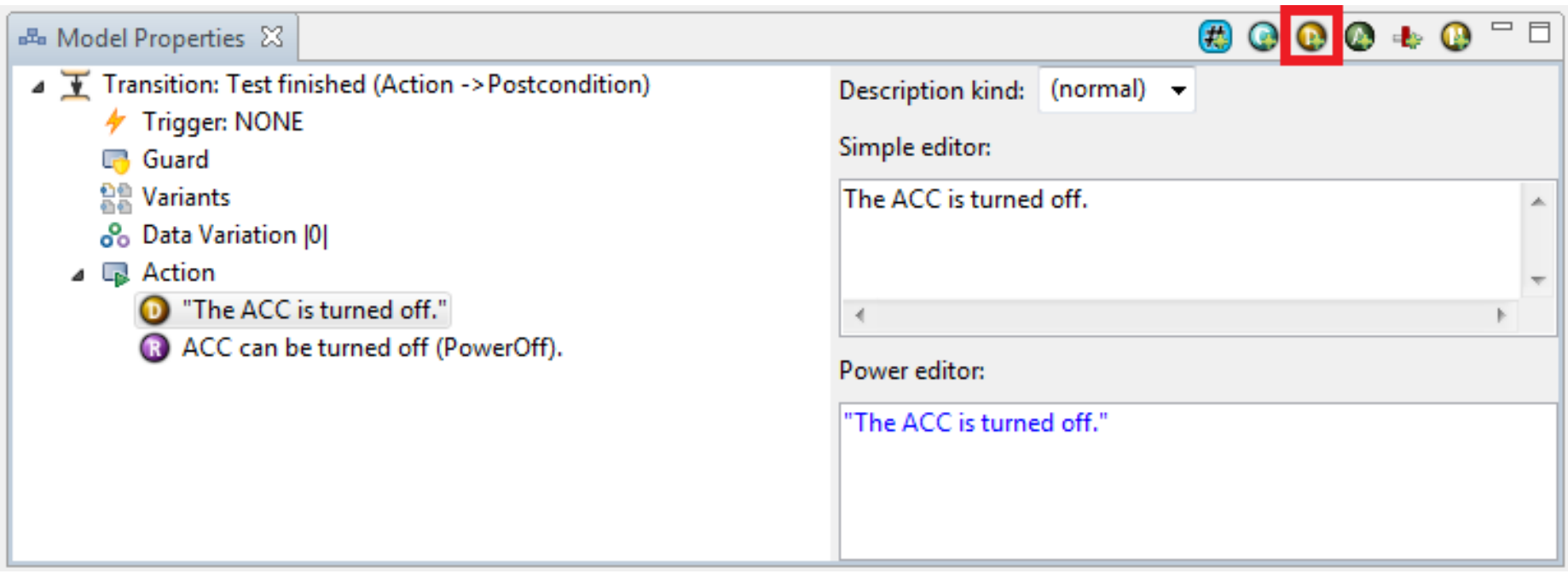
Beschreibungen (oder auch *Descriptions)* dienen dazu, in einem Testfall zu beschreiben, was darin passiert. Dazu lassen sich an Zuständen und Transitionen in MODICA Beschreibungen hinterlegen, die im endgültigen Testfall als Text zusammengefasst werden. Diese sind zur genaueren Dokumentation des Systems und zum Review und Debugging.

Beschreibungen werden im *Model Properties View* zu *Action* Elementen hinzugefügt. Um eine Beschreibung hinzuzufügen, wählt man das entsprechende Element im *Editor* aus, wechselt in die *Model Properties View*, selektiert das *Action* Element und klickt auf den Button *Add Description* . Im *Simple Editor* kann der gewünschte Text direkt eingegeben werden. Benutzt man den *Power Editor*, muss beachtet werden, dass **reiner Text in Anführungszeichen** geschrieben wird. Hier können jedoch Variablen über ein '+' an die Beschreibung angehängt werden. Varibalen werden dann per Drag&Drop aus dem *Varibales View* in den *Power Editor* gezogen. Um zwei in Anführungszeichen gesetzte Passagen

in derselben *Description* aneinander zu hängen, müssen sie ebenfalls mit einem '+' verbunden werden.

Weitere Informationen zur Formatierung der Beschreibungen finden sich in dem Kapitel Model Properties View.

Als Beschreibung eignet sich zum Beispiel der Inhalt der Anforderungen.



Modelleigenschaften befüllen, Description hinzufügen

In den folgenden Tabellen wird angezeigt welche Elemente welche Beschreibungen enthalten. Bei den Beschreibungen gibt es zwei Typen. Normal, welche den Inhalt im Fließtext darstellt. Und Steps, welche die Schritte wie eine Aufzählung anzeigt.

Substate "Action"

Element	Beschreibung
Transition turn on	"The ACC is turned on." "\nAction:\n- The ACC is turned on."
Transition turn off	"The ACC is turned off." "\nAction:\n- The ACC is turned off. "

Substate "Turned On"

Element	Beschreibung
Transition recognize other vehicle	"A vehicle is newly recognized." "\nAction:\n- A vehicle is newly recognized."
Transition recognize leaving vehicle	"A recognized vehicle leaves the detection range." "\nAction:\n- A recognized vehicle leaves the detection range."
Transition recognize acceleration	"A recognized vehicle accelerates." "\nAction:\n- A recognized vehicle accelerates"
Transition recognize deceleration	"A recognized vehicle slows down." "\nAction:\n- A recognized vehicle slows down."
Transition deactivate	"The ACC is deactivated." "\nAction:\n- The ACC is deactivated."
Transition activate	"The ACC is activated." "\nAction:\n- The ACC is activated."
	"The desired speed is increased."

Transition increase desired speed	"\nAction:\n- The desired speed is increased."
Transition decrease desired speed	"The desired speed is decreased." "\nAction:\n- The desired speed is decreased."

Substate "active"

Element	Beschreibung
State adjust own speed	"\nCheck:\n-The actual speed should change to " + g._currentSpeed
State do not change	"\nCheck:\n-The actual speed should stay at " + g._currentSpeed
State adjust speed	"\nCheck:\n-The actual speed should change to " + g._currentSpeed

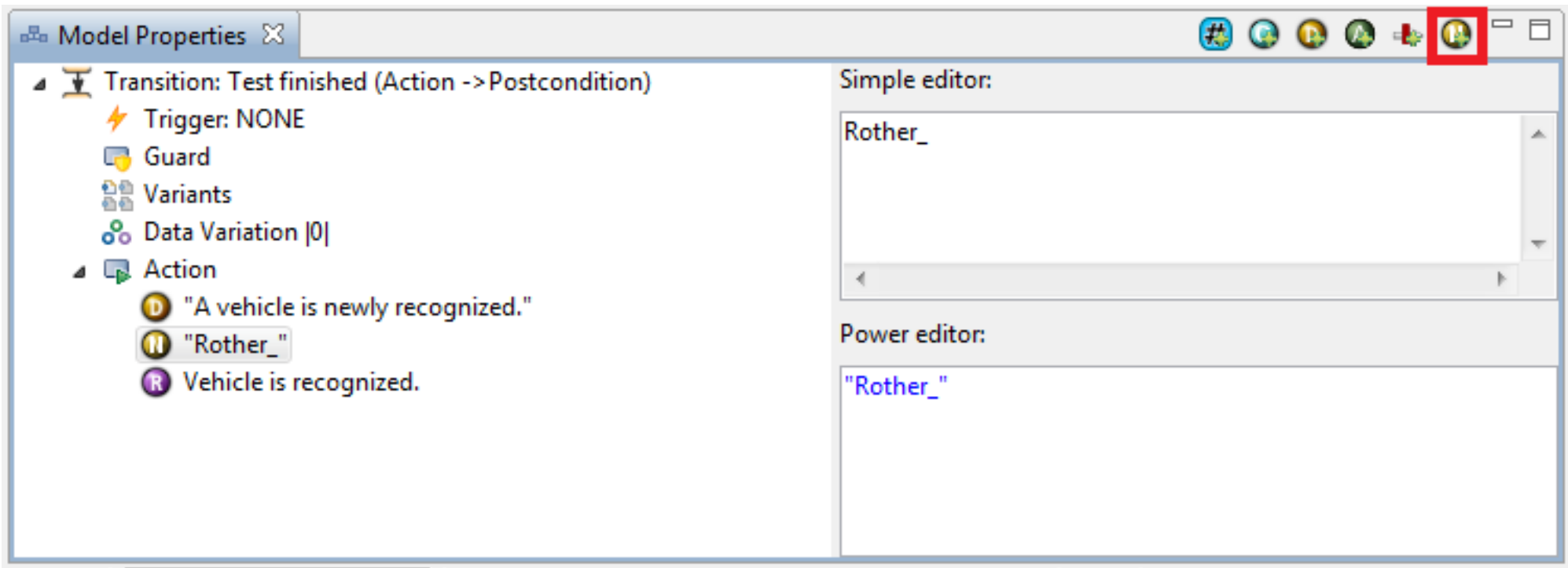
Naming Fragments hinzufügen

Unter *Naming Fragment* versteht man einen Bestandteil des Testfallnamens.

Ähnlich wie bei Beschreibungen können auf diese Art und Weise Testfallnamen zusammengesetzt werden. Hierzu müssen an allen Stellen im Modell, an denen ein Teil zu dem Testfallnamen hinzugefügt werden soll, ein Element *Naming Fragment* hinzugefügt werden. Die Fragmente sind meist kryptisch geschrieben, da die Namen ansonsten zu lang werden. Dennoch sollte ersichtlich sein, welcher Pfad gegangen wurde.

Die Fomatierung der Naming Fragments ist genauso wie die der Descriptions. Außerdem sollte jedes Fragment mit einem Unterstrich enden, damit, wenn sie aneinander gefügt werden, die einzelnen Abschnitte klar erkennbar bleiben. Um ein *Naming Fragment* hinzuzufügen, wählt man das entsprechende Element im *Editor* aus, wechselt in die *Model Properties View*, selektiert das *Action* Element und klickt auf den Button *Add Naming Fragment* .

In dem Beispiel sieht man das *Naming Fragment* für die Transition recognize other vehicle. Hier wurde der Name zusammengekürzt, lässt aber noch erahnen, worum es geht.



Modelleigenschaften befüllen, Naming Fragment hinzufügen

Folgende Elemente werden mit *Naming Fragments* versehen:

Substate "Action"

Element	Naming Fragment

Transition turn on "ON_"

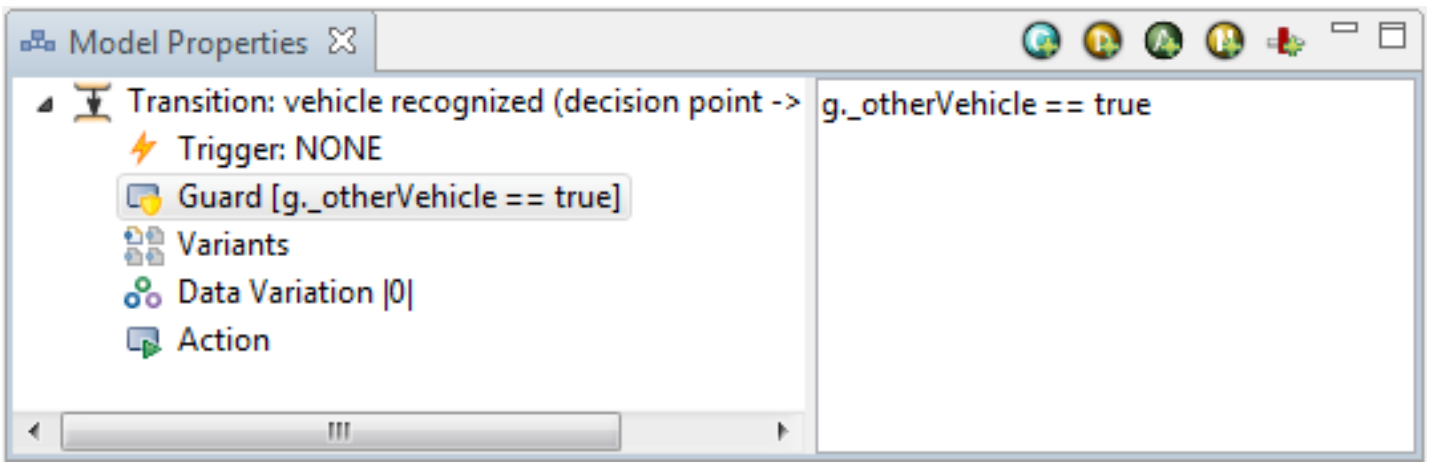
Substate "Turned On"

Element	Naming Fragment
Transition activate	"ACT_"
Transition deactivate	"DEACT_"
Transition increase desired speed	"DesFaster_"
Transition decrease desired speed	"DesSlower_"
Transition recognize other vehicle	"Rother_"
Transition recognize leaving vehicle	"RLeave_"
Transition recognize acceleration	"RAccel_"
Transition recognize deceleration	"RDec_"

Guards definieren

Der *Guard* kann als Wächter einer *Transition* verstanden werden.

Im Guard-Feld werden Bedingungen eingestellt, unter welchen Umständen eine *Transition* durchgeführt werden darf. Zum Beispiel darf eine *Transition* nur durchlaufen werden, wenn eine bestimmte Variable auf *true* gesetzt ist.



Achtung: Im Gegensatz zu den *Calculations* darf bei den *Guards* am Ende des Ausdrucks kein Semikolon gesetzt werden!

Generell müssen *Guards* nicht befüllt sein. Ist ein *Guard* leer, kann die *Transition* immer durchlaufen werden. Ist er befüllt, kann die *Transition* nur ausgeführt werden, wenn die Bedingungen erfüllt sind. Dadurch lassen sich im Modell weitere Abhängigkeiten modellieren.

Folgende *Guards* müssen gesetzt werden.

Substate "Turned On"

Element	Guard	Erläuterung
Transition recognize leaving vehicle	g._otherVehicle == true	Um das Verschwinden eines Fahrzeuges zu bemerken, ist die Bedingung, dass eines vorhanden ist.
Transition recognize acceleration	g._otherVehicle == true	Um die Erhöhung der Geschwindigkeit eines Fahrzeuges zu bemerken, ist die Bedingung, dass eines vorhanden ist.
Transition recognize deceleration	g._otherVehicle == true	Um die Verringerung der Geschwindigkeit eines Fahrzeuges zu bemerken, ist die Bedingung, dass eines vorhanden ist.

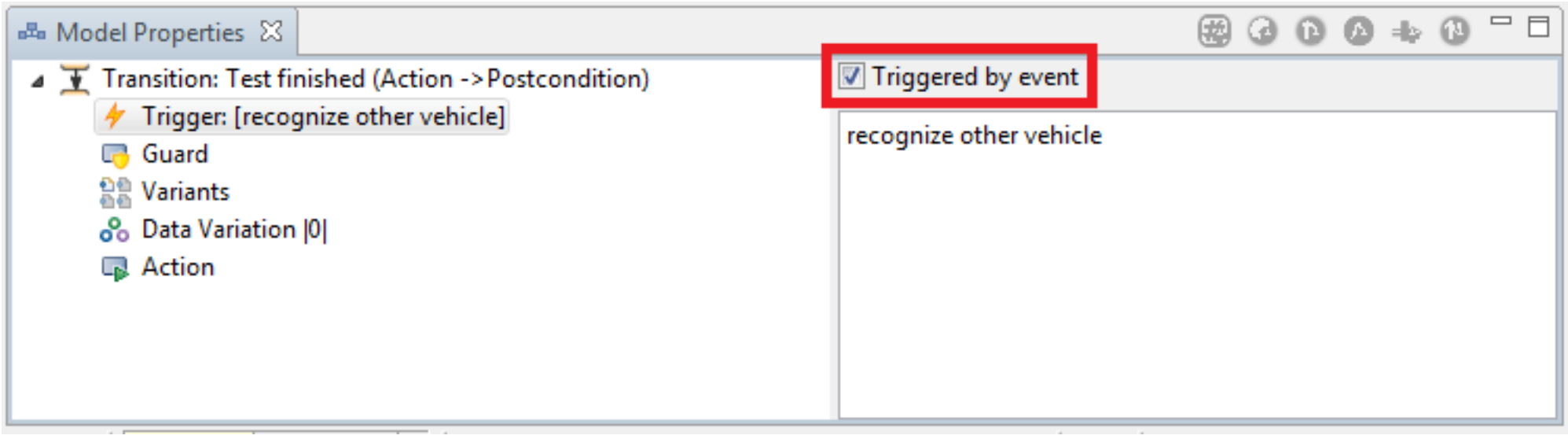
Substate "active"

Element	Guard	Erläuterung

Transition vehicle recognized	`g._otherVehicle == true`	Für die Überprüfung des Abstands und der Geschwindigkeit eines vorausfahrenden Fahrzeuges, ist die Bedingung, dass ein Fahrzeug vorhanden ist.
Transition own speed is larger than other speed	`g._currentSpeed > g._otherSpeed`	Um diese Transition ausführen zu können, muss die aktuelle Geschwindigkeit größer sein als die Geschwindigkeit des vorausfahrenden Fahrzeugs.
Transition own speed is smaller than other speed and desired speed	`g._currentSpeed < g._otherSpeed && g._currentSpeed < g._desiredSpeed`	Um diese Transition ausführen zu können, muss die aktuelle Geschwindigkeit kleiner sein als die Geschwindigkeit des vorausfahrenden Fahrzeugs und die gewünschte Geschwindigkeit.
Transition own speed is smaller than other speed but already at desired speed	`g._currentSpeed < g._otherSpeed && g._currentSpeed >= g._desiredSpeed`	Um diese Transition ausführen zu können, muss die aktuelle Geschwindigkeit kleiner sein als die Geschwindigkeit des vorausfahrenden Fahrzeugs, aber schon der gewünschten Geschwindigkeit entsprechen.
Transition speed is equal	`g._currentSpeed == g._otherSpeed`	Um diese Transition ausführen zu können, muss die aktuelle Geschwindigkeit gleich der Geschwindigkeit des vorausfahrenden Fahrzeugs sein.
Transition else	`else`	Else ist ein Befehl, der verwendet werden kann, um klar zu stellen, dass die Transition ausgeführt werden kann, wenn die anderen verfügbaren Transitionen aufgrund ihres Guards nicht ausgeführt werden können.
Transition current speed is smaller than desired speed	`g._currentSpeed<g._desiredSpeed`	Diese Transition kann ausgeführt werden, wenn die aktuelle Geschwindigkeit kleiner ist als die gewünschte Geschwindigkeit.
Transition current speed is larger than desired speed	`g._currentSpeed>g._desiredSpeed`	Diese Transition kann betreten werden, wenn die aktuelle Geschwindigkeit größer ist als die gewünschte Geschwindigkeit.
Transition current speed and desired speed is equal	`g._currentSpeed == g._desiredSpeed`	Diese Transition kann betreten werden, wenn die aktuelle Geschwindigkeit und die gewünschte Geschwindigkeit gleich sind.

Trigger setzen

Ein *Trigger* ist in UML das Ereignis, welches dazu führt, dass ein *State* verlassen wird. Damit der Testfallgenerator bei mehreren ausgehenden Transitionen aus einem Zustand die Auswahl der Transition treffen kann, müssen die Bedingungen der Ausführung der Transitionen eindeutig sein. Diese Eindeutigkeit kann entweder über sich unterscheidende *Guards* oder über *Trigger* erreicht werden. *Trigger* werden als Text angegeben, der das Ereignis, das zu diesem Übergang führt, symbolisiert. Die *Trigger* dienen dem Testfallgenerator dazu, die Testabläufe aus dem Benutzungsmodell zu erzeugen. **Grundsätzlich gilt**: Wenn ein State über mehrere Transitionen verlassen werden kann und sich diese aufgrund der Guards nicht gegenseitig eindeutig ausschließen, muss ein Trigger für jede Transition gesetzt werden.



Modelleigenschaften befüllen, Trigger setzen

Auf jeder Modellebene müssen die *Trigger* eindeutig sein. D.h. es ist sinnvoll, sie nach dem Ereignis zu benennen, das in der *Transition* passiert. An dieser Stelle wird nicht vorgegeben, was als *Trigger* eingetragen wird, da der Inhalt keine Auswirkung auf die Generierung hat.

Die Trigger werden gesetzt, indem man das Häkchen rechts oben bei *Triggered by event* anwählt. Also folgt nur eine Liste, der Elemente, die einen Trigger benötigen.

Substate "Action"

Element
Transition turn on
Ausgehende Transition zum Final State

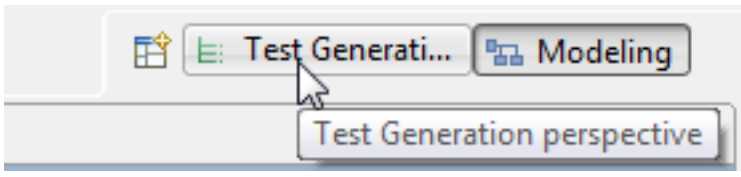
Substate "Turned On"

Element
Transition deactivate
Transition activate
Transition recognize other vehicle
Transition recognize leaving vehicle
Transition recognize acceleration
Transition recognize deceleration
Transition increase desired speed
Transition decrease desired speed
Ausgehende Transition zum Final State

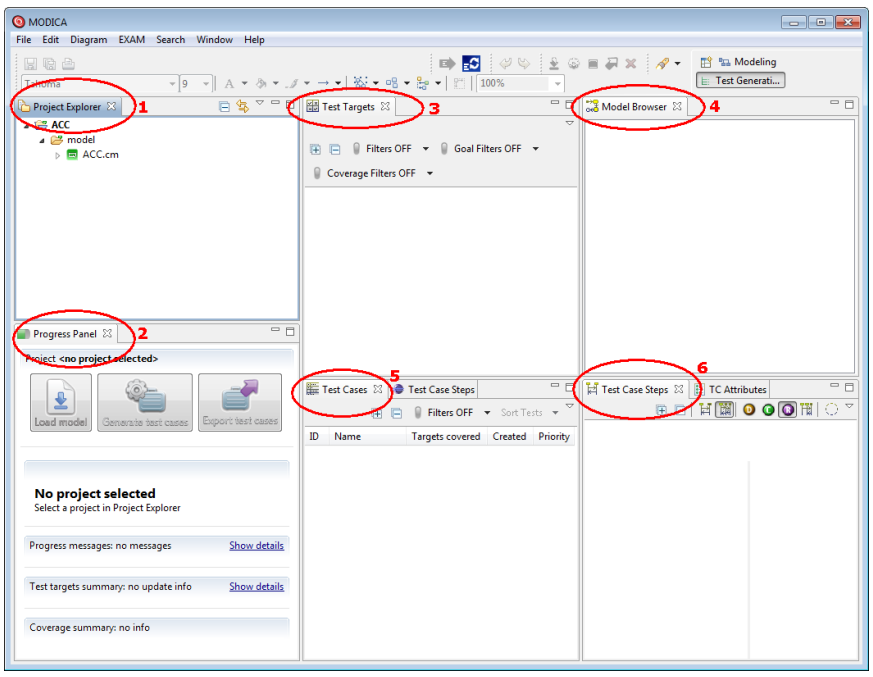
Damit sind die Modelleigenschaften des Benutzungsmodell mit allen wichtigen Elementen befüllt, die für eine erste Testfallgenerierung zur Requirementsabdeckung erforderlich sind. Im nächsten Schritt wird die Testfallgenerierung beschrieben und exemplarisch durchgeführt.

