AutoFocus 2

Das Bilderbuch

Doris Wild

Technische Universität München
Institut für Informatik
Lehrstuhl Prof. Dr. Dr. h. c. Manfred Broy
Software & Systems Engineering

Technischer Bericht TUM-I0610
Mai 2006

Inhaltsverzeichnis

1 Motivation

AutoFOCUS 2 (AF2) ist ein Werkzeugprototyp, der am Lehrstuhl für Software & Systems Engineering der TU München entwickelt wird [1]. AF2 bietet dem System-Entwickler eine umfassende Unterstützung bei der Spezifikation softwareintensiver Systeme an. AF2 beinhaltet dazu u.a. folgende wichtige Funktionalitäten:

- Unterstützung bei Anforderungserhebung und -strukturierung
- Graphische, sichtenbasierte System-Modellierung
- Simulation und Rapid Prototyping
- Testfallgenerierung und formale Verifikation

Dieses Bilderbuch wendet sich an Leser, die einen ersten Überblick über das Werkzeug AF2 erhalten wollen. Anhand eines einfachen Fallbeispiels werden Kernfunktionalitäten des Werkzeugs durch ausgewählte, kommentierte Abbildungen illustriert und ein möglicher methodischer Einsatz von AF2 im SW-Entwicklungsprozess aufgezeigt. Erweiterte Funktionalitäten des Werkzeugs wie z.B. eine begleitende Prozessunterstützung, Anbindungen an einen Modellchecker oder Testfallgenerierung usw. werden hier nicht behandelt.

Der Aufbau dieses Dokuments orientiert sich an den Phasen eines systematischen Entwicklungsprozesses. Nach einer kurzen Einführung in AF2 im nächsten Abschnitt werden im Abschnitt 3 zunächst die Konzepte zur Anforderungsanalyse erklärt. Darauf aufbauend werden im Abschnitt 4 die modellbildenden Konzepte für eine formale Spezifikation und für die Simulation veranschaulicht. Im Abschnitt 5 wird auf die Schnittstelle zwischen Anforderungsanalyse und formaler Spezifikation eingegangen.

Die behandelten Konzepte werden in allen Abschnitten anhand eines einfachen, durchgängigen Fallbeispiels (einer Fensterheberfunktion) veranschaulicht; grundlegende Anforderungen des Fallbeispiels können dem Dokument „Das Türsteuergerät – eine Beispielspezifikation“ aus dem Projekt Quasar [2] entnommen werden. Sinn dieses Fallbeispiels ist nicht, eine perfekte Modellierung zu zeigen, sondern die Vorteile des Werkzeuges AF2 in einem speziellen Anwendungsfall zu demonstrieren.

2 Einführung in AF2

AF2 ist ein modellbasiertes Werkzeug zur Entwicklung von verteilten eingebetteten Systemen. Es bietet die Möglichkeit, das zu entwickelnde System graphisch zu spezifizieren und zu simulieren. Darüber hinaus dient AF2 als Werkzeug für die strukturierte Anforderungsanalyse.

AF2 ist prototypisch am Lehrstuhl Software & Systems Engineering der TU München von Studenten, Mitarbeitern des Lehrstuhls und der Validas AG entwickelt worden und ist eine „Neuentwicklung“ von AutoFOCUS (AF) [3], deren Ursprung und Hintergrund die formale Spezifikations-Methodik FOCUS [4] ist.

Die funktional bedeutendste Erweiterung von AF2 gegenüber AF ist die Einbindung des Werkzeuges AutoRAID [5], das im Sommersemester 2004 in einem Softwaretechnik-Praktikum an der TU München entstanden ist. AutoRAID unterstützt die modellbasierte Anforderungsanalyse und den Übergang zum Design, indem es die zielorientierte Identifikation und Strukturierung von Anforderungen einfordert und die schrittweise Abbildung dieser Anforderungen auf funktionale Modelle ermöglicht. Weitere Informationen und Hintergründe zum Werkzeug AutoRAID sind ausführlich in [6] und kurz zusammengefasst in [7] beschrieben.

Projekte (Abbildung 1), die in AF2 modelliert werden, bestehen aus zwei eng miteinander verknüpften Spezifikationen: Einem Analyseteil, in dem sich alle für dieses Projekt relevanten Anforderungen befinden, und einem Modellierungsteil, bestehend aus einer formalen Spezifikation des zu entwickelnden Systems.

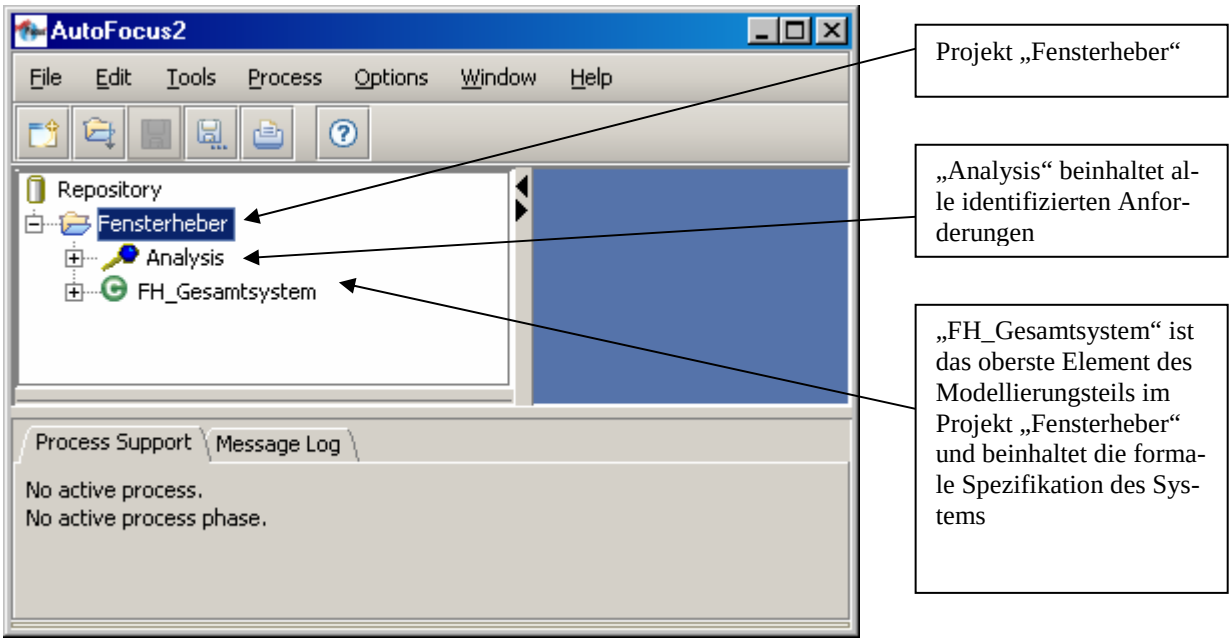

Abbildung 1: Struktur der Projekte in AF2

In den folgenden Abschnitten werden die Konzepte beider Teile erläutert, sowie explizit auf die Beziehung zwischen den Elementen dieser beiden Teile eingegangen.

3 Anforderungsanalyse

In diesem Kapitel werden die AF2-Konzepte erklärt, die die Anforderungsanalyse unterstützen. Die Darstellung anhand der Fallstudie dient gleichzeitig der Einführung grundlegender Anforderungen an die Funktion „Fensterheber“.

3.1 Überblick über die Konzepte zur Anforderungsanalyse

Wie in Abschnitt 2 erwähnt, werden alle Spezifikationen, die mittels der Konzepte zur Anforderungsanalyse erstellt werden, im Analyseteil („Analysis“) eines Projekts abgelegt. Abbildung 2 zeigt die Struktur und gibt einen ersten Überblick über die verwendeten Konzepte.

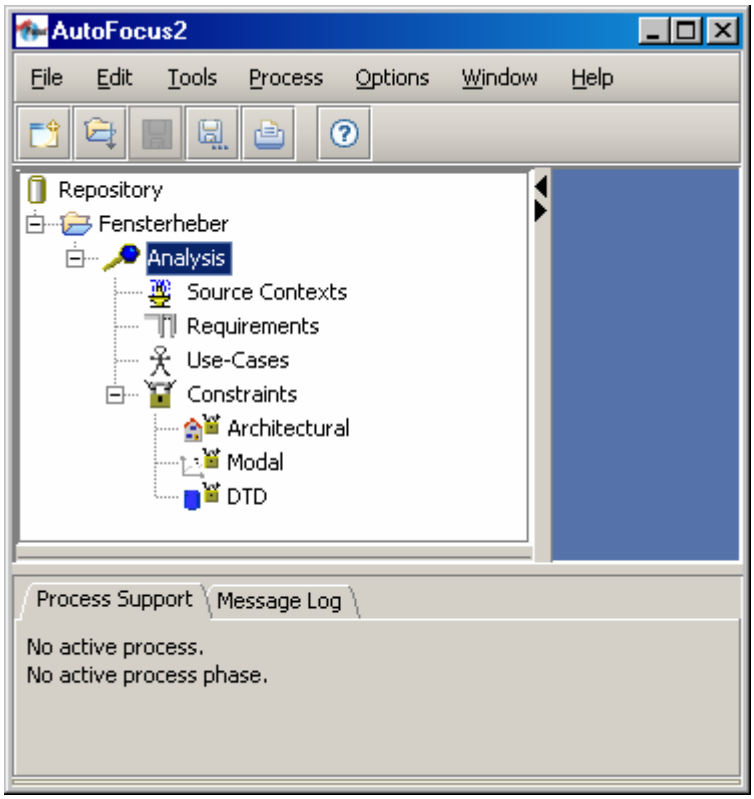

Abbildung 2: Struktur des Analyseteils

Die Konzepte werden unterteilt in:

- Source Contexts,
- Requirements,
- Use-Cases,
- Constraints.

In den folgenden Abschnitten werden diese Konzepte kurz erklärt.

3.2 Requirements

Im Teilbaum *Requirements* sind in einer hierarchischen Struktur **alle** Anforderungen aufgeführt, die das zu entwickelnde System beschreiben. Auch Einträge unter *Constraints* und unter *Use-Cases* finden sich hier noch einmal wieder. Durch die hierarchische Struktur wird ersichtlich, woraus sich die einzelnen Anforderungen ableiten, und – in der entgegengesetzten Richtung – veranschaulicht, zu welchem Ziel eine Anforderung einen Beitrag leistet.

In AF2 können für alle Anforderungen folgende Informationen angegeben werden:

- Pflichtfelder:
  - Titel
  - Verantwortlicher
- Optionale Felder:
  - Beschreibung
  - Status
  - Priorität
  - Begründung

Um die Anforderungsanalyse sinnvoll durchführen zu können, ist es ratsam, alle Felder gewissenhaft auszufüllen.

Abbildung 3 zeigt das Eingabeformular für Anforderungen, veranschaulicht am Beispiel eines *BusinessRequirement.*

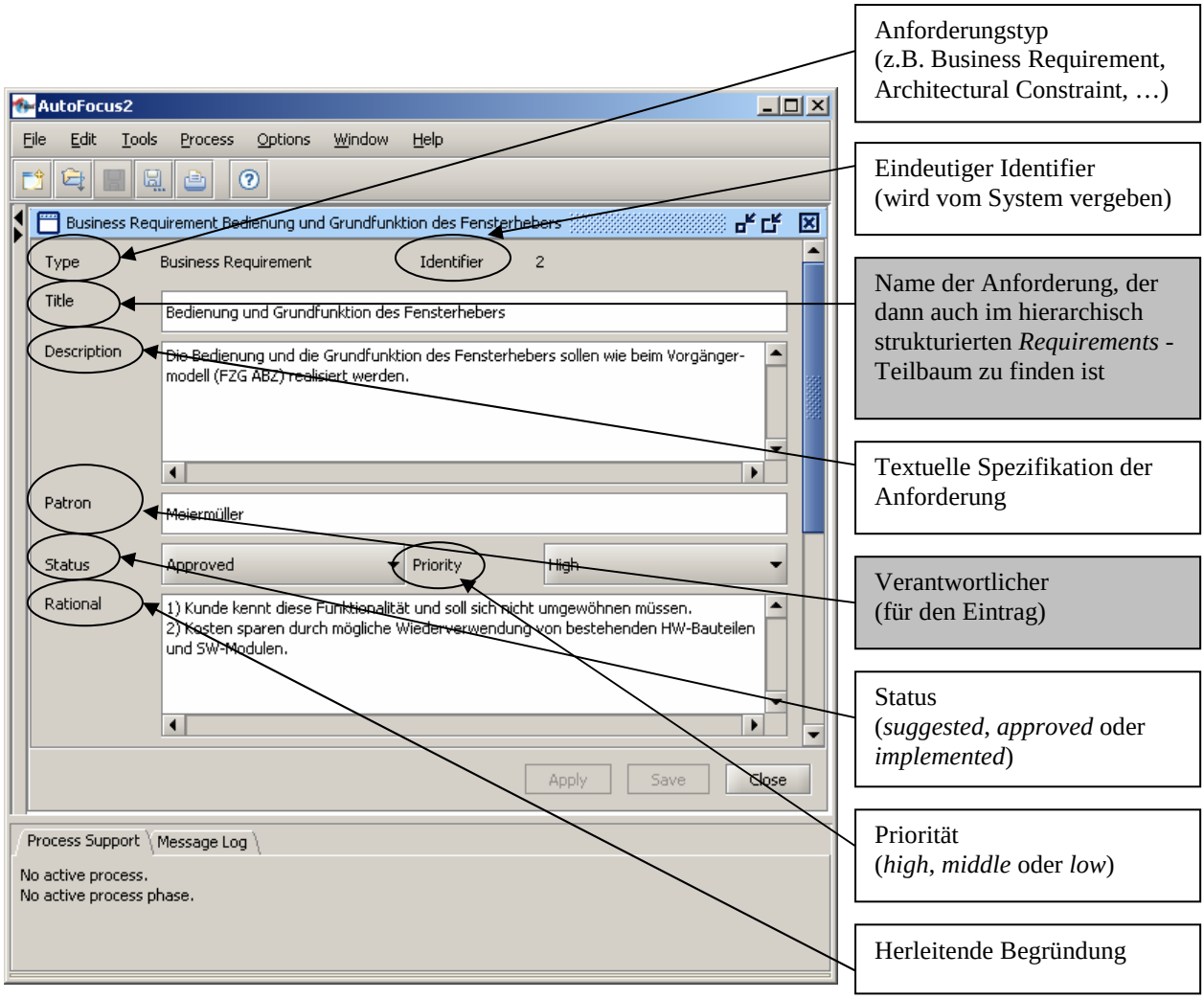

Abbildung 3: Eingabeformular für Anforderungen (Pflichtfelder sind dunkel hinterlegt)

Im *Requirements* Teilbaum werden zunächst zwei Arten von *Requirements* unterschieden, nämlich *BusinessRequirements* ( ) und *ApplicationRequirements* ( ). Abbildung 4 zeigt beispielhaft einige *BusinessRequirements* des Fallbeispiels.

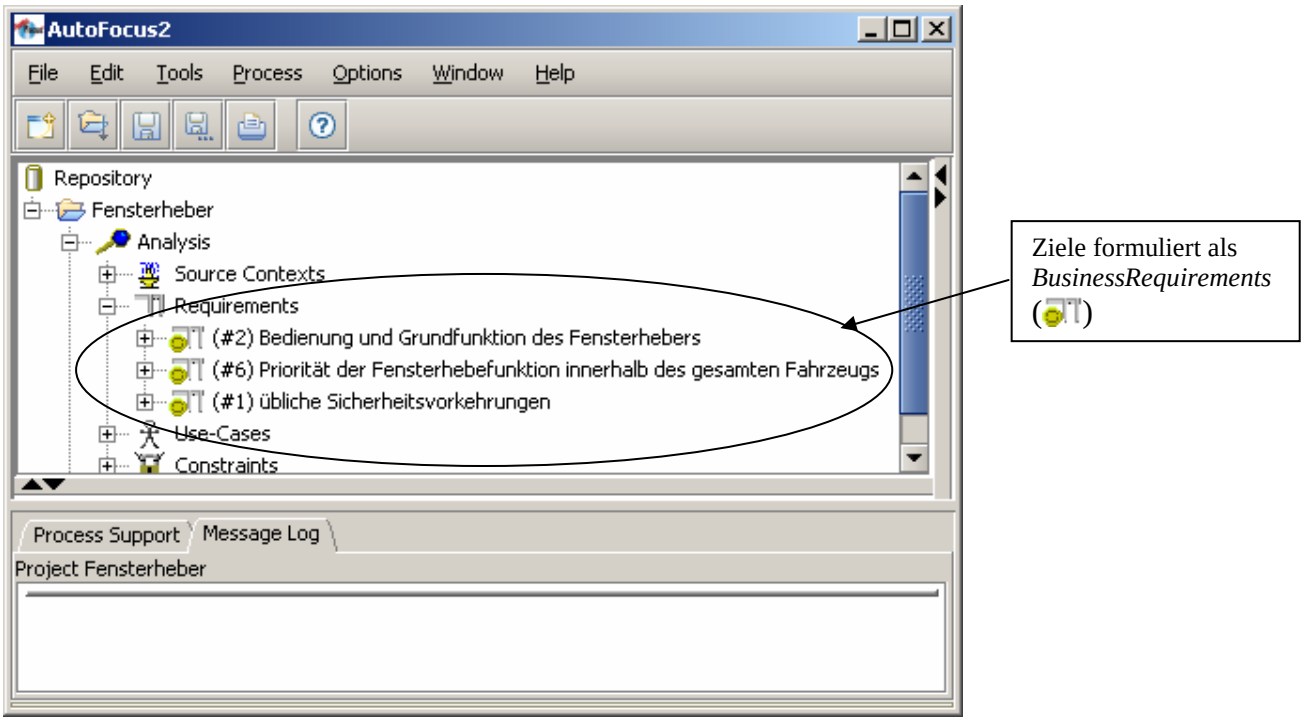

Abbildung 4: *BusinessRequirements* für das Fallbeispiel „Fensterheber“

BusinessRequirements beinhalten die Ziele, die mit der Entwicklung des entsprechenden Produkts erreicht werden sollen. Sie stellen damit „High-level“-Anforderungen dar, die meist vage formuliert sind und im Normalfall nicht direkt umsetzbar sind. Diese Ziele werden solange verfeinert, bis messbare Anforderungen formuliert werden können, d.h. Anforderungen, die hinsichtlich ihrer Erfüllung bewertet werden können.

