Anhang zur Arbeit

„<TRADEKNOWLEDGE/>"

KONZEPTION, ENTWICKLUNG,
TEST UND BETRIEB
VON HANDELSSYSTEMEN
AUF BASIS EINES WISSENSBASIERTEN MODELLS

Detlev Matthes, *dmatthes@web.de*
Oktober 2022

1 INHALT

4 Iterationen des Entwicklungsprozesses ... 5

5 Wissensabbildung ... 7

5.1 Attribute ... 7

5.2 Properties ... 8

5.3 Operationen ... 8

5.4 Beziehungen ... 8

6 Weitere Bestandteile von TradeKnowledge ... 10

6.1 Abbildung von Werten ... 10

6.2 Abbildung von Funktionen ... 11

6.2.1 Terme für Werte ... 12

6.2.2 Terme für Bedingungen ... 13

6.2.3 Kombinierte Terme ... 14

6.3 Meldungen von TradeKnowledge ... 15

6.3.1 Meldungen innerhalb der Modellierungsphase ... 15

6.3.2 Meldungen zur Laufzeit ... 15

7 Weiterführende Ideen mit TradeKnowledge ... 16

7.1 Profile-matching ... 16

7.2 Konzeption und Ideenfindung ... 17

7.3 Modell-Prozessierung ... 18

7.4 Funktionstests von Modellbestandteilen ... 20

7.5 Einheitliche Dokumentation ... 21

7.6 Tools zur Unterstützung des praktischen Tradings ... 22

7.7 Erkennung von Zukunftsblicken ... 23

7.7.1 Überprüfung und Zertifizierung von Termen ... 23

7.7.2 2-Phasen Modellprozessierung ... 23

8 Prototypische Umsetzung ... 25

9 Vollständiges Beispiel ... 28

9.1 Modellierung des Handelssystem (3. Phase) ... 29

9.1.1 Instanz TradingSystem ... 29

9.1.2 Instanz PositionsManagement ... 31

9.1.3 Instanz RisikoAndMoneyManagement ... 32

9.1.4 Instanz Var ... 34
9.1.5 Indikatoren ... 35
9.1.6 Szenarien ... 37
9.1.7 Einstiegsszenario ... 37
9.1.8 Ausstiegsszenario für die Verlustbegrenzung ... 39
9.1.9 Alternatives Ausstiegsszenario ... 41
9.1.10 Zusammenfassung ... 44
9.1.11 Optionale Schritte ... 44
9.2 Transformation in ProBuilder-Code für ProRealtime (4. Phase) ... 45
9.3 Deployment des Handelssystems (5. Phase) ... 48

2 ABBILDUNGSVERZEICHNIS

Abbildung 1: Iterationen innerhalb des gesamten des Entwicklungsprozesses ... 6
Abbildung 2: Ausgewählte Aspekte der Wissensabbildung ... 7
Abbildung 3: Beispiel Assoziation mit Multiplizität in der Wissensabbildung ... 8
Abbildung 4: Beispiel Komposition ... 9
Abbildung 5: Abbildung von Werten als Ableitungen der Var-Klasse ... 10
Abbildung 6: Speicherung von Werten innerhalb der Vars-Instanz ... 11
Abbildung 7: Abbildung von Termen für unterschiedliche Ergebnis-Typen ... 12
Abbildung 8: Terme für einfache Bedingungen ... 13
Abbildung 9: Abbildung von einfachen Operationen ... 14
Abbildung 10: Profilematching ... 16
Abbildung 11: Konzeption und Ideenfindung ... 17
Abbildung 12: Ausführung Backtest innerhalb eines Modellprozessors ... 19
Abbildung 13: Generierung von Dokumentation aus einem Modell heraus ... 22
Abbildung 14: Modellierungstool als Java-Anwendung ... 27
Abbildung 15: Modellierungstool: Instanz des Handelssystems ... 30
Abbildung 16: Modellierungstool: Positionsmanagement ... 31
Abbildung 17: Modellierungstool: Attribute für das Risko- und Moneymanagement ... 32
Abbildung 18: Modellierungstool: fehlender Verweis auf ein Ausstiegsszenario für die Riskobegrenzung .. 33
Abbildung 19: Modellierungstool: Var-Instanz ... 34
Abbildung 20: Modellierungstool: Parameter für die Indikatoren ... 35
Abbildung 21: Modellierungstool: Menüpunkt zur Erstellung eines SMA-Indikators ... 36
Abbildung 22: Modellierungstool: modellierte Indikatoren ... 36
Abbildung 23: Modellierungstool: Modellierungsmeldungen für fehlende Szenarien ... 37
Abbildung 24: Modellierungstool: noch unvollständiges Einstiegsszenario ... 38
Abbildung 25: Modellierungstool: CrossOver-Term im Einstiegsszenario ... 39
Abbildung 26: Modellierungstool: vorgegebener SignalPreis (Close der Woche) im Einstiegsszenario ... 39
Abbildung 27: Modellierungstool: vordefinierter DefaultExitLimitSzenario ... 40
Abbildung 28: Modellierungstool: DefaultExitLimitSzenario mit Stopp-Loss-Preis ... 40
Abbildung 29: Modellierungstool: Zuweisung zum RisikoAndMoneyManagement ... 41

Abbildung 30: Modellierungstool: Attribute des Exitszenarios und Signalterm 42
Abbildung 31: Modellierungstool: vorgegebener SignalPreis (Close der Woche) im Ausstiegsszenario 43
Abbildung 32: Modellierungstool: Gegenüber TradeKnowledge gültiges Modell 44
Abbildung 33: Modellierungstool: Transformation in Programmcode 45
Abbildung 34: Der generierte Programmcode des modellierten Handelssystems in ProRealtime deployed .. 48
Abbildung 35: Handelssystem innerhalb eines Charts 49

3 TABELLENVERZEICHNIS

Tabelle 1: Meldungen während der Modellierung 15
Tabelle 2: Meldungen zur Laufzeit 15
Tabelle 3: beispielhafte Testfälle für einen Trendfilter 20
Tabelle 4: Verwendete Tools und Technologien für die prototypische Umsetzung 26

4 ITERATIONEN DES ENTWICKLUNGSPROZESSES

Ergänzend zum, in der Arbeit im Kapitel 6.1 gezeigten, Entwicklungsprozess verdeutlicht die folgende Abbildung die verschiedenen Iterationen innerhalb jeder Phase. So ist erkennbar, dass in Richtung des Entwicklungsprozesses eine Aktualisierung innerhalb einer Phase ebenfalls eine Aktualisierung in den nachfolgenden Phasen verursacht. Wird z.B. das Tradingwissen innerhalb von TradeKnowledge durch neue Erkenntnisse oder Korrekturen aktualisiert, können bereits fertig modellierte Handelssysteme erneut modelliert, transformiert und deployed werden. Somit profitieren diese von den aktualisierten Tradingwissen.

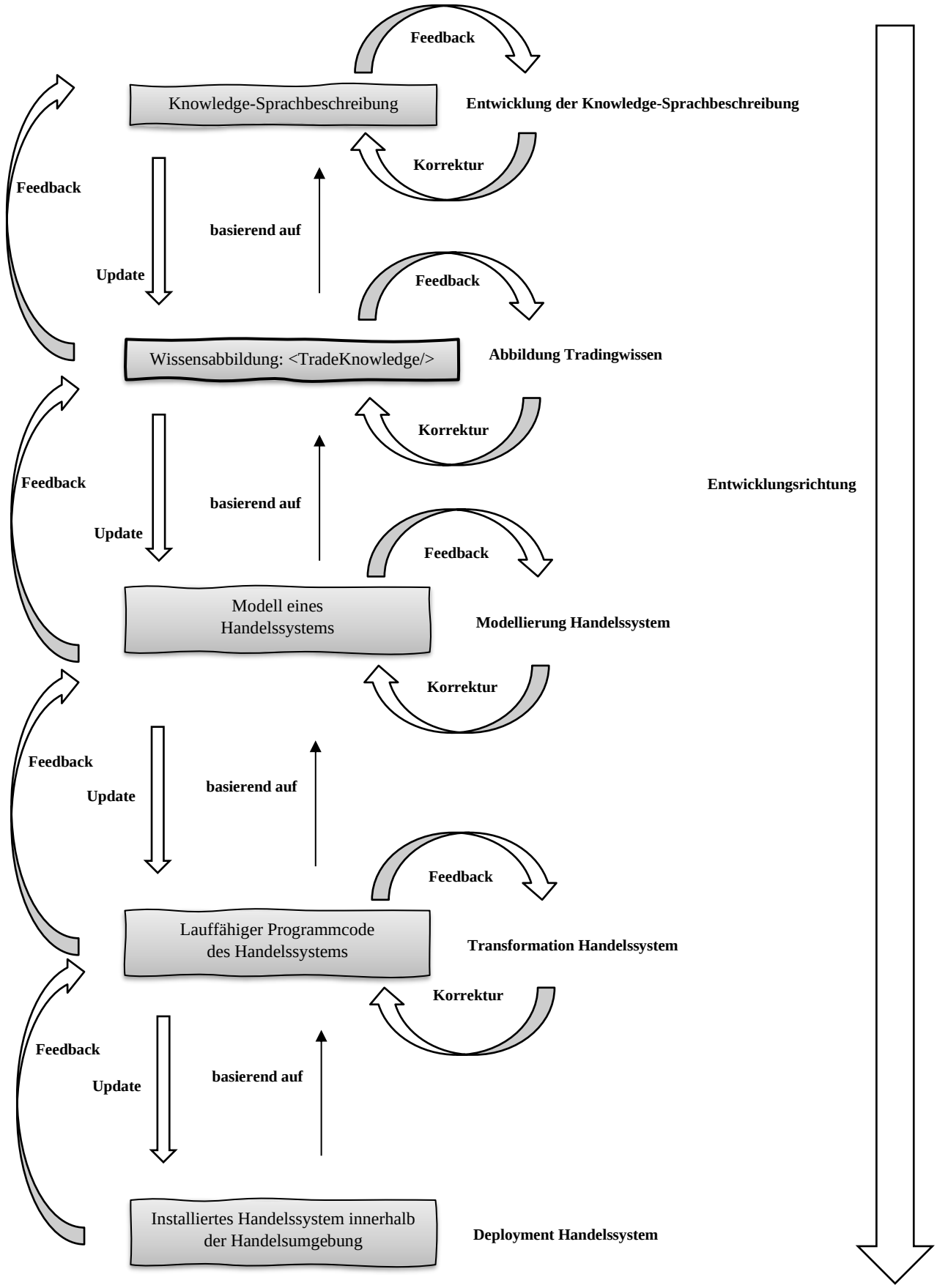


Abbildung 1: Iterationen innerhalb des gesamten des Entwicklungsprozesses

5 WISSENSABBILDUNG

Ergänzend zu der in der Arbeit im Kapitel 6.2. erwähnten Sprachbeschreibung zur Wissensbeschreibung werden in diesem Kapitel ausgewählte Aspekte erläutert.