Schritt 4 - Testfallgenerierung

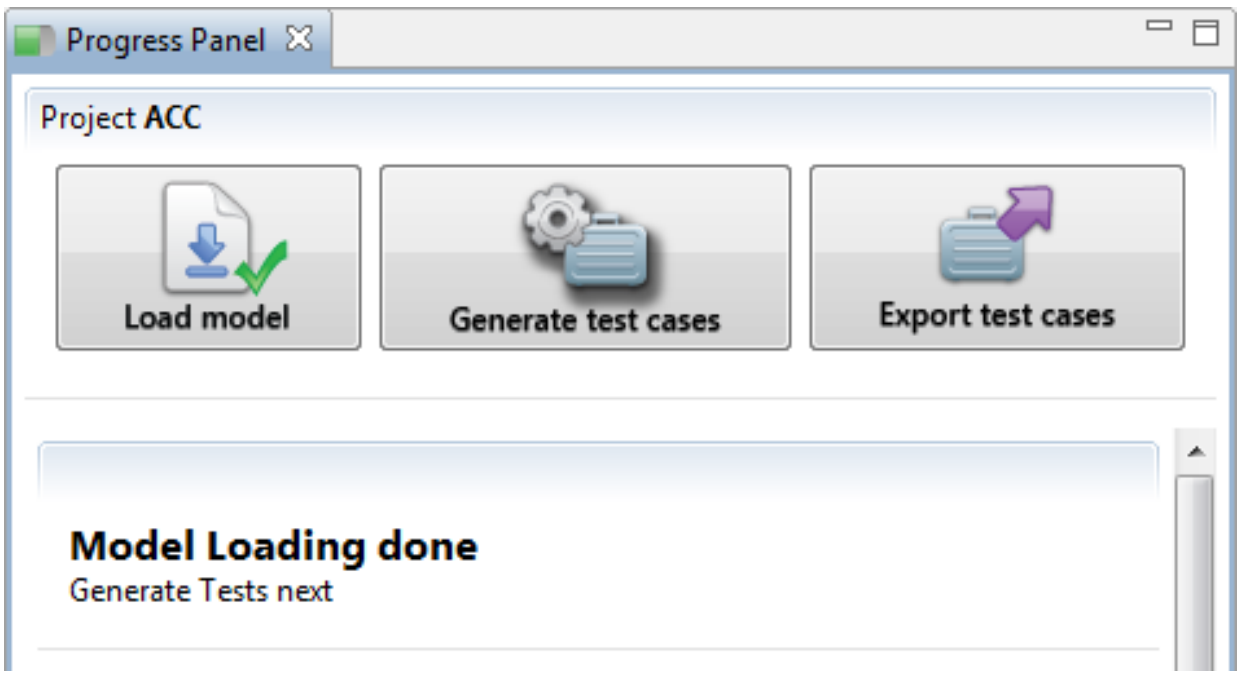
Für die Testfallgenerierung gibt es eine eigene Perspektive in MODICA. Um die Perspektive zu wechseln, klickt man auf den entsprechenden Button, der sich oben rechts im Programmfenster befindet.



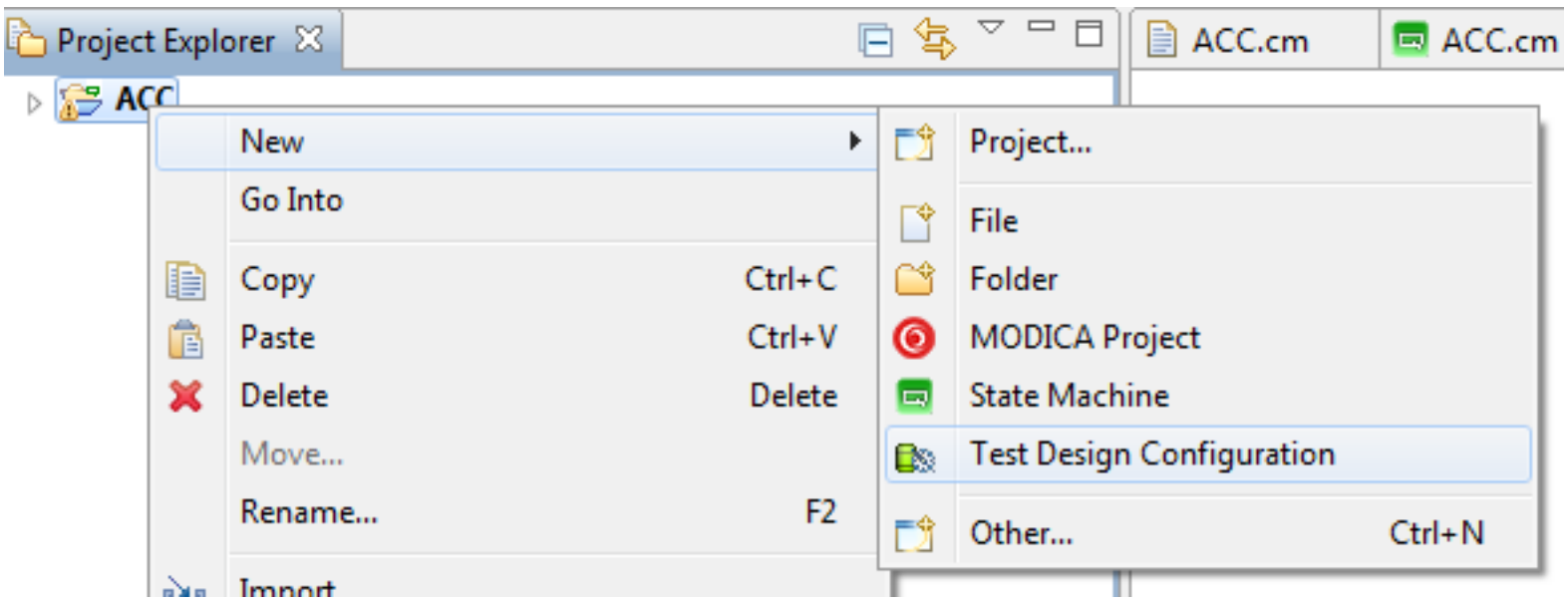
Diese Perspektive unterscheidet sich grundlegend von der Perspektive *Modeling*. Neben dem *Project Explorer* (1) beinhaltet sie das *Progress Panel* (2), mit dem das Modell geladen und die Testfallgenerierung gestartet wird. Die *Test Target View* (3) zeigt die Testabdeckung an. Der *Model Browser* (4) zeigt eine schematische Darstellung des Models, bei der man sich einzelne Pfade, die der Testfall genommen hat, anzeigen lassen kann. Unter *Test Cases* (5) werden die erstellten Testfälle aufgelistet, hier werden die aus den Naming Fragments erstellten Namen angezeigt. Im *Test Case Steps View* (6) kann man die einzelnen Test Schritte eines Testfalles nachverfolgen und sich die Aufrufe und Variablen-Werte anschauen.



Bevor Tests generiert werden können, muss das Model geladen werden. Dabei wird das Modell auf korrekte Syntax und Fehler geprüft. Dazu klickt man auf *Load model* in dem *Progress Panel*. Danach erhält man eine Übersicht, die u.a. Fehler im Modell anzeigt.

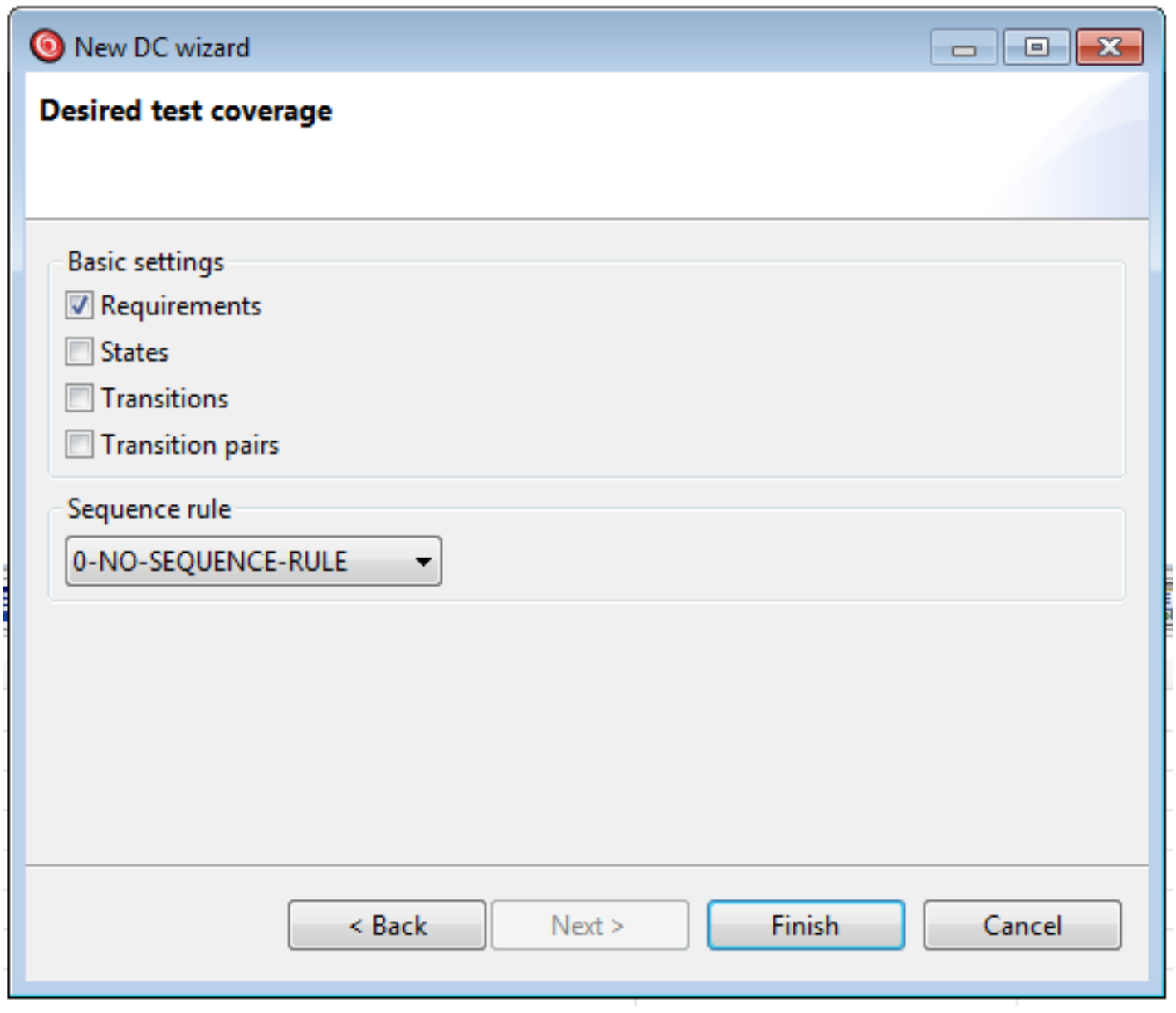


Um Testfälle zu generieren, muss definiert sein, welche Testziele erreicht werden sollen. Dafür muss eine *Test Design Configuration* erstellt werden. Bevor diese nicht erstellt ist, können keine Tests generiert werden. In der *Test Design Configuration* definiert man die Testziele, in diesem Fall die Abdeckung der Anforderungen. Um eine *Test Design Configuration* hinzuzufügen, wählt man ACC im *Project Explorer* aus, öffnet das Kontextmenü und wählt *New -> Test*



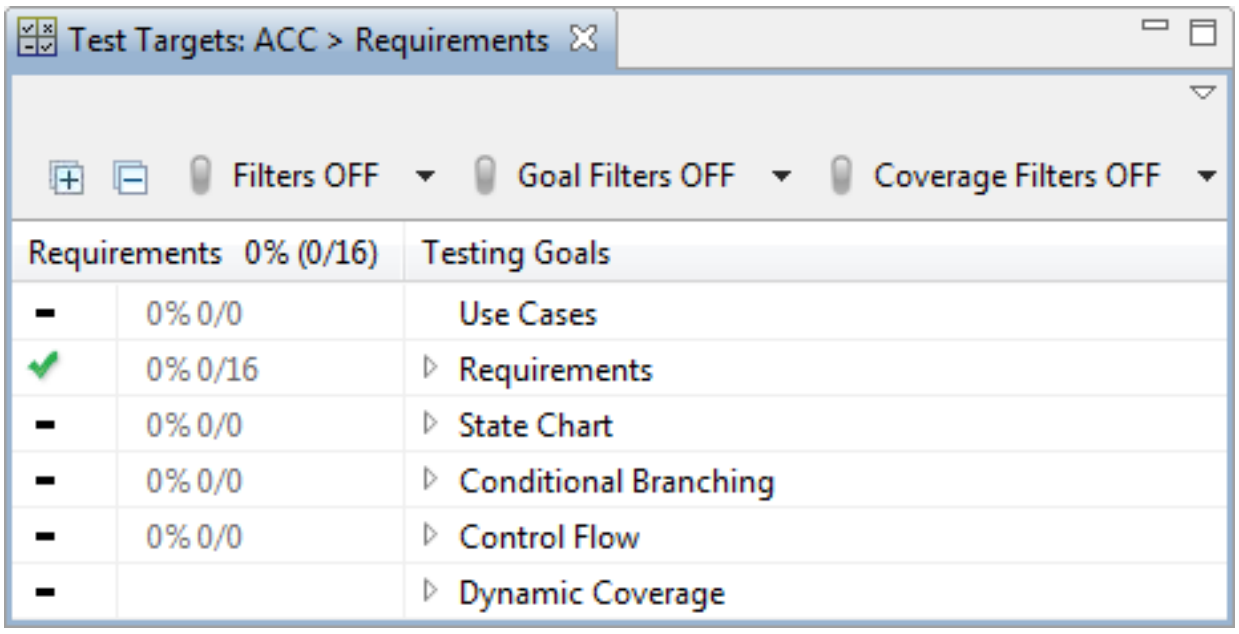
Neue DC-Konfiguration erstellen

In dem darauf folgenden Dialog kann der Name der Design Configuration geändetr werden - beispielsweise auf `Requirements`. Nachdem man auf *Next* geklickt hat, gibt es eine Vorauswahl an vorgefertigten Testzielen. Neben den hier angebotenen Möglichkeiten, kann man im *Test Target View* seine Testziele noch weiter verfeinern. In diesem Beispiel wählt man nur *Requirements* aus und klickt auf *Finish*.

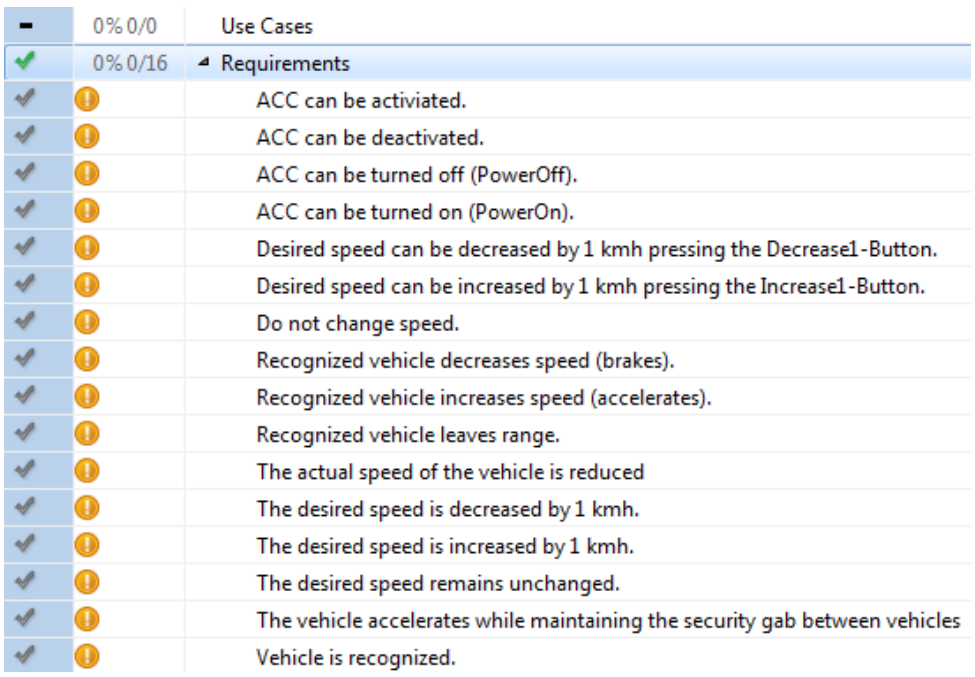


neue DC-Konfiguration erstellen - Einstellungen

Sobald die *Test Design Configuration* erstellt wurde, erscheinen die Testziele im *Test Target View*. Hier können mit einem Klick auf die erste Spalte weitere Testziele gesetzt oder vorhandene entfernt werden.



Ebenso lassen sich auch die einzelnen Requirements betrachten.



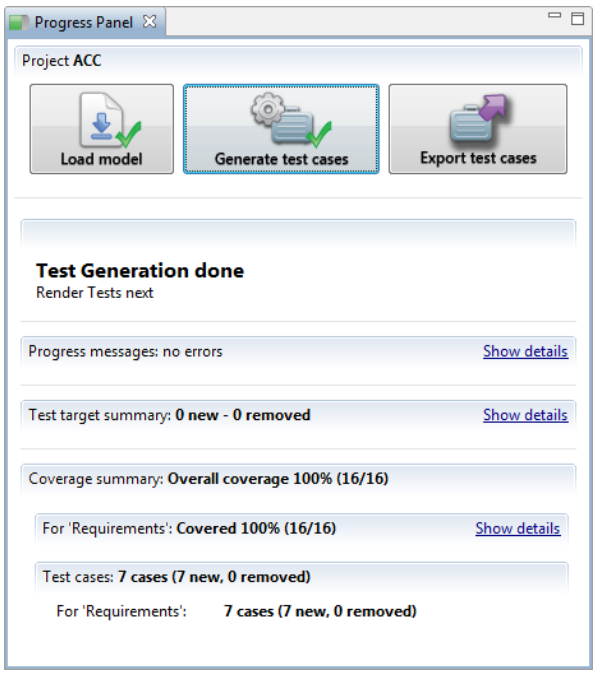
Mit den festgelegten Kriterien lassen sich nun Testfälle generieren.

Dazu wechselt man ins *Progress Panel* und klickt auf *Generate test cases*. Die Testfälle werden generiert, danach zeigt das *Progress Panel* eine Zusammenfassung der Abdeckung, die Anzahl der Testfälle und mögliche Fehler. Weitere Informationen sind in den [Modellierungsrichtlinien] in Kapitel 3 zu finden.

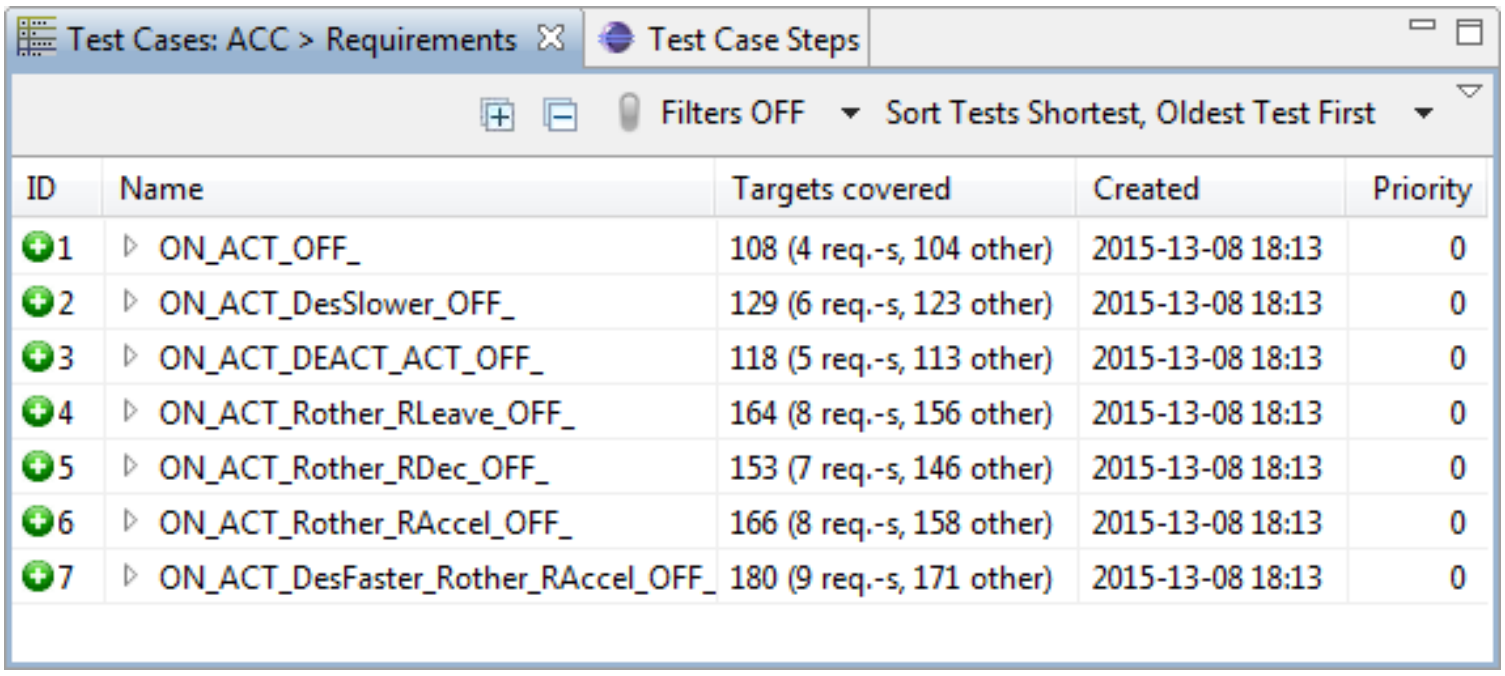
Tipp

Um mögliche Fehler frühzeitig zu erkennen, sollte vor dem Schritt *Load Model* eine Projekt-Validierung durchgeführt werden, um die Fehler besser lokalisieren und beheben zu können.

Dazu muss auf die *Modeling-Perspektive* gewechselt und auf den -Button geklickt werden. Sind Fehler vorhanden, so werden sie u.a. im *Problems View* angezeigt. Eine ausführlichere Dokumentation zur Projektvalidierung befindet sich im Kapitel Prüfung der Modellierung.