Diese verfeinerten oder auch abgeleiteten Anforderungen werden in AF2 durch *Application-Requirements* dargestellt. Dazu zählen auch *Use-Cases* und *Constraints*, die beide weiter unten noch erklärt werden. In Abbildung 5 ist die Ableitung von Anforderungen aus Zielen für ein *BusinessRequirement* beispielhaft dargestellt.

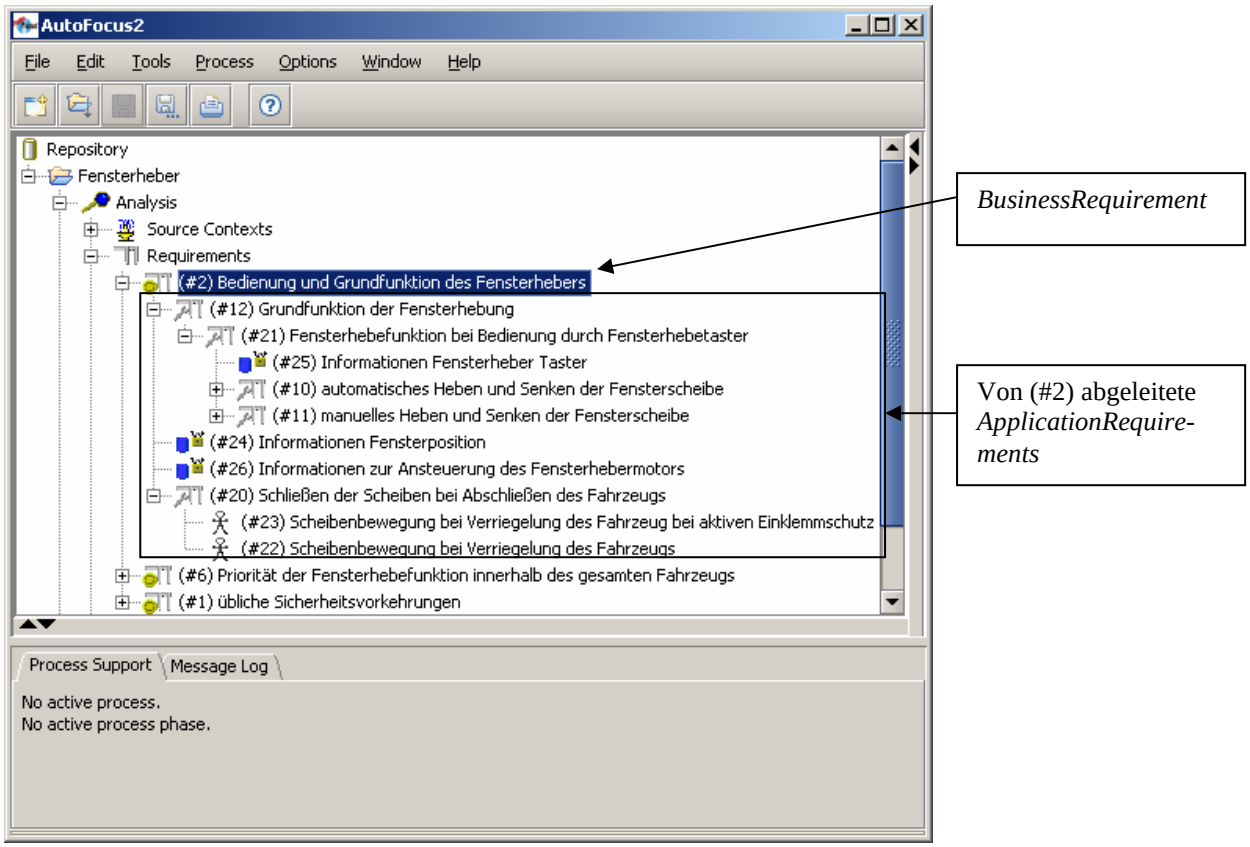

Abbildung 5: Ableitung von *ApplicationRequirements* aus *BusinessRequirements*

Anhand dieser Hierarchiestruktur können schon erste, einfache Überprüfungen in Bezug auf die Vollständigkeit und Konsistenz der gesammelten Anforderungen gemacht werden:

- Gibt es zu jedem *BusinessRequirement* auch verfeinerte *ApplicationRequirements*? D.h. konkret: Hat jedes *BusinessRequirement* mindestens eine Ableitung?

- Ist jedes *ApplicationRequirement* eine berechtigte Anforderung und trägt somit zur Erreichung eines *BusinessRequirement* bei? D.h. konkret: Ist jedes *ApplicationRequirement* mindestens die Ableitung eines *BusinessRequirements*?

3.3 Use-Cases

Use-Cases repräsentieren spezifische Ausprägungen von *ApplicationRequirements*. Sie werden benötigt, um geforderte Systemfunktionen und Nutzungsprozesse genauer beschreiben zu können. Aus *Use-Cases* können durch eine Analyse der Szenarien (siehe weiter unten) MSCs (Message Sequence Charts) generiert werden, die dann z.B. zur weiteren Analyse, zur Validierung oder auch zur Testfallgenerierung eingesetzt werden können.

Um schneller und gezielter auf wichtige Anforderungsklassen zugreifen zu können und um die Anordnung der Anforderungen übersichtlich darzustellen, sind alle *Use-Cases* – außer im Requirements-Teilbaum – zusätzlich unter dem Teilbaum „*Use-Cases*“ aufgelistet.

In Abschnitt 5.4 wird gezeigt, wie *Use-Cases* und *Szenarien* benutzt werden können, um Unstimmigkeiten in Anforderungen zu entdecken.

3.4 Constraints

Constraints sind besondere *ApplicationRequirements*, die messbare Vorgaben an den Entwurf und an die Entwicklung des Systems beinhalten. Durch diese Vorgaben wird der potentielle Lösungsraum durch die Vorwegnahme von Designentscheidungen eingeschränkt. Es lassen sich 3 Arten von *Constraints* unterschieden:

- *ArchitecturalConstraints*:
  Diese Anforderungen beinhalten Strukturanforderungen an das zu entwickelnde System und an seine Umgebung. Hier werden die beteiligten Komponenten, ihre Schnittstellen und die Kommunikationsbeziehungen zwischen diesen festgelegt.

- *ModalConstraints*:
  Diese Anforderungen beinhalten die Betriebsmodi der Anwendung. Hier können verschiedene Zustände und Zustandsübergänge definiert werden.

- *DTDConstraints*:
  Diese Anforderungen beinhalten Datentypen, die im zu entwickelnden System benutzt werden.

Die bei der Spezifikation von *Use-Cases* und *Constraints* getroffenen Entwurfsentscheidungen müssen in nachfolgenden Entwurfsschritten nochmals überprüft und weiter detailliert werden. In den Abschnitten 4 und 5 wird demonstriert, wie AF2 diese Überprüfung und Detaillierung unterstützen kann.

3.5 Source Contexts

Anforderungen können in AF2 auf zwei Arten eingegeben werden: Einmal durch einfache Eingabe in das Anforderungsformular und einmal durch ein „copy&paste“- Verfahren, das erlaubt, Textpassagen direkt aus Dokumenten in das Anforderungsformular zu übernehmen. In diesem Abschnitt wird am Beispiel gezeigt, wie Dokumente in AF2 importiert werden und daraus Anforderungen gewonnen werden.

Hinter dem Eintrag *Source Contexts* verbergen sich alle in AF2 importierten Dokumente, wie z.B. Lastenhefte, Besprechungsprotokolle oder Telefonnotizen. Diese wurden – wie in Abbildung 6 demonstriert – in AF2 eingelesen[1] und können so als direkte Grundlage für die Anlage von Anforderungen dienen.

[1] Einschränkung: Es können nur sehr einfache Texte im „html“-Format eingelesen werden.

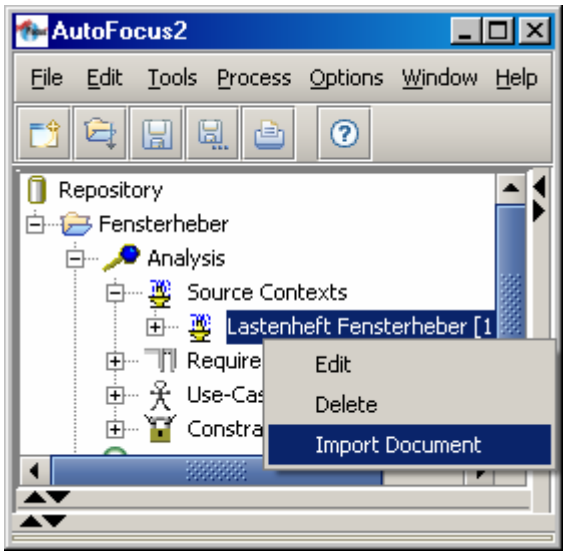

Abbildung 6: Dokumentenimport in AF2

In den eingelesenen Dokumenten gibt es die Möglichkeit einzelne Textstellen per „copy&paste“ zu extrahieren (Abbildung 7) und unmittelbar als *BusinessRequirement*, *ApplicationRequirement*, *Use-Case* oder *Constraint* geordnet abzulegen (Abbildung 8).

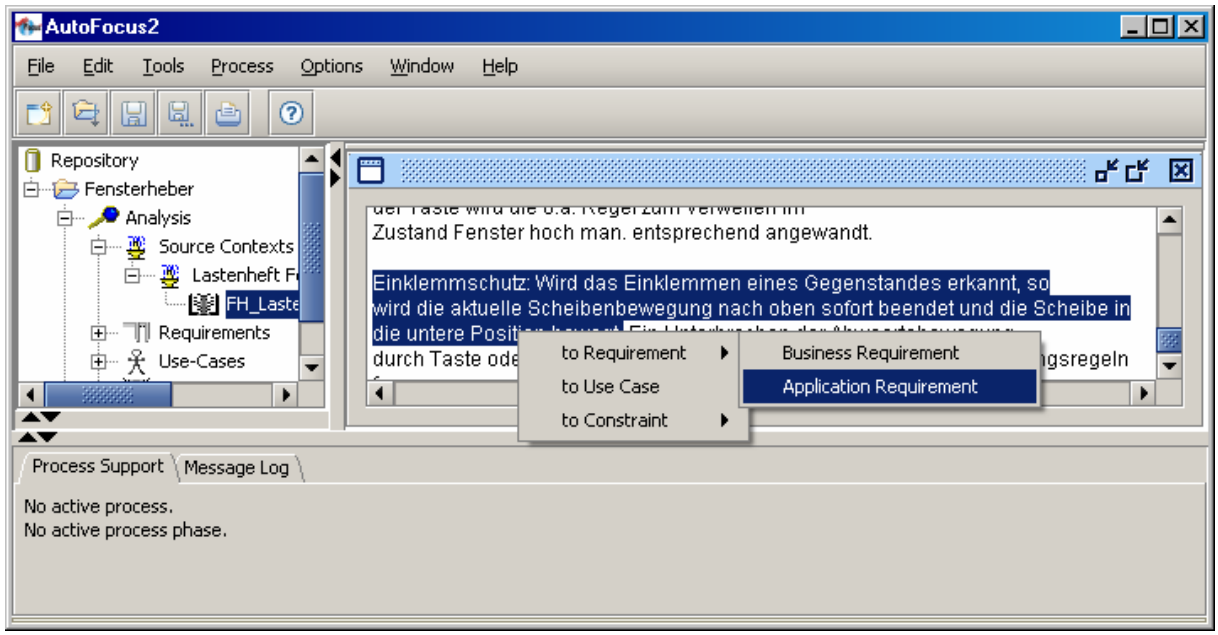

Abbildung 7: Anlegen einer Anforderung durch "copy&paste" - Verfahren

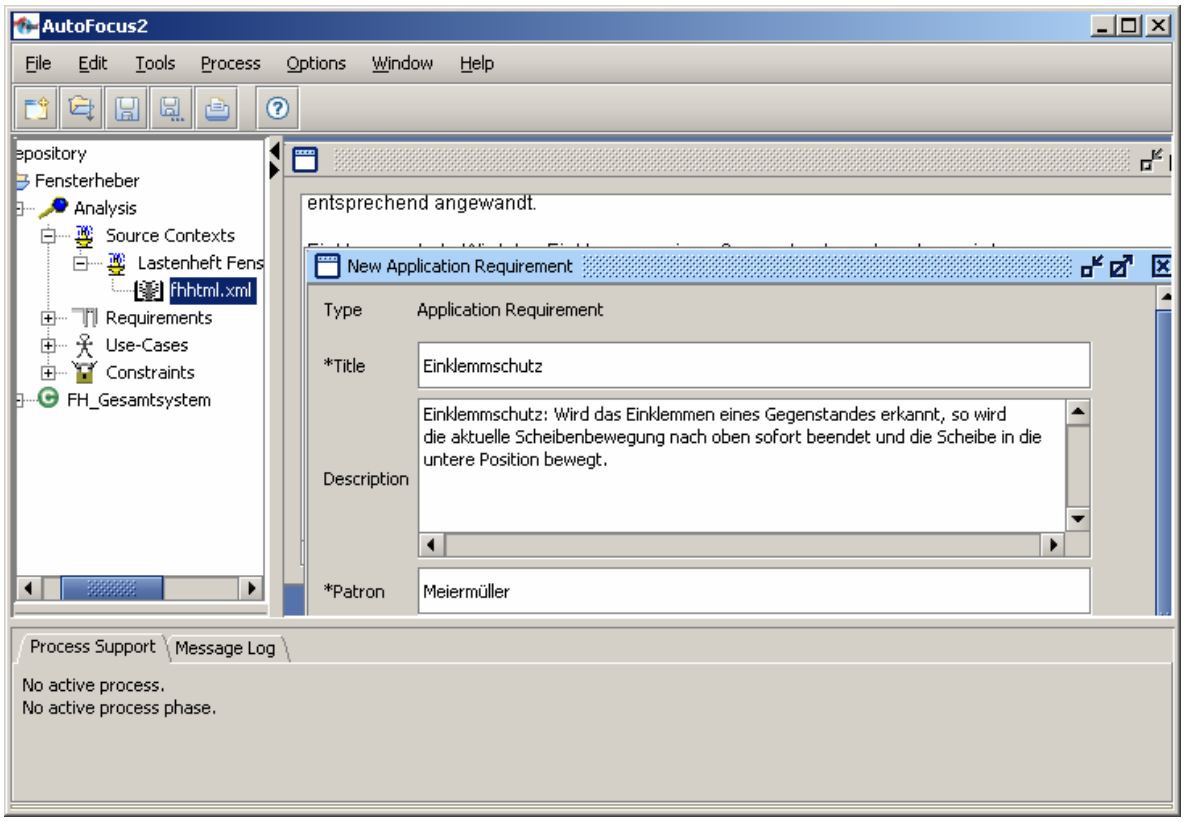

Abbildung 8: als ***ApplicationRequirement*** abgelegte Anforderung

Die entsprechenden Textstellen werden – wie in Abbildung 9 zu sehen – im eingelesenen Dokument markiert. Die Farbe der Markierung hängt von der Klassifizierung der extrahierten Anforderung ab (blau: *ApplicationRequirement*, gelb: *BusinessRequirement*, grau: *Use-Case*, rosa: *ArchitecturalConstraint*, dunkelgelb: *ModalConstraint*, hellrosa: *DTDConstraint*).