Die Entitäten, so z.B. ein Handelssystem, werden durch Klassen beschrieben. So wird ein Handelssystem durch die Klasse „Tradingsystem“ beschrieben (siehe Arbeit, Kapitel 7.1.1).

Klassen enthalten Attribute, Operatoren (Funktionen) und Beziehungen:

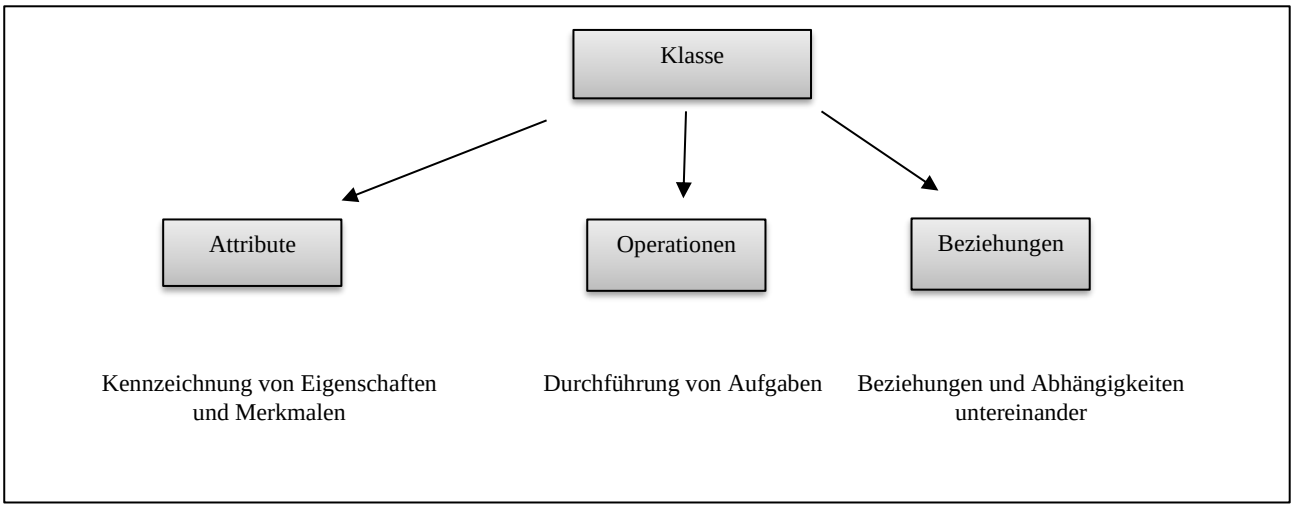


Abbildung 2: Ausgewählte Aspekte der Wissensabbildung

Zum Verständnis der im Folgenden genannten Begriffe sind Kenntnisse in der UML (https://de.wikipedia.org/wiki/Unified_Modeling_Language) notwendig.

5.1 ATTRIBUTE

Mit Attributen werden die Eigenschaften und Merkmale beschrieben.

Attribute steuern das Verhalten in der entsprechenden Modellierungsphase. Sie bestimmen somit den Aufbau des zu erstellenden Modells und beschreiben also den Anwendungszweck eines Modells näher.

Ein Beispiel aus Trade-Knowledge: Die Klasse TradingSystem enthält das Attribut „@TradeDirection“. Dieses Attribut kann die zwei Werte „LONG“ und „SHORT“ enthalten und bestimmt somit, dass innerhalb der Modellierung nur entsprechende Bestandteile für die gesetzte Handelsrichtung gültig sind.

Die Werte von Modellattributen sind entsprechend vor der Modellierung zu setzen: Besitzt dieses Attribut den Wert „LONG“, so kann das Modell eben nur Bestandteile für LONG-Ausführungen enthalten, z.B. ein LONG-Einsteigszenario. Dieses Modell kann dann keine SHORT-Szenarien aufweisen. Der aus dem Modell generierte Handelssystemcode enthält dann nur Programmcode zur Ausführung von LONG-Positionen.

Die in der Modellierung verwendeten Attribute dürfen jedoch nicht mit verwendeten Variablen (so z.B. die Anzahl Perioden für einen gleitenden Durchschnitt) innerhalb eines Handelssystems verwechselt werden. Variablen innerhalb eines Handelssystems werden durch die Klasse „Var“ beschrieben (siehe 6.1).

5.2 PROPERTIES

Bei Properties handelt es sich um nicht änderbare Werten, die nur innerhalb einer Klassendefinition gültig sind und somit eine Klassendefinition näher beschreiben können.

Ein Beispiel aus Trade-Knowledge: Die Klasse SellStopOrder für die Ausführung einer Stopp-Verkaufsorder wird mit dem Property OrderType = „stop“ (es handelt sich um eine Stopp-Order) und mit dem Property Typ = „sell“ (es handelt sich um eine Verkaufsorder) beschrieben (siehe Arbeit, Kapitel 7.1.3).

5.3 OPERATIONEN

Operationen steuern das Verhalten in der entsprechenden Modellierungsphase und bestimmen somit den Aufbau des zu erstellenden Modells.

Ein Beispiel aus Trade-Knowledge: Die Klasse ExitSzenario für die Abbildung eines Ausstieg-Szenarios verwendet die Methode „***recommendation***“, um während der Modellierungsphase eine Empfehlung zur Verwendung einer Stopp-Order auszugeben. Die Klasse „Indicator“ für die Abbildung eines Indikators verwendet die Methoden „***usage***“ und „***alternatives***“ um während der Modellierungsphase eine Beschreibung zur Verwendung und zu möglichen Alternativen und Ideen auszugeben.

Über Operationen werden ebenfalls Beschränkungen und Regeln für Attributwerte und Beziehungen (genannt **Restricions**) beschrieben, so muss z.B. die Beziehung (siehe Beispiel Assoziation 5.4) „lossLimitExitSzenario“ auf ein ExitSzenario verweisen, dessen Property „#Type“ den Wert „LossLimitExit“ ausweist.

Es gibt Operationen für Transformationen; z.B. für die Generierung von Programmcode eines Handelssystems aus dem Modell heraus.

Funktionen innerhalb eines Handelssystems, so z.B. die Berechnung eines gleitenden Durchschnittes, werden durch entsprechende Klassen (gen. Terme) beschrieben (siehe 6.2).

5.4 BEZIEHUNGEN

Mit einer Beziehung (Assoziation) zwischen Entitäten wird eine Abhängigkeit gekennzeichnet.

Mit der Multiplizität einer Beziehung wird angegeben, mit wie vielen (minimalen und maximalen) Objekten der gegenüberliegenden Klasse eine Entität in Beziehung stehen kann. Beziehungen können ebenso eine Verantwortung (Aggregation) oder einen Besitz (Komposition) darstellen.

Durch eine entsprechende Beschreibung kann das Verhaltens der Relation gesteuert werden. So kann z.B. die Multiplizität über bestimmte Werte von Modellattributen oder von definierten Zuständen abhängig gemacht werden.

Beispiel einer Assoziation: Das Entry für das Risiko- und Moneymanagement (RiskoAndMoneyManagement) verweist auf Ausstiegs-Szenarien, die einen Ausstieg bzgl. einer Verlustbegrenzung ausführen. Durch Angabe der Multiplizität 1..n muss es mindestens ein Ausstiegszenario (1..) geben und es können beliebig viele (..n) Ausstiegszenarien für Verlustbegrenzungen im Modell vorhanden sein.

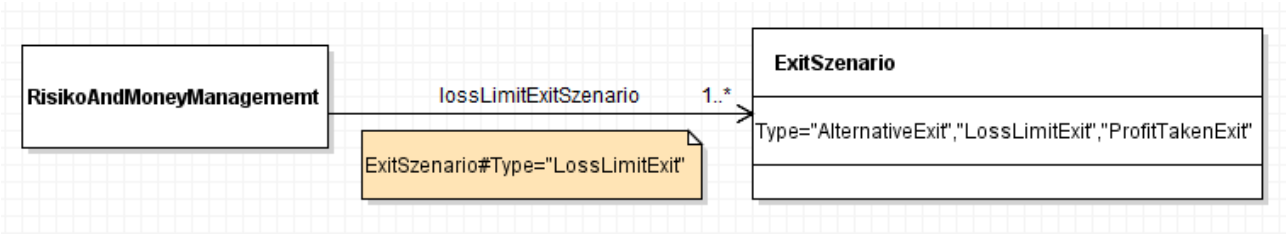


Abbildung 3: Beispiel Assoziation mit Multiplizität in der Wissensabbildung

Beispiele einer Komposition: Das Risko- und Moneymanagemt (RisikoAndMoneyManagement) besitzt eine Funktionalität zur Bestimmung von Positionsgrößen (DetermineStockAmount).

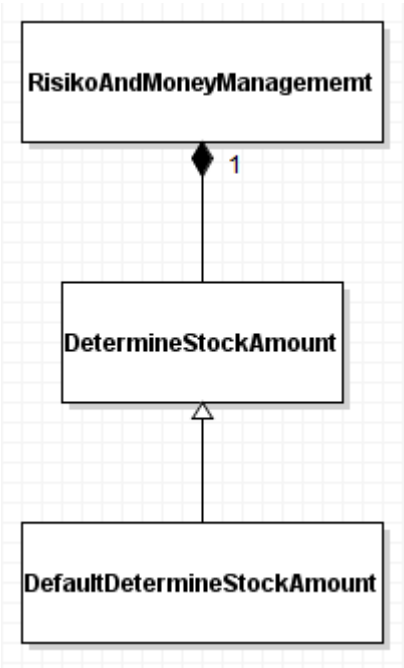


Abbildung 4: Beispiel Komposition

6 WEITERE BESTANDTEILE VON TRADEKNOWLEDGE

Dieses Kapitel stellt weitere Bestandteile von TradeKnowledge detailliert vor.

6.1 ABBILDUNG VON WERTEN

Werte von Preisen, Indikatoren, Konstanten u.a. werden als Ableitungen der Klasse „Var“ abgebildet. So wird z.B. der Close-Preis mit der Klasse „Close“ abgebildet (zweite Ableitung).