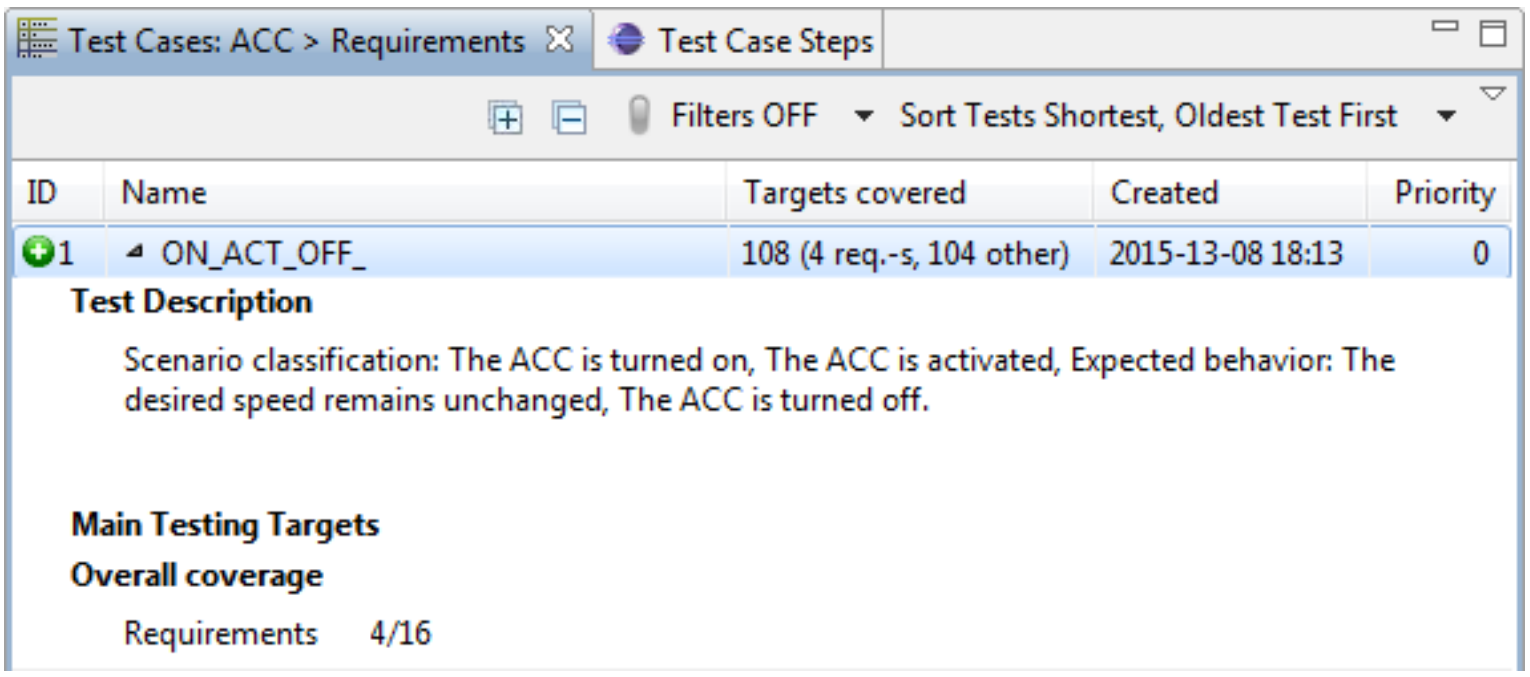


Die erstellten Testfälle erscheinen in dem *Test Cases View*. Der dort angezeigte Testfallname besteht aus den aneinandergereihten *Naming Fragments*.



ID	Name	Targets covered	Created	Priority
1	ON_ACT_OFF_	108 (4 req.-s, 104 other)	2015-13-08 18:13	0
2	ON_ACT_DesSlower_OFF_	129 (6 req.-s, 123 other)	2015-13-08 18:13	0
3	ON_ACT_DEACT_ACT_OFF_	118 (5 req.-s, 113 other)	2015-13-08 18:13	0
4	ON_ACT_Rother_RLeave_OFF_	164 (8 req.-s, 156 other)	2015-13-08 18:13	0
5	ON_ACT_Rother_RDec_OFF_	153 (7 req.-s, 146 other)	2015-13-08 18:13	0
6	ON_ACT_Rother_RAccel_OFF_	166 (8 req.-s, 158 other)	2015-13-08 18:13	0
7	ON_ACT_DesFaster_Rother_RAccel_OFF_	180 (9 req.-s, 171 other)	2015-13-08 18:13	0

Die Testfälle können aufgeklappt werden und enthalten Informationen wie die Beschreibung und die Testzielabdeckung. Die Test *Description* ist, ebenso wie die *Naming Fragments*, aus den einzelnen Sätzen zusammengefügt.



Die Testfälle, die hier entstanden sind, sind abstrakte Testfälle. D.h., dass diese Testfälle noch keine Funktionen zur Interaktion mit dem **System under Test** haben, da bislang noch keine Funktionen der Testautomatisierung hinzugefügt wurden, sondern an dieser Stelle Beschreibungen enthalten sind. Dennoch legen die hier generierten Testfälle den Ablauf fest, den der Testfallgenerator entsprechend der Generierungsregeln aus dem Benutzermodell ableitet. Die erzeugten Testabläufe können daher als Testspezifikation angesehen werden.

Schritt 4a - Beeinflussung des Testfallgenerators und Sequenzregeln

Dieses Kapitel baut insbesondere auf dem Wissen aus dem Handbuch-Kapitel Testfallgenerierung auf und erklärt am Beispiel fortgeschrittene Möglichkeiten, den Testfallgenerator zu beeinflussen. Zusätzliche Informationen zu diesem Thema finden sich in Handbuch-Kapitel Sequenzregeln (Ablaufregeln).

Die mit MODICA modellierten Benutzungsmodelle erlauben häufig eine Vielzahl von aufeinanderfolgenden Aktionen. Im ACC Modell kann das System beispielsweise beliebig oft zwischen active und inactive umgeschalten werden. Obwohl eine lange Kette von solchen Umschaltungen möglich ist, stellt sie möglicherweise kein ökonomisch sinnvolles Testszenario dar (aus Sicht der Ausführungskosten). Ebenso enthalten MODICA Modelle häufig Stellen, an denen verschiedene Stimuli zur Verfügung stehen und der Testfallgenerator sich entscheiden kann, welcher als nächstes durchgeführt wird. Immer wenn bei der Generierung mehrfach solche Entscheidungen anstehen, ist die Menge der zu generierenden Testfälle potentiell sehr groß. Im ACC Beispiel ist hier der State „active“ mit seinen ausgehenden Transitionen eine solche Stelle, da fast alle ausgehenden Transitionen zurück zum gleichen State führen.

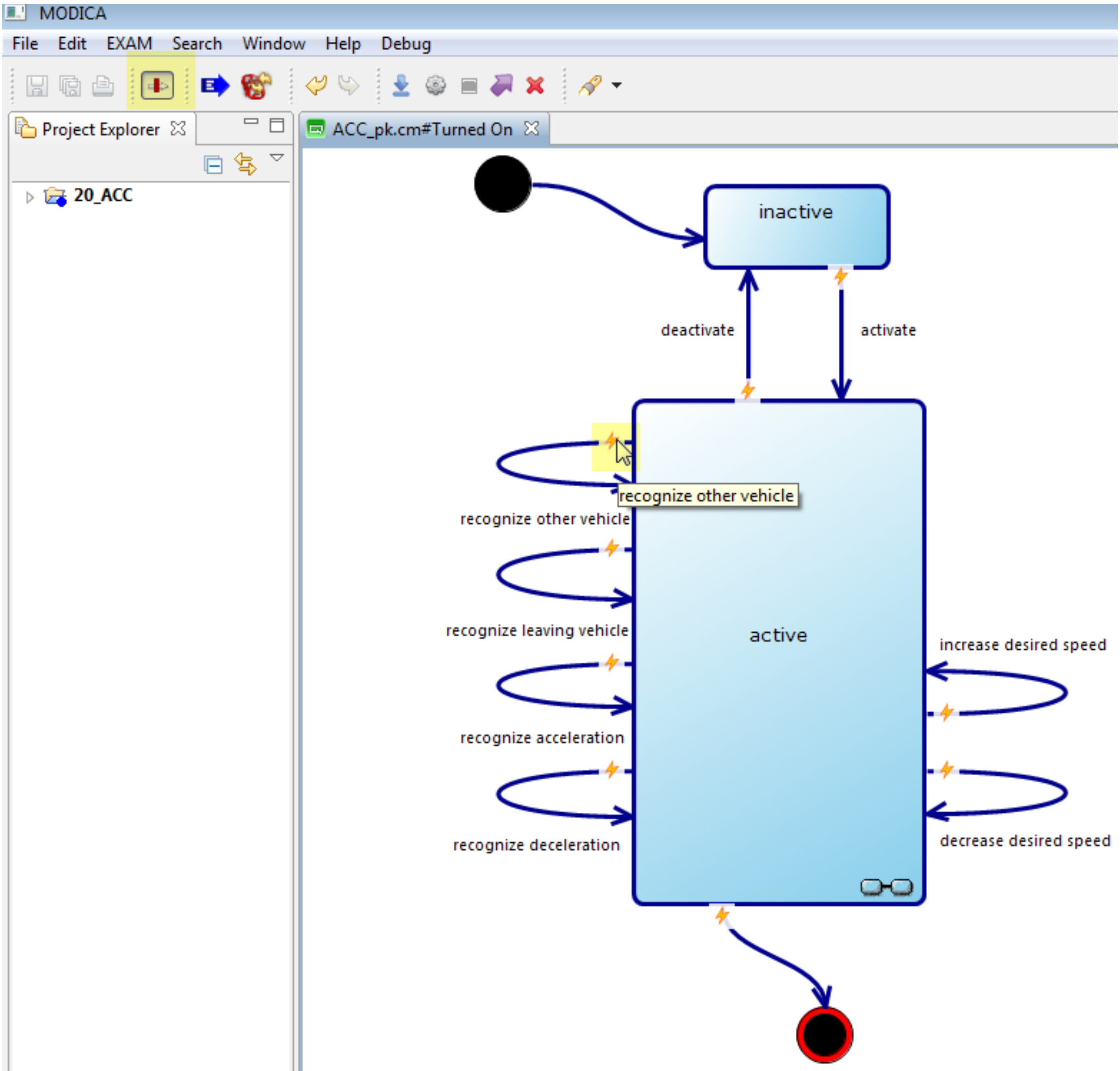
Da die Kombinationsmöglichkeiten in dem Benutzungmodell sehr groß sind, favorisiert die Generierung Testfälle, die mit dem „kürzesten“ Weg die angegebenen Testziele erreichen. Dies hat allerdings zur Folge, dass unter Umständen Fehler nicht gefunden werden, die nur bei bestimmten Abläufen bzw. einer bestimmten Sequenz von Stimuli auftreten. Hat man als Tester hier tieferes Domänenwissen und kennt daher solche potentiellen gefährlichen Sequenzen, so kann man dieses in Form von *Sequenzregeln* formalisieren und dem Generator mit auf den Weg geben.

Im konkreten Beispiel des ACC könnte es beispielsweise interessant sein dafür zu sorgen, dass Testfälle generiert werden, die zunächst

- nach dem Anschalten des Fahrzeugs das ACC aktivieren
- das ACC direkt wieder deaktivieren
- das ACC wieder aktivieren
- ein anderes Fahrzeug erkennen

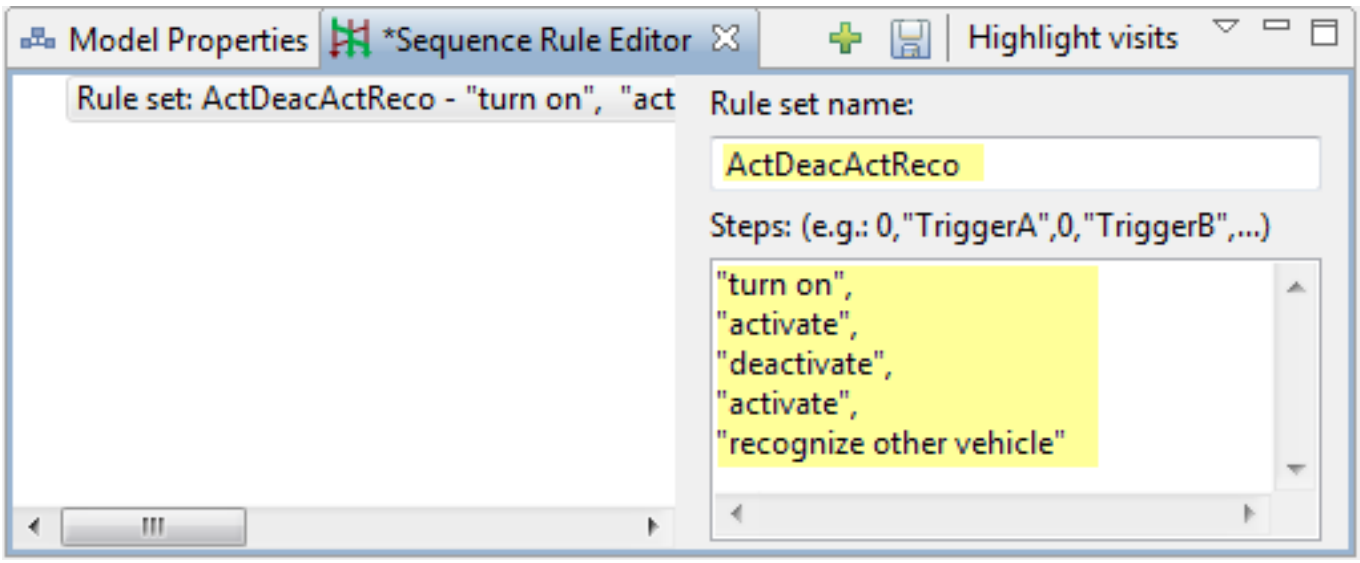
Erst nach diesem Prefix soll der Testfallgenerator selbst versuchen, die noch offenen Testziele abzudecken.

In vielen Fällen müssen an Modell keine weiteren Änderungen vorgenommen werden, um für das beschriebene Testziel eine passende Sequenzregel zu erzeugen. Wichtig sind jedoch die im Modell vorhandenen Trigger, da diese Entscheidungsmöglichkeiten für den Testfallgenerator darstellen. Sie können mit einem Klick auf das folgenden Screenshot markierte Icon in allen Diagrammen sichtbar gemacht werden:



Hauptdiagramm mit Annotationen für Trigger

Dazu wird zunächst per ein neues Regel-Set angelegt und mit einer textuellen Zusammenfassung der zu schreibenden Regel, ähnlich den *Naming Fragments*, benannt. In diesem Fall ActDeacActReco.

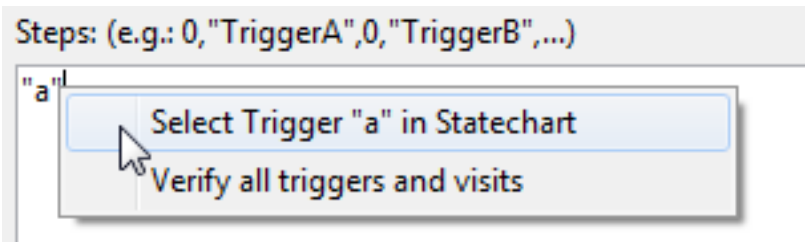


Benennen und füllen der Sequenzregel

In der größeren Textbox werden die vom Generator zu passierenden Trigger wie im Screenshot eingetragen:

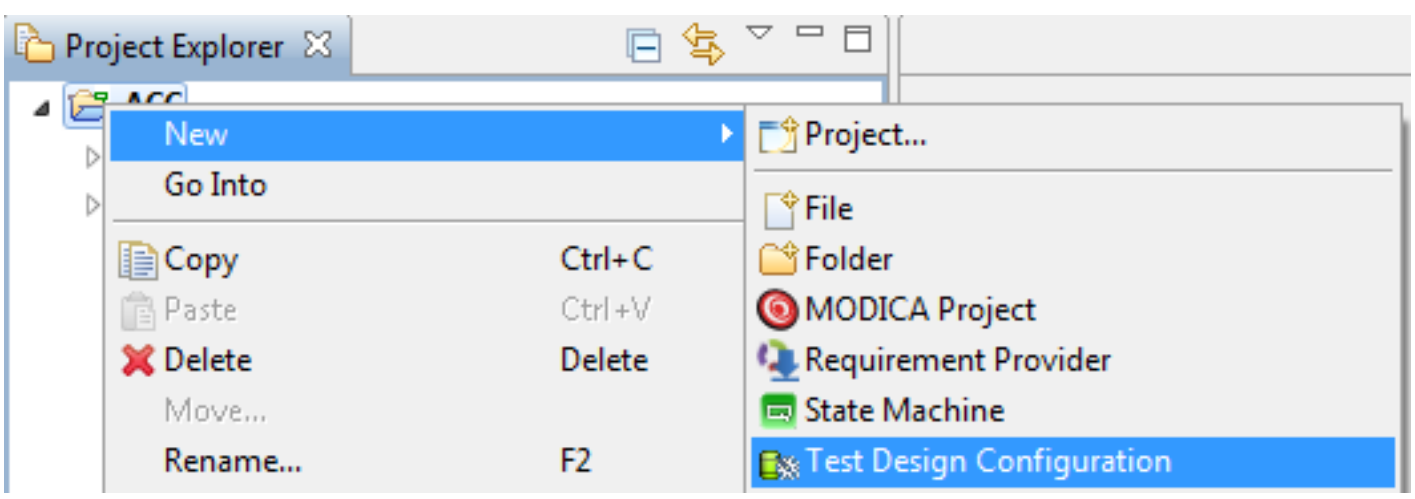
```
"turn on",
"activate",
"deactivate",
"activate",
"recognize other vehicle"
```

Es ist hierbei essentiell, dass die angegebene abfolge auch tatsächlich im Modell möglich ist. Das prinzipielle Vorhandensein der angegebenen Trigger lässt sich aber bereits mit dem Kontextmenü prüfen:



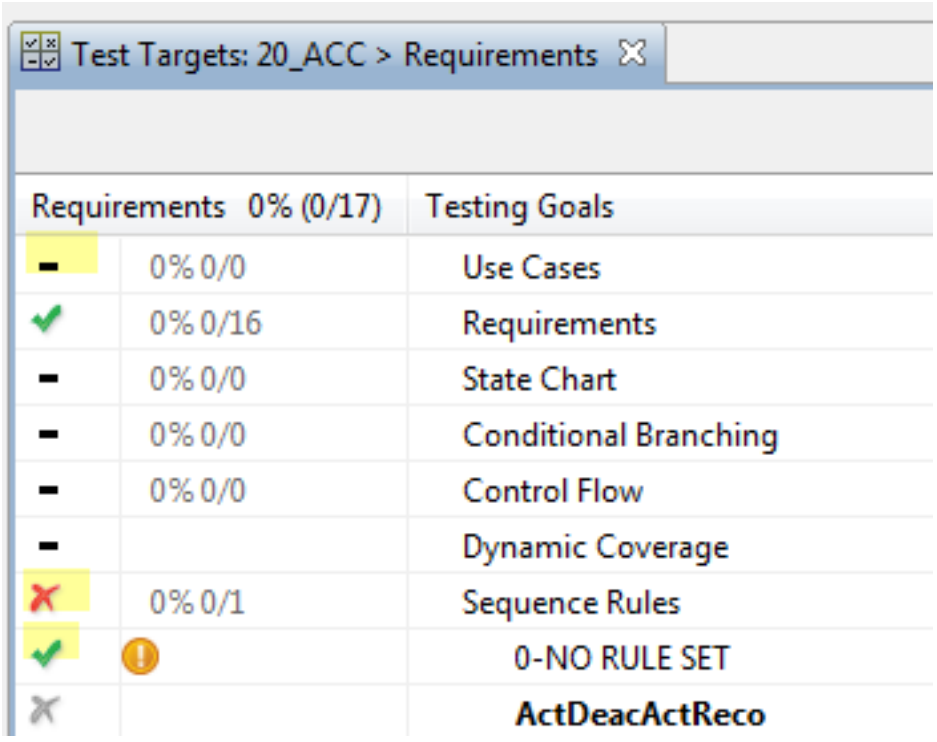
Select Trigger in Statechart

Um die Sequenzregel zu verwenden, wird eine weitere *Design Configuration* im Projekt angelegt, deren Name identisch mit dem der Sequenzregel ist. Dazu wählt man erneut das Projekt im *Project Explorer* aus, öffnet das Kontextmenü und wählt *New → Test Design Configuration*.



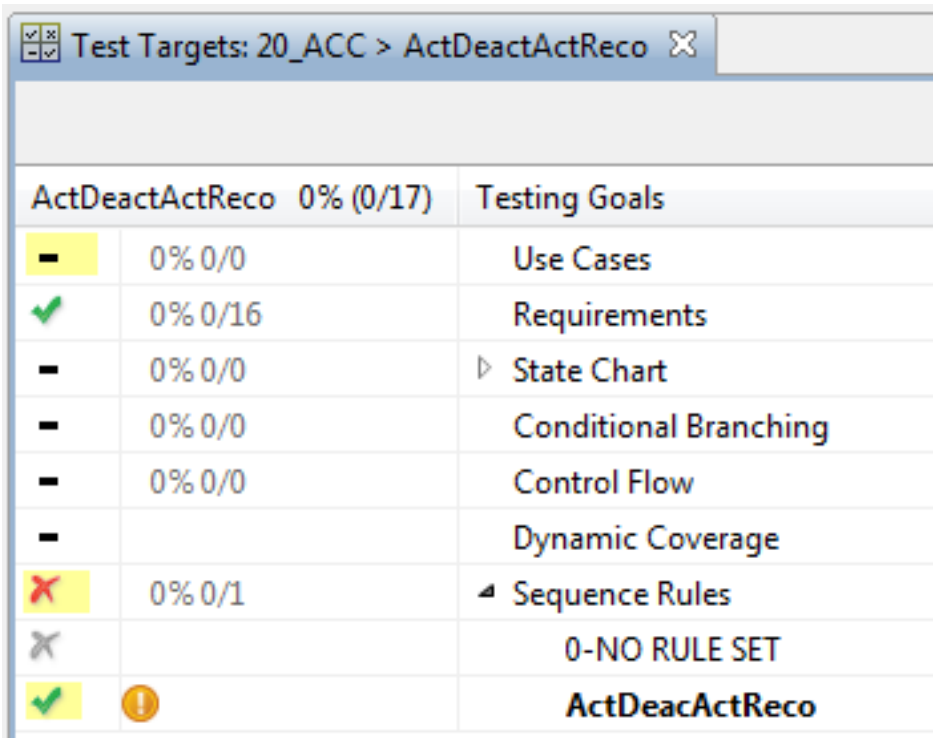
Im folgenden Wizard ist *Requirements* als Testziele auszuwählen. Der empfohlene Name ist ActDeacActReco.