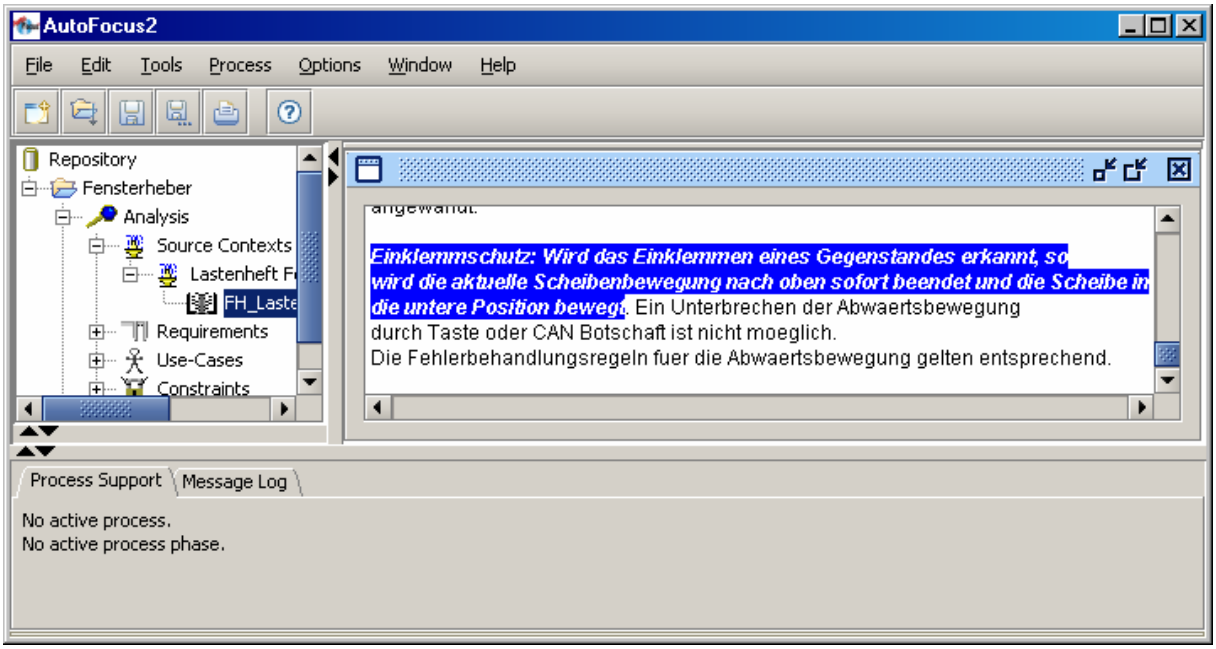

Abbildung 9: markierte Textstellen im importierten Dokument

Zwischen der markierten Textstelle und der aus ihr extrahierten Anforderung existiert nun ein Link, der erlaubt, direkt zur entsprechenden Anforderung zu navigieren. Ein Mausklick auf die markierte Stelle (Abbildung 10) erlaubt einfaches Auffinden und Identifikation der aus ihr entstandenen Anforderung

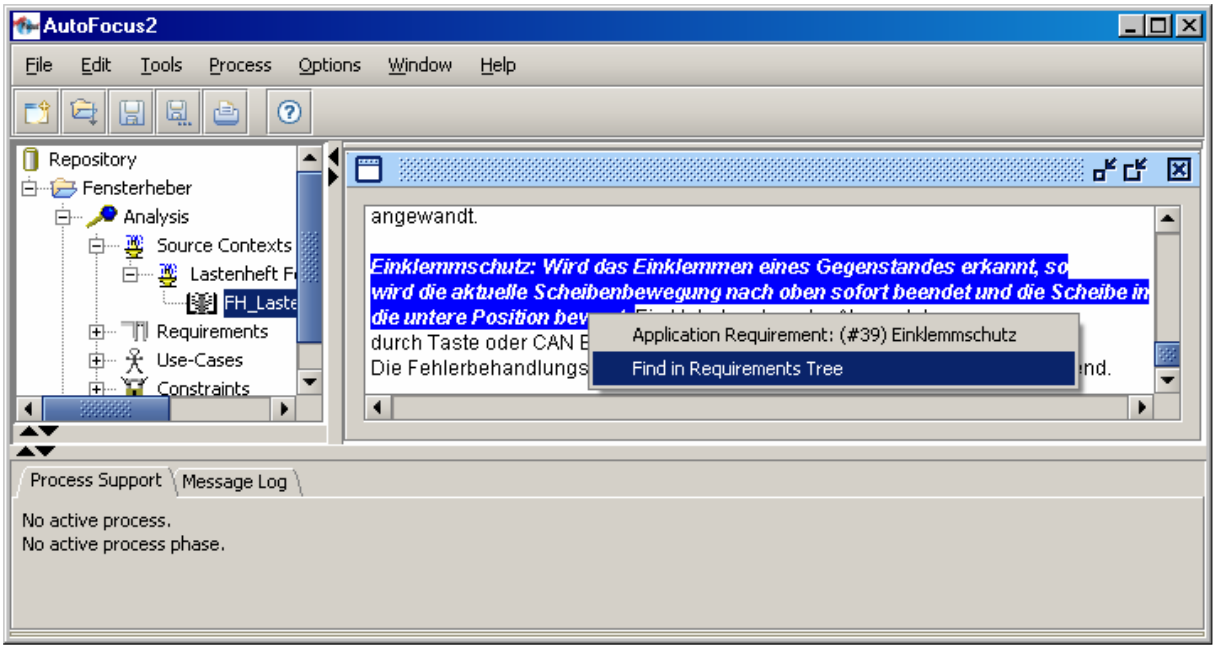

Abbildung 10: Link zur angelegten Anforderung

4 Formale Spezifikation

In Kapitel 3 wurden alle AF2-Konzepte vorgestellt, die die Anforderungsermittlung und -verwaltung unterstützen. Bis hierher wird das betrachtete System noch textuell, anhand von strukturiert abgelegten Anforderungen, beschrieben. In diesem Kapitel werden diejenigen AF2-Konzepte vorgestellt, die für die formale Spezifikation des zu entwerfenden Systems benötigt werden.

4.1 Überblick über die modellbildenden Konzepte

Folgende modellbildende Konzepte bietet AF2 zur formalen Spezifikation an:

-  Komponenten
-  Ports
-  Kanäle
-  Datentypen
-  Zustand
-  Transitionen (Zustandsübergänge)

Diese Konzepte, die im Folgenden noch genauer erläutert werden, reichen aus, um auch komplexe Systeme beschreiben zu können. Sie befinden sich im Modellierungsteil (Abbildung 1). Im Fallbeispiel existiert ein Teilbaum „FH_Gesamtsystem“, der den Modellierungsteil darstellt. Abbildung 11 gibt einen ersten Überblick über die Struktur des Modellierungsteils in AF2.

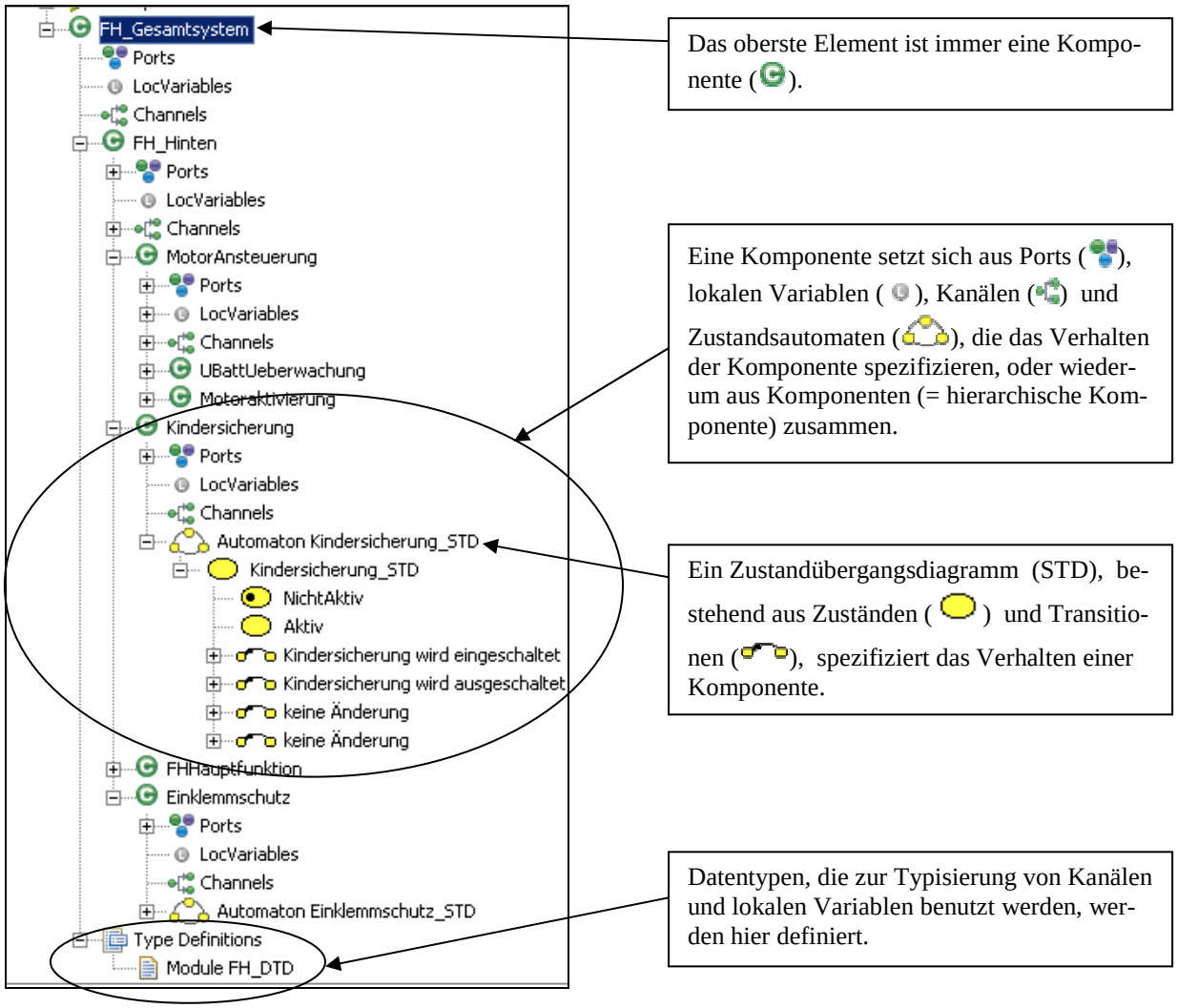

Abbildung 11: Darstellung der Struktur der AF2-Konzepte im Teilbaum „FH_Gesamtsystem“

Der Entwickler bearbeitet ein AF2-Modell nicht in seiner Gesamtheit, d.h. er muss nicht immer alle Elemente gleichzeitig im Auge behalten, sondern betrachtet vielmehr nur bestimmte Aspekte des Systems (z.B. Struktur oder Verhalten), die momentan im Blickpunkt seines Interesses stehen. Diese Aspekte bilden die Sichten auf das System.

In AF2 werden drei Sichten unterschieden:

- Die strukturelle Sicht, repräsentiert durch Systemstrukturdiagramme (SSD)
- Die Verhaltenssicht, repräsentiert durch Zustandsübergangsdiagramme (STD)
- Die Datensicht, repräsentiert durch Datentypdefinitionen (DTD)

Abbildung 12 zeigt Beispiele der einzelnen Sichten. Die einzelnen Sichten werden sowohl graphisch als Diagramme, als auch namentlich im Modellierungsteil repräsentiert.

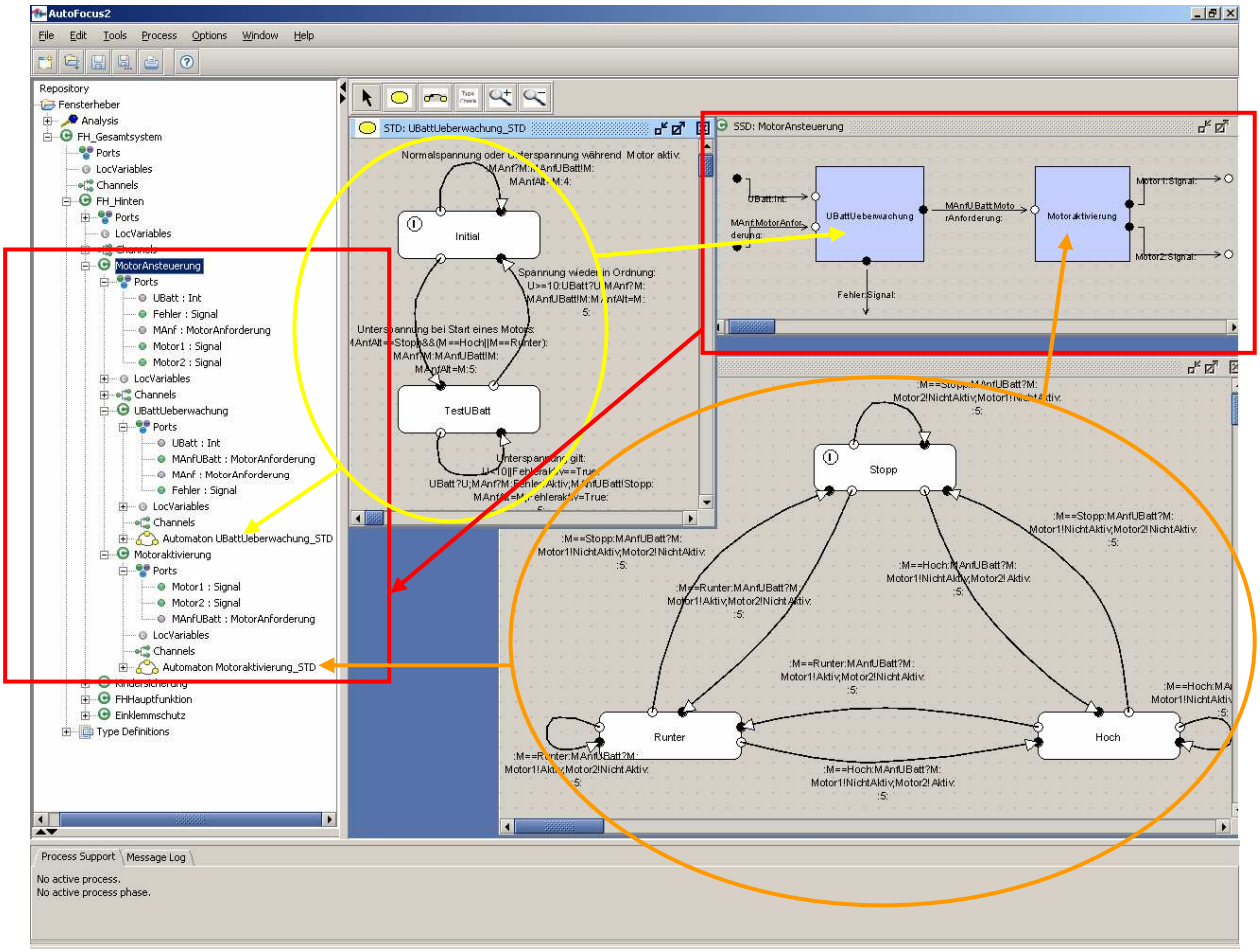

Abbildung 12: Konzepte und Sichten in AF2

Die einzelnen Beschreibungstechniken unterstützen die hierarchische Entwicklung eines Systems, d.h. eine *Komponente* kann wiederum *Komponenten* enthalten und ein *Zustand* kann Unterzustände enthalten.

4.2 Die strukturelle Sicht

In der strukturellen Sicht wird der funktionale Aufbau des zu entwickelnden Systems durch SSDs (SSD: System Structure Diagram) beschrieben. Das zentrale Element in einem SSD ist die *Komponente* (). Diese kann hierarchisch aufgebaut sein und selbst aus miteinander kommunizierenden *(Sub-)Komponenten* bestehen. *Komponenten* kommunizieren über *Ports* () mit der Umgebung und sind über *Kanäle* () miteinander verbunden.

Das System stellt sich somit in der strukturellen Sicht als Netzwerk von *Komponenten* dar, die untereinander über *Kanäle* kommunizieren. Dieses Netzwerk wird im Systemstrukturdiagramm (SSD) repräsentiert.

Abbildung 13 zeigt den Aufbau der *Komponente* „FH_Hinten“ des Fensterhebermodells.

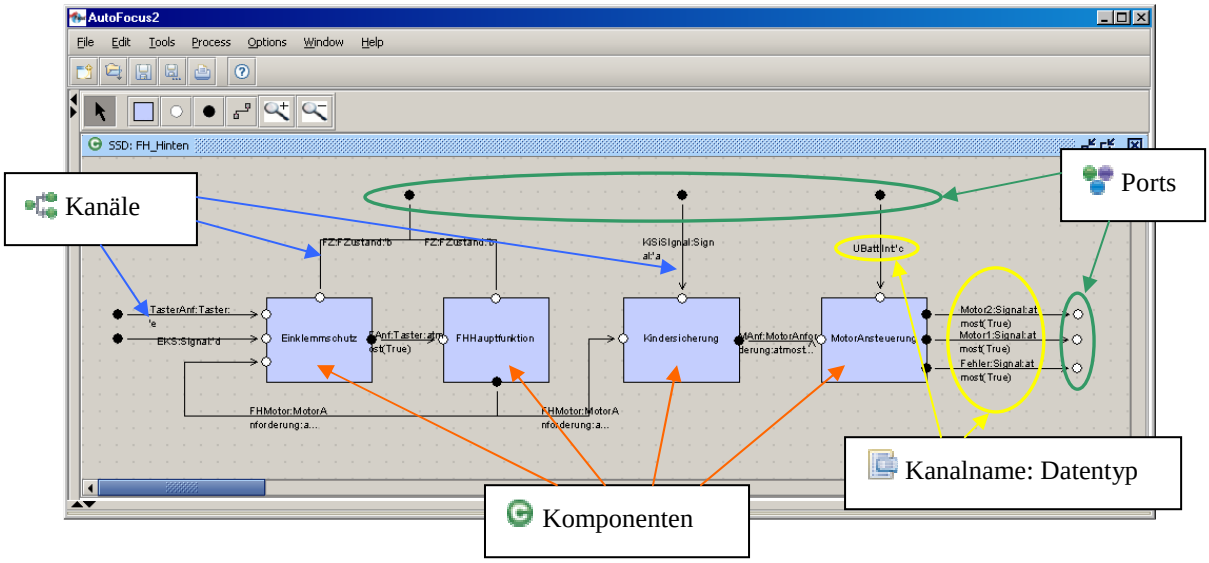

Abbildung 13: SSD der Komponente „FH_Hinten“

Komponenten repräsentieren die wesentlichen Bausteine eines Systems (im Diagramm symbolisiert durch blaue Kästen, z.B.: „Einklemmschutz“, „FHHauptfunktion“, „Kindersicherung“ und „MotorAnsteuerung“), sie besitzen einen eindeutigen Namen und sind über *Ports* und *Kanäle* miteinander verbunden. Die interne Struktur, das Verhalten und lokale Daten werden durch *Komponenten* gekapselt.

Über *Ports* (im Diagramm symbolisiert durch weiße und schwarze Kringel, im Modellierungsteilbaum durch grüne und weiße) kommunizieren die *Komponenten* mit der Umwelt. Sie haben einen Namen und einen Typ und werden unterschieden in *InputPorts* ( ) und *OutputPorts* ( ).