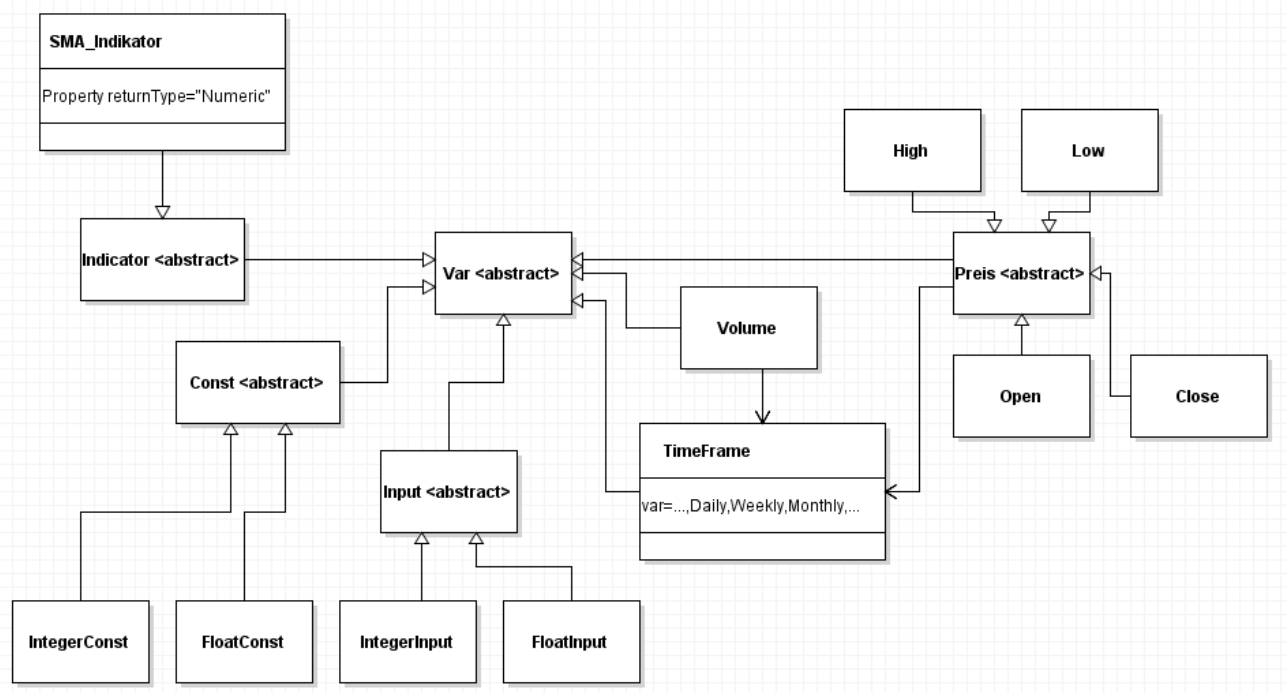


Abbildung 5: Abbildung von Werten als Ableitungen der Var-Klasse

Der Vars-Bestandteil befindet sind innerhalb einer Instanz von „Handelssystem“.

Es bündelt alle im Handelssystem verwendeten Werte (numerische, boolesche, u.a.).

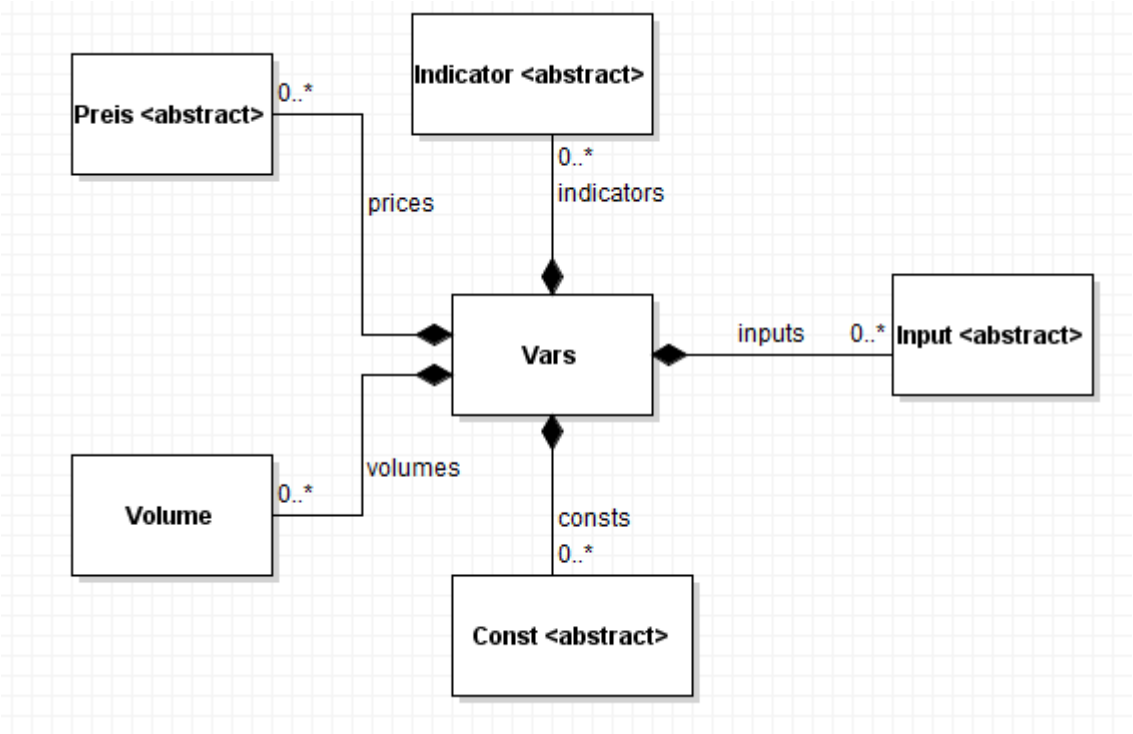


Abbildung 6: Speicherung von Werten innerhalb der Vars-Instanz

Beispiel: Ein Handelssystem besteht aus zwei SMA-Indikatoren (Close tagesbasis) mit den einstellbaren Perioden „shortPeriod“ und „longPeriod“.

Vars enthält dann u.a. die folgenden Instanzen:

- „Close“ mit der Bezeichnung „closePrice“ für den Schlusskurs
- zwei Eingabeparameter
  - IntegerInput „shortPeriod“
  - IntegerInput „longPeriod“
- Für die Indikatoren
  - SMA_Indicator „shortSMA“ mit einem Verweis auf IntegerInput „shortPeriod“ und „closePrice“
  - SMA_Indicator „longSMA“ mit einem Verweis auf IntegerInput „longPeriod“ und „closePrice“

6.2 Abbildung von Funktionen

Werden algorithmische Ausdrücke in verschiedenen Programmiersprachen formuliert, so haben diese meist eine unterschiedliche Syntax. Um Algorithmen innerhalb eines Modells bzw. innerhalb eines Indikators abbilden zu können, ist somit eine allgemeine Beschreibung notwendig.

In TradeKnowledge können mit der Klasse „Term“ einfache Algorithmen abgebildet werden. Ein modellierter Term wird durch den entsprechenden Transformer in die entsprechende Programmiersprache (so z.B. Equilla) abgebildet.

Ein Term kann wiederum andere Terme enthalten und sich auf Var-Elemente (Werte von Konstanten und Indikatoren) beziehen.

6.2.1 Terme für Werte

Mit der Klasse Term können einfache Algorithmen abgebildet werden. Es wird unterschieden zwischen Termen die einen numerischen Wert oder Boolean als Ergebnis liefern:

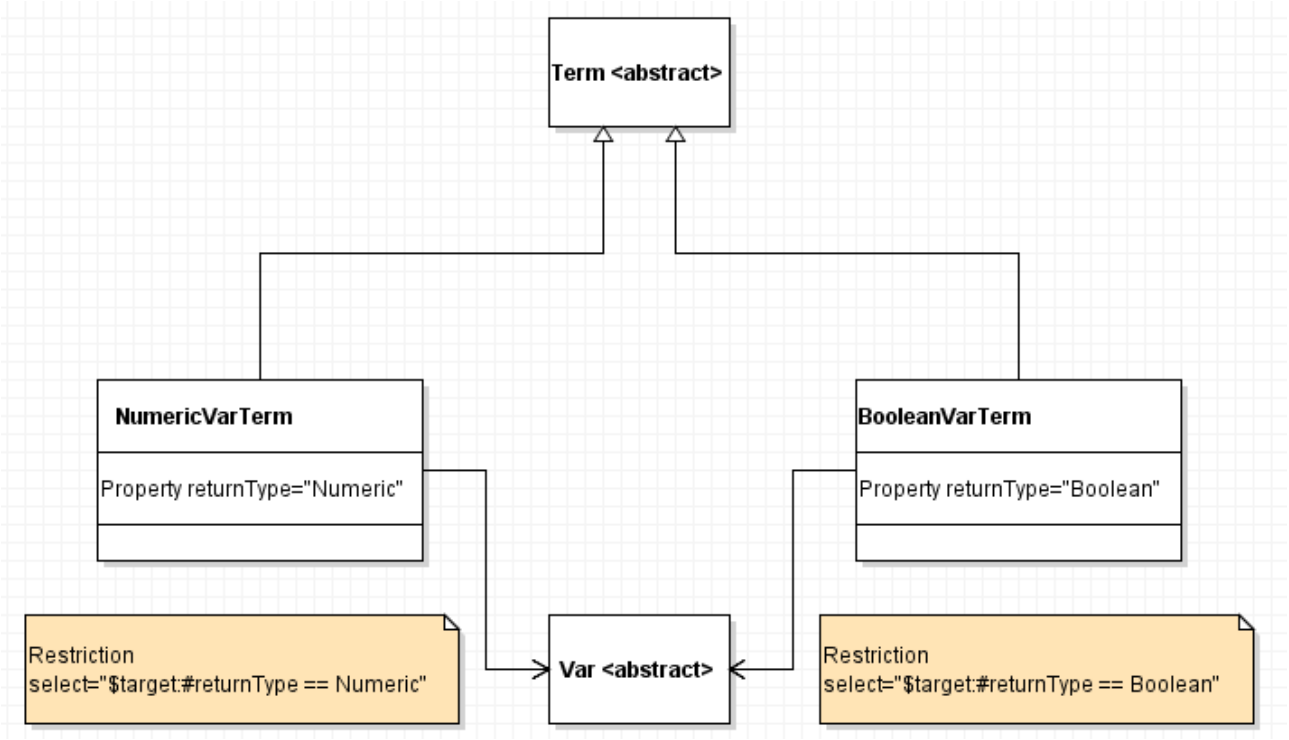


Abbildung 7: Abbildung von Termen für unterschiedliche Ergebnis-Typen

Über die interne Restriction kann ein BooleanVarTerm nur boolsche Werte und ein NumericVarTerm nur numerische Werte referenzieren.

6.2.2 Terme für Bedingungen

Mit vorgefertigten Termen für Bedingungen können Prüfungen abgebildet werden. So z.B. liefert „isNull“ einen Boolean-Wert, wenn der enthaltene Wert einen Null-Wert (NaN) enthält.