Man wechselt in die Perspektive *Test Generation* und klickt im *Progress Panel* auf *Load Model*. Danach erscheint die Sequenzregel im *Test Targets View*. Für die bestehende Design Configuration Requirements werden die neuen Sequenzregeln deaktiviert wie im Screenshot ersichtlich (alle gelb markierten Einstellungen sind vorzunehmen):



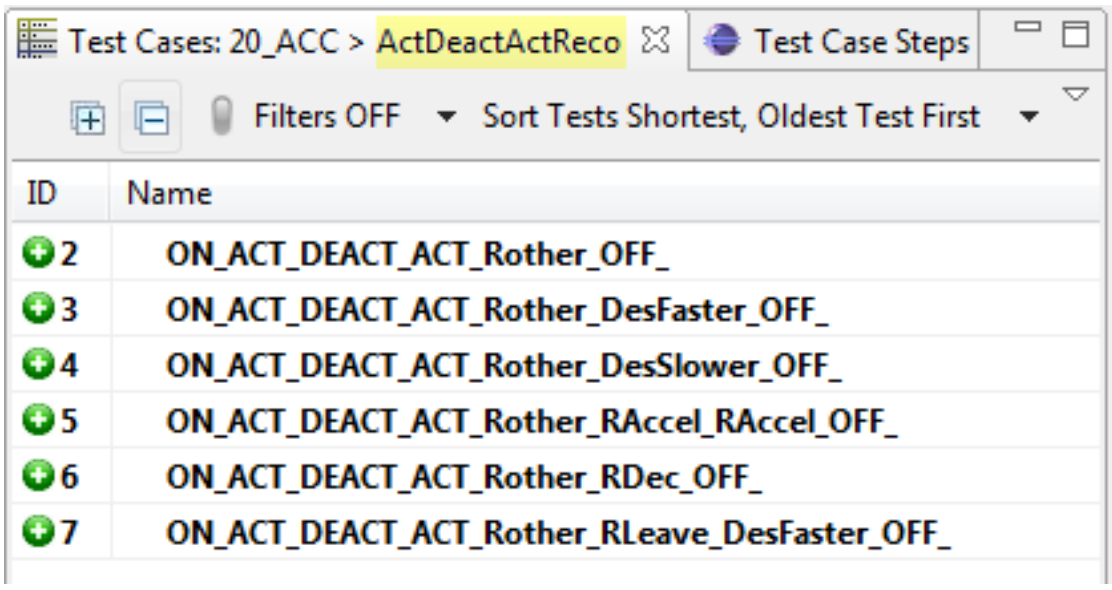
TestTargetsView Einstellungen Requirements DC

Für die neue Design Configuration ActDeactActReco wird die gleichnamige Sequenzregel ausgewählt indem die markierten Einstellungen gesetzt werden:



TestTargetsView Einstellungen für Sequenzregel-DC

Generiert man nun Testfälle, so erhält man neben den Testfällen für die Requirementsabdeckung auch Testfälle für die Sequenzregeln. Markiert man im Project Explorer die neue Design Configuration, so werden im Test Cases view nur die zugehörigen Testfälle angezeigt:



Generierte Tests für ActDeactActReco

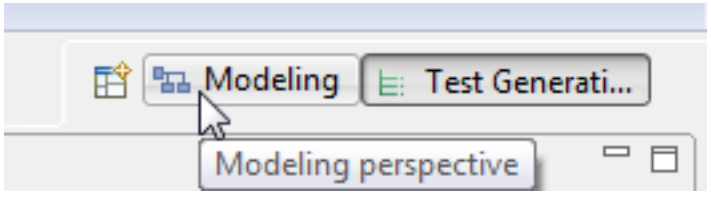
Neben der hier entworfenen Sequenzregel findet sich im beiliegendem ACC Projekt auch noch weitere Sequenzregeln samt zugehöriger *Design Configuration* als zusätzliche Anschauungsbeispiele. Im Handbuch sind zusätzlich Informationen zu finden. Diese erklären unter anderem die Definition von Alternativen oder auch dem Einsatz von Visits im Modell für explizite Lücken in den Sequenzregeln.

Schritt 5 - EXAM Funktionen dem Modell hinzufügen

Die folgenden Schritte beschreiben die Vorbereitung der Ausführung von ablauffähigen Testfällen, die in MODICA erzeugt werden. Hier wird die Ausführung unter Verwendung des Werkzeugs EXAM (EXtended Automation Framework) beschrieben.

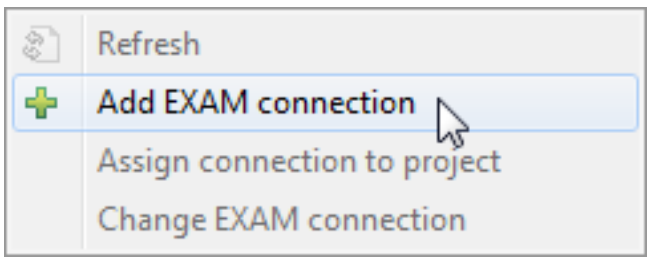
Damit die Testfälle exportiert werden können und später auch in EXAM ausgeführt werden, müssen vor der Generierung einige EXAM Funktionsaufrufe mit Modellelementen verknüpft werden. Das kann natürlich schon vorher während der Erstellung des Benutzungsmodells gemacht werden.

Dafür wird eine Verbindung mit dem EXAM Server benötigt. Hierfür sollten die entsprechenden Serverinformationen und Logindaten vorliegen. Die Zuweisung von EXAM Objekten findet in der Perspektive *Modeling* statt, deshalb wird an dieser Stelle die Perspektive wieder gewechselt.

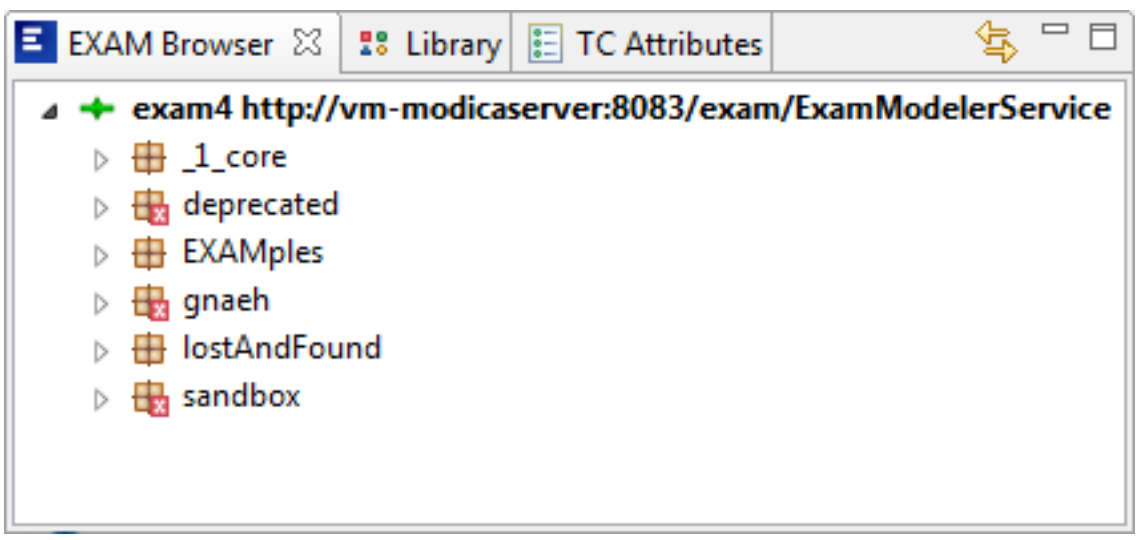


Erstellung einer EXAM Verbindung

In der Perspektive *Modeling* befindet sich der *EXAM Browser* unten links. Falls der *Variables* View noch ausgewählt ist, klickt man auf *EXAM Browser*, um die Ansicht zu ändern. Um eine neue EXAM Verbindung herzustellen, öffnet man das Kontextmenü der View mit Rechtsklick und wählt *Add EXAM connection*.



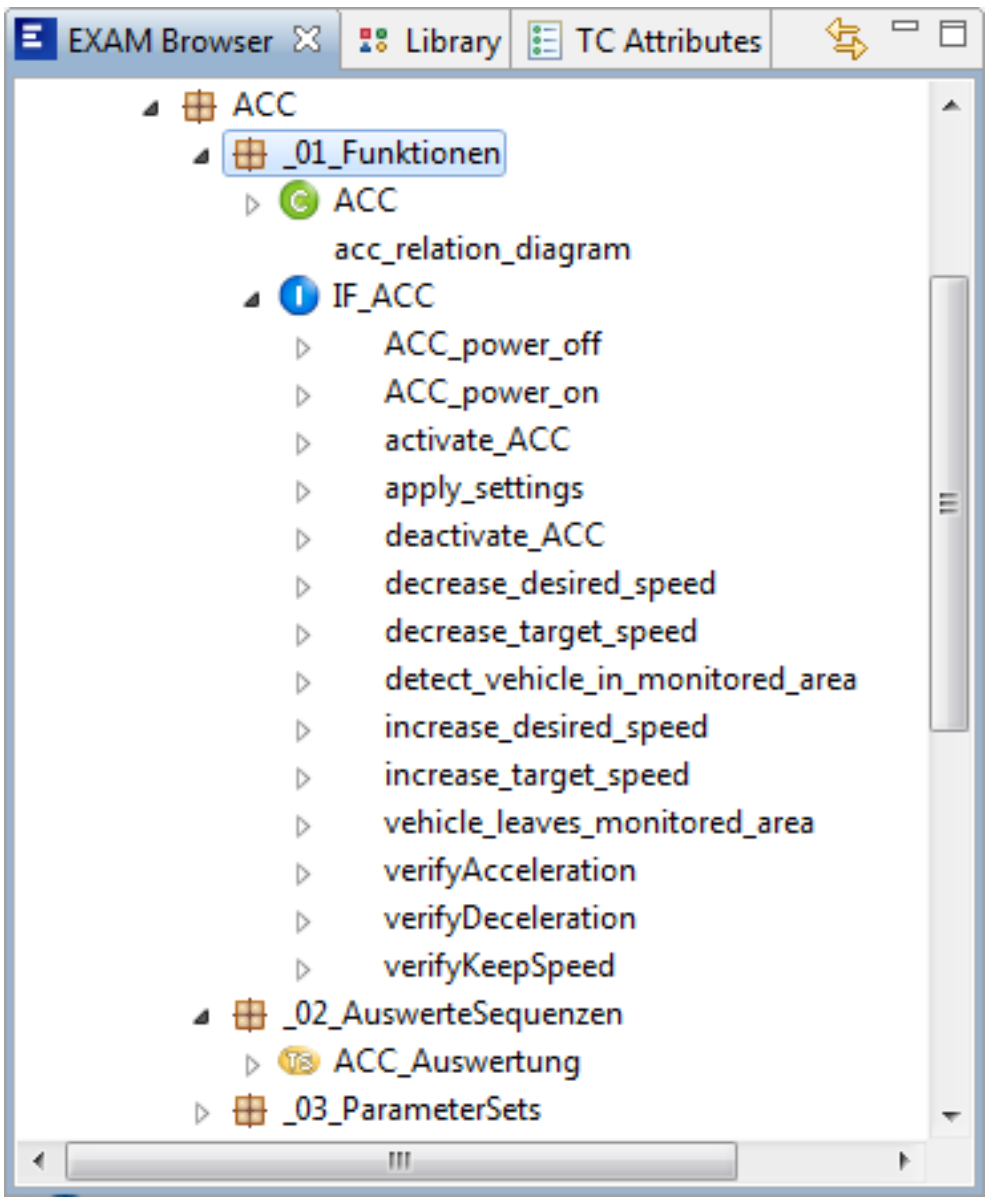
In dem Dialog werden Server und Modellname eingetragen und mit *OK* bestätigt. Die Verbindung wird dem *EXAM Browser* hinzugefügt. Der *EXAM Browser* stellt nach erfolgreicher Verbindung die verfügbaren Daten vom Server in der gewohnten Baumstruktur dar.



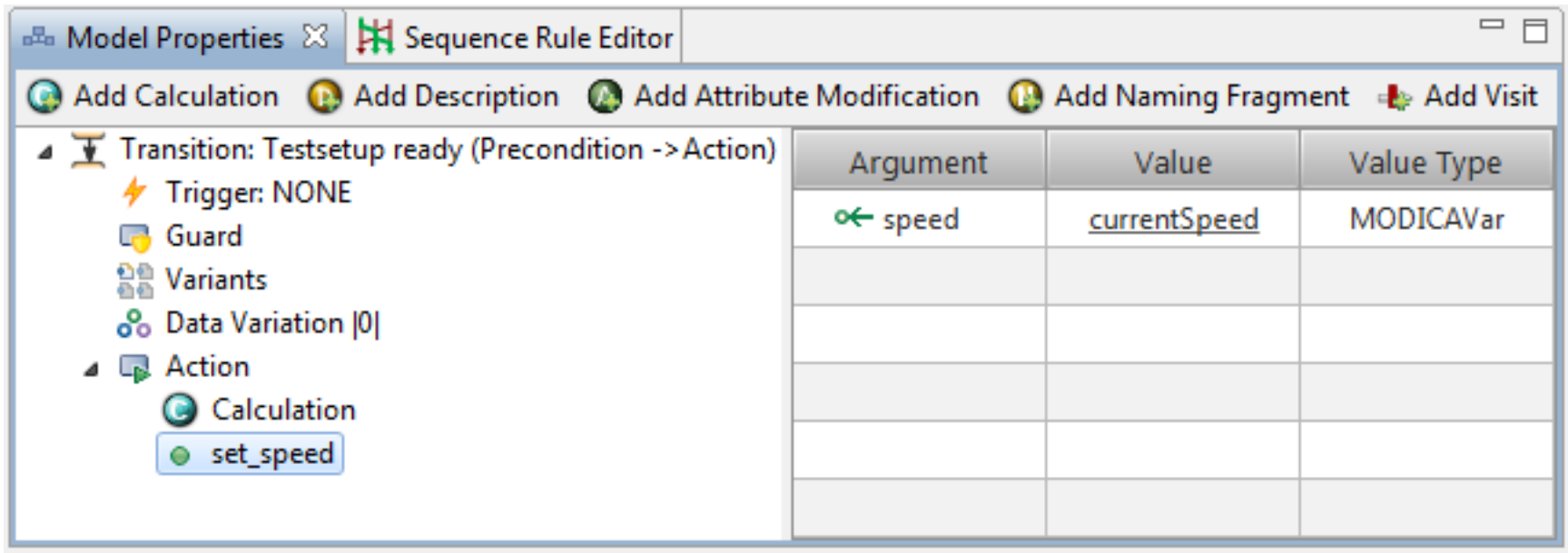
EXAM Objekte werden genau wie Anforderungen oder Variablen per Drag&Drop den Elementeigenschaften im *Model Properties View* hinzugefügt. Für das Tutorial sollten die Packages ACC und Vehicle_actions zur Verfügung stehen. Sollte im verwendeten EXAM Modell der Pfad sandbox.acc_Durchstich.ACC nicht vorhanden sein, kann er durch Import einer XMMDL Datei für EXAM erzeugt werden.

Verknüpfung von EXAM Objekten

Alle verfügbaren EXAM Objekte haben bestimmte Funktionen. Diese werden mit Zuständen und Transitionen im Modell verknüpft. Diese Objekte müssen vorab in EXAM angelegt sein. Zu jeder Funktion gibt es zugehörige Interfaces, die für die Zuweisung verwendet werden können. Die hier verwendeten EXAM Objekte sind speziell für das Tutorial angelegt worden.



Manchen EXAM Objekten werden, nachdem sie per Drag & Drop den Elementen hinzugefügt wurden, noch zusätzliche Werte zugewiesen. Dazu klickt man auf das verknüpfte Objekt im *Model Properties View*. Im rechten Teil der View erscheint eine Tabelle. Dort klickt man auf das leere Feld unter Value und kann einen Wert eintragen oder eine Variable per Drag & Drop einfügen.



Folgende weitere EXAM Objekte müssen dem Modell hinzugefügt werden:

ACC Statechart

Editor Element	EXAM Objekt
State Postcondition / *Entry Action*	ACC/02_AuswerteSequenzen/ACC_Auswertung

Substate "Precondition"

Editor Element	EXAM Objekt	Value
Transition set default values	Vehicle_actions/_01_Funktionen/Vehicle_performance/set_speed	Variable currentSpeed

Substate "Action"

Element	EXAM Objekt
Transition turn on	ACC/IF_ACC/ACC_power_on
Transition turn off	ACC/IF_ACC/ACC_power_off.

Substate "Turned On"

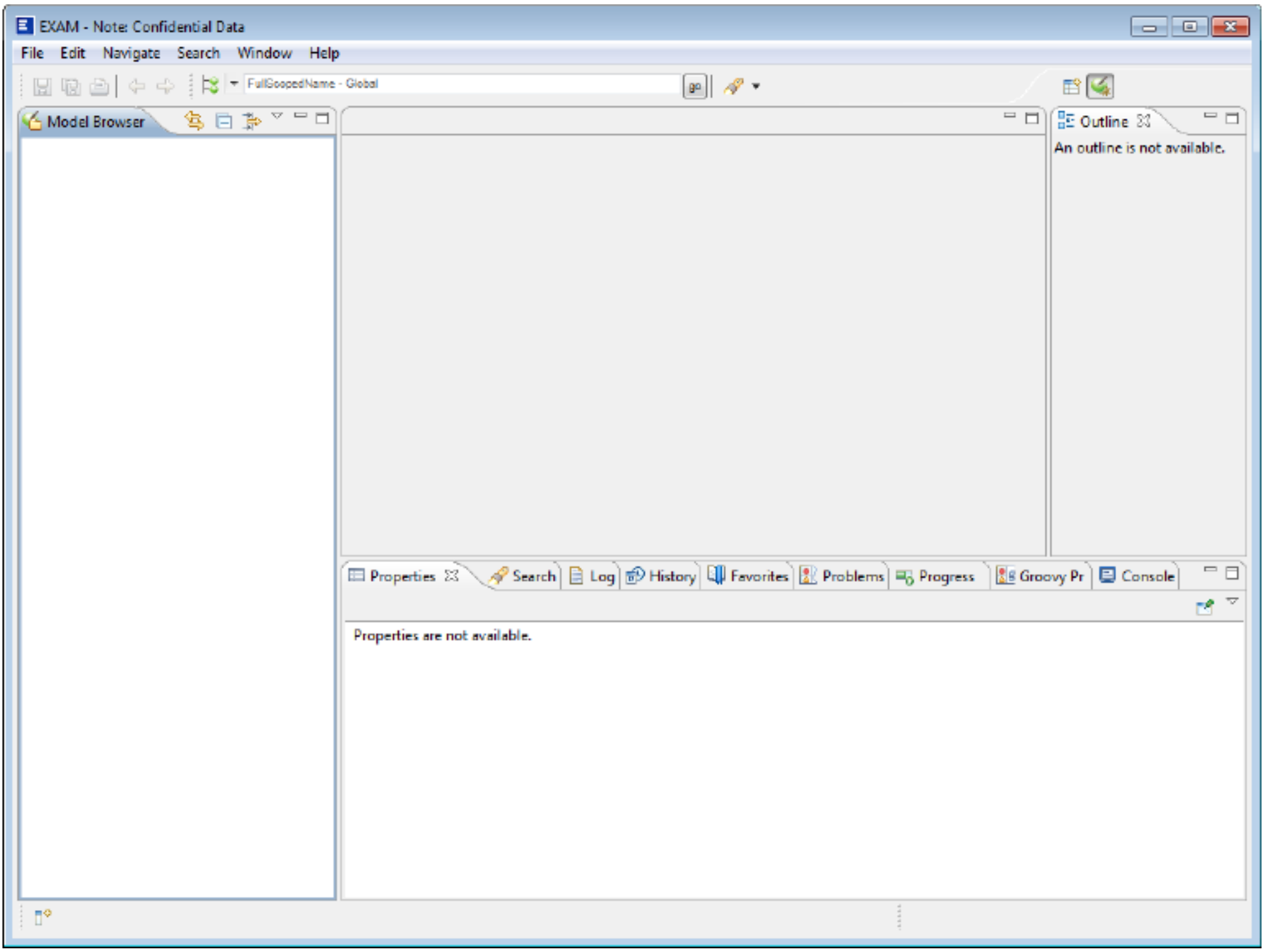
Element	Anforderung	Value
Transition recognize other vehicle	ACC/IF_ACC/detect_vehicle_in_monitored_area	-
Transition recognize leaving vehicle	ACC/IF_ACC/vehicle_leaves_monitored_area	-
Transition recognize acceleration	ACC/IF_ACC/increase_target_speed	5
Transition recognize deceleration	ACC/IF_ACC/decrease_target_speed	5
Transition deactivate	ACC/IF_ACC/deactivate_ACC	-
Transition activate	ACC/IF_ACC/activate_ACC	-

Damit ist das Modell in MODICA für den Export nach EXAM bereit. Als nächstes wird EXAM für den Import vorbereitet.

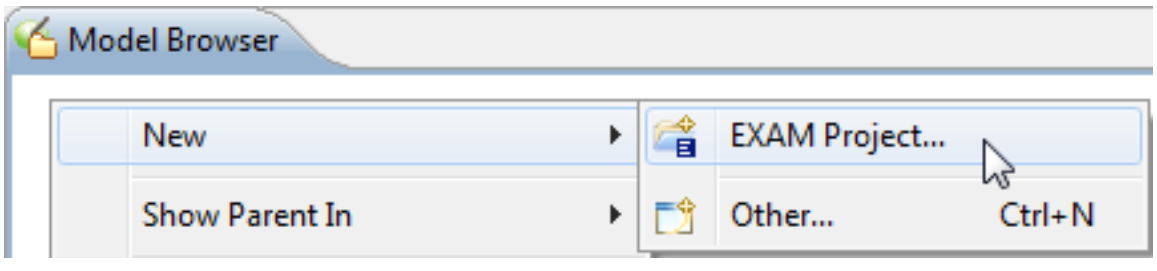
Schritt 6 - Vorbereitung in EXAM

Damit die Testfälle nach EXAM exportiert werden können, müssen in EXAM ebenfalls einige Dinge vorbereitet werden bzw. Modellelemente vorhanden sein.

Dazu wird der **EXAM Client** gestartet.

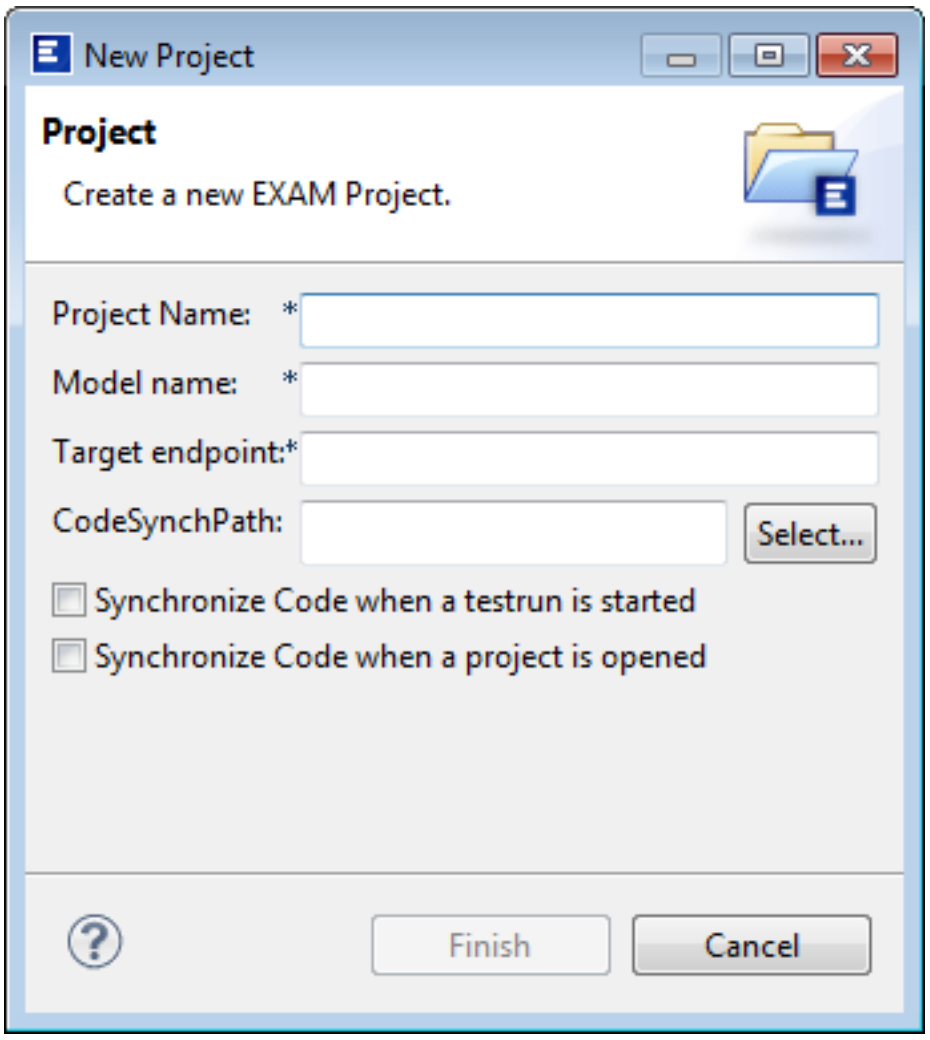


Der *Model Browser* listet die verbundenen Server und deren Inhalte auf. Um eine Verbindung zum Server herzustellen, öffnet man das Kontextmenü und wählt *New → EXAM Project*.