Die *Kanäle* (im Diagramm symbolisiert durch Pfeile) verbinden die *Komponenten* über die *Ports* miteinander. Sie sind unidirektional und haben ebenfalls einen Typ und einen Namen. *Kanäle* definieren die Kommunikationsstruktur des Systems.

4.3 Die Verhaltenssicht

Das Verhalten einer Komponente wird durch Zustandübergangsdiagramme (STD: State Transition Diagram) beschrieben. Die STDs stellen graphisch endliche Zustandsautomaten dar.

STDs bestehen aus *Zuständen* ( ) und *Transitionen* ( ). Jede *Transition* verbindet genau zwei *Zustände* und besteht aus vier Teilen:

- Pre Vorbedingung und Post Nachbedingung (Aktionen):
  Vor- bzw. *Nachbedingungen* sind Prädikate über Datenelementen, die vor bzw. nach der Ausführung einer *Transition* gelten müssen.

- ○ Eingabemuster und ● Ausgabemuster:
  Ein- und *Ausgabemuster* bestimmen, welche Werte oder Muster (Datentypen) am

InputPort vorhanden sein müssen, damit die *Transition* ausgeführt werden kann und welche Werte nach der Ausführung der *Transition* auf die *OutputPorts* geschrieben werden.

In Abbildung 14 wird anhand eines einfachen STDs (der *Komponente* „Kindersicherung“) die Darstellung in AF2 aufgezeigt.

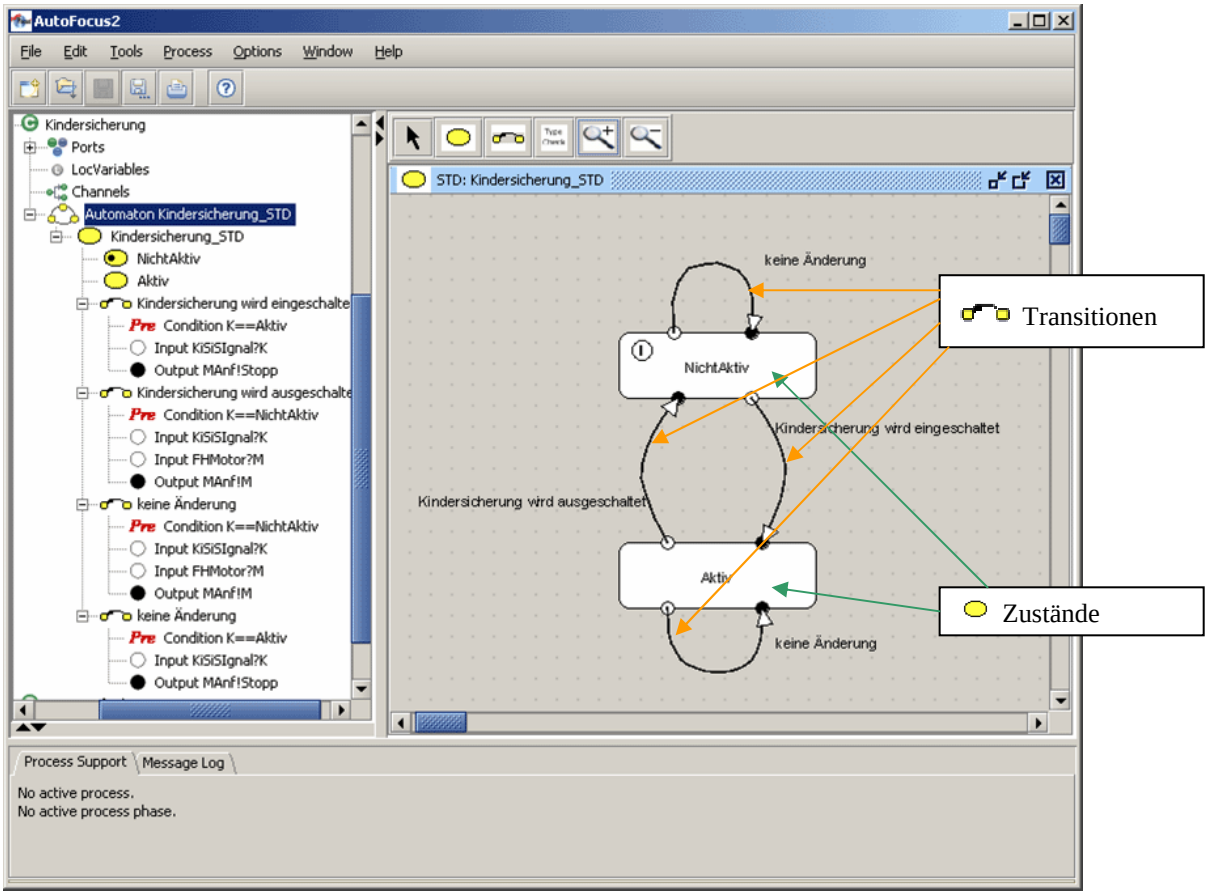

Abbildung 14: STD der Komponente „Kindersicherung“

Abbildung 15 zeigt das Eingabeformular einer *Transition*, welches neben der Eingabe der Transitionsbedingungen auch die Angabe eines Transitionsnamens und einer Priorität erlaubt.

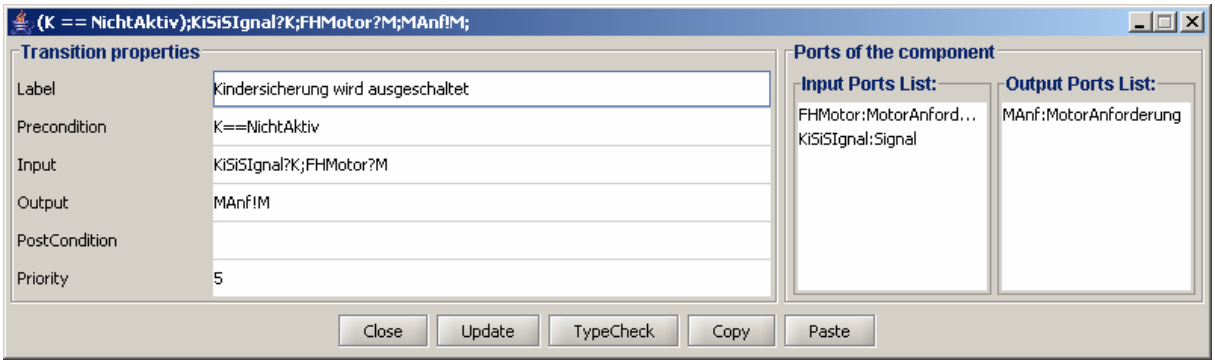

Abbildung 15: Eingabeformular für eine Transition

4.4 Datensicht

In AF2 ist es möglich, benutzerdefinierte *Datentypen* ( ) anzulegen. Diese werden über einen einfachen Texteditor (siehe Abbildung 16) eingegeben. Mit den hier festgelegten *Datentypen*, die durchaus auch wesentlich komplexer gestaltet sein können als im Fallbeispiel gezeigt, können *lokale Variablen* oder *Kanäle* typisiert werden.

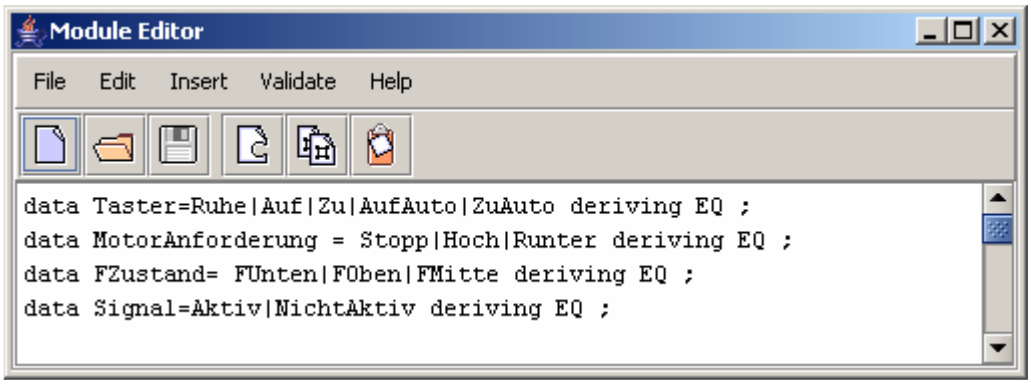

Abbildung 16: Beispiele für einfache Datentypdefinitionen

4.5 Simulation

Anhand einer Beispielsimulation wird nachfolgend gezeigt, wie in AF2 die Modelle ausgeführt werden. In AF2 kann das gesamte Modell aber auch jede *Komponente* für sich simuliert werden. Als Beispiel wird die *Komponente* „MotorAnsteuerung“ simuliert. Abbildung 17 zeigt die *Komponente* zum Startzeitpunkt mit der Defaultbelegung.

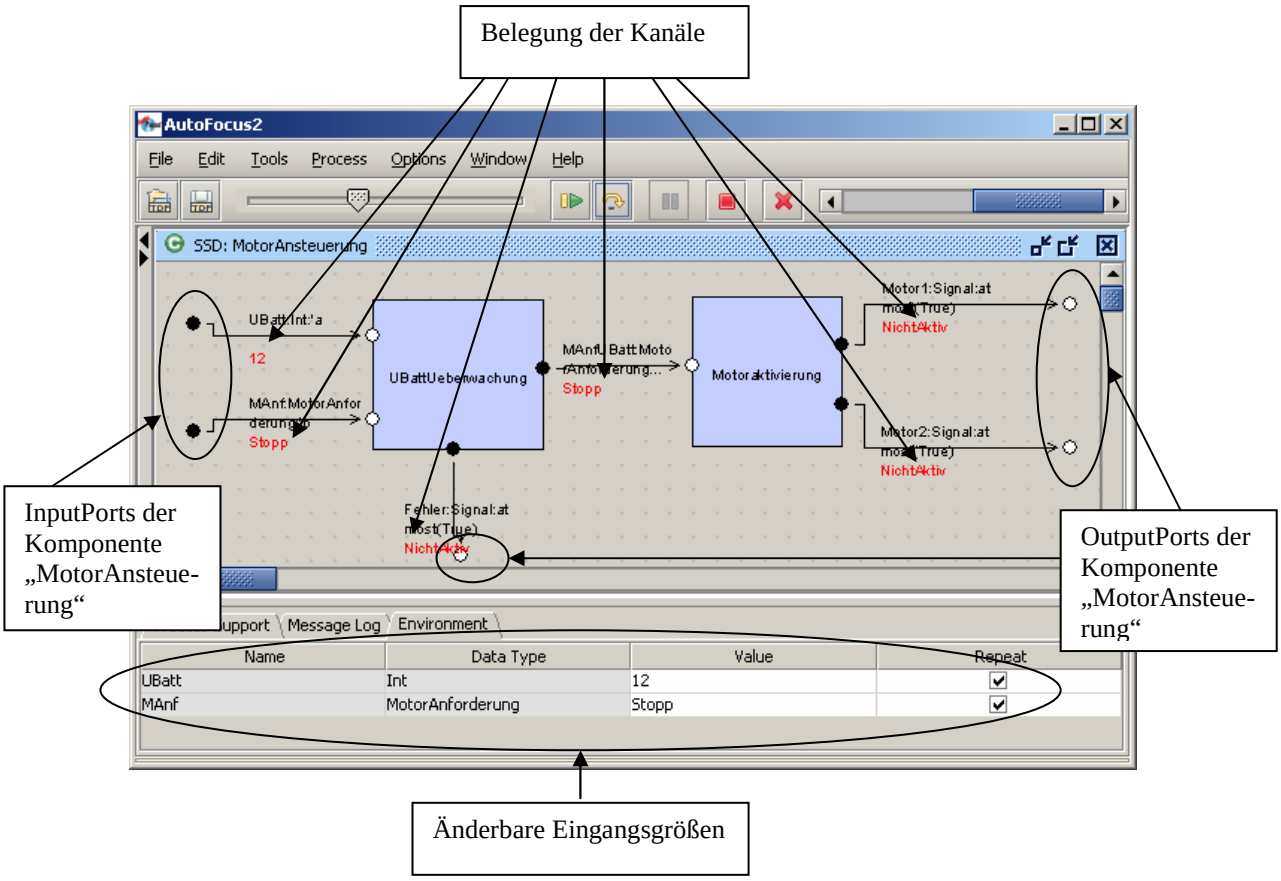

Abbildung 17: Komponente „MotorAnsteuerung“ bei Start der Simulation

In den folgenden Abbildungen (Abbildung 18 bis Abbildung 21) wird die Simulation schrittweise für die Motoranforderung „Hoch“ gezeigt. Die sich jeweils ändernden Daten sind zum leichteren Auffinden in den Abbildungen durch Ellipsen markiert. Dadurch ist gut zu sehen, wie sich die Änderung bei jedem Schritt auswirkt.

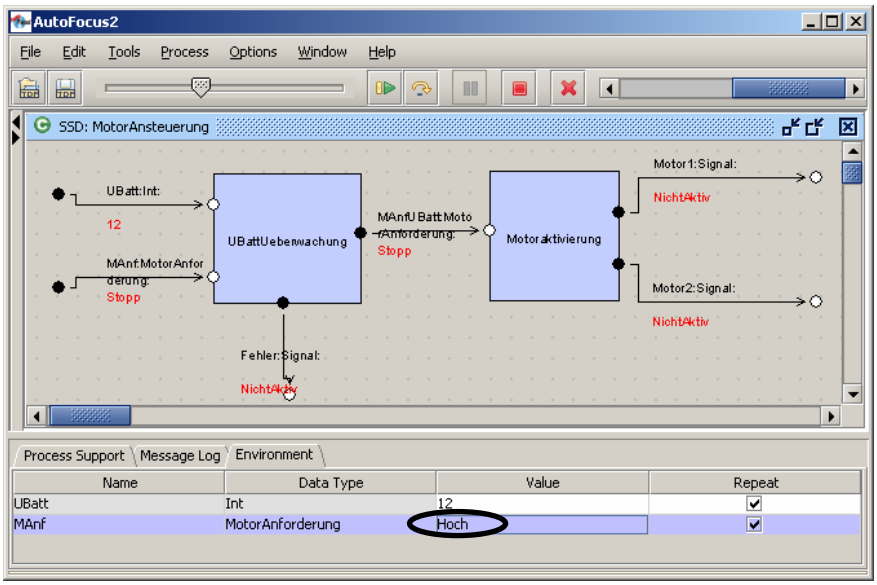

Abbildung 18: Eingang wird von „Stopp“ auf „Hoch“ gesetzt

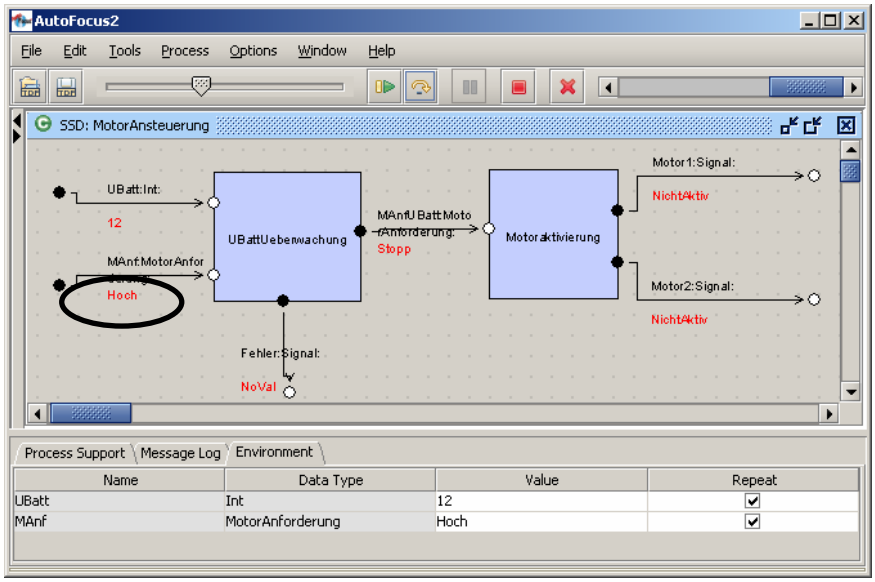

Abbildung 19: Schrittweise Simulation: Step 1

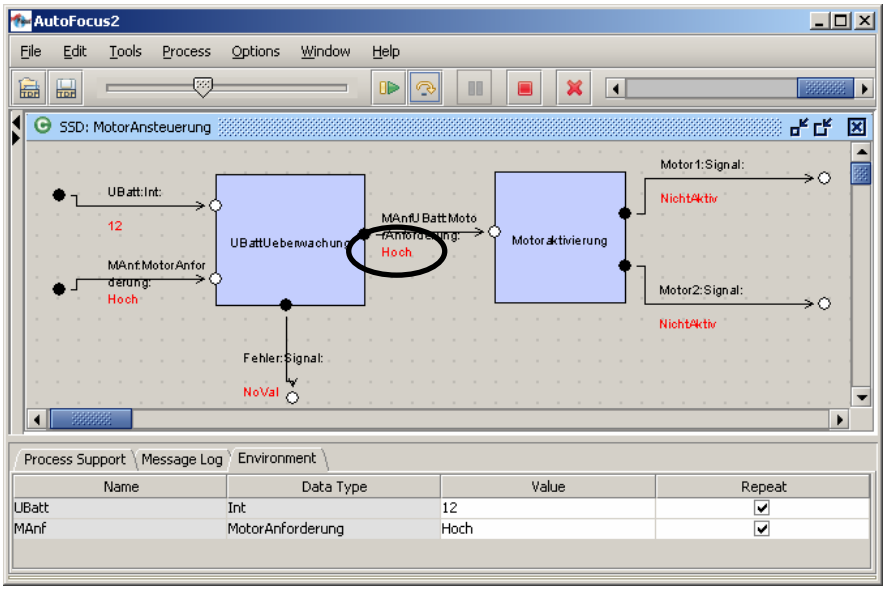

Abbildung 20: Schrittweise Simulation: Step 2

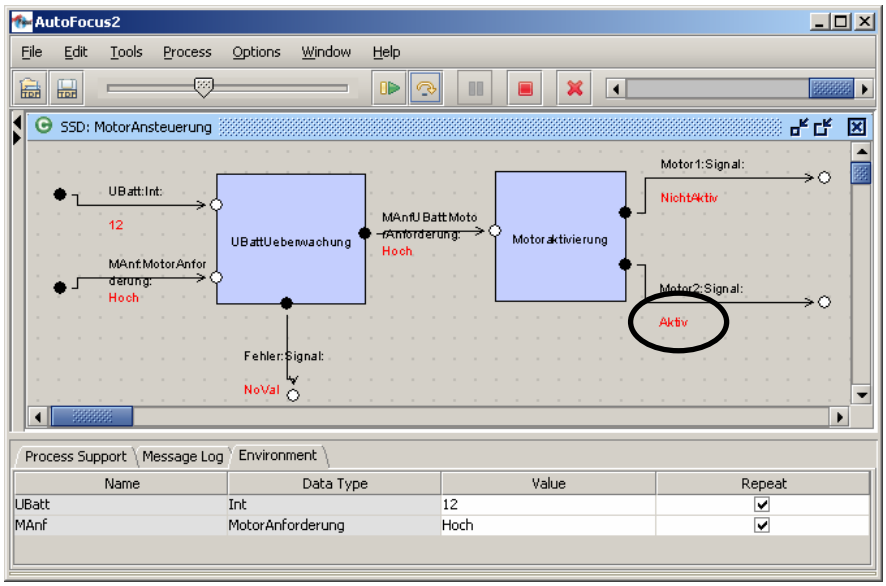

Abbildung 21: Schrittweise Simulation: Step 3

Am Anfang eines jeden Simulationsschrittes werden zunächst die Werte an den *InputPorts* übernommen. Der zur *Komponente* gehörende Zustandsautomat wird ausgeführt, d.h. falls eine Vorbedingung einer *Transition*, die vom aktuellen Zustand ausgeht, zutrifft, wird diese *Transition* ausgeführt und die zu ihr gehörenden Ausgaben werden an den *OutputPorts* bereitgestellt. Im nächsten Schritt steht dann dieser aktualisierte Wert am entsprechenden *InputPort* der „nächsten“ *Komponente* zur Verfügung.