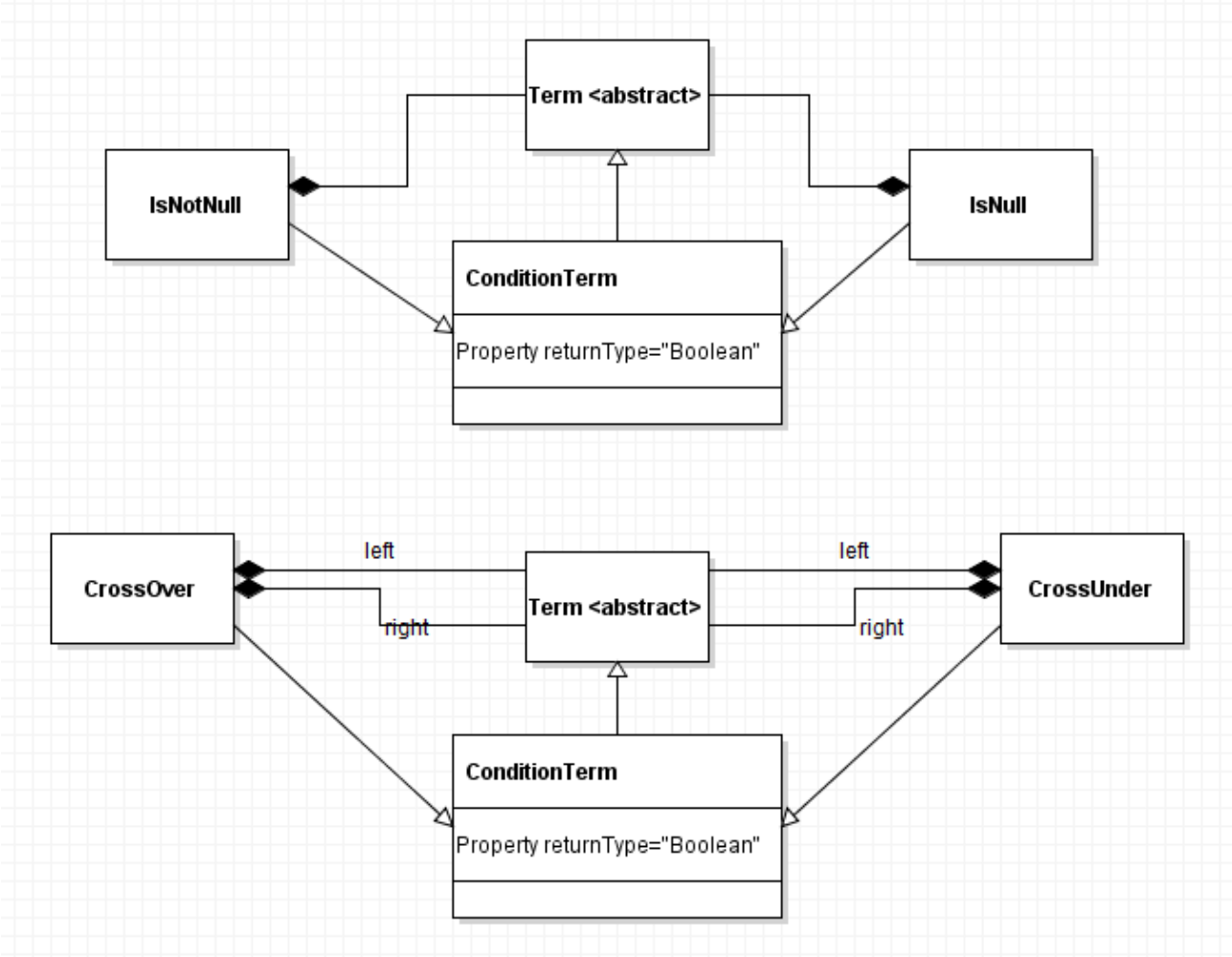


Abbildung 8: Terme für einfache Bedingungen

Mit CrossOver kann eine Kreuzung zweier numerische Terme erkannt werden, z.B. als Term um ein Handelssignal zur erzeugen, das auf einer Kreuzung zweier Indikatoren (z.B. zwei gleitende Durchschnitte) basiert.

6.2.3 Kombinierte Terme

Mit vorgefertigten Termen für einfache Operationen können die Addition, Subtraktion, das Logische Oder u.a. von zwei referenzierten Termen (leftTerm und rightTerm) abgebildet werden:

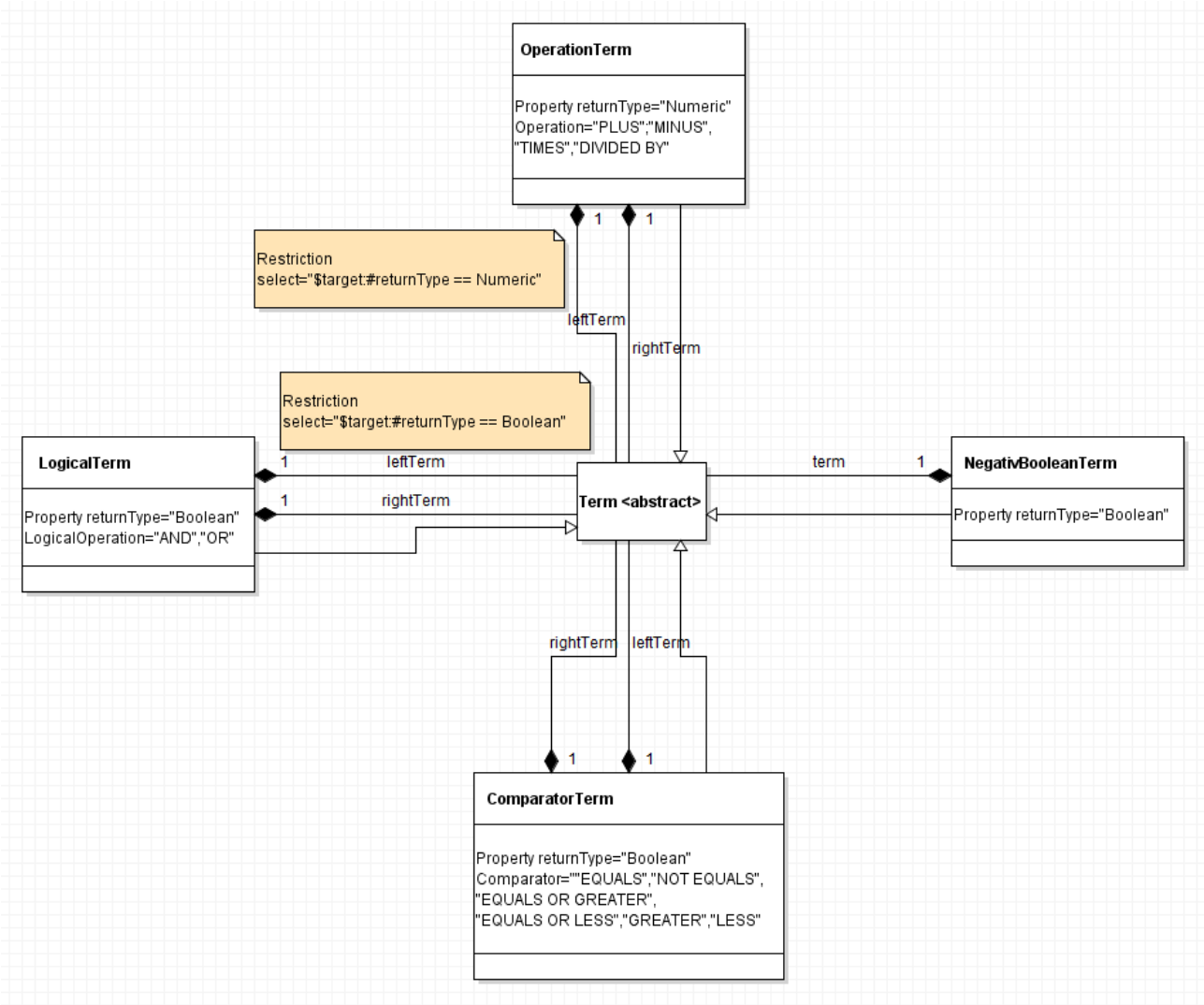


Abbildung 9: Abbildung von einfachen Operationen

Durch die Modellierung von Termen und deren Beziehungen zu anderen Modellelementen ist ein Überblick über dessen Verwendung gegeben. So können z.B. alle Input-Werte eines Handelssystems in einer Eingabe gesammelt und geordnet werden.

6.3 MELDUNGEN VON TRADEKNOWLEDGE

Ergänzend zu denen, im Kapitel 7.3. der Arbeit erwähnten Meldungen werden hier weitere mögliche Überprüfungen und Meldungen genannt.

Die Liste der Meldungen kann in weiteren Versionen von TradeKnowledge kontinuierlich ausgebaut und verbessert werden.

6.3.1 Meldungen innerhalb der Modellierungsphase

Zustand	Meldung
Verwendung von SHORT-Positionen	Es muss damit gerechnet werden, dass wegen regulatorischer Einschränkungen der Finanzaufsichtsbehörden Short-Trades ggf. nicht ausgeführt werden können. Der Kurs kann theoretisch ins Unendliche steigen kann, so dass ohne eine entsprechende Absicherung ein erhöhtes Risiko besteht.
Es gibt nur eine Exit-Szenario für die Verlustbegrenzung.	Es sollte noch eine weiteres Exit-Szenario verwendet werden. Mit nur einem Exit-Szenario für die Verlustbegrenzung erfolgt eine Schließung der Position zu seinem initialen Stopp-Preis, was stets zu einem Verlust führen würde.

Tabelle 1: Meldungen während der Modellierung

6.3.2 Meldungen zur Laufzeit

Ereignis	Hinweis/Warnung
Es soll eine Stopp-/Limit-Order für einen Einstieg in eine Position verwendet werden und der Preis innerhalb dieser Order ist zu weit vom aktuellen Preis entfernt.	Solch eine Order muss ggf. sehr lange an der Börse aktiv sein und das Kapital muss bei der Ausführung verfügbar sein.
Wegen eines Aktiensplits von 1:4 fällt der Kurs einer Aktie um 75%	Bei einem durchgeführten Aktiensplit sind alle historischen Daten und die Preise bestehender Orders entsprechend zu normieren. Erfolgte diese Normierung jedoch noch nicht, so besteht die Gefahr fälschlicherweise ein Signal für die Verlustbegrenzung zu generieren. TradeKnowledge kann dieses nicht verhindern; aber es werden bei Preissprüngen mit den Verhältnissen 1:n und n:1 entsprechende Warnungen zur Laufzeit ausgegeben. Durchgeführte Aktiensplits, aktive Orders und die historischen Daten sollten dann überprüft werden.
Eine bestehende Stopp-Order zur Verlustbegrenzung einer LONG-Position wird ausgeweitet, so dass dieser unterhalb des initialen Stopp-Preises liegt. Damit soll ein Ausstoppen der Position vermieden werden.	Es wird nur das initiale Risiko erhöht und damit die Strategie für das Risiko- und Money-Management nicht eingehalten.
Der Kurs einer Aktie ist gefallen und die entsprechende LONG-Position liegt bereits im Verlust. Durch den Nachkauf von Aktien soll so der Einstiegspreis verbilligt werden.	Die Verbilligung ist ein Trugschluss. Es muss auf die Einhaltung der Exitstrategie geachtet werden. Eine Aufstockung einer Position sollte nur mit einem Kaufsignal erfolgen.

Tabelle 2: Meldungen zur Laufzeit

7 WEITERFÜHRENDE IDEEN MIT TRADEKNOWLEDGE

Dieses Kapitel stellt weiterführende, noch nicht umgesetzte Ideen und sinnvolle Aspekte von TradeKnowledge vor.

7.1 PROFILE-MATCHING

Zusätzlich zu dem in der Arbeit (siehe Kapitel 7.2.) dargestellten Profilen von Tradern, können ebenfalls Profile von Handelssystemen, von Brokern und von Handelsumgebungen in TradeKnowledge abgebildet werden. In den Profilen können relevante Eigenschaften beschrieben werden, z.B. die beabsichtigten Intervalle eines Handelssystems, die unterstützten Orderarten eines Brokers oder die Latenzzeiten einer Handelsumgebungen.