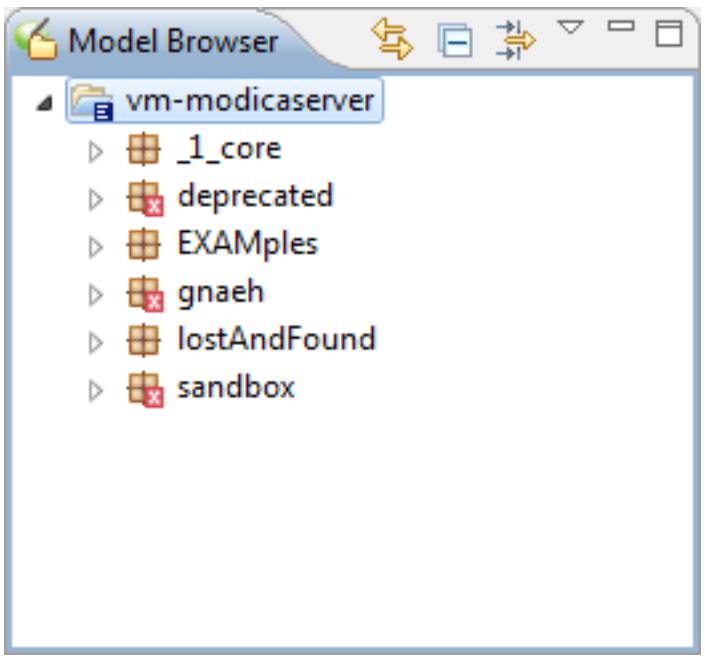


Es öffnet sich ein Dialog, in dem die Verbindungsdaten eingetragen werden.

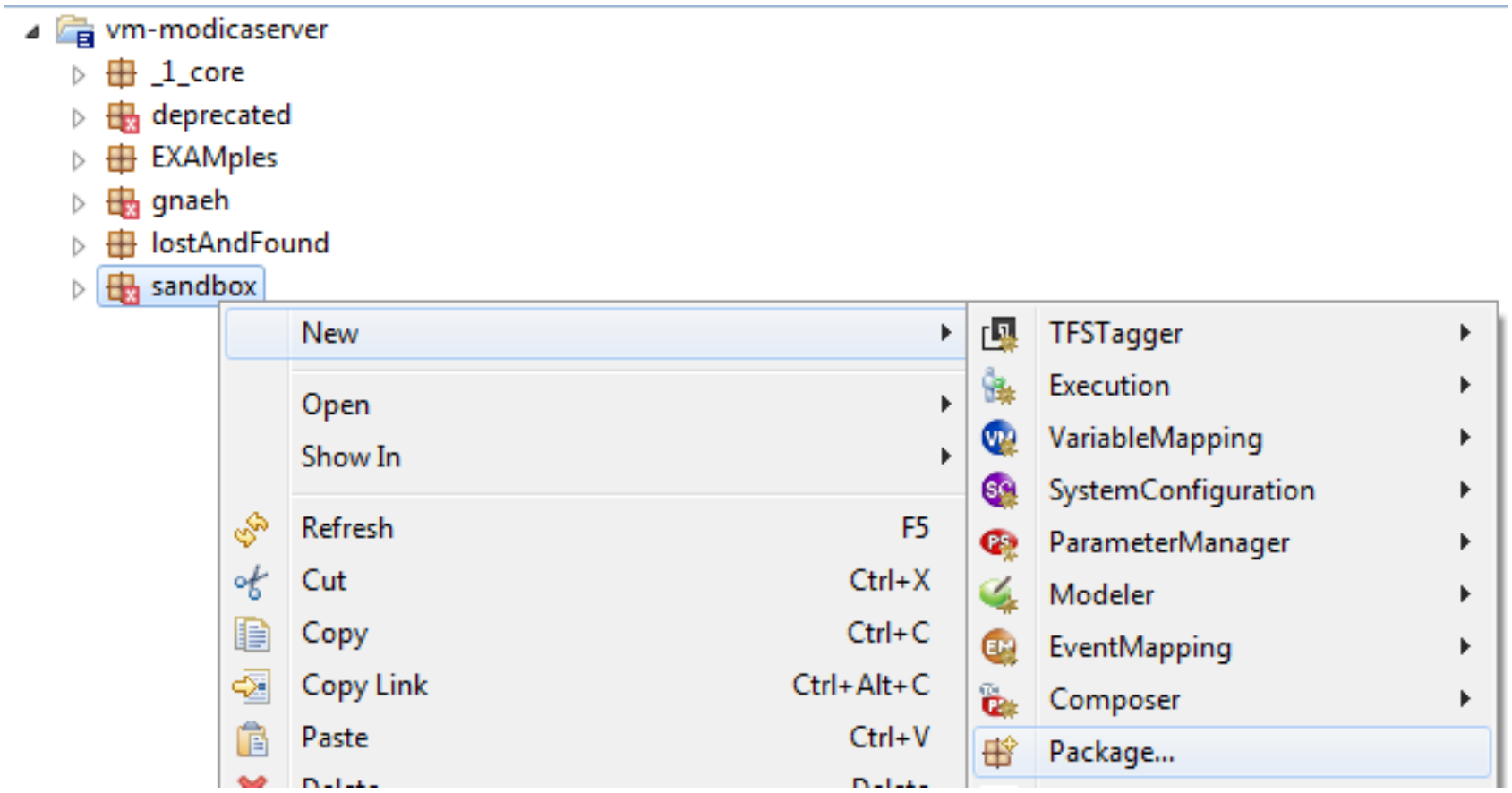
- *Project Name*: Dieser Name ist frei wählbar und wird als Bezeichnung im *Model Browser* angezeigt.
- *Model name*: Dies ist der Name des Models, der auch in MODICA verwendet wurde.
- *Target endpoint*: Hier wird die Adresse zum Server eingetragen wie sie auch in MODICA verwendet wurde.



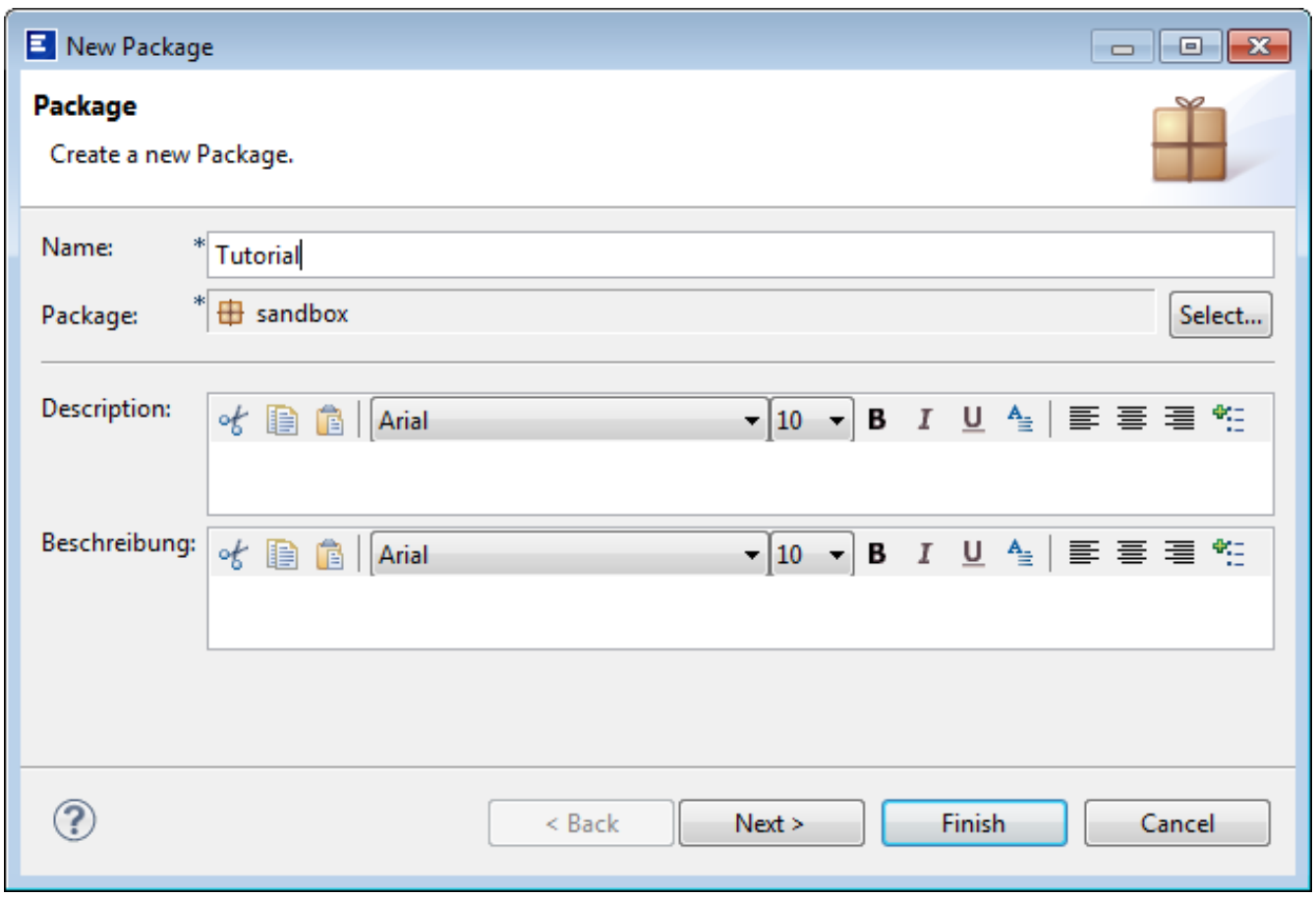
Im *Model Browser* erscheint daraufhin das Projekt und es kann geöffnet werden. An dieser Stelle erfolgt die Authentifizierung. Diese kann abhängig von der Einrichtung per PKI Karte stattfinden. Details hierzu finden sich in der Dokumentation von EXAM.



Für die Testfälle und Ergebnisse wird ein neues *Package* erstellt. Dazu öffnet man das Kontextmenü und wählt *New* → *Package*.



In dem folgenden Dialog wird der Name für das *Package* angegeben. Man trägt Tutorial ein und klickt auf Finish.



Das Package Tutorial wird dem ausgewählten Package hinzugefügt.

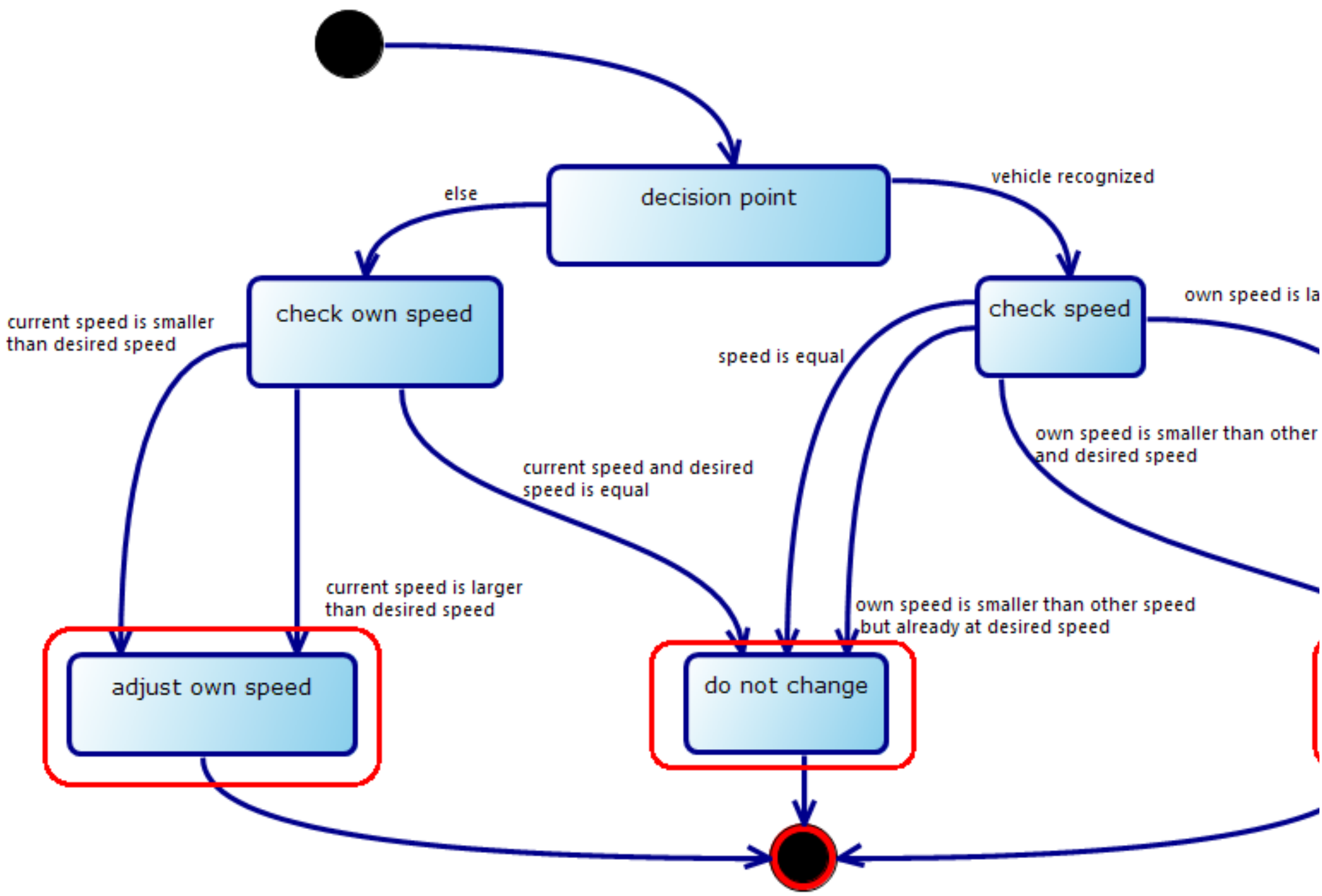
Schritt 7 - Hinzufügen der checkSpeed Funktion und Testfallgenerierung

Bisher enthält das Benutzungsmodell lediglich Stimuli, aber keine Überprüfungen für das tatsächliche Verhalten. Diese sind aber für einen sinnvollen Test zwingend erforderlich und werden im Folgenden in das Modell integriert. Sie liefern in den generierten EXAM Testfällen die EXAM Reportinhalte.

Es gibt verschiedene Möglichkeiten, diese Prüfungen durchzuführen (siehe Modellierungsrichtlinien). Eine der einfachsten wird in diesem Beispiel verwendet: sofortige Überprüfung während der Ausführung. Gerade bei asynchronen Abläufen und komplexeren oder voneinander abhängigen Prüfungen muss in der Praxis häufig auf den Analyse-Service zurückgegriffen werden.

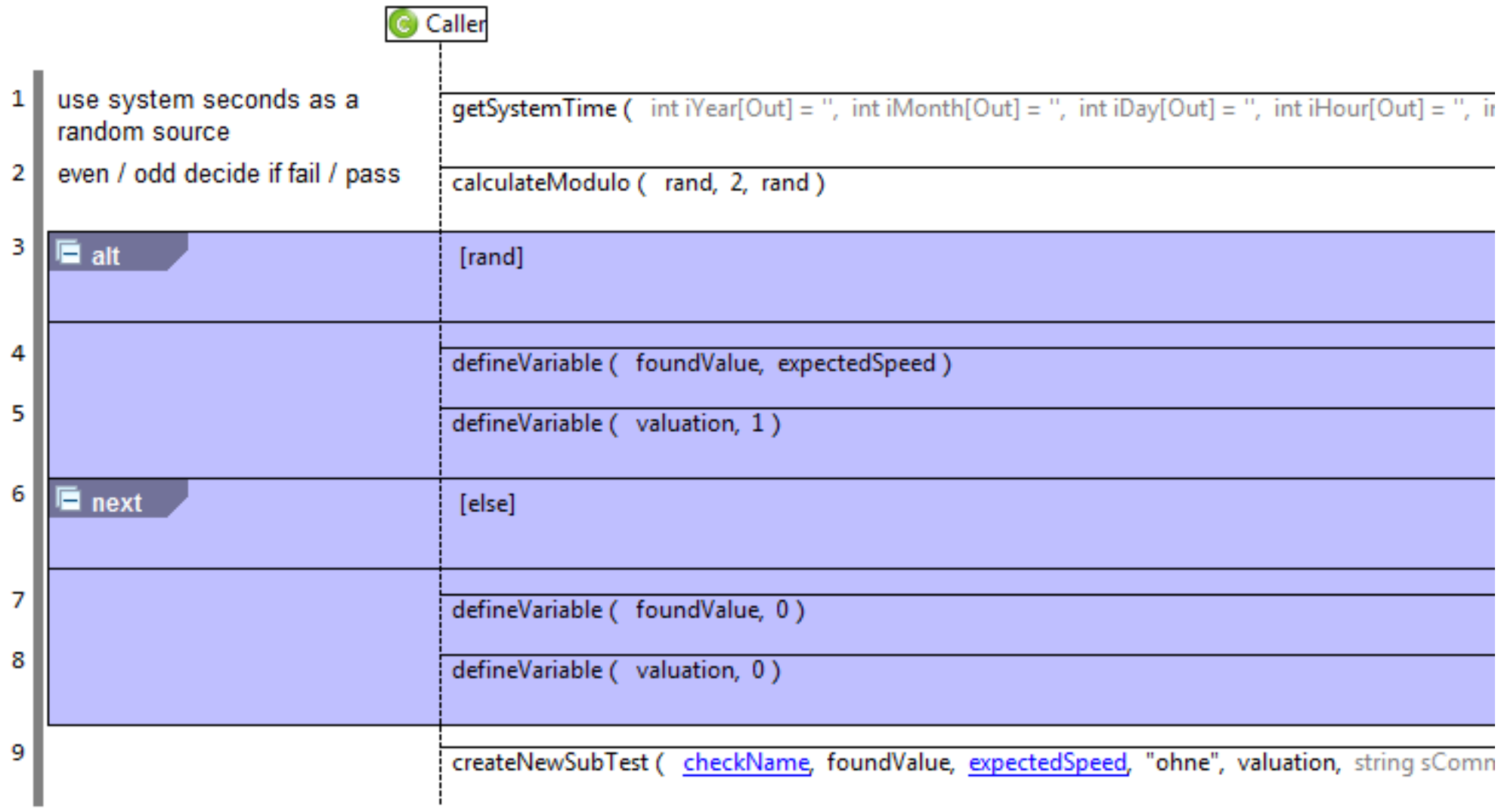
Ein möglicher Angriffspunkt für die sofortige Prüfung ist im ACC Modell der Substate active. In diesem ist je nach dort besuchtem *State* bekannt, ob das System die Sollgeschwindigkeit verändert und warum es das tut.

An den drei letzten *States* dieses Diagramms wird jeweils ein Aufruf der EXAM Funktion `checkSpeed()` eingefügt. Diese gleicht die tatsächliche Geschwindigkeit mit einem Sollwert ab und erzeugt entsprechend des Ergebnisses die *SubTestValuation* im EXAM-Report.