Die zu den *Komponenten* gehörenden Zustandsautomaten können während der Simulation der *Komponenten* auch beobachtet werden. In Abbildung 22 und Abbildung 23 ist der Zustandsautomat zur *Komponente* „MotorAktivierung“ zu Beginn der Simulation und nach Ausführung des Simulationsschrittes 2 zu sehen.

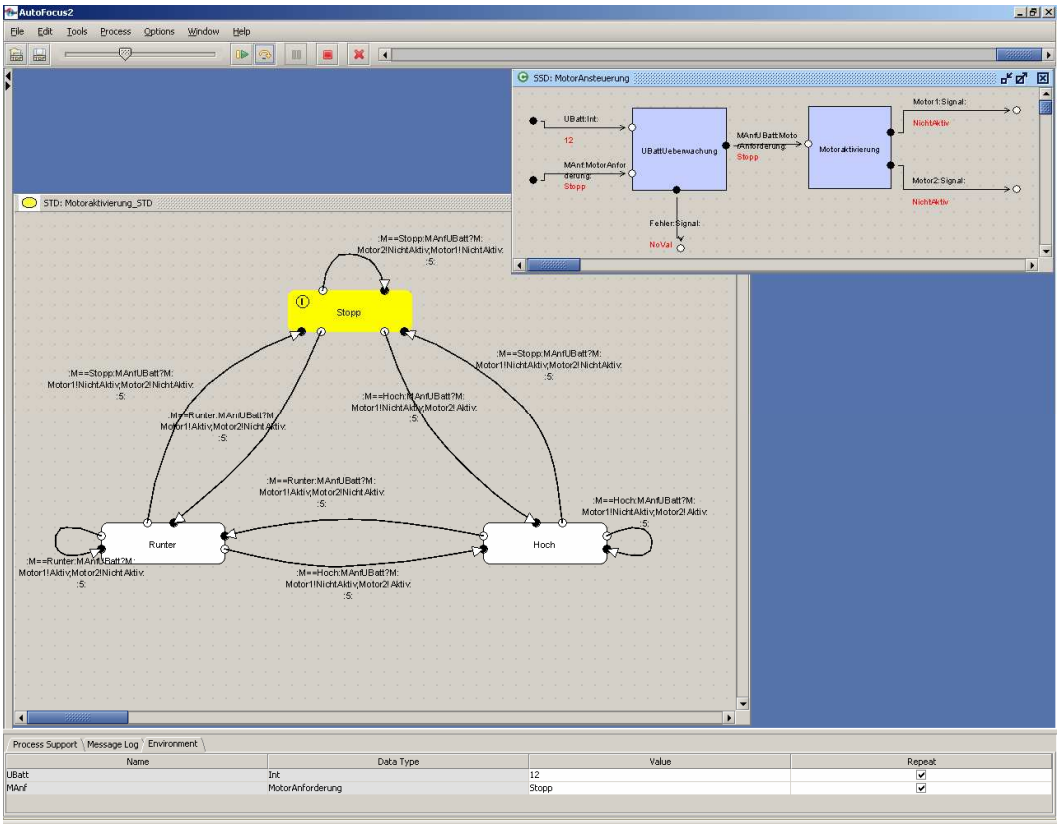

Abbildung 22: Simulationsstart bei STDs (gelb: aktive Zustände)

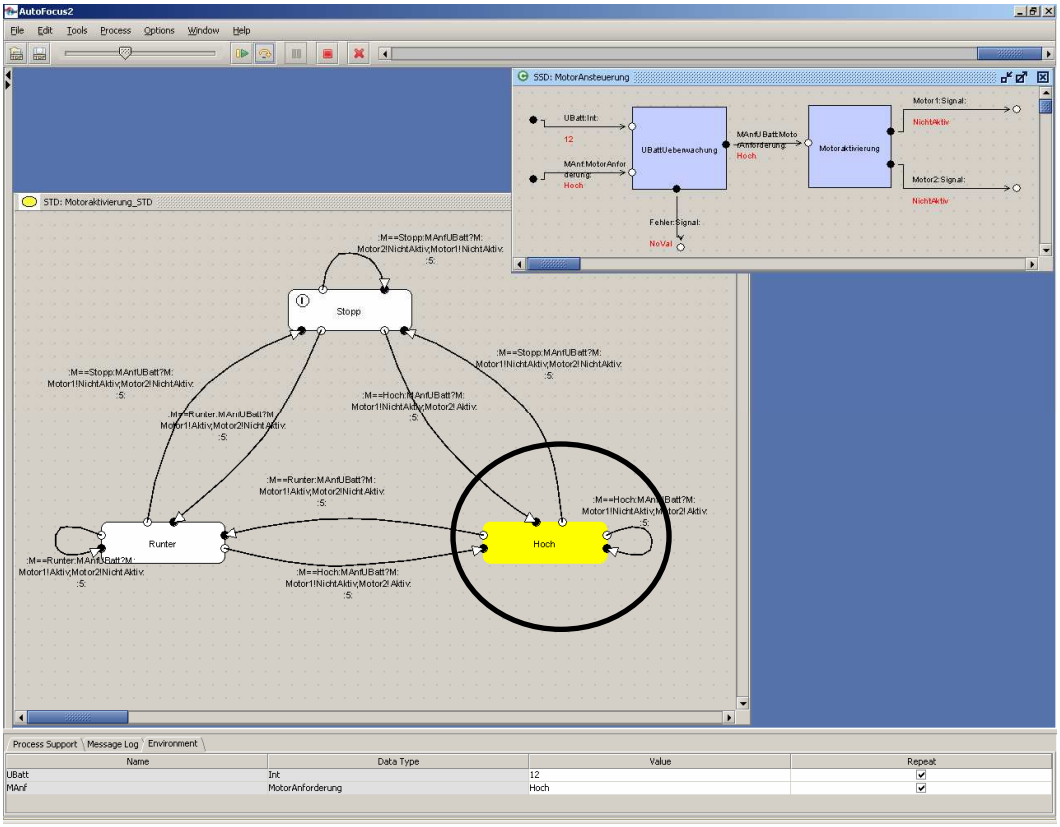

Abbildung 23: Step 2 bei STD für Komponente „Motoraktivierung“

4.6 Zusammenfassung

Durch die formale Spezifikation des zu entwickelnden Systems in AF2 erhält man ein Multi-sichten-Modell. Eine gemeinsame mathematische Basis integriert diese Schichten zu einem ausführbaren Gesamtmodell. Ein integriertes Simulationswerkzeug erlaubt die deterministische Ausführung und damit das Testen dieses Modells.

Der hierarchische Aufbau der Sichten SSD und STD unterstützt den Entwickler bei einem Top-Down-Entwurf, in dem er das System in den frühen Entwicklungsphasen zunächst nur grob beschreibt und in der weiteren Entwicklung durch Hinzufügen weiterer *Komponenten* bzw. weiterer *Zustände* nach und nach verfeinert, bis das System detailliert beschrieben ist.

5 Schnittstelle Anforderungsanalyse und formale Spezifikation

In den beiden vorangegangenen Abschnitten wurden die Konzepte für die Anforderungsanalyse und für die formale Spezifikation kurz vorgestellt. In diesem Abschnitt wird nun auf die Schnittstelle zwischen diesen beiden Teilen eingegangen. Es wird gezeigt, dass AF2 den Bruch zwischen den textuellen Anforderungen und dem formalen Modell überbrücken kann.

Dazu gibt es in AF2 zum einen die methodischen Konzepte der Szenarienanalyse und zum anderen zwei Beziehungstypen zwischen den *Requirements* aus der Anforderungsanalyse und den einzelnen Modellelementen aus der formalen Spezifikation, nämlich die *Motivation* und die *Association*. Im Folgenden werden zunächst die beiden Beziehungstypen erläutert. Auf die Szenarienanalyse wird dann beispielhaft im Abschnitt 5.4 eingegangen.

5.1 Beziehungstyp „Motivation“

Die *Motivation* ist eine Konstruktionsbeziehung zwischen den *Constraints* (= Forderung nach bestimmten Modellelementen) auf der Anforderungsseite und den daraus zu modellierenden Systemelementen (*Komponenten*, *Zuständen* und *Datentypen*) andererseits.

Im Einzelnen können folgende Elementpaare in einer *Motivation*-Beziehung stehen:

- *ArchitecturalConstraints* () – *Komponenten* ()
- *ModalConstraints* () – *Zustände* ()
- *DTDConstraints* () – *Datentypen* ()

Durch die *Motivation*-Beziehung von *Constraints* zu Systemelementen werden in den entsprechenden Sichten im Modellierungsteil die gewünschten Modellelemente angelegt und anschließend mit dem *Constraint*, welches das Modellelement motiviert, in Beziehung gesetzt.

In Abbildung 24 und Abbildung 25 wird am Beispiel eines *ArchitecturalConstraint* gezeigt, wie einfach diese *Motivation* von Modellelementen in AF2 durchgeführt werden kann. Abbildung 24 zeigt am Fallbeispiel in AF2 die Bedienführung zum Anlegen einer neuen *Komponente*, die aus einem *ArchitecturalConstraint* motiviert wurde.

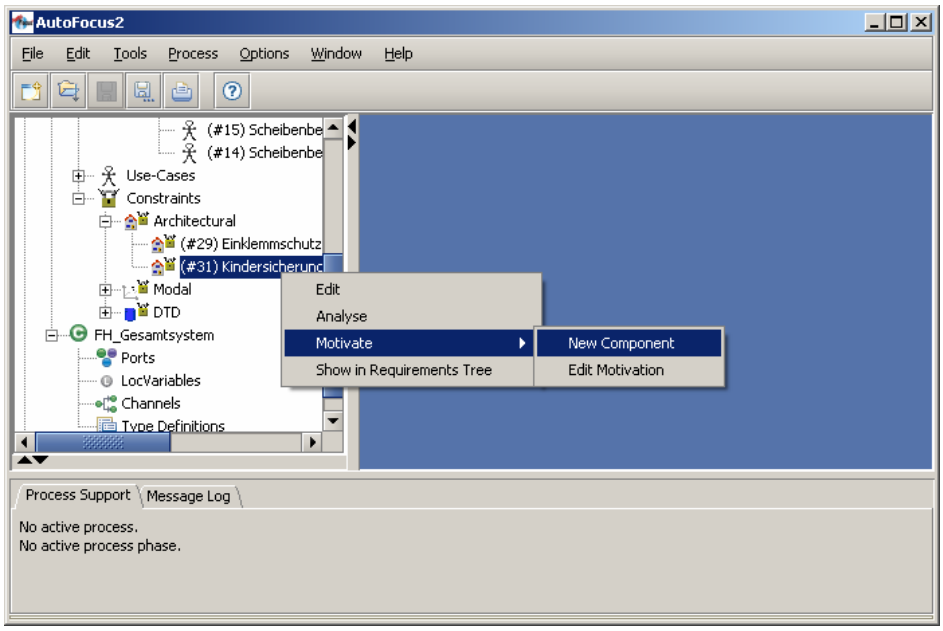

Abbildung 24: Motivation von neuen Modellelementen

Abbildung 25 zeigt die in der *Komponente* „FH_Gesamtsystem“ neu angelegte *Komponente* „Kindersicherung“.

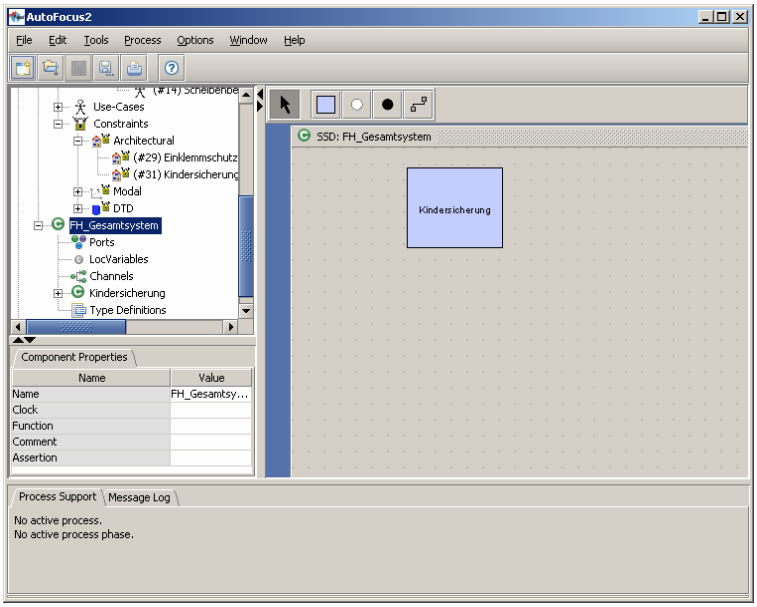

Abbildung 25: Neue Komponente „Kindersicherung“

In Abbildung 26 sieht man die sowohl im *Constraint* als auch in der *Komponente* angelegte *Motivation*-Beziehung.

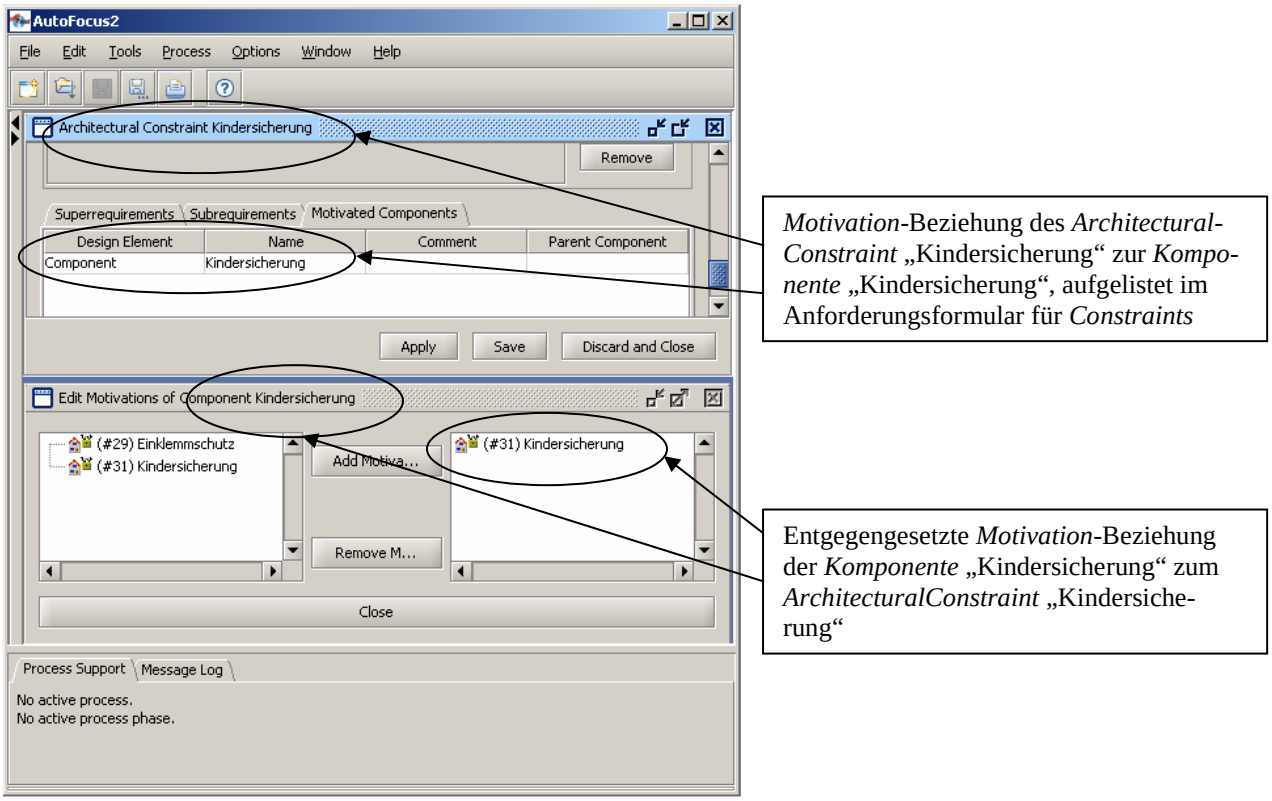

Abbildung 26: *Motivation*-Beziehung in *Constraint* und *Komponente*

Das Anlegen und Verwalten von *Motivation*-Beziehungen von *DTDConstraints* und *Modal-Constraints* erfolgt analog zu denen von *ArchitecturalConstraints*.[2]

5.2 Beziehungstyp „Association“

Durch die *Association*-Beziehung können von jedem möglichen *ApplicationRequirement* Verbindungen zu den Modellelementen *Komponente*, *Zustand*, *Datentyp*, *Kanal* und *Transition* angelegt werden. Dadurch sind alle relevanten Informationen direkt den entsprechenden Modellelementen zugeordnet. Es ist nicht möglich, *BusinessRequirements* durch *Association*-Beziehungen mit Modellelementen zu verbinden. Dadurch wird der Zielcharakter dieser Art von Anforderungen noch einmal hervorgehoben.