Über entsprechende Matching-Methoden können für Trader geeignete Handelssysteme und für Handelssysteme passende Broker und Handelsumgebungen ermittelt werden.

So ist z.B. im Trader-Profil enthalten, dass der Trader berufstätig ist und den Handel auf Aktien des DAX und auf Wochenbasis vorzieht. Die entsprechende Matchfunktion liefert dann nur Modelle von Handelssystemen, die diese Eigenschaften in ihrem Handelssystem-Profil vermerkt haben. Ein Handelssystem, das z.B. eine Kauf-Limit-Order für einen Einstieg verwendet, benötigt einen Broker, der diese Kauf-Limit-Order unterstützt. In einem weiteren Beispiel benötigt ein Handelssystem für EURUSD-Handel im 15Min-Chart eine Handelsumgebung mit hoher Zuverlässigkeit, geringen Latenzzeiten zur Börse und einer ununterbrochenen Betriebsumgebung während der Handelszeiten (so z.B. eine Metatrader-Umgebung innerhalb einer gemieteten virtuellen Maschine).

Der Trader erhält die passenden Broker und Handelsumgebungen indirekt über die Matchfunktionen des Handelssystems.

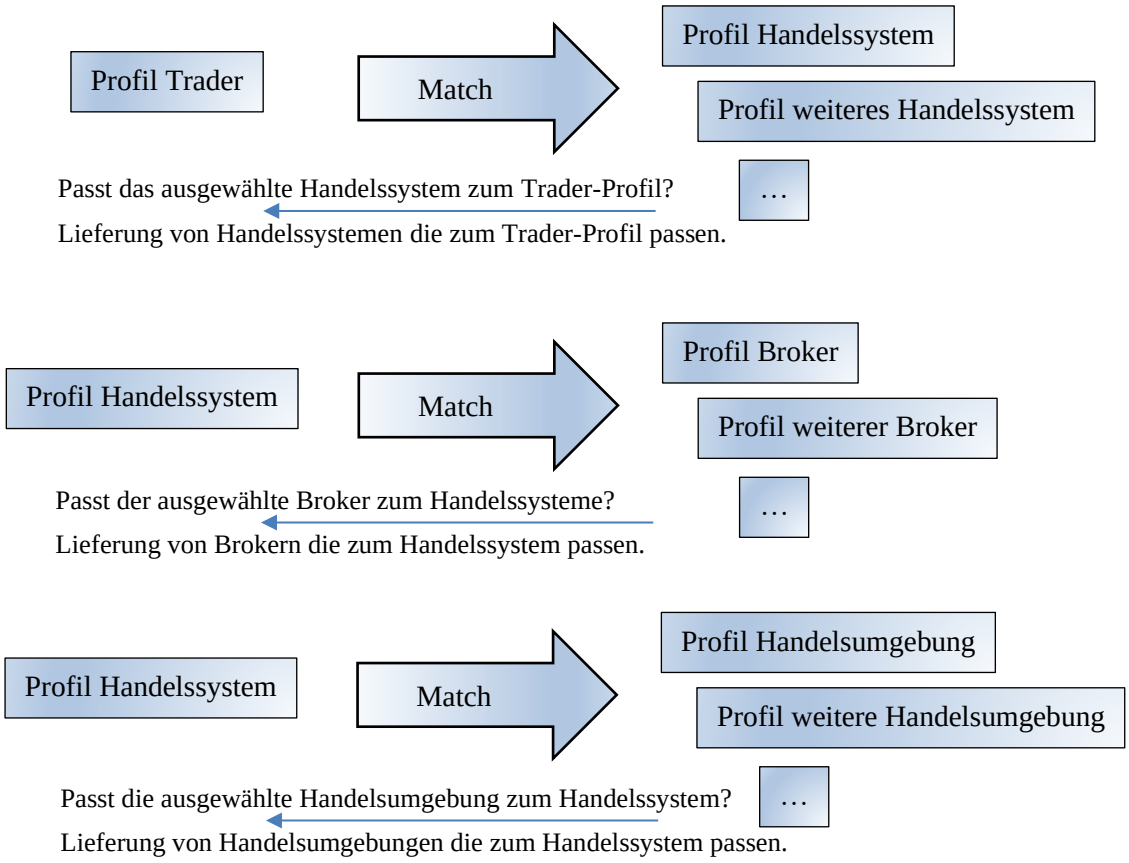


Abbildung 10: Profilematching

Voraussetzung für diese Vorgehensweise sind von den Brokern und Betreibern von Handelsumgebungen stets aktuell gehaltenen und bereitgestellten Profile. Ändern sich z.B. die Eigenschaften einer Handelsumgebung (z.B. wenn ein ununterbrochener Betrieb nicht gewährleistet werden kann), können ggf. die Handelssysteme aufgrund unpassender Eigenschaften nicht mehr eingesetzt werden.

Innerhalb einer Community können somit Suchfunktionen angeboten werden, so dass dann über die Eingabe von bestimmten Eigenschaften (Suchkriterien) passende Handelssysteme, Broker oder Handelssystemumgebungen gefunden werden können.

7.2 KONZEPTION UND IDEENFINDUNG

Großes Kopfzerbrechen bereitet es wohl, einen Überblick über gewonnene Ideen, Skizzen und Programmierexperimente von Handelssystemen (und auch weiteren Modellbestandteilen) zu behalten.

Um dem vorzubeugen, können mit TradeKnowledge Komponenten beschrieben werden, mit denen eine geordnete Sammlung und Ordnung von Ideen, Skizzen und (analog zu Programmierexperimenten) Verweisen auf bestehende (und auch halbfertige) Handelssystemmodelle erfolgen kann. Mit dem Verweis auf Ihre ursprüngliche Idee ist für modellierte Handelssysteme deren Fertigstellungs- und Reifegrad dokumentierbar.

Ideen/Skizzen können im Freitext beschrieben und mit Dokumenten versehen werden. Ebenso kann auf bereits existierende Ideen/Skizzen oder anderer Modelle von Handelssystemen verweisen werden.

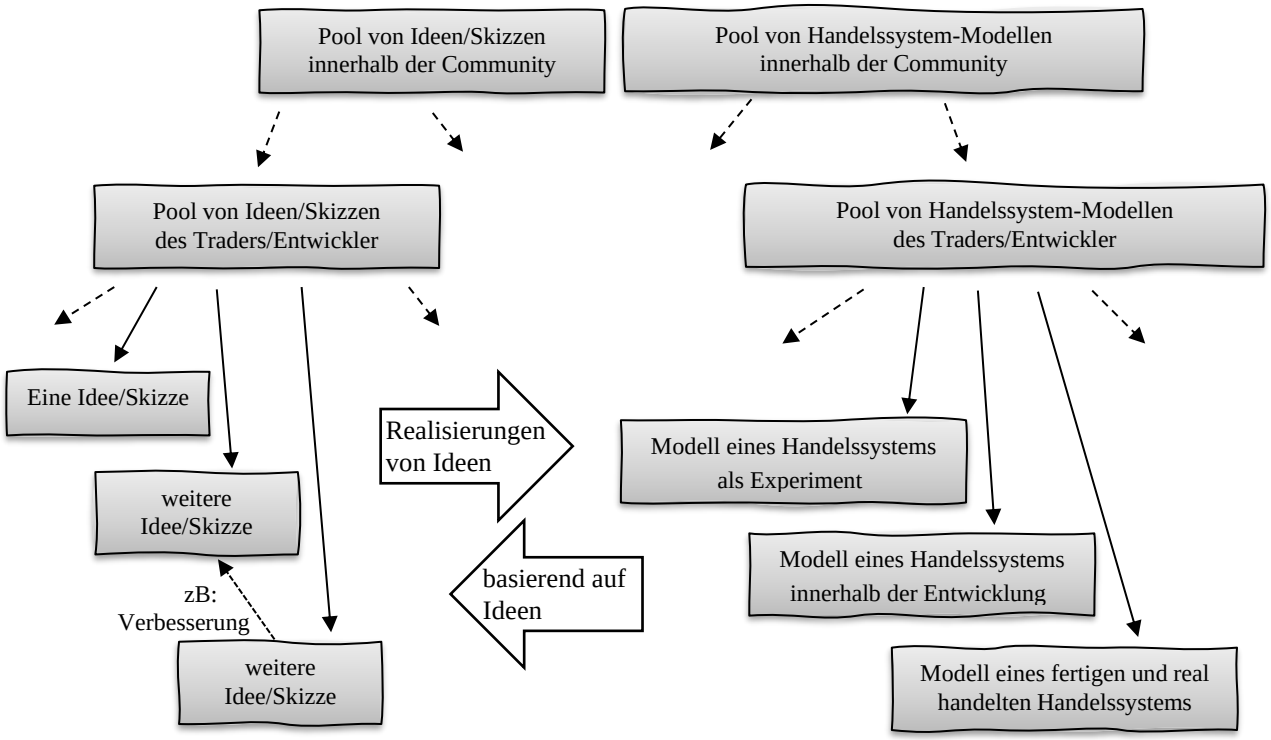


Abbildung 11: Konzeption und Ideenfindung

Über einen Pool von Ideen/Skizzen bleibt der Überblick gewährt. Werden Ideen oder Skizzen mit entsprechenden Profilen versehen (siehe 7.1), ist deren Selektion und ihr Wiederauffinden mit gewünschten Eigenschaften über entsprechende Matching-Methoden möglich. Ebenso ist über einen Pool von Handelssystemen ein Zugriff auf bereits modellierte Handelssysteme möglich. Anhand deren Profile ist ebenfalls eine gezielte Selektion möglich, so z.B. „Liefere mir eine Liste von Handelssystemen, die für einen Aktienhandel geeignet sind“.

Ein gemeinsamer und öffentlicher Ideen- und Handelssystempool innerhalb der Community ermöglicht einen gemeinsamen und übergreifenden Austausch von Ideen- und Handelssystemen. So können Entwickler und Trader von den Ideen und Handelssystemen anderer Entwickler und Trader profitierten, diese kommentieren und verbessern. Voraussetzung hierfür ist, dass der entsprechende Trader/Entwickler seine Ideen und Handelssysteme als öffentlich und entsprechend mit der Erlaubnis zur Kommentierung und/oder Erweiterbarkeit kennzeichnet.

7.3 MODELL-PROZESSIERUNG

Für die Ausführung eines Handelssystems innerhalb der Handelsumgebung sind die Transformation in einen Programmcode (z.B. ProBuilder für ProRealtime) für eine ausgewählte Zielumgebung (z.B. in ProRealtime) und das Deployment des erzeugten Programmcodes in der Zielumgebung notwendig. Dann ist die Durchführung eines Backtests für das Handelssystem in dieser Zielumgebung möglich.

Innerhalb der Entwicklungzeit eines Handelssystems ist es notwendig wiederholt Backtests durchzuführen, um die Performance des Handelssystems zu ermitteln. Dazu müssen die Transformation, das Deployment, und die Durchführung des Backtests innerhalb der Zielumgebung ebenfalls wiederholt durchgeführt werden.