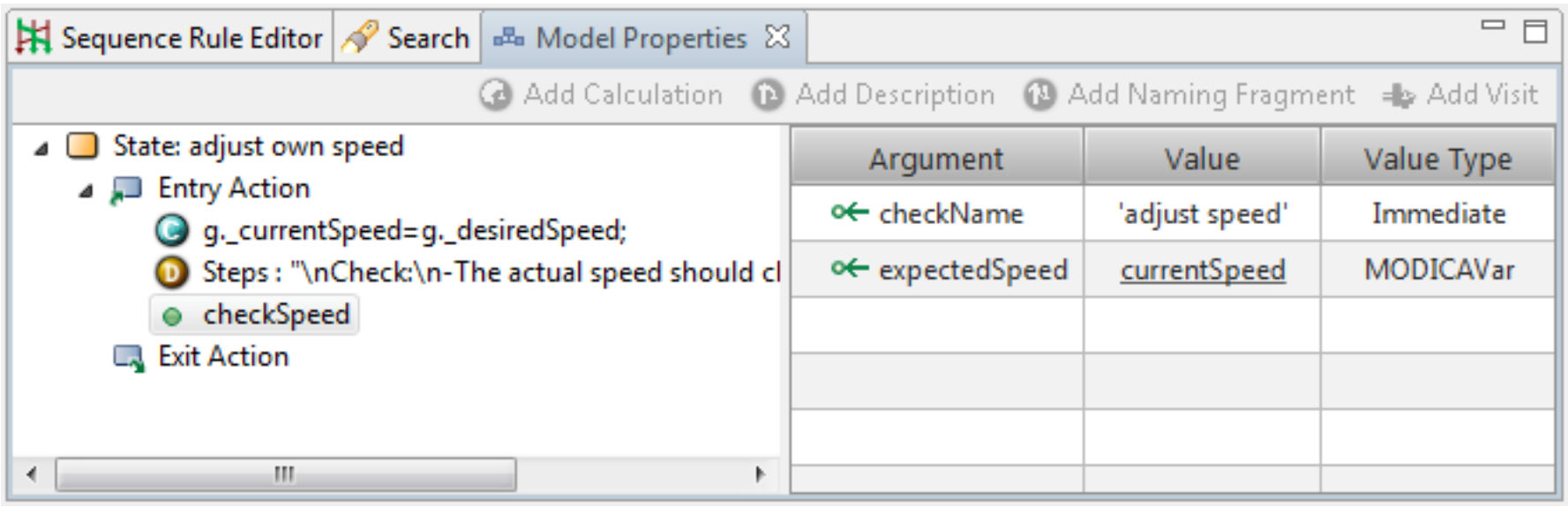


Im Rahmen dieses Tutorials wird nicht erklärt, wie eine solche Funktion in EXAM entworfen werden kann. Eine ohne Prüfstand ausführbare Implementierung dieser Check-Funktion ist in der ACC Funktionsbibliothek vorhanden. Tatsächlich ist es für die hier verwendete Offline-Ausführung auch nicht ohne weiteres möglich, die aktuelle Geschwindigkeit zu

erfahren, da es im einfachsten Fall wie hier noch gar kein SuT gibt. Stattdessen ist diese Funktion mit einem Pseudo-Zufallszahlengenerator implementiert, der in 50% der Prüfungen das Ergebnis ablehnen wird. Der Vollständigkeit halber ist die hier genutzte Stub-Implementierung zur Offlineausführung im Folgenden abgebildet:

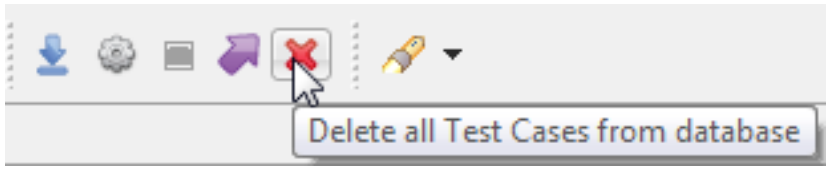


Um diese Funktion nun tatsächlich aufzurufen, wird sie im *Model Properties View* der *Entry Action* für jeden der drei States adjust own speed, do not change und adjust speed eingefügt und konfiguriert

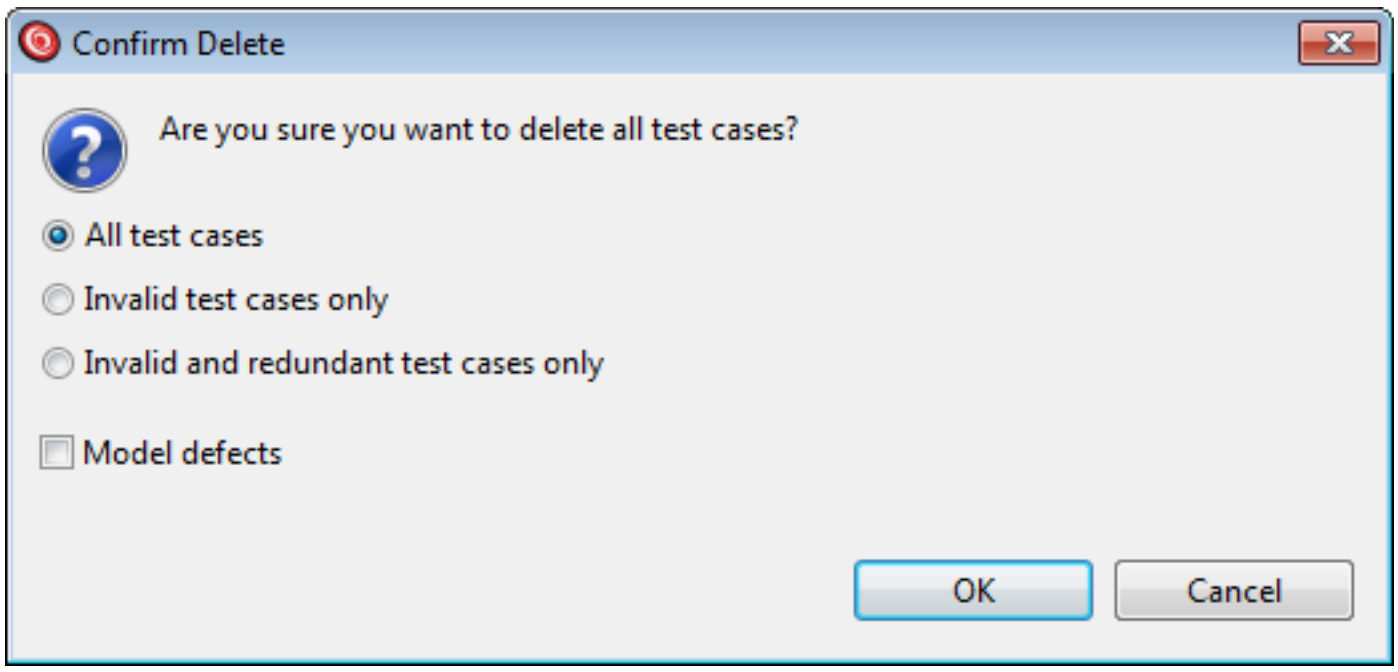


Hierbei ist darauf zu achten, dass *checkSpeed* nach der jeweiligen *Calculation* (nicht vorhanden in do not change) aufgerufen wird, damit es die richtige Geschwindigkeit übergeben bekommt. Weiterhin ist das Argument *checkName* mit einem String zu befüllen, der die konkrete Aufrufstelle markiert. Dieser String wird später auch bei der Ausführung des Testfalls als SubTest in den Reports auftauchen.

Dann speichert man das Modell und wechselt in die Perspektive *Test Generation*. Da eine Änderung am Modell vorgenommen wurde, wird es erneut geladen. Dazu klickt man auf den Button *Load Model* im *Progress Panel*. Die Testfälle, die vorher erstellt wurden, werden gelöscht. Dazu klickt man in der Toolbar auf das rote *X*.



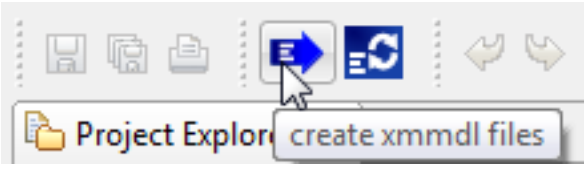
Gegebenenfalls öffnet sich ein Dialog, der die Perspektive wechseln möchte. Diesen schließt man mit Cancel. Der Dialog *Confirm Delete* bietet die Möglichkeit nur einen Teil der Testfälle zu löschen. Hier bleibt *All test cases* ausgewählt und man klickt auf *OK*.



Sind die Testfälle gelöscht, klickt man im *Progress Panel* auf den Button *Generate test cases*. Die neuen Testfälle, jetzt mit den Aufrufen den Check-Funktion bestückt, erscheinen in der View *Test Cases*.

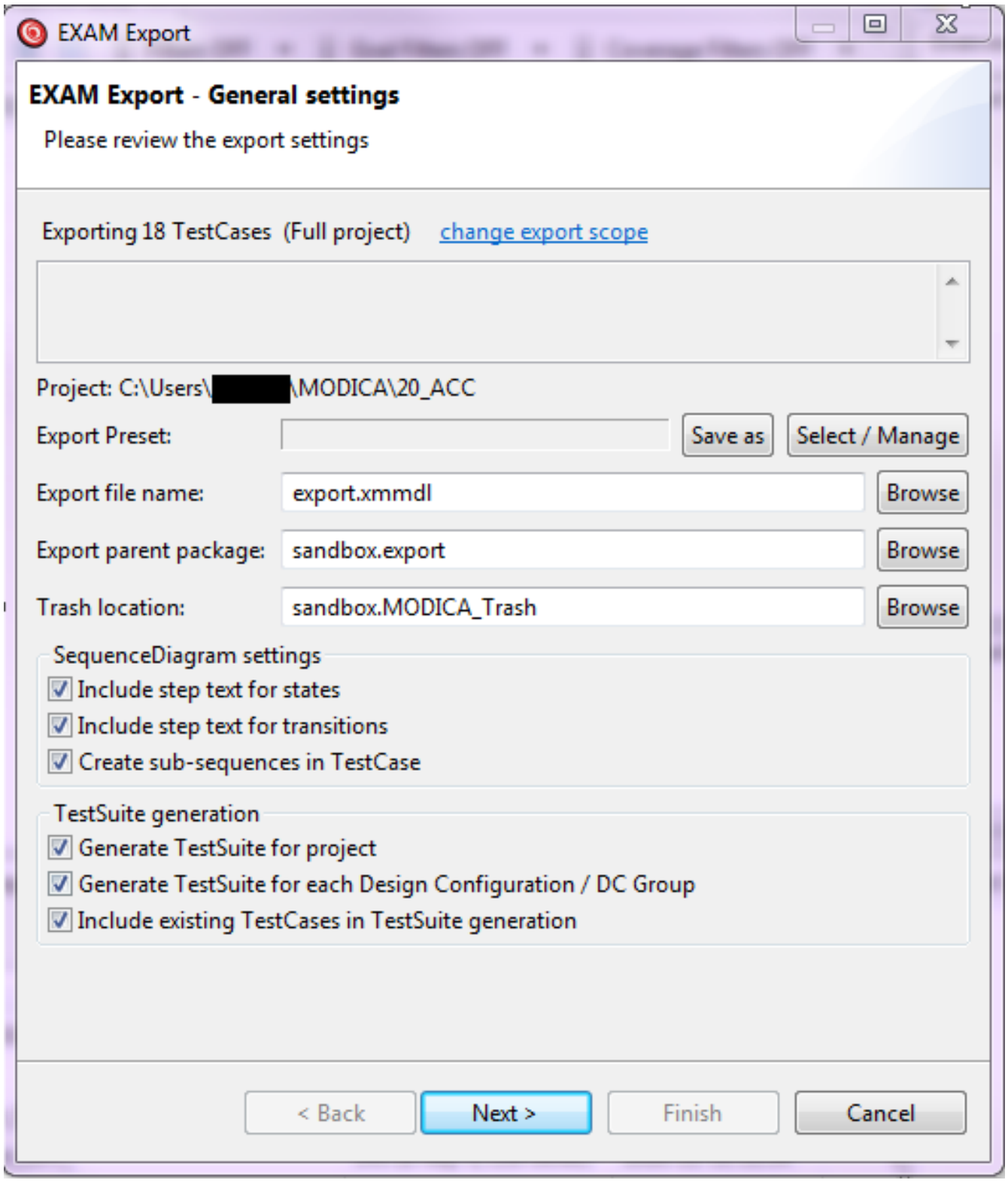
Schritt 7a - Erzeugung der XMMDL Datei

Der Exporter generiert eine *XMMDL Datei* mit den Testfällen und allen relevanten Daten, die in EXAM importiert werden kann. Um die *XMMDL Datei* zu generieren, klickt man in der Toolbar auf den großen blauen Pfeil *create xmmdl files*.



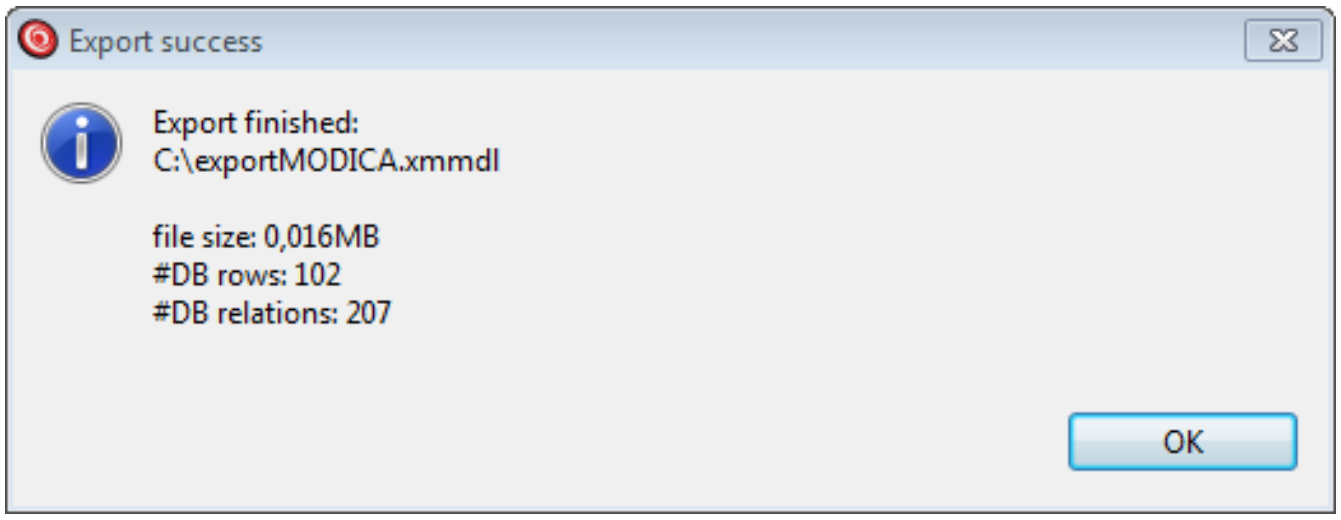
Im darauf folgenden Dialog werden die Exporteinstellungen angepasst.

- Über den Link change export scope können individuell die zu exportierenden Testfälle ausgewählt werden.
- *Export file name*: Das ist der lokale Speicherort für die exportiere Datei. Dieser kann beliebig gewählt werden.
- *Export parent package*: Hier wird das *Package* in EXAM ausgewählt, in das die Testfälle exportiert werden sollen. Dazu klickt man auf den Button *Browse* und navigiert in EXAM zum Package Tutorial.
- *Trash location*: Die trash location ist per default gesetzt und kann beibehalten werden. Testfälle, die durch einen neuen Import ersetzt werden, werden in diesen Ordner verschoben.



Mit einem Klick auf *Finish* startet man den Export. In einem Dialog wird der Fortschritt angezeigt, während die Datei geschrieben wird. Natürlich kann die exportierte Datei auch an andere Stellen gespeichert werden als hier genannt.

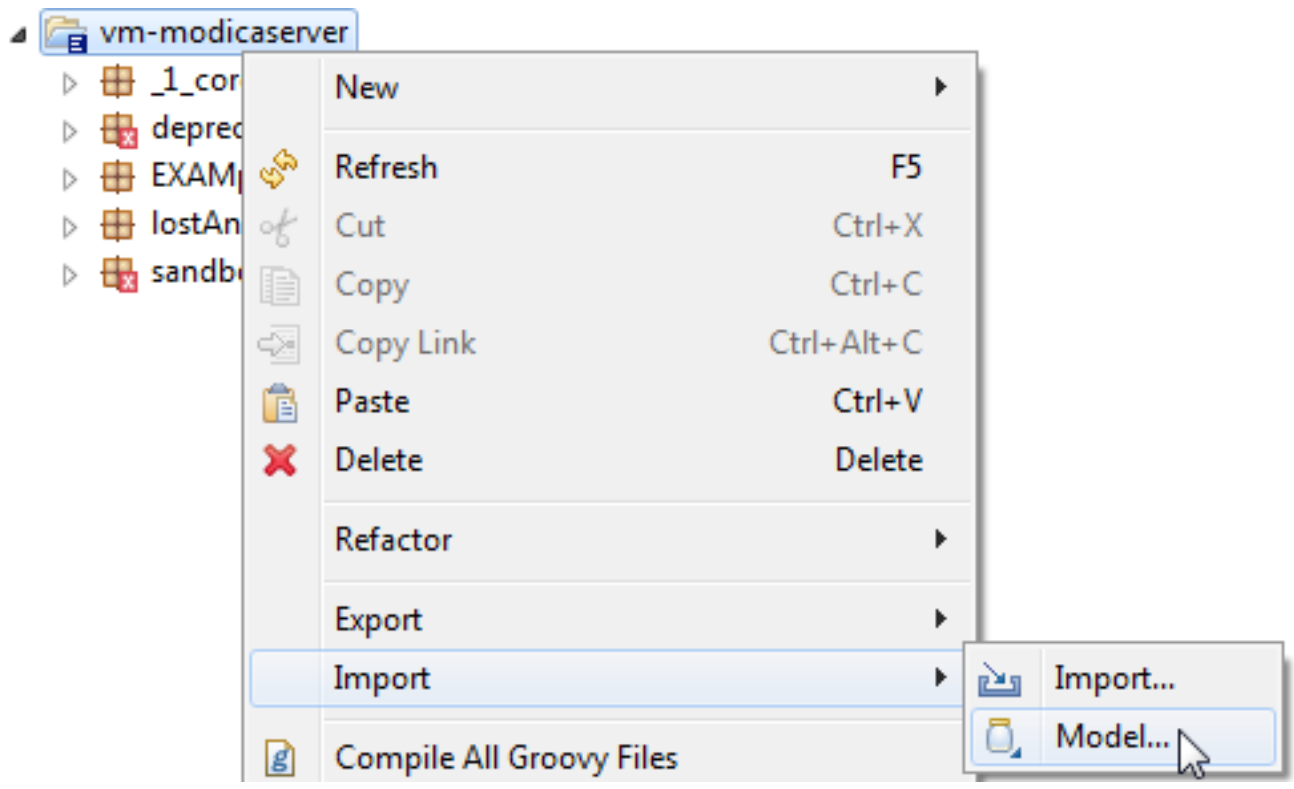
Der Export kann, je nach Anzahl der Testfälle, einige Minuten dauern. Wenn er fertig ist, wird eine Übersicht über den Umfang der Datei angezeigt.



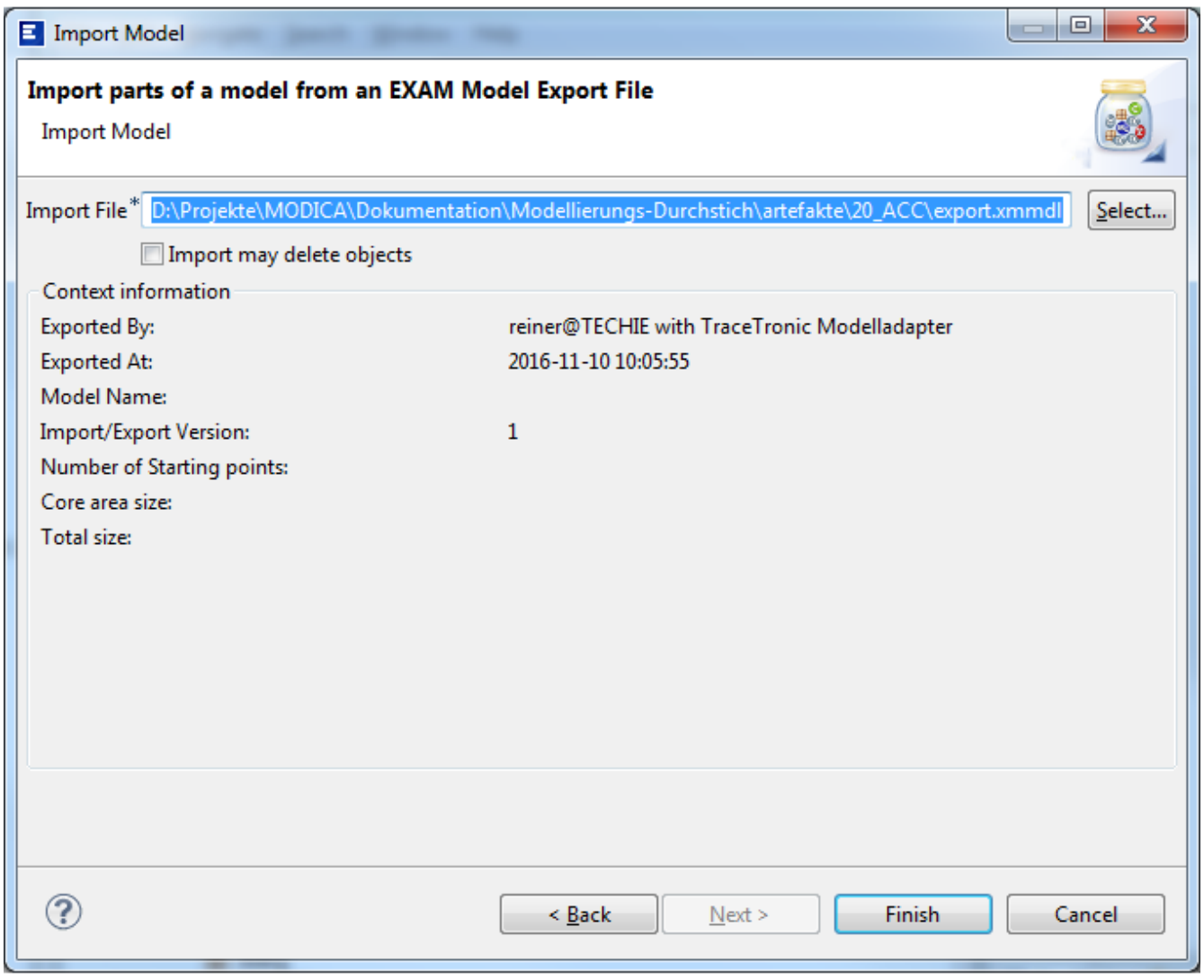
Damit ist die Arbeit in MODICA beendet. Nun werden die Testfälle in EXAM importiert, ausgeführt und die Ergebnisse betrachtet. Bei erneuter Generierung und zusätzlichen Importen in EXAM werden **Test Cases**, sofern sich der Name nicht geändert hat, aktualisiert oder verworfen. Mehr dazu findet sich im Kapitel Export von Tetsfällen (EXAM spezifisch).

Schritt 8 - Import der Testfälle und Vorbereitung der Ausführung in EXAM

Um die Testfälle zu importieren, öffnet man in EXAM mit Rechtsklick das Kontextmenü des *Model Browsers* und wählt *Import → Model*.



Es öffnet sich der *Import Wizard*. In diesem Dialog klickt man auf *Select*, um die erstellte *XMMDL Datei* auszuwählen.

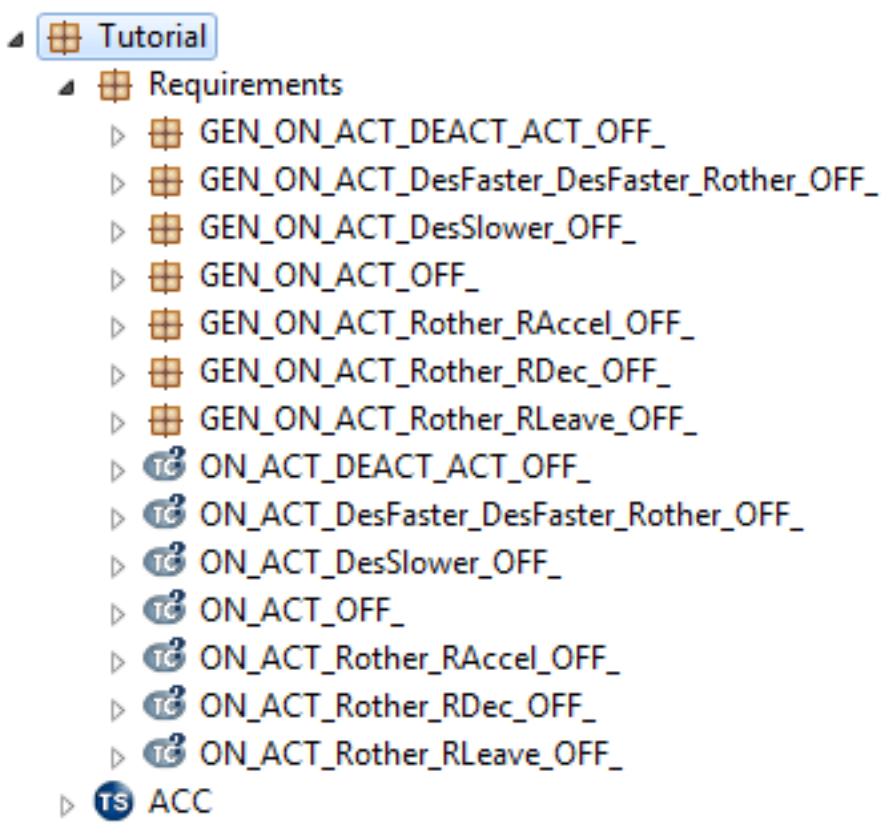


Mit einem klickt auf *Finish* schließt man den Dialog und importiert die Testfälle.