Das Anlegen und Verwalten der *Association*-Beziehung von den einzelnen *ApplicationRequirements* erfolgt analog zum Anlegen und Verwalten der *Motivation*-Beziehung. Abbildung 27 zeigt ein Beispiel.

[2] Das Anlegen von *Motivation*-Beziehungen zwischen *DTD-Constraint* und DTD funktioniert in der dem AF2-Bilderbuch zugrunde liegenden Version (März 2006) nicht.

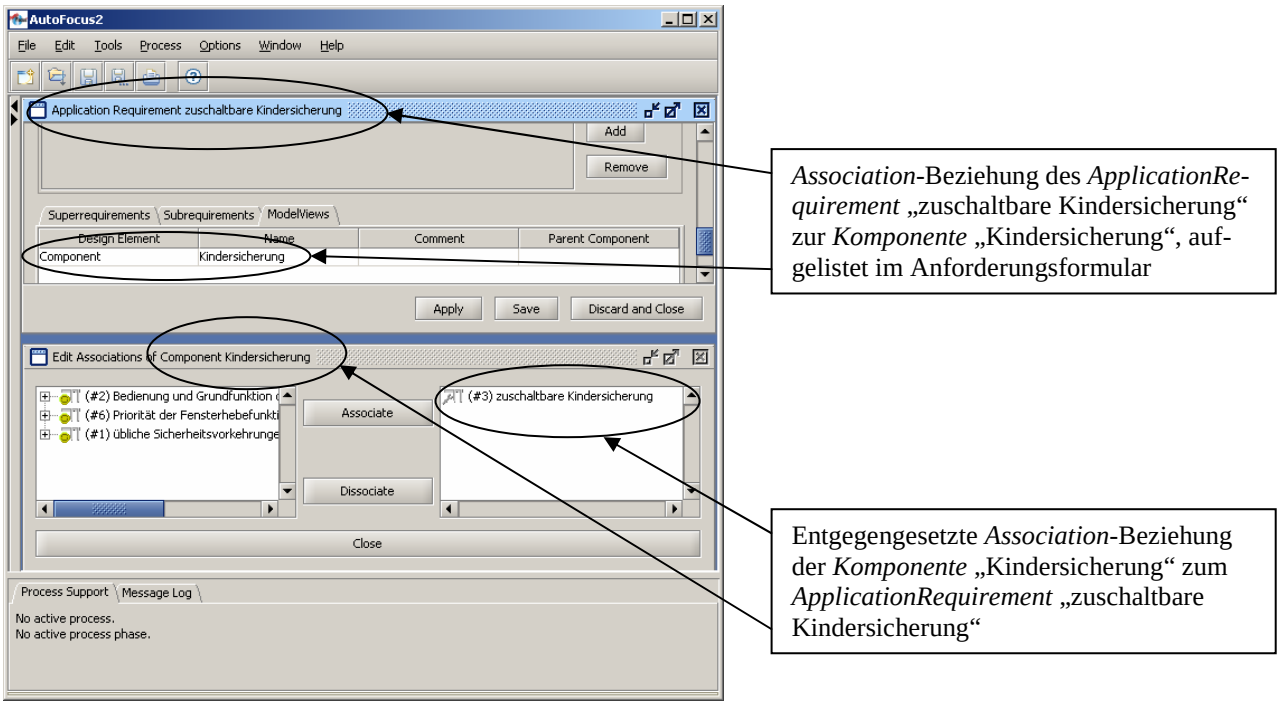

Abbildung 27: „Association“-Beziehung (analog zu „Motivation“-Beziehung)

5.3 Ein kleines Anwendungsbeispiel für die Schnittstelle: Fehlersuche

Anhand eines fiktiven Fehlers in einer möglichen Realisierung[3], wie sie aufgrund des AF2-Modells durchgeführt werden könnte, soll mit Hilfe einer exemplarischen Fehlersuche gezeigt werden, wie die Konzepte für die Anforderungsanalyse und der formalen Spezifikation in AF2 zusammenwirken. Dazu enthält das begleitende Beispielmodell des Fensterhebers einen (zunächst nicht offensichtlichen) Widerspruch zu den bisher festgelegten Anforderungen.

Aus dem Versuch wird folgende Fehlermeldung gemeldet:

„Bei Kälte funktioniert der Einklemmschutz nicht richtig! Das Fenster stoppt zwar, bewegt sich jedoch nicht nach unten bis zum Anschlag.“

Durch ein Protokoll über die Ein- und Ausgabewerte, das bei diesem Einklemmschutz-Test aufgenommen wurde, können die Eingabewerte nachvollzogen werden und es kann die im Folgenden gezeigte Simulation des Verhaltens durchgeführt werden.

Die Simulation (siehe Abbildung 28 und Abbildung 29) ergibt, dass der Fehler im Modell nachvollzogen werden kann: Auch im Modell wird das Fenster nicht – wie gefordert – nach unten bewegt!

[3] Die Realisierung des AF2-Modells wurde nicht durchgeführt. Die beschriebene Fehlermeldung ist erfunden, aber dennoch denkbar!

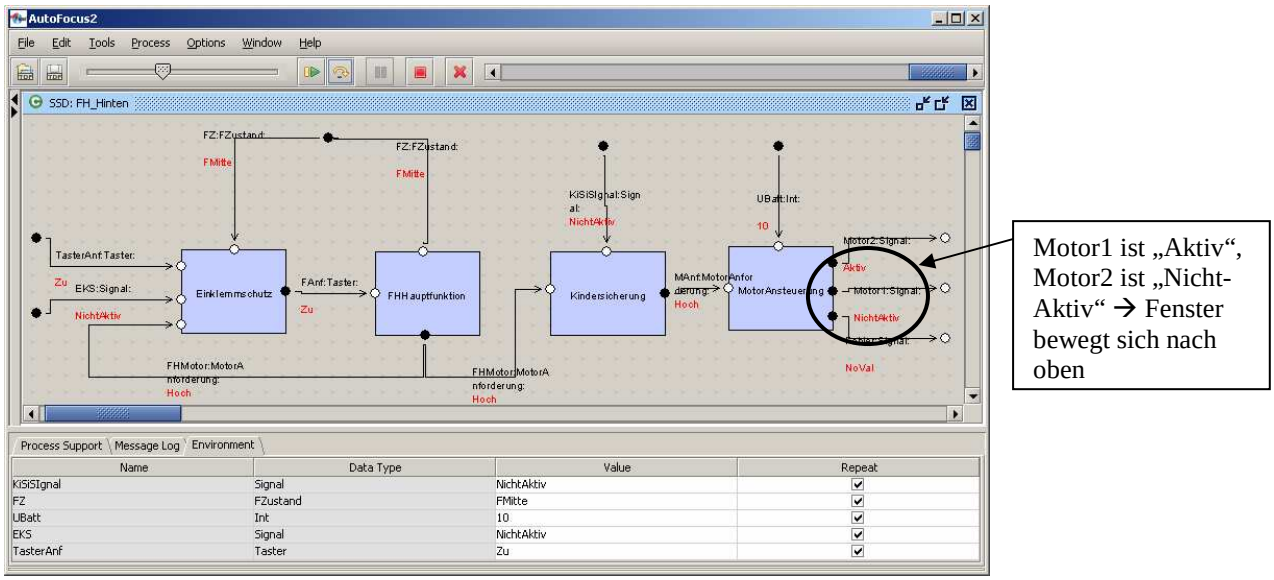

Abbildung 28: Fenster bewegt sich nach oben

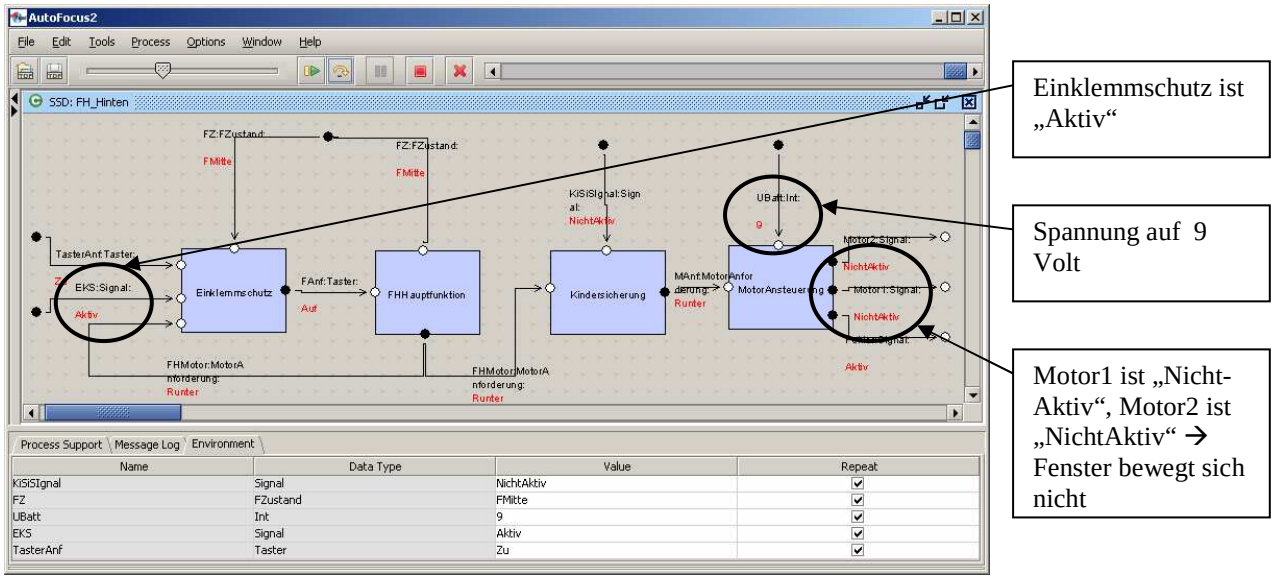

Abbildung 29: Einklemmschutz wird aktiv und Spannung auf 9 Volt (Vorgaben aus Fehlermeldung)
➔ Fenster geht nicht nach unten

Bei Analyse des Kommunikationsflusses fällt auf, dass in der *Komponente* „MotorAnsteuerung“ die OutputPorts für die Motoren auf „NichtAktiv“ gesetzt sind und der OutputPort „Fehler“ auf aktiv, obwohl die Anforderung „Runter“ an die *Komponente* „MotorAnsteuerung“ gestellt wurde.

Bei der weiteren Analyse des momentanen Zustands der *Komponente* „MotorAnsteuerung“ (siehe Abbildung 30) wird ersichtlich, dass in der *Komponente* „UbattUeberwachung“ die Anforderung „Runter“ an die *Komponente* „MotorAnsteuerung“ in „Stopp“ umgewandelt wird und der Ausgang „Fehler“ auf „aktiv“ gesetzt wurde.

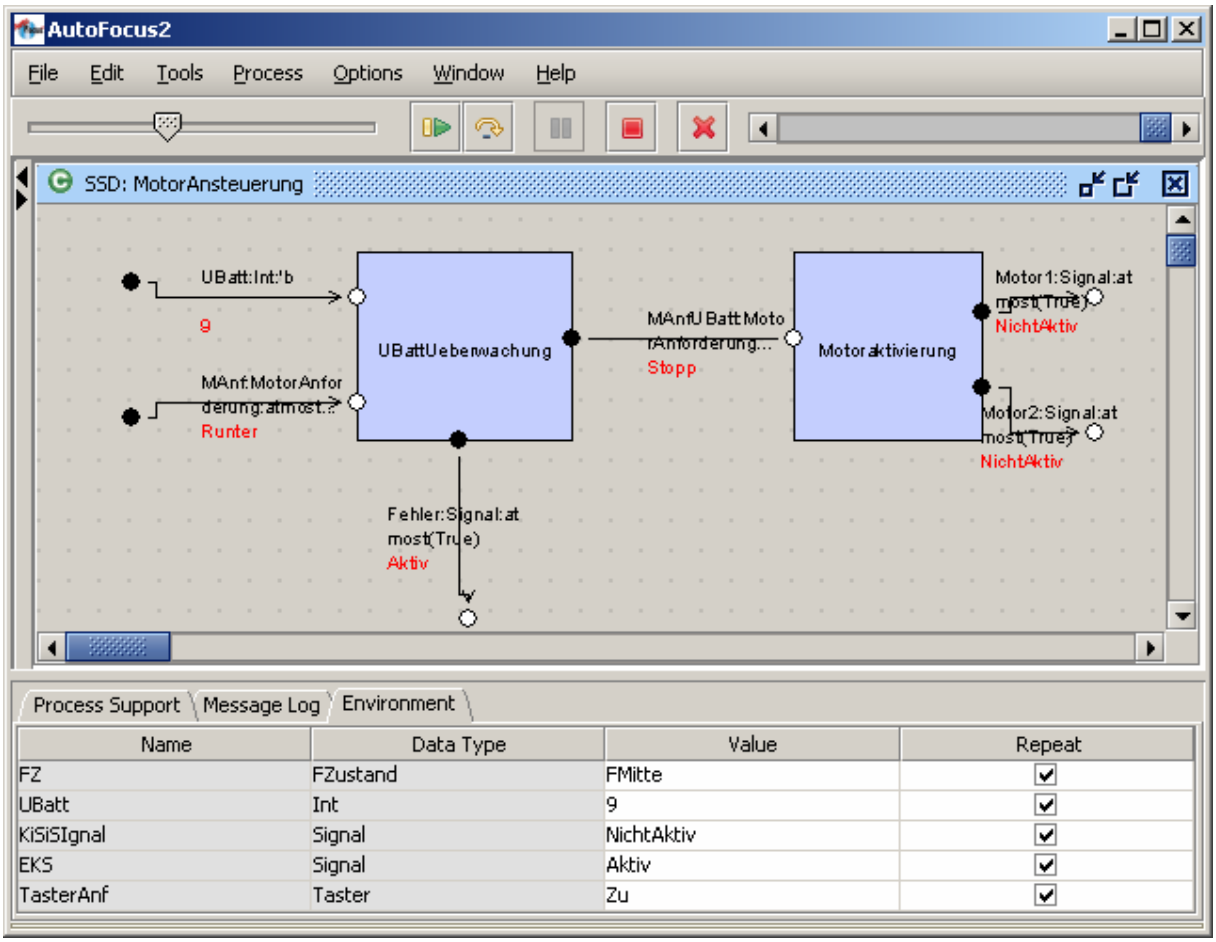

Abbildung 30: Zustand der Komponente „MotorAnsteuerung

Nun wird das zur *Komponente* „UbattUeberwachung“ gehörige STD überprüft (siehe Abbildung 31) und festgestellt, dass die Spannung bei Beginn der Bewegung des Fensters nach unten zu niedrig war und sich deshalb das Fenster nicht bewegt hat.