Mit einer direkten Interpretation, der im Modell abgebildeten, Algorithmen, ist jedoch eine Simulation des Tradingverlaufes (Backtest) ohne Transformation und Deployment möglich. Ein Backtest kann damit direkt aus dem Modell heraus und unabhängig von einer Zielumgebung erfolgen. Auf diese Weise ist ein schnellerer Entwicklungsprozess möglich und weitere interessante Möglichkeiten ergeben sich (nachfolgend erläutert).

Die direkte Interpretation und Ausführung der im Modell abgebildeten Algorithmen (Terme) erfolgt durch eine entsprechende Software (genannt: Modellprozessor). Die enthaltenen funktionellen Bestandteile werden dabei interpretiert und ausgeführt, so z.B. die Interpretation und Berechnung eines Terms für die Kreuzung von zwei Werten (so für ein CrossOver zweier Durchschnitte). Die Simulation des Tradingverlaufes erfolgt dann mit bereitgestellten historischen Daten und der im Modell abgebildeten Struktur (Szenarien und das verwendete Risiko- und Moneymanagement).

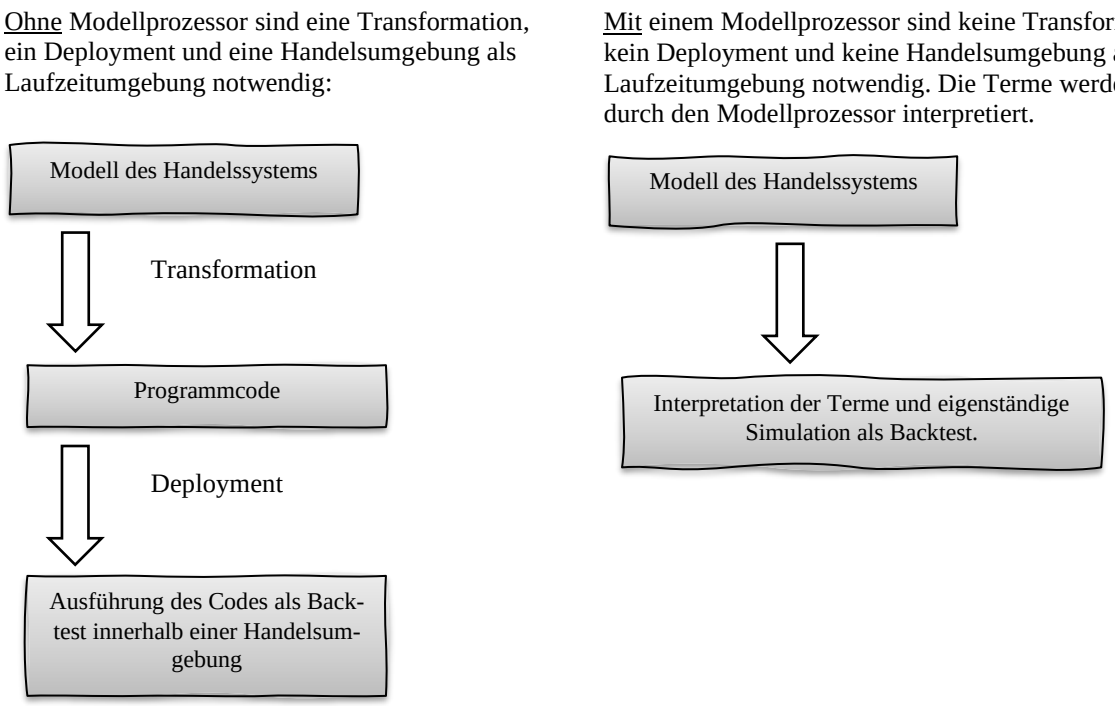


Abbildung 12: Ausführung Backtest innerhalb eines Modellprozessors

Dieser Ansatz setzt allerdings voraus, dass alle im Handelssystem verwendeten Algorithmen als Terme im Modell abgebildet werden oder der Modellprozessor diese Algorithmen auch ohne deren Abbildung ausführen kann. Üblicherweise stellt jede Programmiersprache vorgefertigte Algorithmen als Methoden zur Verfügung. So z.B. die Methode „Average“ in ProBuilder zur Berechnung eines gleitenden Durchschnittes. Für eine Anwendung des Modellprozessors müssten also entweder die Methode „Average“ als Term im Modell nachmodelliert werden oder der Modellprozessor hat bereits eine Methode zur Berechnung eines gleitenden Durchschnittes implementiert.

Basierend auf diesem Ansatz sind die Entwicklung und Einsatz weiteren Tools denkbar, wie z.B.

- Ausführung von Funktionstests von Modellbestandteilen (siehe 7.4),
- Optimierung von Parametern während der Durchführung von Backtests,
- Profiler (für die Laufzeitoptimierung von Algorithmen),
- Auffinden von Zukunftsblicken (siehe 7.7.2) und
- Code-Coverage (https://en.wikipedia.org/wiki/Code_coverage) für die Bestimmung der Testabdeckung von, im Modell abgebildeten Algorithmen. Somit kann ermittelt werden, für welche Algorithmen-Bestandteile noch keine Funktionstest vorhanden sind.

Betrachten wir Software für die Programmier- und Handelsumgebung (so z.B. MetaTrader oder AmiBroker) werden ja bereits durch die simulierte Ausführung von Orders auf Basis von historischen Börsendaten Backtests durchgeführt. In unserem Fall werden aber anstelle von Programmcode (so MQL für MetaTrader und AFL für AmiBroker) die im Modell abgebildeten, Algorithmen und ebenso die Orders auf Basis von historischen Börsendaten simuliert ausgeführt.

Leider sind Tools für die Funktionstests, Profiler und Code-Coverage nicht Bestandteile von z.B. Tradesignal, MetaTrader oder AmiBroker. Der hier vorgestellte modellorientierte Ansatz mit der Modellprozessierung ist somit eine große Bereicherung für die Handelssystementwicklung. Anstatt Tools für z.B. das Profiling, Backtest oder für die Optimierung von Parametern für jede Programmier- und Handelsumgebung zu implementieren, können diese mit Hilfe des Modell-Prozessors zentralisiert entwickelt und verwendet werden.

7.4 FUNKTIONSTESTS VON MODELLBESTANDTEILEN

Zusätzlich von Backtests zur Überprüfung der Funktionsweise und Performanz eines Handelssystems, ist eine separate Überprüfung von einzelnen Modellbestandteilen sinnvoll (z.B. die separate Überprüfung der Funktionsweise eines Einstiegsszenarios oder der Funktionsweise des Algorithmus zur Berechnung der Größe für eine einzugehende Position).

Der Funktionstest von Modellbestandteilen (genannt UnitTests *https://de.wikipedia.org/wiki/Modultest*), die im Ganzen die vollständige Funktionsweise des Handelssystems gewährleisten, ist Standard und „State of the Art" in der Softwareentwicklung.

Ein UnitTest testet also die Funktionsfähigkeit eines Modellbestandteiles; und zwar unabhängig (also isoliert) von anderen Modellbestandteilen. Mit dieser Isolation und Abgrenzung werden die Tests auf nur den entsprechende Modellbestandteil beschränkt und Abhängigkeiten zu anderen Modellbestandteilen ausgeschlossen. Bestehende Abhängigkeiten zu funktionellen Bestandteilen werden dabei simuliert und als Mock-Objekte (*https://de.wikipedia.org/wiki/Mock-Objekt*) abgebildet, so z.B. die Bereitstellung von historischen Börsendaten oder die Simulation von Orderanweisungen. Die Ergebnisse der Tests schließen demnach nur auf das isolierte und abgegrenzte Modellbestandteil, womit sich Fehlerquellen gezielt eingrenzen lassen.

Mit Sätzen von Testdaten für verschiedenste Test-Szenarien (wie z.B. enthaltene GAPs, starke Abwärtstrends oder auch rein zufällige Muster) lassen sich gezielt Testmethoden entwerfen, die sicherstellen, dass die im Modell enthaltenen Algorithmen, in den entsprechenden Mustern und Szenarien wie erwartet funktionieren. Testdaten können aus ausgewählten realen historischen Daten oder computergeneriert Mustern entsprechen (so z.B. ein idealer Auswärtstrend, Full-GAP, Partial-GAP, Candlestick patterns, schwaches oder starkes Rauschen).

Betrachten wir als Beispiel einen stark vereinfachten Trendfilter als einen zu testenden Modellbestandteil. Ein Aufwärtstrend wird wie folgt definiert: Der einfach gleitende Durchschnitt der letzten 200 Perioden (MA(200)) ist größer als der einfach gleitende Durchschnitt der letzten 200 Perioden vor 5 Perioden (MA(200)[5]).

Für diesen Trendfilter werden die zwei Testfälle definiert:

Testfall	**Testmethode**	**Satz mit Testdaten**	**Erwartetes Ergebnis**
testTrendUp Test auf einen bestehenden Aufwärtstrend	IsTrendUp = MA(200) > MA(200)[5]	MA(200) = 100 und MA(200)[5] = 95	IsTrendUp liefert TRUE für einen aufsteigenden Trend
testTrendNotUp Test auf einen nicht bestehenden Aufwärtstrend	IsTrendUp = MA(200) > MA(200)[5]	MA(200) = 100 und MA(200)[5] = 100	IsTrendUp liefert FALSE für einen nicht-aufsteigenden Trend

Tabelle 3: beispielhafte Testfälle für einen Trendfilter

In diesem Beispiel bekommt jeder Testfall für seine Ausführung unterschiedliche Testdaten. Für testTrendUp stellen die Daten einen Aufwärtstrend und für testTrendNotUp stellen die Daten keinen Aufwärtstrend dar. Für jeden ausgeführten Test wird ein entsprechendes Ergebnis erwartet (hier der Wert von IsTrendUp). Liefert eine Testmethode nach deren Ausführung nicht das erwartete Ergebnis, so gilt der Test als fehlgeschlagen.

Sind bei einer Testdurchführung Tests fehlgeschlagen, so kann der Fehler eingegrenzt und behoben werden. Mit entsprechenden Testfällen wird eine testgetriebene Entwicklung eines Modellbestandteiles möglich (https://de.wikipedia.org/wiki/Test_Driven_Development).

Mit einer automatisierten Ausführung der Tests durch entsprechende Continuous-Integration-Tools (https://en.wikipedia.org/wiki/CI/CD) nach jeder Anpassung, Re-Factoring (https://de.wikipedia.org/wiki/Refactoring) oder Fehlerfix am Modellbestandteil wird die Funktionsfähigkeit des Modells fortlaufend gewährleistet.

Außerdem besitzen Unittests dokumentativen Charakter, indem diese das Verhalten von Modellbestandteilen am besten wiedergeben. So können die Beschreibungen der vorhandenen und ausgeführten Tests mit in die generierte Dokumentation des Handelssystems einfließen, z.B. die Tests eines Einstiegsszenarios (siehe 7.5).