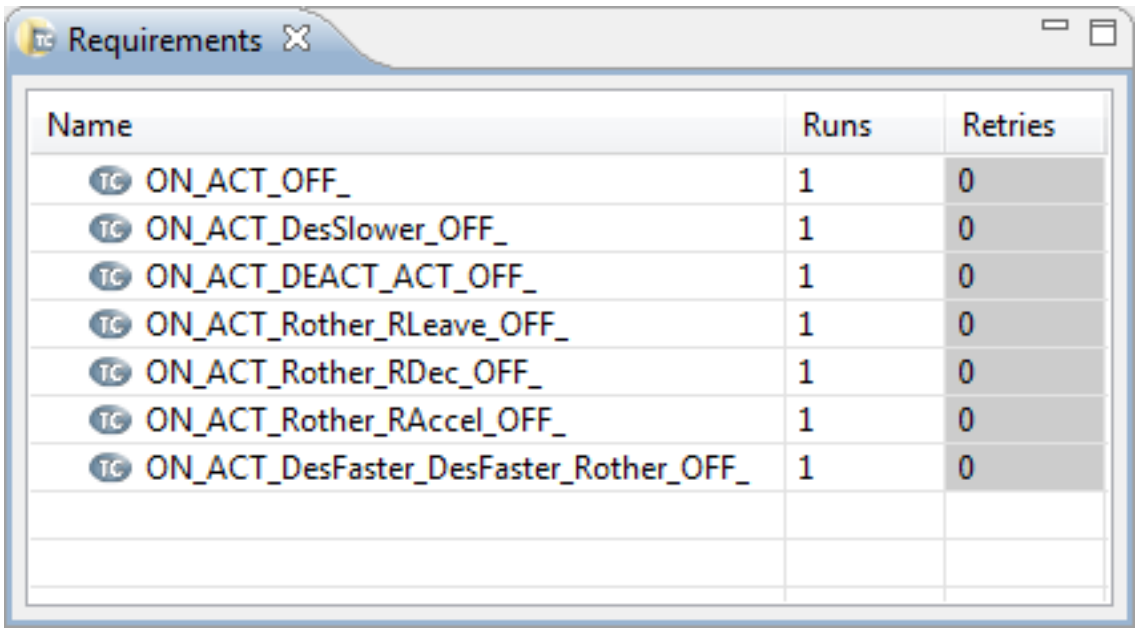
Achtung: Die Funktion *Import may delete objects* darf niemals ausgewählt werden. Es können unbeabsichtigt fremde Daten gelöscht werden. Mehr dazu findet sich im Handbuch .

Die Testfälle werden dem Package Tutorial hinzugefügt. Dort wird ein weiteres *Package* mit dem Namen der *Design*

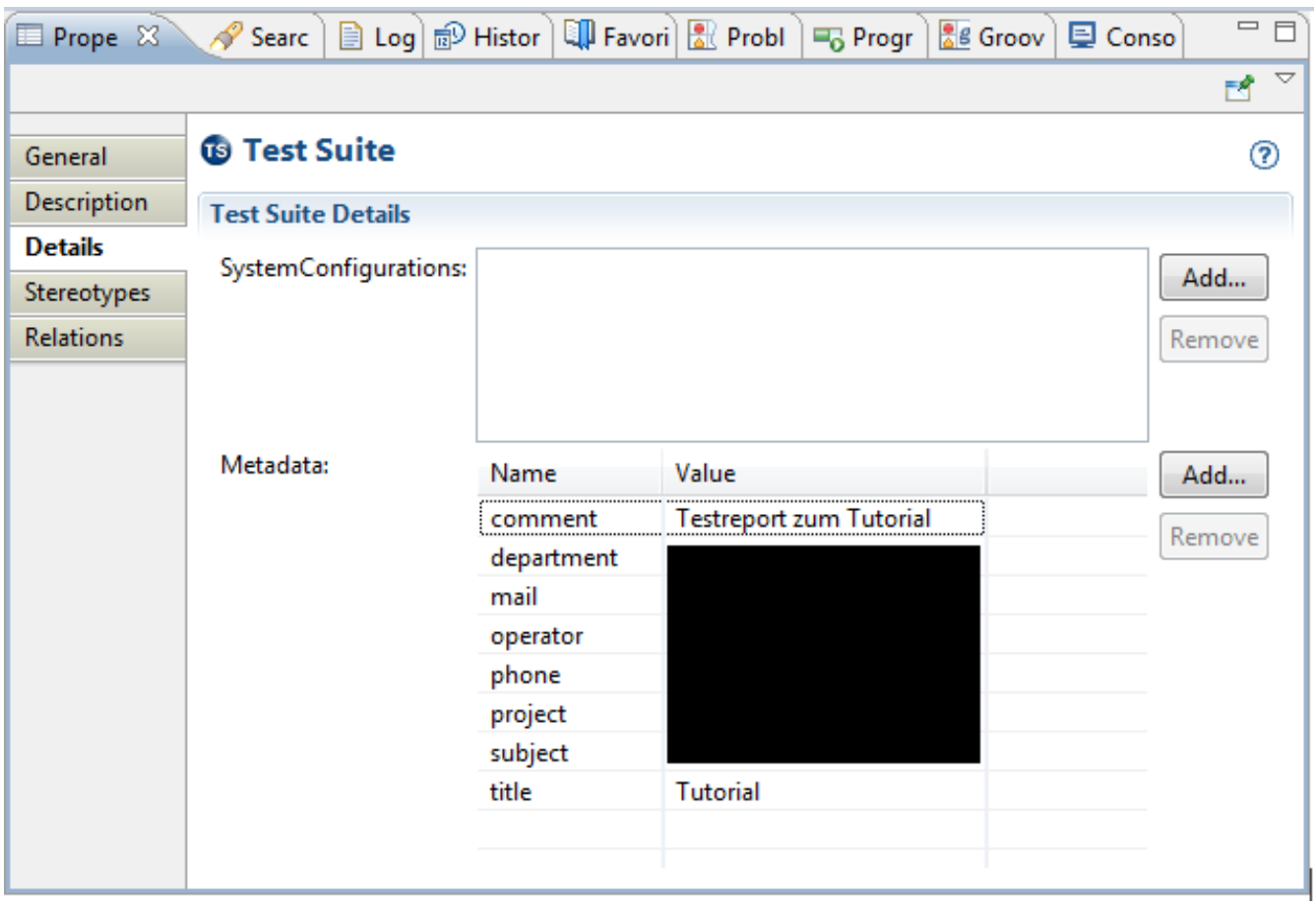
Configuration aus MODICA erstellt und befüllt. Sollten die Testfälle nicht sichtbar sein, kann man mit Rechtsklick das Kontextmenü öffnen und *Refresh* auswählen. Dadurch wird der Inhalt des *Model Browsers* neu geladen.



Zusätzlich erscheint eine Liste der Testfälle in der Testgruppe Requirements.

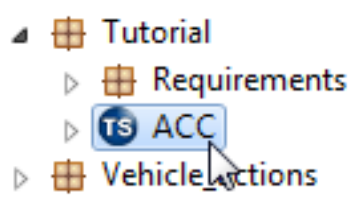


Zur Vorbereitung der späteren Testreports lässt sich an dieser Stelle die *Test Suite* bearbeiten, damit relevante Daten später zur Verfügung stehen. Dazu wählt man die Test Suite ACC aus und wechselt in die *Properties View*. Dort können unter *Details* Metadaten eingetragen werden, die später im Report erscheinen.



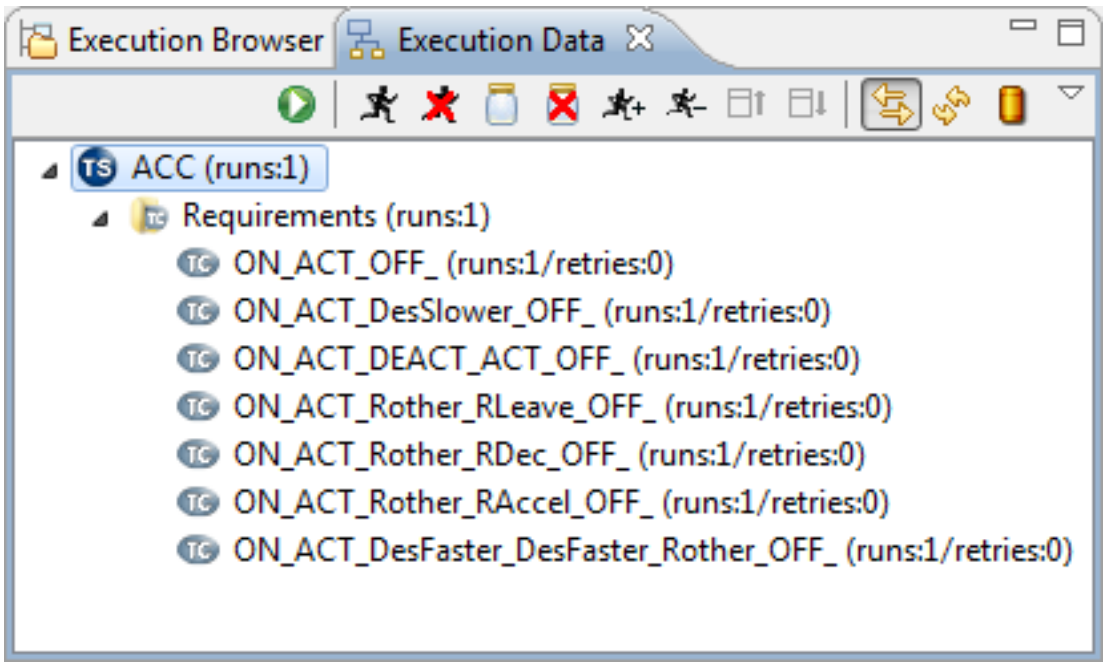
Es ist sinnvoll, das zu diesem Zeitpunkt zu tun, da man später ansonsten die *Perspektive* wechseln müsste, um es nachzutragen. Als nächstes fügt man die *Test Suite* einer Testkampagne hinzu, um die Ausführung vorzubereiten.

Dazu wird die Perspektive *Testrunner* benötigt. Diese öffnet sich mit einem Doppelklick auf die Test Suite ACC.

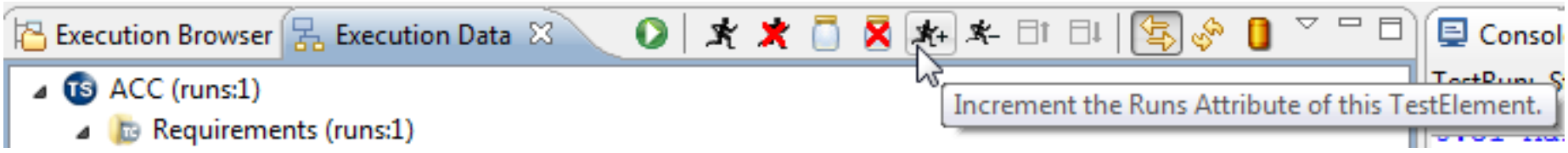


Die Perspektive *Testrunner*, enthält die View *Execution Data*, in der die Testfälle der gewählten *Test Suite* aufgelistet sind und bearbeitet werden können. In der Konsole erscheint der Output der Testdurchläufe während der Ausführung.

Die Testgruppe Requirements lässt sich aufklappen und zeigt die vorhandenen Testfälle. Hinter den Testfällen steht die Anzahl der Durchläufe.



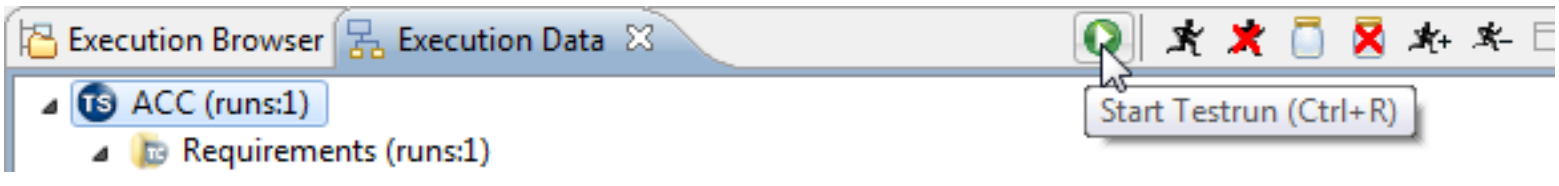
An dieser Stelle kann die *Test Suite* weiter editiert werden. Zum Beispiel lassen sich weitere Testdurchläufe mit dem *Runner-Icon* hinzufügen. Dazu klickt man auf das Icon, das ein rennendes Männchen mit einem Plus-Zeichen zeigt.



Des Weiteren lassen sich auch einzelne Tests deaktivieren, die nicht benötigt werden. Als nächstes werden die Testfälle ausgeführt.

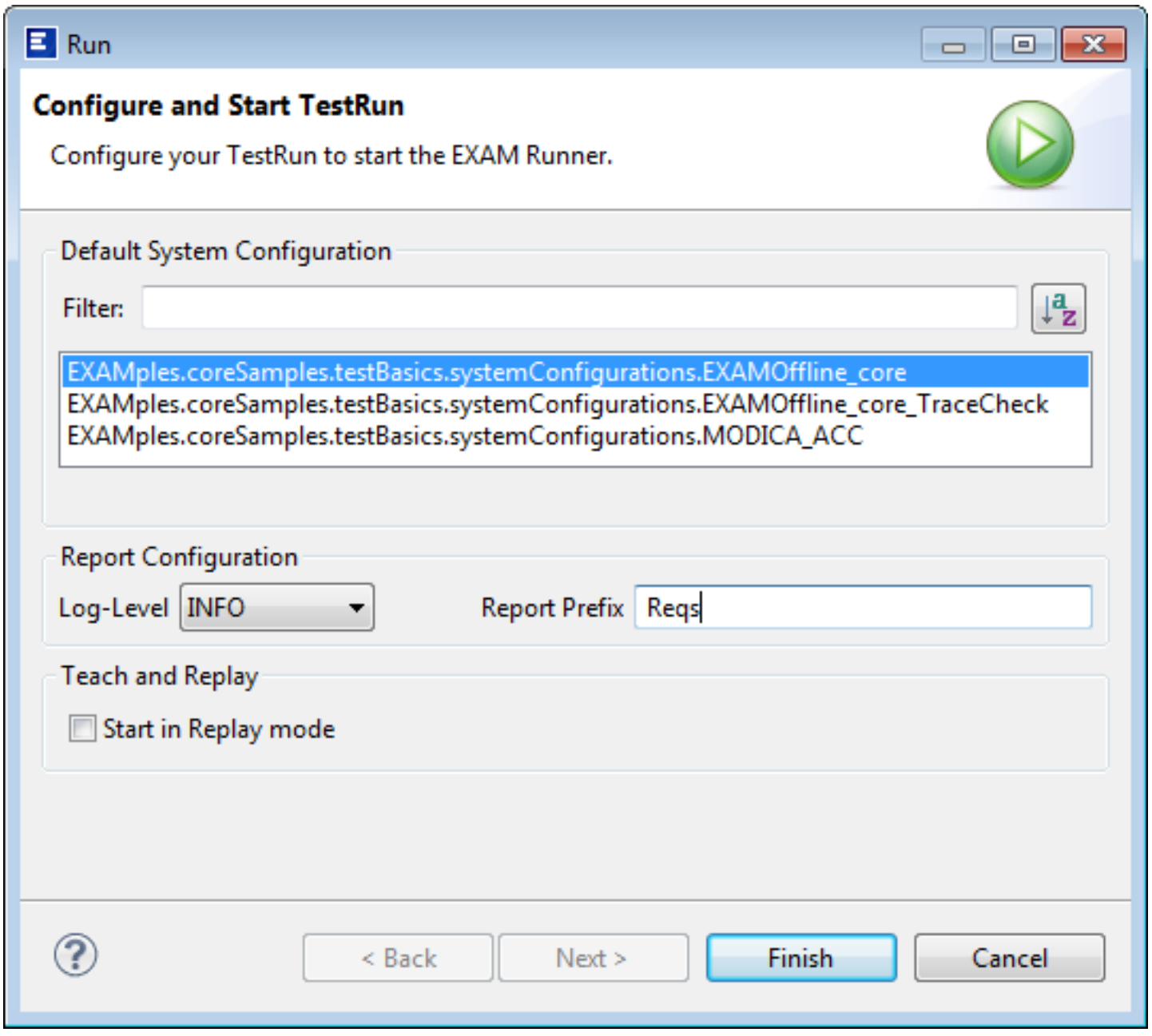
Schritt 9 - Ausführung und Auswertung in EXAM

Der Testdurchlauf wird mit einem Klick auf den grünen *Play-Button* gestartet. Es öffnet sich ein Dialog, in dem die Einstellungen für den Testlauf eingegeben werden.



Zuerst wählt eine bereits vorhandene Systemkonfiguration für die Offline-Ausführung aus, in diesem Fall *EXAMOffline_core*. Falls keine solche vorhanden ist, finden sich weitere Informationen in der EXAM Dokumentation.

Zusätzlich wird ein *Report Prefix* gesetzt, um die Zuordnung der Testreports zu erleichtern.



Klickt man auf *Finish* startet die Ausführung.

In der Konsole kann man zusehen, welche Testschritte ausgeführt werden und welche Testresultate ausgegeben werden.

```
Console
TestRun:  Started at: Sat Sep 26 21:19:37 CEST 2015
2015-09-26 21:19:43 MainThread ... message: ACC wird eingeschaltet
2015-09-26 21:19:43 MainThread ... append a new sub-test: activate ACC  -  pass
2015-09-26 21:19:43 MainThread ... append a new sub-test: speed stays   50   50   ohne   pass
2015-09-26 21:19:43 MainThread ... append a new sub-test: detect vehicle in monitored area  -  pa
2015-09-26 21:19:43 MainThread ... append a new sub-test: speed change behind vehicle   49   49
2015-09-26 21:19:43 MainThread ... append a new sub-test: increase target speed of vehicle  -  pa
2015-09-26 21:19:43 MainThread ... append a new sub-test: speed change behind vehicle   50   50
2015-09-26 21:19:43 MainThread ... append a new sub-test: increase target speed of vehicle  -  pa
2015-09-26 21:19:43 MainThread ... append a new sub-test: speed stays   50   50   ohne   pass
2015-09-26 21:19:43 MainThread ... append a new sub-test: ACC power off  -  pass
```

Die Testreport-Daten werden auf der Festplatte gespeichert und können anschließend betrachtet werden.

Insgesamt gibt es zwei Dokumente, die sich auswerten lassen. Zum einen die *Logfiles*, welche sich auf der Festplatte befinden und zum anderen der Inhalt des *Reportmanagers*.

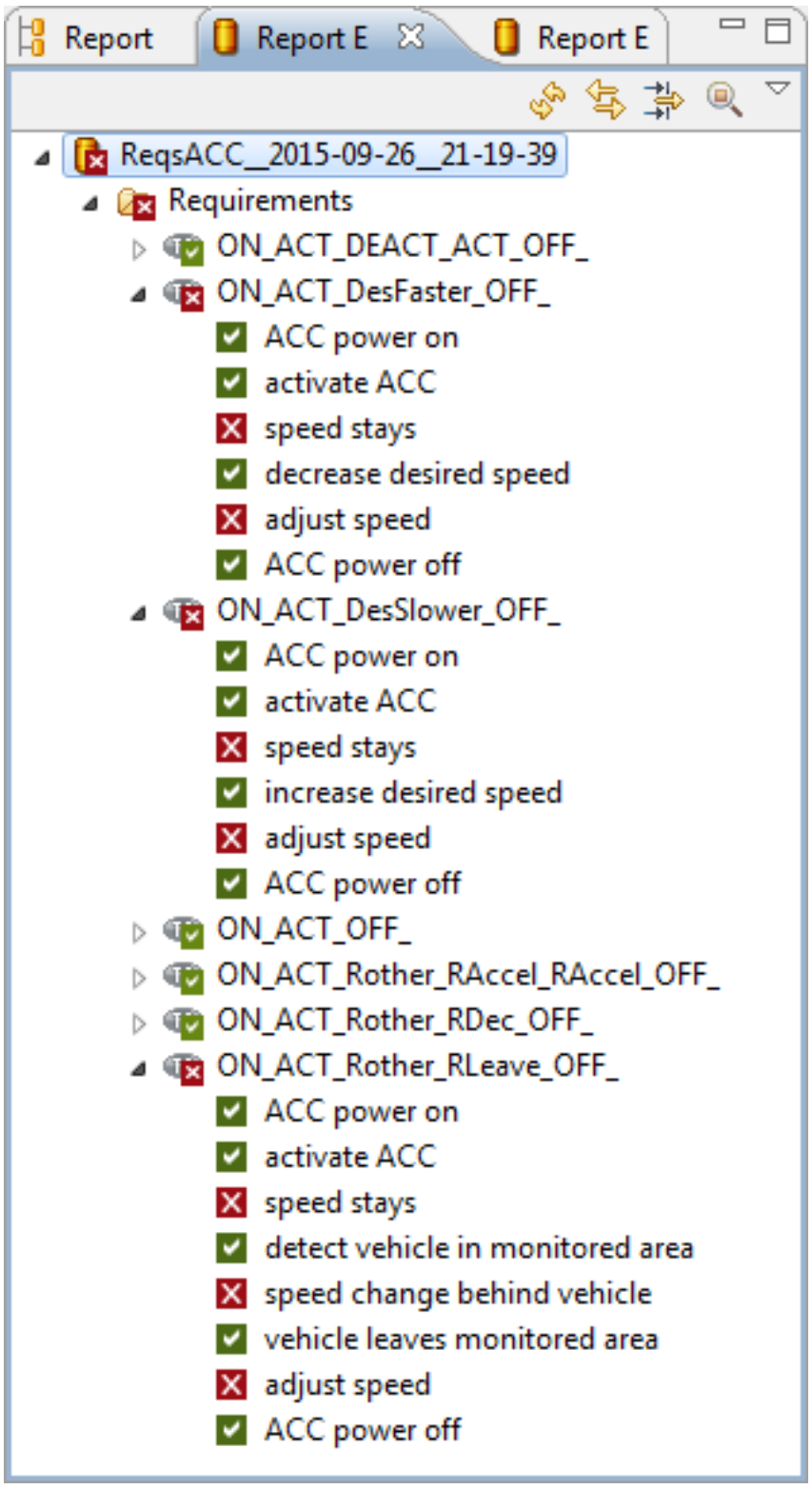
Die Logfiles findet man im Installationsverzeichnis unter `...\userData\reports\sandbox\Tutorial`. Dort lässt sich die logs.zip entpacken und die Index Datei im Browser betrachten. Man bekommt eine Übersicht über den Ablauf mit Verlinkungen zu den einzelnen Tests. Allerdings sieht man an dieser Stelle nicht viel, da die verknüpften EXAM Funktionen leer sind.

```
INFO

ReqsACC__2015-09-26__21-19-39

2015-09-26 21:19:39   EXECUTION   Testrunner   test-suite started !!!
2015-09-26 21:19:39   EXECUTION   Testrunner   test-suite name:        ACC
2015-09-26 21:19:39   EXECUTION   Testrunner   test-suite description:
2015-09-26 21:19:39   EXECUTION   Testrunner   test-suite system-config.: EXAMples.coreSamples.testBasics.systemConfigurations.EXAI
2015-09-26 21:19:39   EXECUTION   Testrunner   ------------------------------------------------------------------------
2015-09-26 21:19:39   EXECUTION   Testrunner   project:
2015-09-26 21:19:39   EXECUTION   Testrunner   subject:
2015-09-26 21:19:39   EXECUTION   Testrunner   comment:
2015-09-26 21:19:39   EXECUTION   Testrunner   operator:
2015-09-26 21:19:39   EXECUTION   Testrunner   start-time:          Sat Sep 26 21:19:39 CEST 2015
2015-09-26 21:19:40   EXECUTION   Testrunner   ------------------------------------------------------------------------
2015-09-26 21:19:40   EXECUTION   Testrunner   ... current-group-name:       Requirements
2015-09-26 21:19:40   EXECUTION   Testrunner   ... current-group-description: TestGroup TestGroup
                                  Testrunner   TestCase: ON_ACT_DEACT_ACT_OFF
                                  Testrunner   TestCase: ON_ACT_DesFaster_OFF
                                  Testrunner   TestCase: ON_ACT_DesSlower_OFF
                                  Testrunner   TestCase: ON_ACT_OFF
                                  Testrunner   TestCase: ON_ACT_Rother_RAccel_RAccel_OFF
                                  Testrunner   TestCase: ON_ACT_Rother_RDec_OFF
                                  Testrunner   TestCase: ON_ACT_Rother_RLeave_OFF
2015-09-26 21:19:44   EXECUTION   Testrunner   ------------------------------------------------------------------------
2015-09-26 21:19:44   EXECUTION   Testrunner   number of executed tests:              7
2015-09-26 21:19:44   EXECUTION   Testrunner   number of tests without script-errors:     7
2015-09-26 21:19:44   EXECUTION   Testrunner   number of tests with detected script-errors: 0
2015-09-26 21:19:44   EXECUTION   Testrunner   number of executed administrativeCases:     0
2015-09-26 21:19:44   EXECUTION   Testrunner   number of executed testCases:           7
2015-09-26 21:19:44   EXECUTION   Testrunner   number of inactive testCases:           0
2015-09-26 21:19:44   EXECUTION   Testrunner   complete test-time: 00:00:00:04
2015-09-26 21:19:44   EXECUTION   Testrunner   ------------------------------------------------------------------------
```