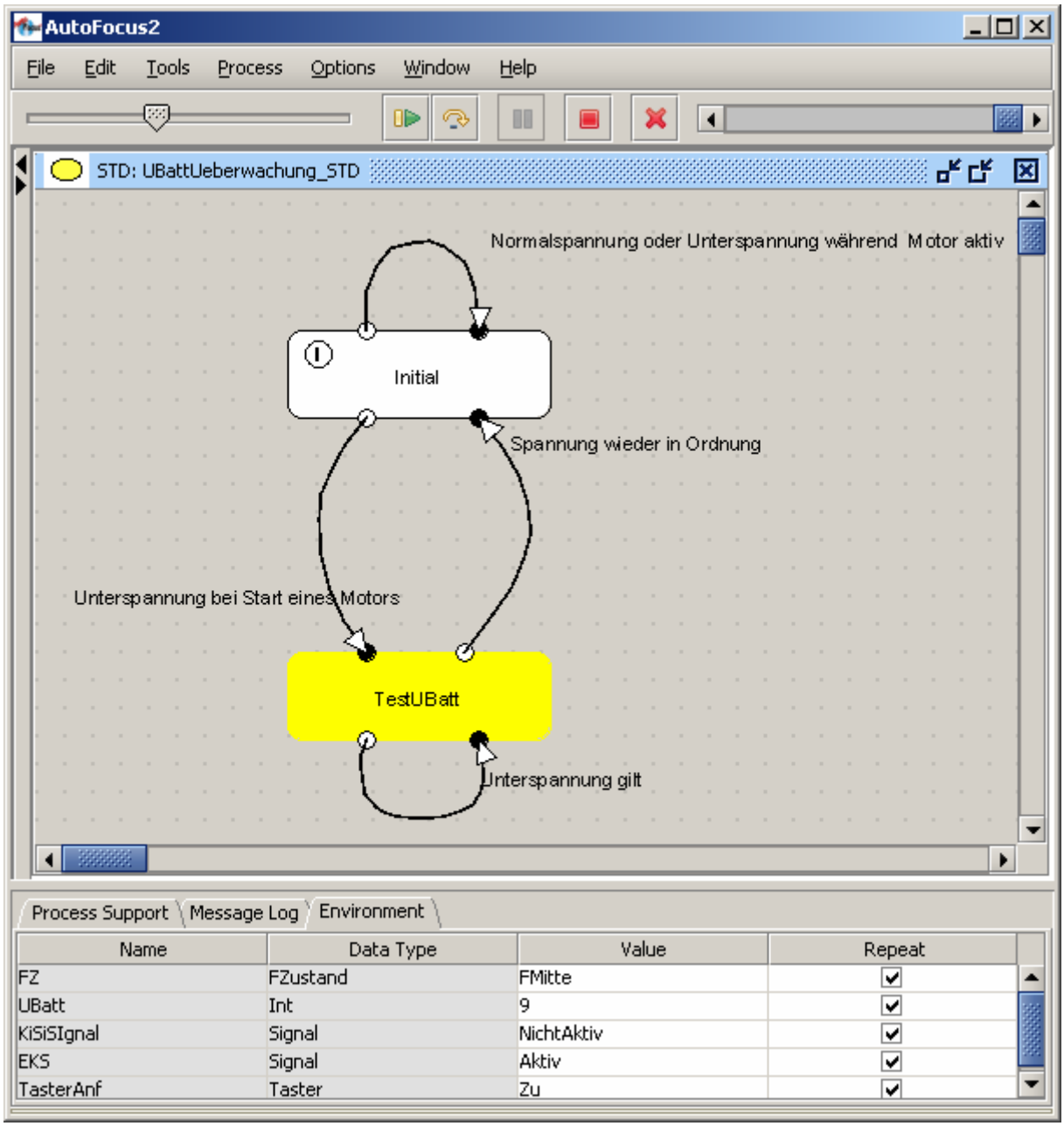

Abbildung 31: STD zur Komponente „UbattUeberwachung“

D.h. man kennt jetzt die Ursache des als Fehler beschriebenen Verhaltens aus dem Versuch: Die Unterspannung, deren Auftreten im Beispielszenario vermutlich durch die Kälte unterstützt wurde, verhindert die Scheibenbewegung nach unten.

Nun muss untersucht werden, warum das Verhalten so modelliert worden ist. Dazu werden die mit der *Komponente* „UbattUeberwachung“ assoziierten Anforderungen herausgesucht und angezeigt (Abbildung 32).

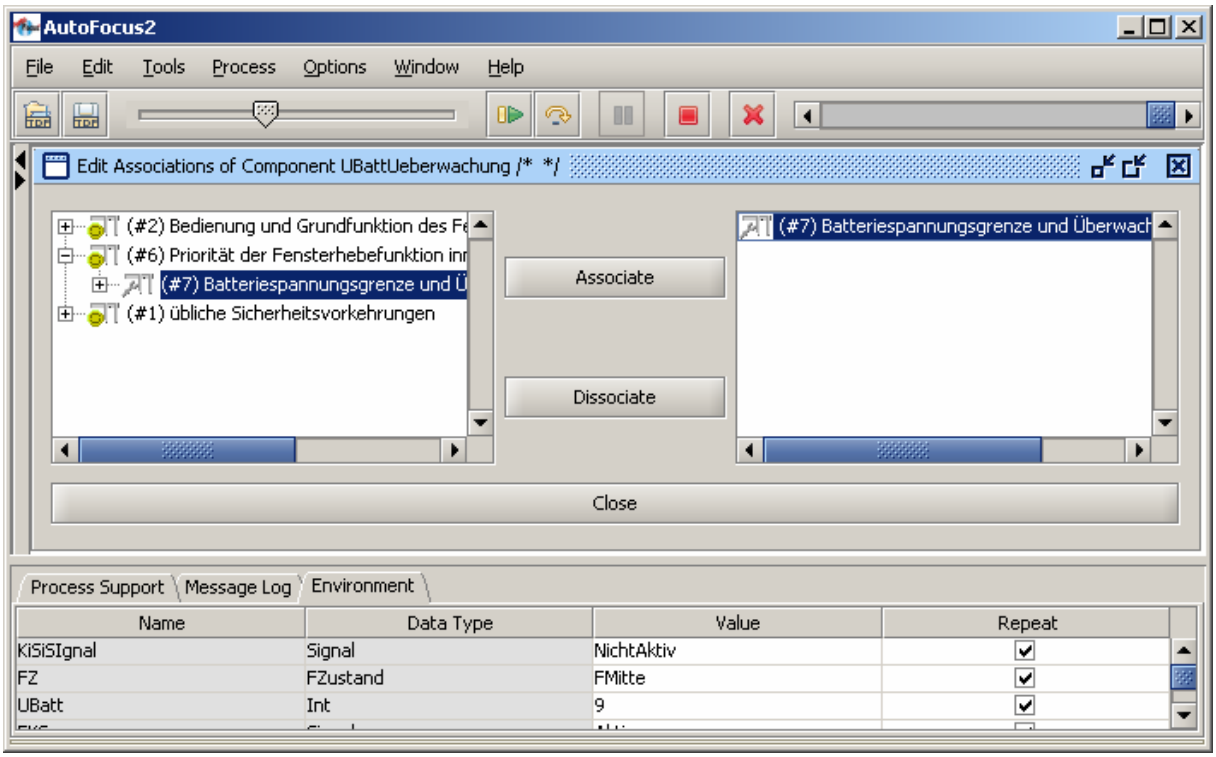

Abbildung 32: Assoziierte Anforderungen

Abbildung 33 zeigt die mit der *Komponente* „UbattUeberwachung“ assoziierte Anforderung (#7 Batteriespannungsgrenze und Überwachungszeitpunkt). Die Prüfung, ob die Anforderung in der entsprechenden Komponente richtig umgesetzt worden ist, ergibt keinen Fehler.

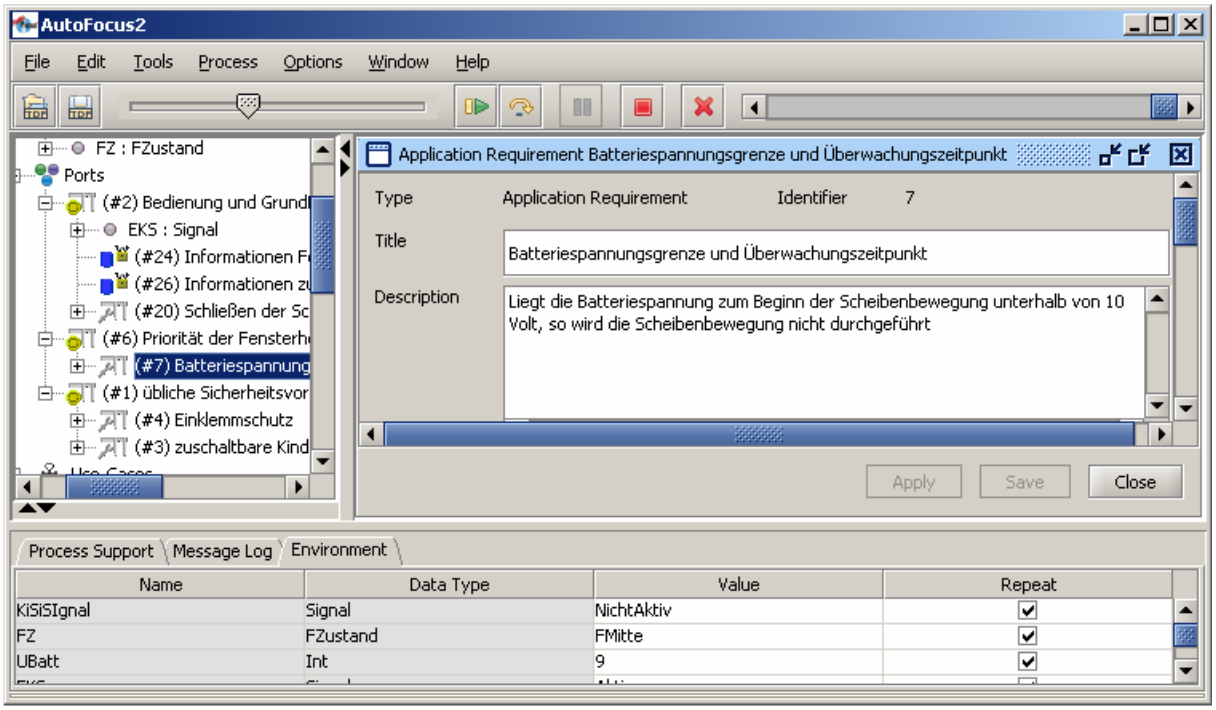

Abbildung 33: Assoziierte Anforderung #7

Die Anforderung wird entlang der Hierarchiebeziehung nach oben verfolgt zum *BusinessRequirement* # 6.

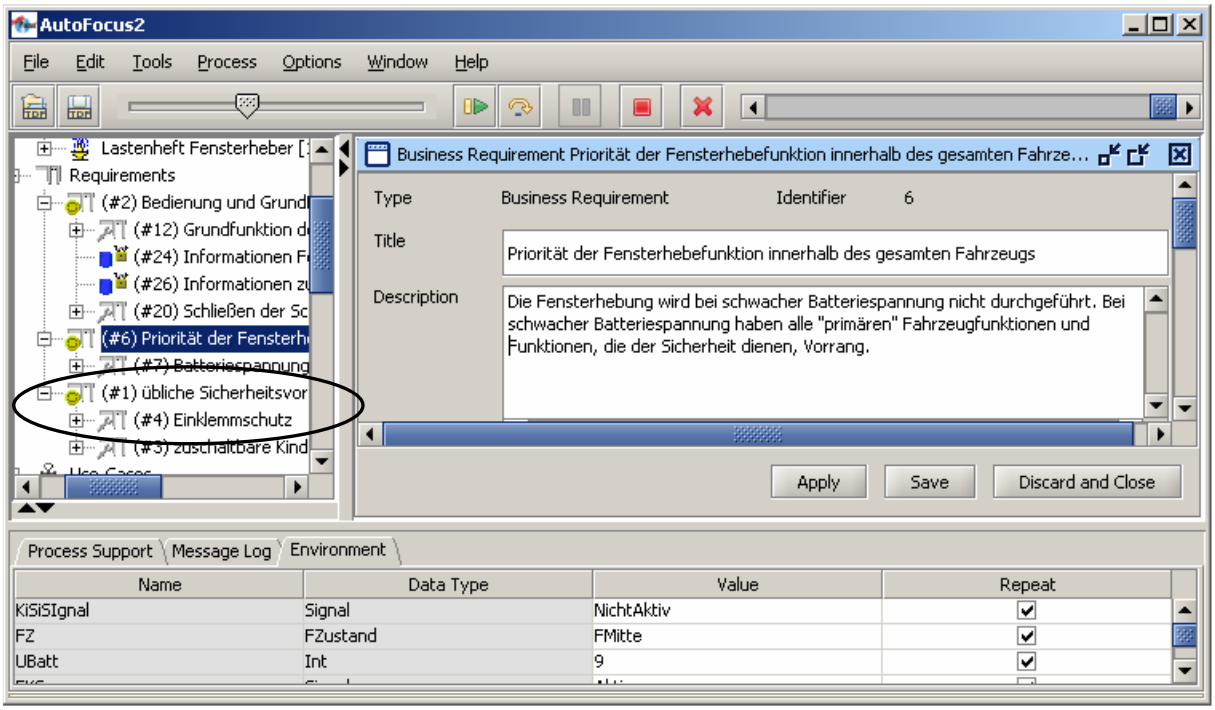

Abbildung 34: Übergeordnetes *BusinessRequirement* #6

Dieser Anforderung #6 „Priorität der Fensterhebefunktion innerhalb des gesamten Fahrzeuges“ (Abbildung 34) kann entnommen werden, dass alle primären Fahrzeugfunktionen und Sicherheitsfunktionen bei schwacher Batteriespannung Vorrang genießen und daher Komfortfunktionen, wie die Fensterhebung nicht durchgeführt werden.

Die Anforderung #4 „Einklemmschutz“ ist aus dem Ziel #1 „übliche Sicherheitsvorkehrungen“ (siehe Ellipse in Abbildung 34) abgeleitet worden und gehört somit zu den Funktionen, die der Sicherheit dienen und sollte deshalb nach #6 auch trotz schwacher Batteriespannung ausgeführt werden. Das Fenster im skizzierten Beispiel sollte also trotz schwacher Batteriespannung nach unten bewegt werden.

Damit wurden die nicht offensichtlichen Widersprüche in den Anforderungen aufgedeckt und es können nun die Anforderung und die dazugehörige Komponente entsprechend geändert werden.

5.4 Auflösen von Widersprüchen in Anforderungen mit Use-Cases

Mit Hilfe des Beispiels im vorangegangenen Abschnitt wurde gezeigt, wie man Fehler, die bei dem Übergang von textuellen Anforderungen zu formalen Modellen auftreten können, anhand der Beziehungen zwischen *Requirements* und Modellelementen in AF2 aufdecken kann.

Nun ist es aber von großem Interesse, diese Art von Fehler von vornherein – also schon bei der Analyse der Anforderungen – zu vermeiden.

AF2 unterstützt den Entwickler bei dieser Analyse durch das Konzept *Use-Case* und durch die Szenarienanalyse. Insbesondere mit Hilfe der detaillierten Szenarioanalyse und der Be-

trachtung der Systeminteraktionen (*Observations*) werden systematisch die konstruierten Modelle analysiert, überarbeitet und vervollständigt.

In AF2 werden Nutzungsprozesse und Systemfunktionen mit Hilfe der *Use-Cases* und der Szenario-Modellierung erarbeitet. Jedem *Use-Case* können mehrere repräsentative Szenarien zugeordnet werden.

Anhand des Beispiels aus Abschnitt 5.3 wird gezeigt, wie die Szenarienanalyse bei der Vermeidung von Fehlern helfen kann.

Abbildung 35 zeigt ein einfaches Szenario für den *Use-Case* „Batteriespannung bei Beginn der Fensterbewegung zu gering“, das nun analysiert werden soll.

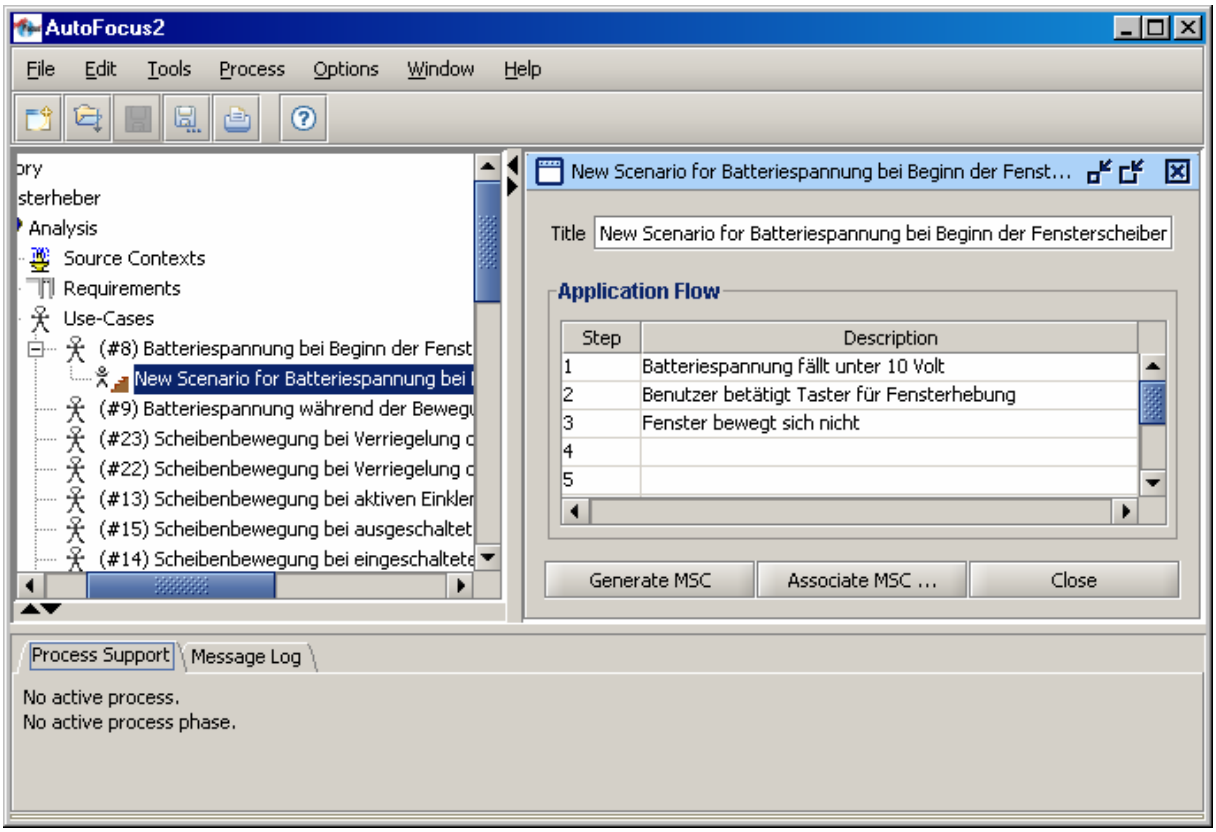

Abbildung 35: Szenario

Mit Hilfe der Szenarienanalyse (Communication Observation, Mode Observation, State Observation[4]) kann nun jeder Schritt analysiert werden (siehe Abbildung 36).