Verglichen mit der Methodik, nach der in der klassischen Handelssystementwicklung u.a. für den Test eines Trendfilters visuelle Sichtungen von Charts notwendig sind oder der Backtest von Handelssystemen unter Umständen ein anderes Ergebnis liefert (bei exakt den gleichen historischen Daten!), können mit dieser Methodik gezielt beliebig viele ausgewählte und unterschiedlichste Szenarien permanent wiederholt und automatisiert getestet werden. Dies unterstützt die Einhaltung und Erhöhung von Qualität, Zuverlässigkeit und Vertrauenswürdigkeit der Modellbestandteile (und somit des Handelssystems). Durch Anwendung von Metriken der Softwarequalität (so z.B. den Grad der Testabdeckung) werden diese messbar gemacht.

In TradeKnowledge werden die Testfälle innerhalb eines Modellbestandteils (so z.B. modellierte Szenarien) modelliert. So enthält der modellierte Trendfilter die zwei Testmethoden als zusätzliche Terme. Zusammen mit den Informationen zu den erwarteten Ergebnissen und der zu verwendeten Testdaten werden somit die Testfälle im Modell abgebildet.

Für die Ausführung von Tests ist ein entsprechender Test-Runner als eine zusätzliche Software erforderlich (wie z.B. https://de.wikipedia.org/wiki/Robot_Framework). Als Erweiterung zum Modellprozessor (siehe 7.3) ruft der Test-Runner in den Modellbestanteilen die entsprechenden modellierten Testmethoden auf, übergibt dabei die dafür erforderlichen Testdaten (z.B. ein Testdatenszenario für einen Aufwärtstrend), sorgt für die Kommunikation mit simulierten Mock-Objekten (z.B. die Simulation von Orders), überprüft das Ergebnis mit dem erwarteten Ergebnis, protokoliert und wertet die Testergebnisse aus.

7.5 EINHEITLICHE DOKUMENTATION

Aufgrund der Beschreibung von Struktur und Aufbau eines Handelssystems innerhalb eines Modells entsprechend der Modellbeschreibung ist die automatisierte Generierung einer Dokumentation mit einheitlichen Formaten möglich. Mit solch einer Dokumentation steht das Modell als lesbares Format zur Verfügung und trägt zum Verständnis des Handelssystems und seiner Bestandteile bei.

Mit einem Verweis auf die Idee (siehe 7.2) können dessen Informationen in der Dokumentation hinzugefügt werden. Mit bestehenden Testfällen können Testfallbeschreibungen (siehe 7.4) ebenfalls der Dokumentation hinzugefügt werden.

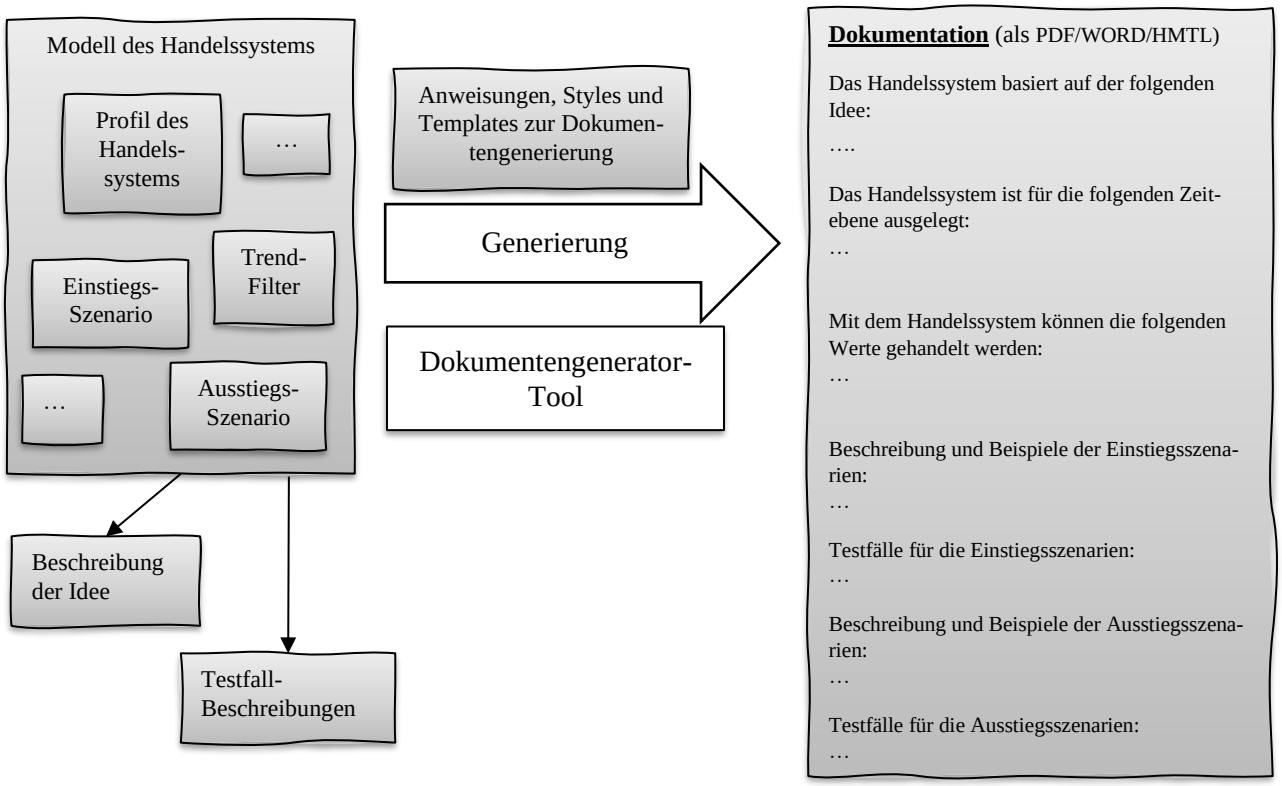


Abbildung 13: Generierung von Dokumentation aus einem Modell heraus

Aus den Modellbestandteilen werden mithilfe einer Software (Dokumentengenerator) entsprechende Dokumente erzeugt. Der Dokumentengenerator verwendet dazu entsprechende Anweisungen, Styles und Templates.

So kann innerhalb der Community eine einheitliche und z.B. Browser-basierte Dokumentation der öffentlichen Handelssysteme und ihrer Bestandteile angeboten werden.

7.6 Tools zur Unterstützung des praktischen Tradings

Bei der praktischen Anwendung eines Handelssystems sind viele Dinge zu beachten. So u.a.:

- ggf. manuell Orders beim Broker eingeben,
- ggf. vergessene oder verpasste Orders nachtragen oder fehlerhaft aufgegebene Orders korrigieren,
- einen Vergleich der SOLL-Positionen und real gehandelte Positionen durchführen,
- das Konto und das Risiko- und Moneymanagement einhalten,
- den Überblick über die gesamten Positionen im Portfolio behalten,
- die Performanz beobachten,
- ein Trading-Tagebuch führen

und vieles mehr.

Begleitende Tools können diese Aufgaben unterstützen oder vollständig übernehmen. Das Konzept von TradeKnowledge liefert eine Basis zur Bereitstellung solcher Tools. Über die im Modell eines Handelssystems angegebenen Informationen (Algorithmen, das verwendete Risiko- und Moneymanagement, Profile und weiterer Eigenschaften) ist die, auf dem Anwendungsfall gezielte, Generierung und Bereitstellung von Tools möglich. Diese würden dann entsprechend dem Handelssystem, der Umgebung und dem Trader angepasst sein.

7.7 ERKENNUNG VON ZUKUNFTSBLICKEN

In diesem Kapitel werden zwei Methoden zur Erkennung und Vermeidung von Zukunftsblicken vorgestellt (siehe Kapitel 7.5.5 in der Arbeit).

7.7.1 Überprüfung und Zertifizierung von Termen

Hierbei erfolgt eine explizite Überprüfung oder Einstufung von Termen bzgl. des Zukunftsblickes. Auf dieser Basis erfolgt eine entsprechende Modellvalidierung.

Jeder im Modell verwendete Term (siehe 6.2) besitzt dazu zusätzlich das Attribut „@forward-looking“. Es kann die folgenden Werte enthalten:

- **„none-prediction“**:
  Der Term wurde noch nicht bzgl. eines Zukunftsblickes überprüft. Es kann daher noch keine Aussage getroffen werden. Solch ein Term kann innerhalb des Modells nicht verwendet werden und verursacht somit ein ungültiges Modell.
- **„audited“**:
  Der Term wurde positiv auf einen nicht vorrausschauenden Blick überprüft und erhält somit ein Zertifikat.
- **„trust“**:
  Dem Term wird ohne eine vorherige Überprüfung vertraut, dass dieser keinen vorrausschauenden Blick enthält.
- **„with intent“**:
  Der Term enthält beabsichtigt einen vorrausschauenden Blick, z.B. die Berechnungsformel für den ZigZag-Indikator.

Wird ein Term modifiziert oder wird ein „trust“ zurückgezogen bedarf es einer erneuten Überprüfung oder Einstufung (der Wert wird auf „none-prediction“ zurückgesetzt).

Die Klasse **Tradingsystem** enthält zusätzlich das Attribut „@forward-looking-allowed“. Mit dem Wert „allowed“ wird explizit angegeben, ob innerhalb des Handelssystems ein Zukunftsblick erlaubt ist. Das kann der Fall sein, wenn z.B. der vorrausschauende Indikator nur zu Analysezwecken verwendet wird, oder man sich sicher ist, nur „sichere“ Indikatorwerte aus der Vergangenheit für die Signalgenerierung zu verwenden.

Per Default ist „forward-looking-allowed“ auf „none“ gesetzt. Es dürfen also keine vorrausschauende Terme verwendet werden.

Für TradeKnowledge ist ein Modell nicht gültig, wenn:

- Mindestens ein Term im Attribut „forward-looking“ den Wert „none-prediction“ aufweist.
- Mindestens ein Term im Attribut „forward-looking“ den Wert „with intent“ aufweist und das Attribut „forward-looking-allowed“ den Wert „none“ enthält.

Wenn in einem gültigen Modell mindestens ein Term im Attribut „@forward-looking“ den Wert „trust“ oder „with intent“ aufweist, wird eine entsprechende Warnung (mit Verweis auf die entsprechenden Terme) ausgegeben. In diesem Fall sollte die nachfolgend beschriebene „2-Phasen Modellprozessierung“ für eine weitere Überprüfung verwendet werden.

7.7.2 2-Phasen Modellprozessierung

Eine weitere Vorgehensweise zur Aufdeckung von Algorithmen, die einen Zukunftsblick enthalten, verwendet die im Kapitel 7.3 beschriebene Modellprozessierung zur Ausführung von Backtests auf Basis eines Modells.