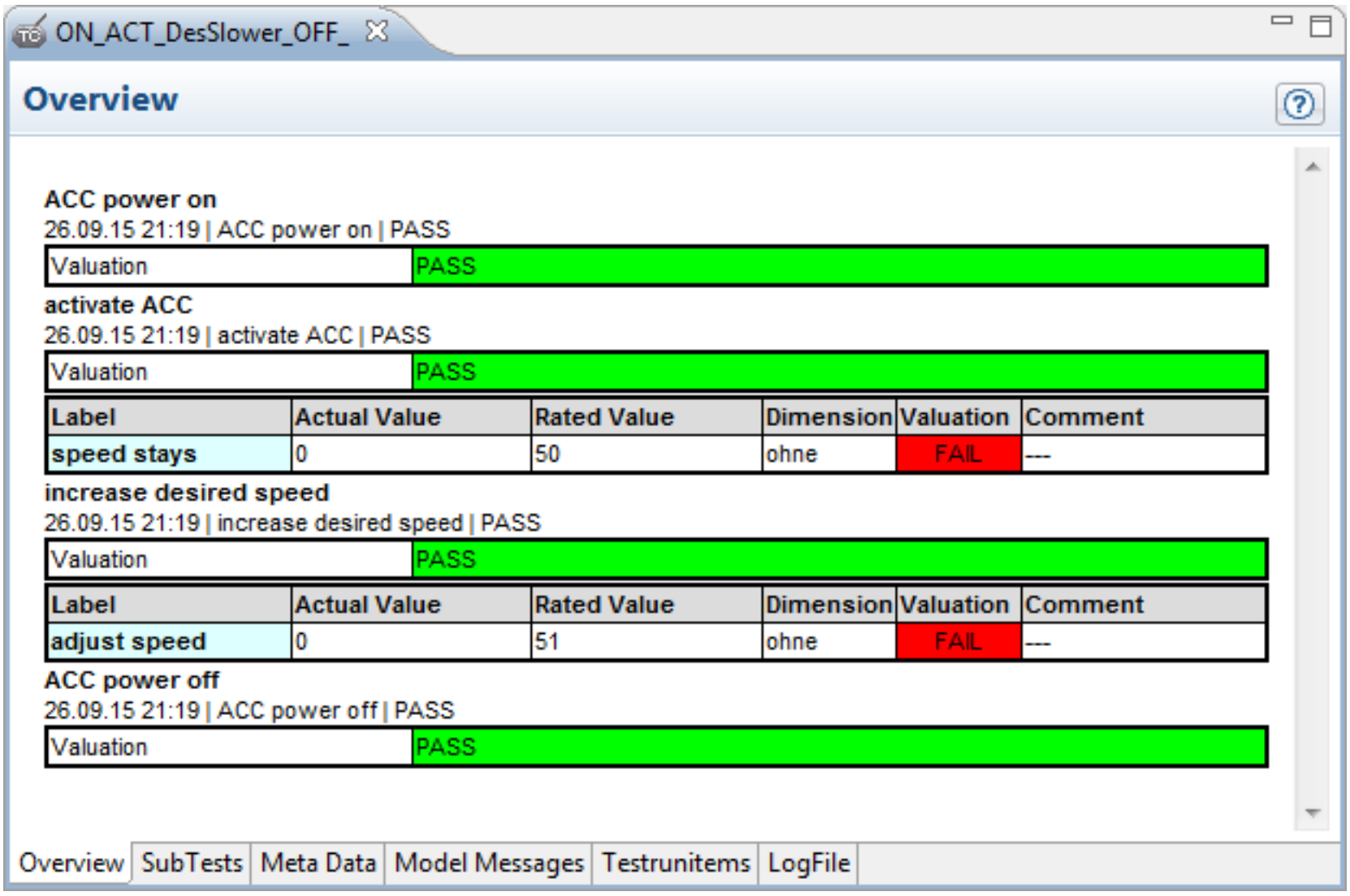
Den Reportmanager findet man in der Perspektivenauswahl oder man klickt auf den entsprechenden Link in der Konsole, nachdem die Testausführung abgeschlossen ist.

```
2015-08-11 11:52:09 Testrunner Open Report: examrm://NBK1359.techie.local:5432/exam_rmdb/Report/1
```

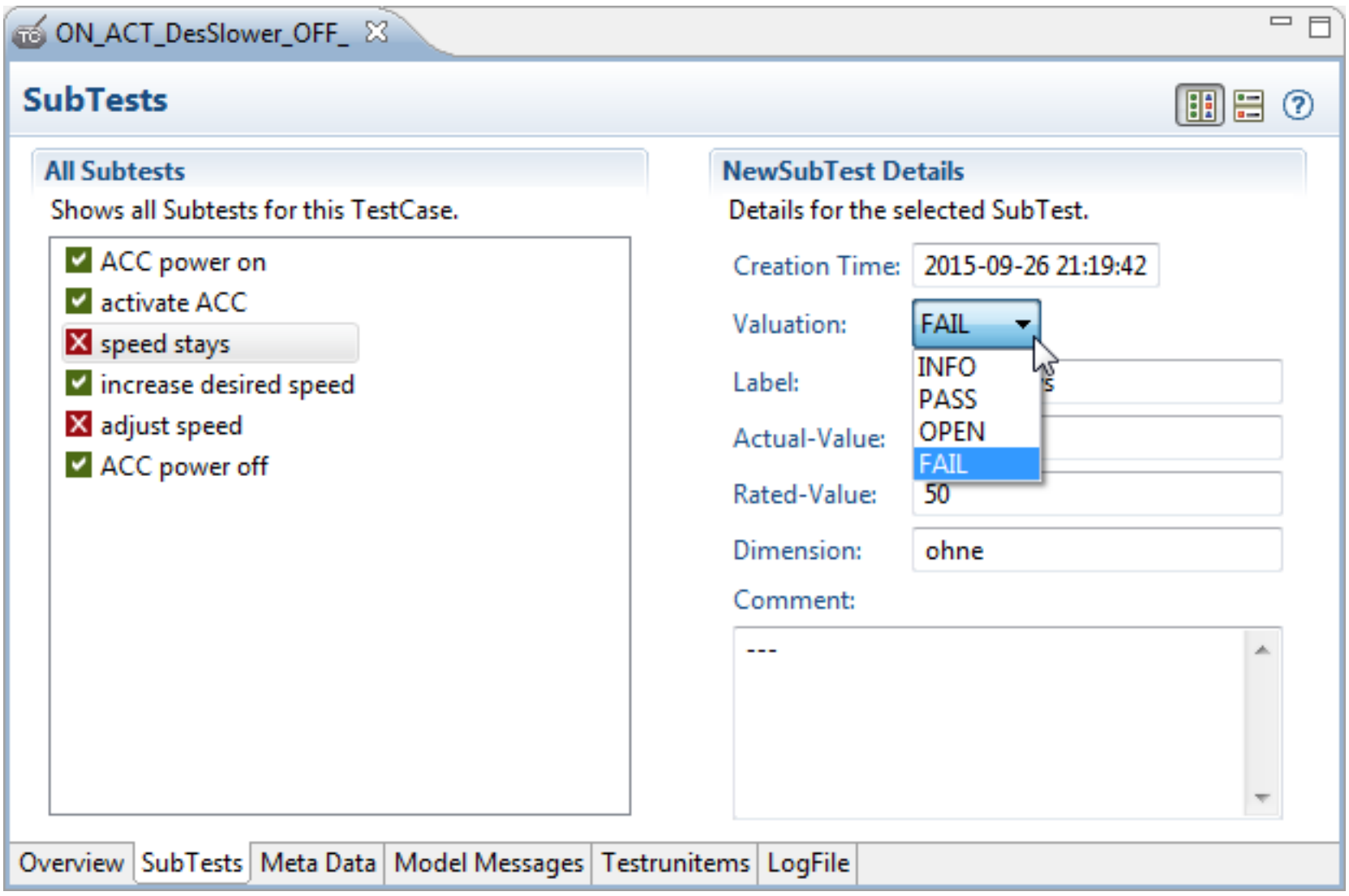
In der Perspektive *Reportmanager* erscheint der Testreport links im *Report Explorer*.



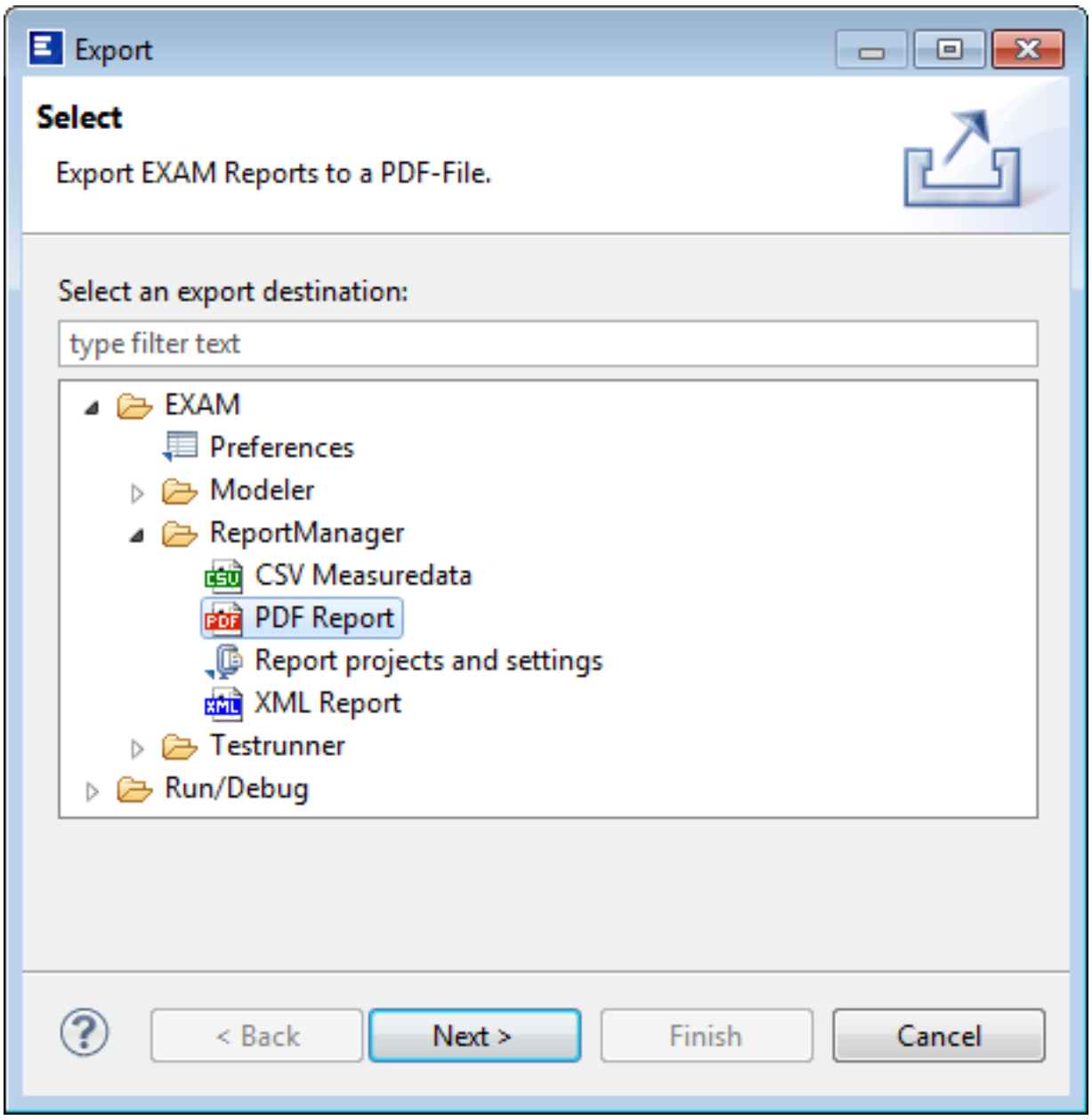
Durch einen Doppelklick auf einzelne Tests oder Testschritte öffnet sich eine weitere View, in der die Testfälle eingesehen und nachbearbeitet werden können.



In den unten angezeigten Tabs lassen sich weitere Details anzeigen oder auch zum Beispiel das Testergebnis einzelner Testschritte anpassen.



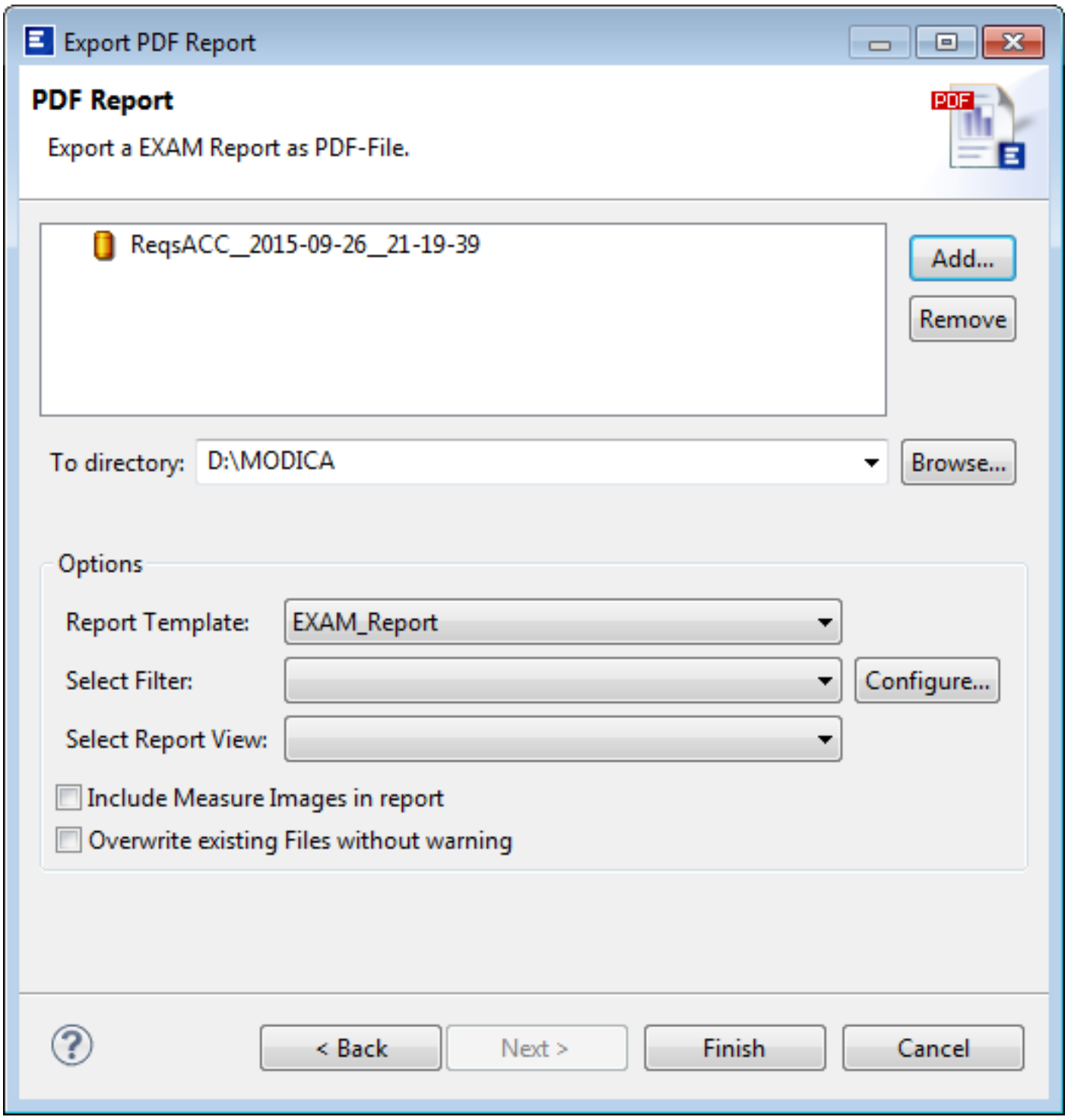
Den *Testreport* kann man ebenfalls als PDF exportieren. Dazu wählt in der Menüleiste man *File* → *Export*. Im *Export Wizard* wählt man *Report Manager* → *PDF Report* aus.



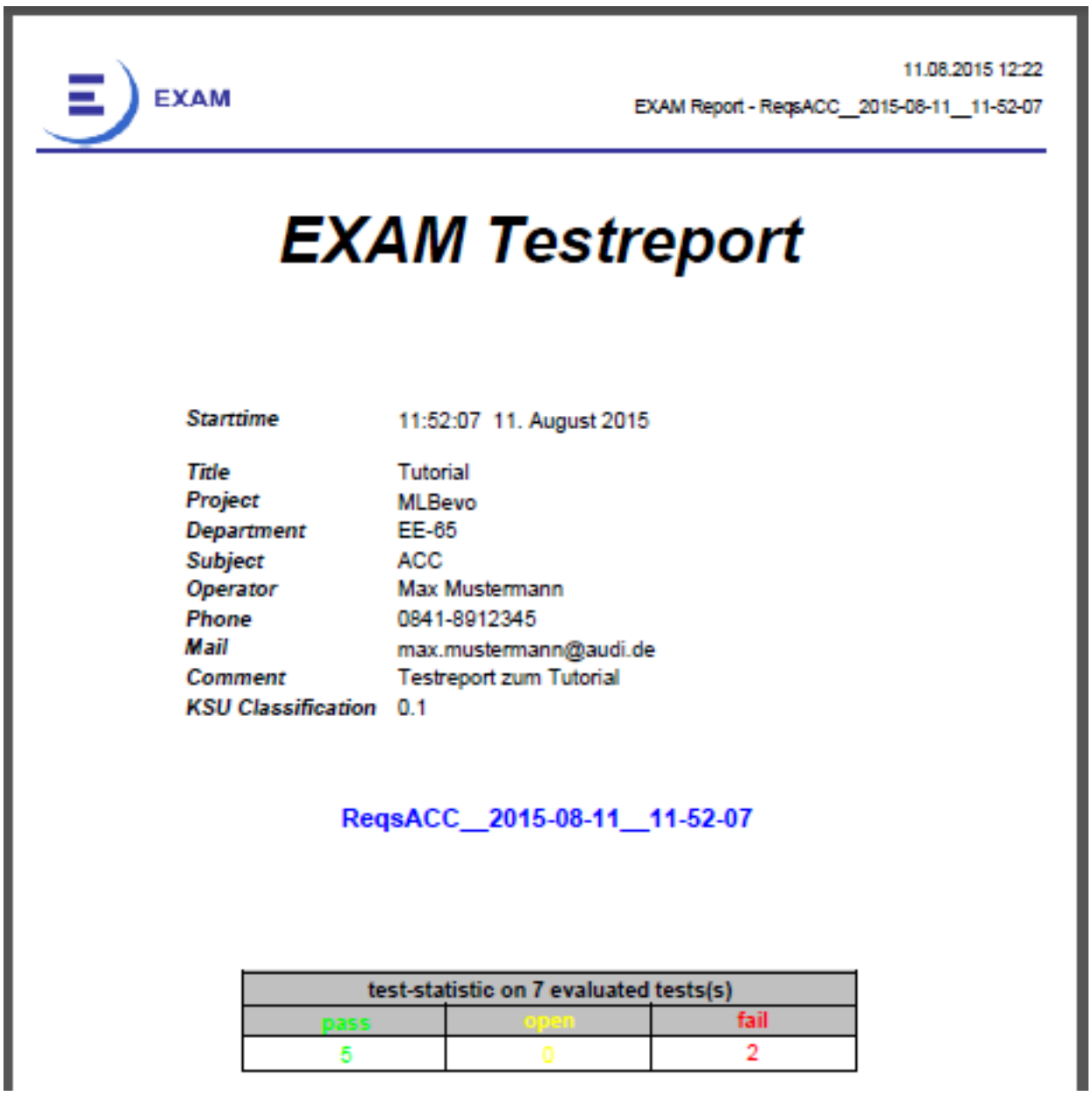
Im *PDF Report Wizard* fügt man mit Add den gewünschten Testreport aus der Datenbank hinzu. Als nächstes bestimmt man ein Zielverzeichnis, wo der Report gespeichert werden soll.

Unter *Options* lässt sich ein *Template* auswählen. Hier ist *EXAM_Report* auszuwählen.

Der Rest bleibt wie voreinstellt und man klickt auf *Finish*.



Der fertige *PDF Export* enthält eine Übersicht der Testfälle, eine Teststatistik und Details zu den einzelnen Testfällen, die u.a. die DOORS Module und Beschreibungen beinhalten.



EXAM

11.08.2015 12:22

EXAM Report - ReqsACC__2015-08-11__11-52-07

EXAM Testreport

Starttime	11:52:07 11. August 2015
Title	Tutorial
Project	MLBevo
Department	EE-65
Subject	ACC
Operator	Max Mustermann
Phone	0841-8912345
Mail	max.mustermann@audi.de
Comment	Testreport zum Tutorial
KSU Classification	0.1

ReqsACC__2015-08-11__11-52-07

test-statistic on 7 evaluated tests(s)		
pass	open	fail
5	0	2

Damit ist das Tutorial abgeschlossen. Details und weitere Informationen sind in den Dokumenten [Handbuch], [Modellierungsrichtlinien] und [Glossar] verfügbar. Für Fragen und Anregungen steht support_MODICA@expleo-germany.com zur Verfügung.