[4] In der dem AF2-Bilderbuch zugrunde liegenden Version geht nur „CommunicationObservation“. Die anderen Analysearten funktionieren nicht.

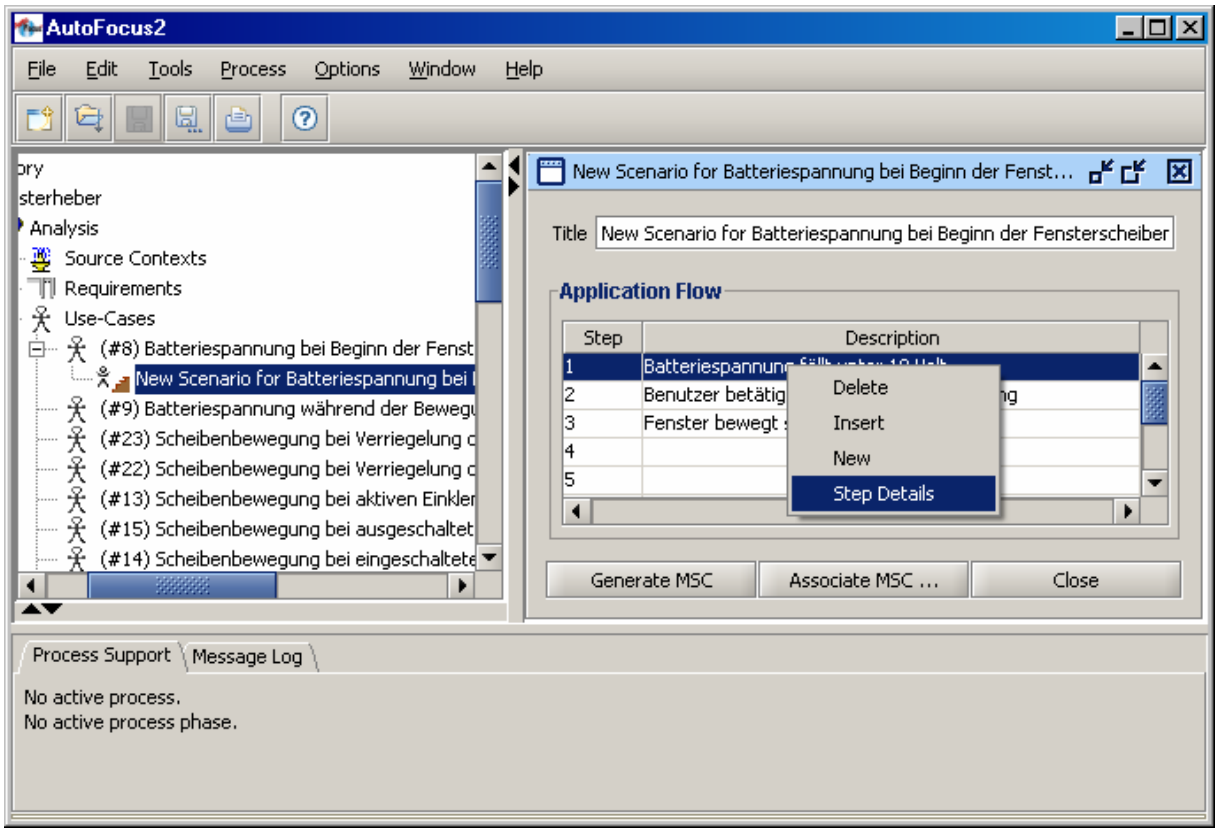

Abbildung 36: Kontextmenü zu einem Step

Nun können, wie in Abbildung 37 gezeigt, für jeden einzelnen Step die betroffenen Komponenten und Kanäle angegeben werden. Falls diese Elemente noch nicht existieren, müssen sie erst neu angelegt werden (z.B. durch eine *Motivation*-Beziehung), um sie dann in der Szenarienanalyse benutzen zu können.

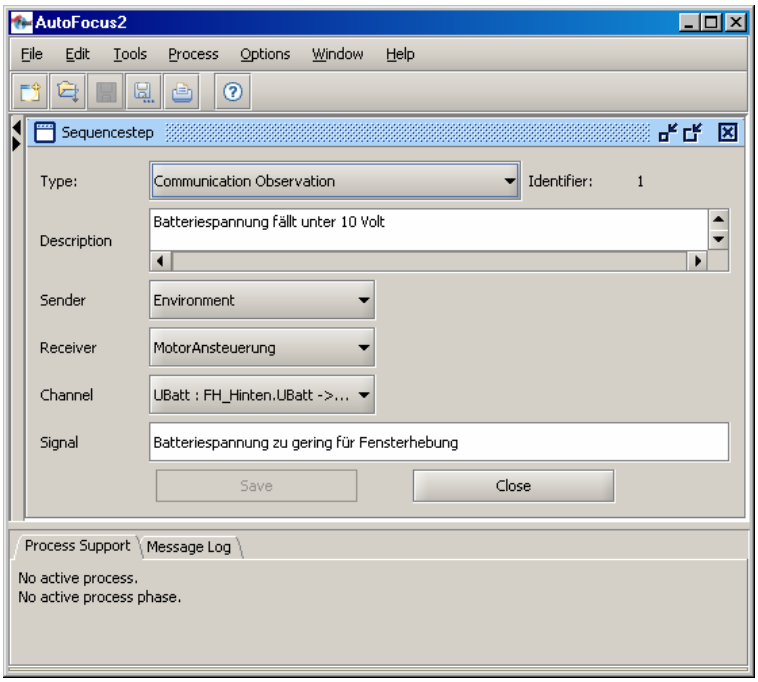

Abbildung 37: Detaillierte Angaben zu einem Step

Wenn für jeden Schritt die entsprechenden Angaben gemacht worden sind, d.h. für das gesamte Szenario wurden die beteiligten Komponenten und der Informationsfluss identifiziert, kann in AF2 aus diesen Angaben ein MSC (Message Sequence Chart) generiert werden, welches das Szenario anschaulich darstellt (siehe Abbildung 38).

In diesem Fallbeispiel wurden bei den Angaben für das Signal nicht die Werte, die für den entsprechenden getypten *Kanal* erlaubt wären, eingegeben, sondern natürlichsprachige Formulierungen, die die Bedeutung der Signale im MSC wiedergeben. Diese Formulierungen sollen das Lesen der einzelnen MSCs erleichtern.

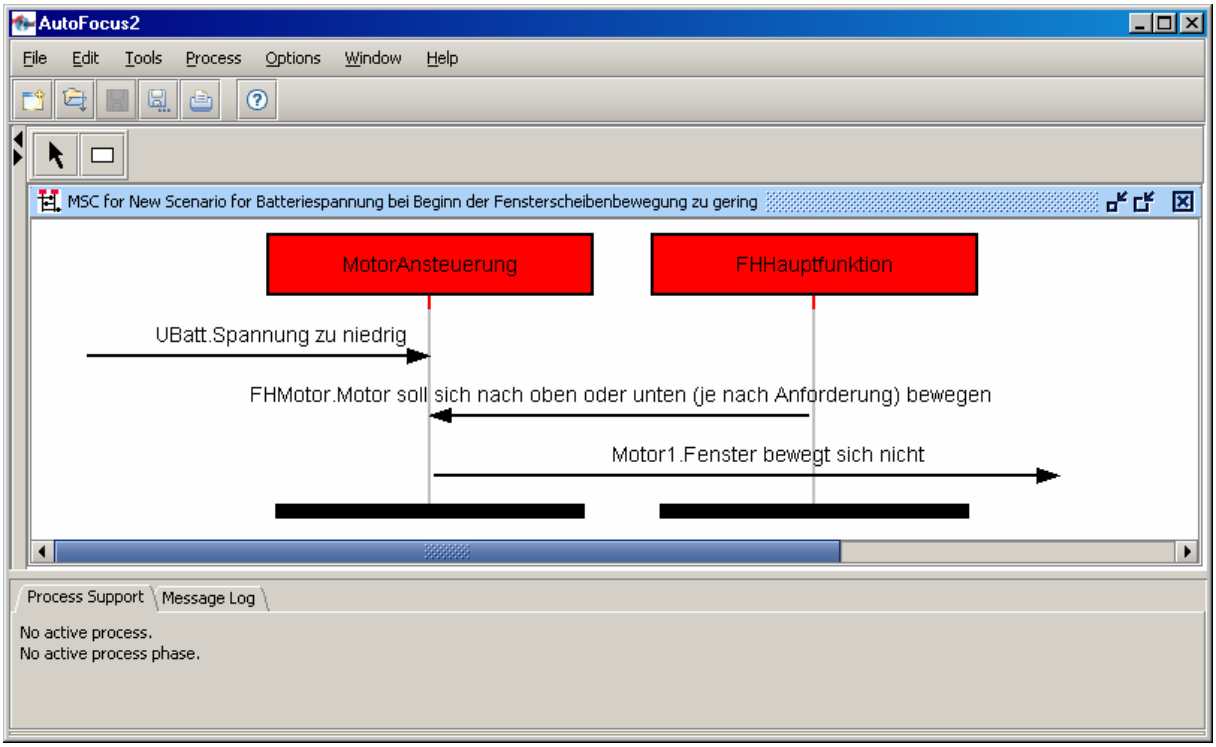

Abbildung 38: Generiertes MSC für Unterspannung

Die gleiche Prozedur wird auch mit der Funktion „Einklemmschutz“ durchgeführt und man erhält ein MSC, so wie es in Abbildung 39 dargestellt wird.

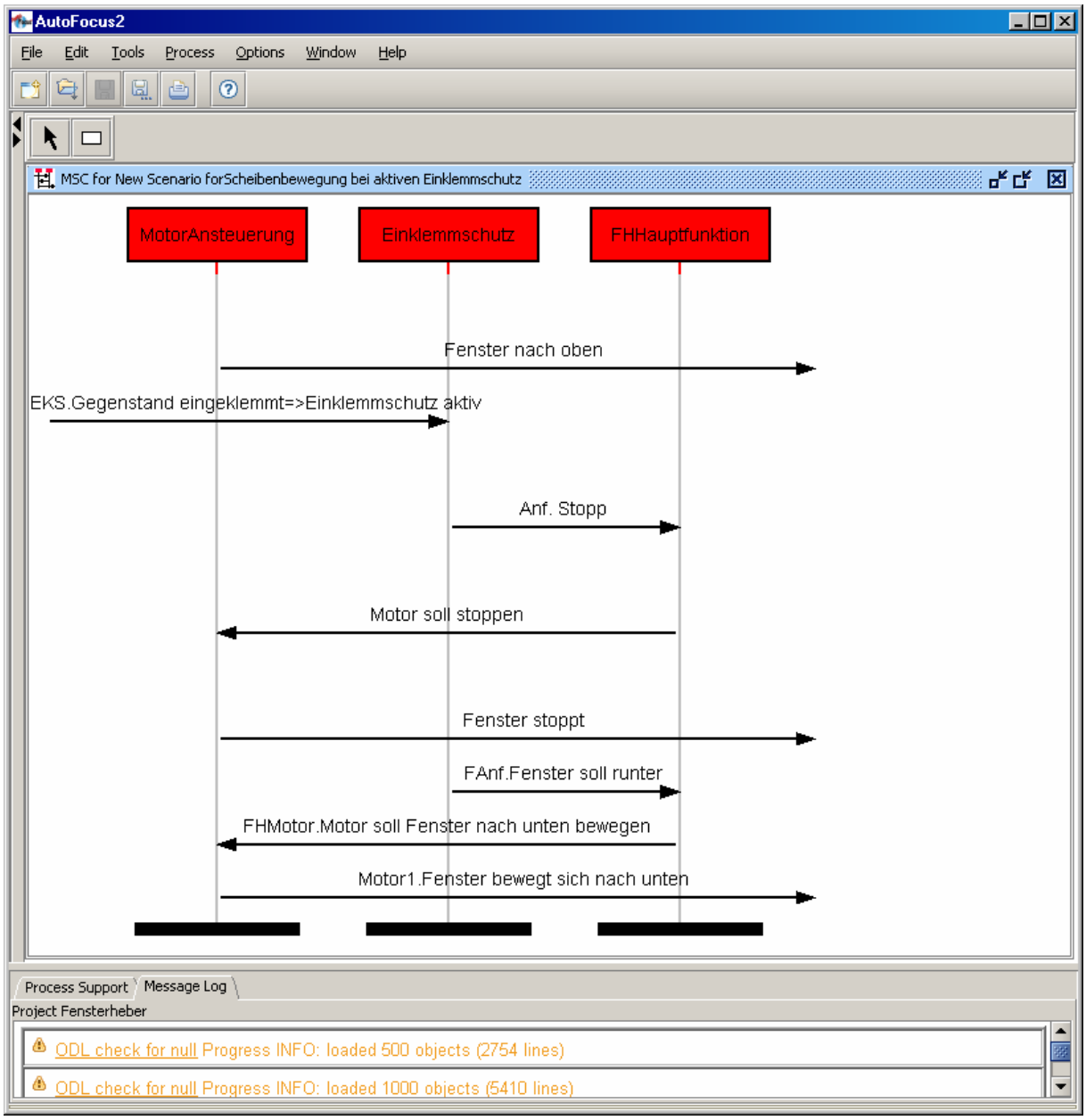

Abbildung 39: MSC für Einklemmschutz

Anhand der beiden MSCs wird sichtbar, dass auch bei dem MSC für den Einklemmschutz eine Bewegungsanforderung von der Komponente „FHHauptfunktion“ zur „MotorAnsteuerung“ gesendet wird, genauso wie beim MSC „Batteriespannung zu klein“.

An dieser Stelle fällt auf, dass hier ein ungewolltes Zusammenspiel von verschiedenen Funktionalitäten auftreten kann. Ein Backward-Tracing zu den entsprechenden Anforderungen kann dann diesen Widerspruch auflösen, so dass dem Modell – und damit der Implementierung – widerspruchsfreie Anforderungen zugrunde liegen.

5.5 Zusammenfassung

Die Verfeinerung der Anforderungen in *Constraints* und in *Use-Cases* ist ein erster Schritt zur Modellerstellung und Systemdefinition. Es werden schrittweise die Komponenten des Systems festgelegt, das System abgegrenzt, die Schnittstellen skizziert und die zu entwickelnden Systemfunktionen identifiziert.

Mit der *Motivation*-Beziehung können aus den einzelnen *Constraints* entsprechende Modellelemente für das formale Modell erzeugt werden, und mit der *Assoziation*-Beziehung können die *ApplicationRequirements* mit den Modellelementen verbunden werden. Dadurch werden diese mit wichtigen Informationen angereichert und detailliert spezifiziert.

Die Beziehungen zwischen Anforderungen und Modellelementen werden auf beiden Seiten festgehalten. Durch *Use-Cases* und durch die Szenarienanalyse können sowohl funktionale als auch nicht-funktionale Anforderungen auf funktionale Modelle (SSD, STD, DTD) abgebildet, weiter verfeinert und strukturiert werden.

Ein wichtiger Aspekt, der durch den Aufbau dieses Beziehungsgeflechts unterstützt wird, ist das Forward- und Backward-Tracing von Anforderungen zum Modell und vice versa.

6 Schlussbemerkung

Dieses Bilderbuch gibt anhand ausgewählter kommentierter Bilder einen ersten Überblick über die Kernkonzepte (Anforderungsanalyse, Formale Spezifikation, Simulation) des Werkzeugs AF2 und zeigt exemplarische Anwendungsszenarien auf.

Bei AF2 handelt es sich - wie eingangs erwähnt - um ein Forschungswerkzeug, das hauptsächlich dazu dient, durchdachte Konzepte zu evaluieren bzw. zu präsentieren. AF2 kann daher kein in allen Belangen ausgereiftes Produkt sein. Mit der Version, die diesem Bilderbuch zugrunde liegt (Mai 2006), ist es jedoch gut möglich, Beispiele im kleineren bis mittleren Umfang zu modellieren.

Detailliertere Informationen über AF2 und AF2 selbst können auf der Homepage von AutoFOCUS 2 heruntergeladen werden [1]. Dort stehen auch wissenschaftliche Veröffentlichungen zur Verfügung, die die theoretische Fundierung von AF2 beinhalten. Ein Benutzerhandbuch für AF2 kann ebenfalls über diese Web-Seite bezogen werden.

7 Literaturverzeichnis

[1] AutoFOCUS 2 – Webseite, http://www4.in.tum.de/~af2/, 2006.

[2] F. Houdek, B. Paech: Das Türsteuergerät – Eine Beispielspezifikation. IESE Report 002.02/D, Januar 2002.

[3] AutoFOCUS – Webseite, http://autofocus.in.tum.de, 2005.

[4] M. Broy, K. Stølen: Specification and development of interactive systems – FOCUS on Streams, Interfaces and Refinement, Springer, 2001.

[5] AutoRAID – Webseite, http://www4.in.tum.de/~autoraid/, 2005.

[6] E. Geisberger: Requirements Engineering eingebetteter Systeme – ein interdisziplinärer Modellierungsansatz, Dissertation TU München, 2005.

[7] E. Geisberger, B. Schätz: Interdisziplinäre Anforderungsanalyse mit dem modellbasierten RM-Werkzeug AutoRAID, eingereicht zur Modellierung 2006 (Innsbruck), 2006.

Kontakt:

AutoFOCUS 2-Team
Lehrstuhl für Software & Systems Engineering
Technische Universität München
Boltzmannstraße 3
85748 Garching b. München

Email: af2@in.tum.de
Web: http://www4.in.tum.de/~af2/