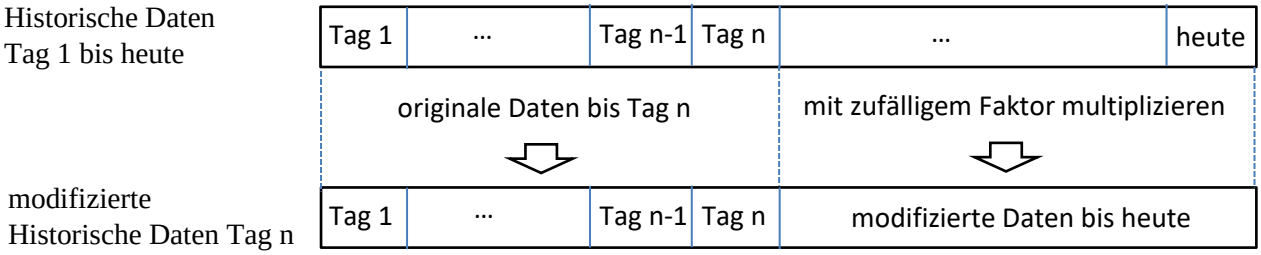
Dazu erfolgt die Ausführung von zwei unterschiedlichen Backtestläufen mit dem Modell-Prozessor auf das gleiche Modell. Liefern die beiden Backtestläufe das gleiche Ergebnis, so kann von einem zukunftsfreien Blick ausgegangen werden.

In einem ersten Backtestlauf werden die originalen historischen Daten verwendet.

In einem zweiten Backtestlauf wird für jeden Datenpunkt (also für alle simulierten Zeitpunkte) ein separater Datensatz erstellt und verwendet. Ein einzelner Datensatz enthält:

- bis einschließlich zum Zeitpunkt des Datenpunktes die identischen Daten wie aus den historischen Daten.
- nach dem Zeitpunkt des Datenpunktes (also, die für diesen Zeitpunkt zukünftigen Daten) modifizierte Werte aus den historischen Daten. Die Modifizierung dieser Datenpunkte erfolgt durch Multiplikation der OHCL-Werte mit einem zufällig gewählten Faktor (>0 bis <1).

Die folgende Abbildung zeigt das beispielhaft an einem historischen Datensatz (Tages-Intervall) mit Beginn Tag 1 bis heute. Der modifizierte Datensatz für den Tag n enthält modifizierte historische Daten ab dem Tag n+1:



In diesem Backtestlauf werden also für jeden Zeitpunkt entsprechend zukünftig modifizierte historische Daten verwendet.

Gibt es nun unterschiedliche Ergebnisse zwischen dem ersten und zweiten Backtestlauf, so kann davon ausgegangen werden, dass auf die modifizierten (also zum Zeitpunkt eines Datensatzes in der Zukunft liegenden Werte) zugegriffen wurde und somit ein Zukunftsblick vorliegt.

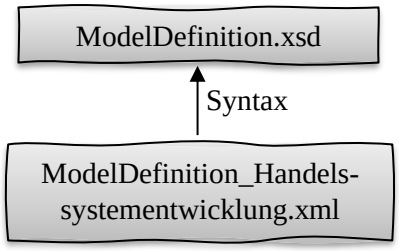
8 PROTOTYPISCHE UMSETZUNG

Zur Durchführung von Untersuchungen und für Überprüfungen auf die praktische Umsetzbar- und Anwendbarkeit des Ansatzes wurde eine prototypische Implementierung erstellt.

Es werden die folgenden Fachbegriffe erwähnt:

- XML: https://de.wikipedia.org/wiki/Extensible_Markup_Language
- XML-Schema: https://de.wikipedia.org/wiki/XML_Schema
- Java (Programmiersprache): https://de.wikipedia.org/wiki/Java_(Programmiersprache)

Die folgende Tabelle zeigt eine Übersicht über die verwendeten Tools und Technologien:

Phase	**Umsetzung**	**Tool zur Bearbeitung der Phase**
Knowledge-Sprachbeschreibung	Darstellung als XML-Schemadatei „ModelDefinition.xsd“ In der XSD-Datei werden XML-Elemente definiert, mit denen eine Wissensabbildung beschrieben werden kann.	<u>Entwicklung der Knowledge-Sprachbeschreibung:</u> XML-Editor zur Bearbeitung der „ModelDefinition.xsd“
Wissensabbildung: <TradeKnowledge/>	Darstellung als XML-Datei mit „ModelDefinition.xsd“ als Syntaxdefinition. „ModelDefinition_Handelssystementwicklung.xml“  Das Tradingwissen wird über XML-Elemente abgebildet. Die Syntaxüberprüfung erfolgt über die Verknüpfung zur XSD-Datei.	<u>Abbildung von Tradingwissen:</u> XML-Editor zur Bearbeitung der „ModelDefinition_Handelssystementwicklung.xml“.

Modell eines Handelssystems	Das Modell eines Handelssystems wird als XML-Datei (hier: „Modell.xml“ als Beispiel eines von vielen möglichen Modellen) mit „Model-Instance.xsd“ als Syntaxdefinition abgebildet. ModelDefinition_Handelssystementwicklung.xml Modellierung Model-Instance.xsd Syntax Modell.xml	<u>Modellierung:</u> Modellierungs-Tools als Java-Anwendung. Über eine grafische Benutzeroberfläche erfolgt die interaktive Zusammenstellung des Modells. ModelDefinition.xsd Interpretation Modellierungs-Tool Das Modellierungs-Tool interpretiert dabei die Sprachbeschreibung „ModelDefinition.xsd“, führt Validierungen aus, zeigt entsprechende Hinweise und Meldungen an und exportiert das Modell als XML-Datei. XML-Dateien von vorhandenen Modellen können importiert und damit weiterhin modelliert werden.
Programmcode des Handelssystems in z.B. Equilla für TradeSignal	Die Transformations-Anweisungen und Templates sind erweiternde Bestandteile der „Model-Definition_Handelssystementwicklung.xml“. Modell.xml Transformations-Anweisungen und -Templates verwendet Transformation Codefragment z.B. „Handelssystem.equ“	Mit dem Transformator-Tool als Java-Anwendung (als erweiternder Bestandteil des Modellierungs-Tools) erfolgt die Erstellung von Codefragmenten für die gewünschte Zielumgebung aus dem Modell heraus. Transformator-Tool Dabei verwendet das Transformator-Tool entsprechende Transformations-Anweisungen und Templates entsprechend der Zielumgebung.

Tabelle 4: Verwendete Tools und Technologien für die prototypische Umsetzung

Die folgende Abbildung zeigt beispielhaft die Anwendung des Modellierungstools bei der Modellierung, des im folgenden Kapitel 9 gezeigten, vollständigen Beispiels eines Handelssystems:

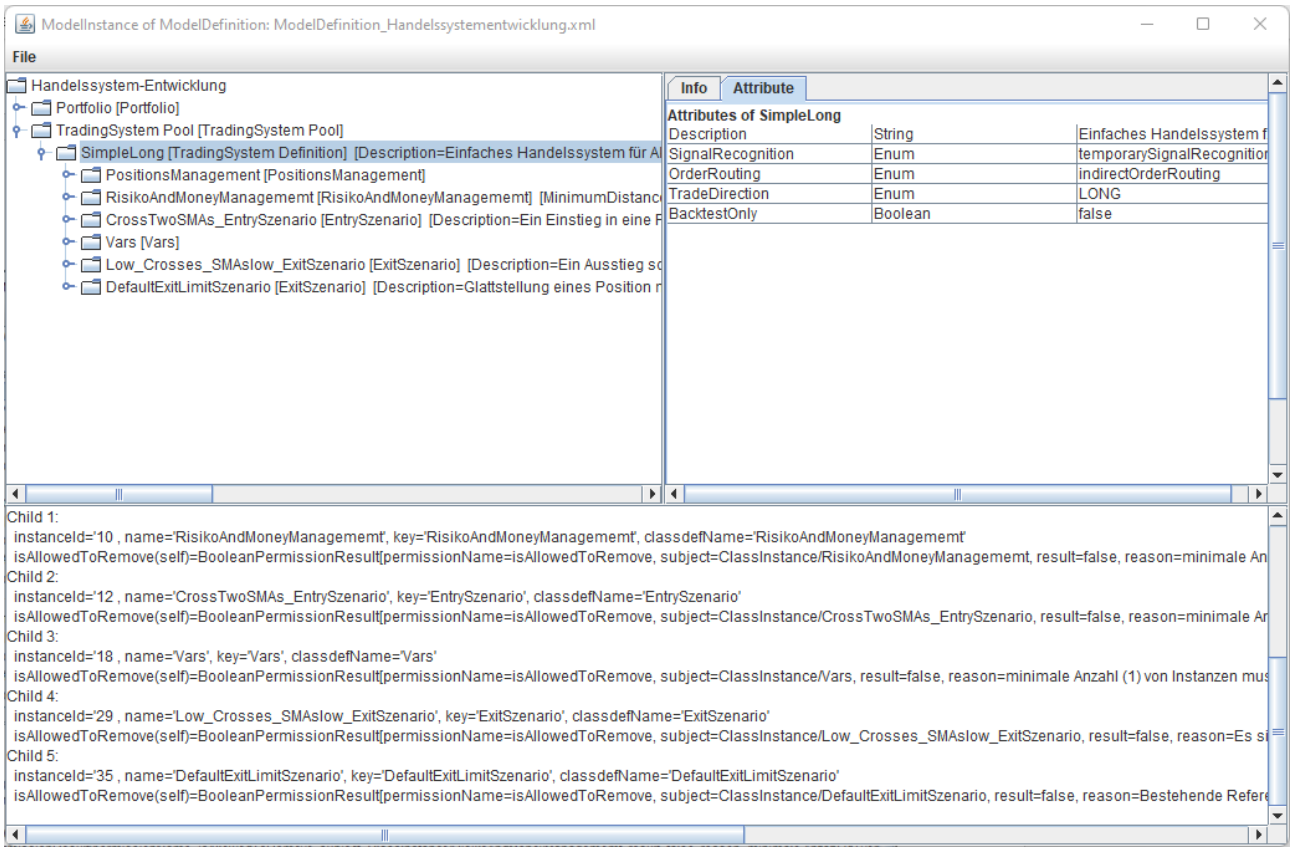


Abbildung 14: Modellierungstool als Java-Anwendung

Der linke obere Bereich zeigt in einer Baumstruktur die modellierten Handelssysteme. Dabei entspricht ein Blatt einer Instanz bzw. Entität einer Klassedefinition. So wird mit „SimpleLong (TradingSystem Definition)“ gezeigt, dass es sich um eine Instanz der Klasse TradingSystem mit dem Namen SimpleLong handelt. Die in dieser Instanz enthaltenen Instanzen (Kompositionen, siehe 5.4) werden als untergeordnete Blätter (z.B. Vars) angezeigt.

Der rechte obere Bereich enthält die zwei Reiter „Info“ und „Attribute“. Unter „Info“ werden die Gültigkeit und Regelverletzungen der modellierten Instanz gegenüber TradeKnowledge angezeigt. Unter „Attribute“ werden die Werte der Instanz angezeigt und sie können mit neuen Werten versehen werden.

Der untere Bereich enthält Debug- und Infomeldungen des Programms der prototypischen Implementierung.

Die Modellierung erfolgt über entsprechende Kontext-Menüeinträge innerhalb der Baumstruktur. Entsprechend der Beschreibung in TradeKnowledge, können hier neue Instanzen angelegt, geändert und gelöscht werden. Ebenso werden über entsprechende Kontext-Menüeinträge Beziehungen zu anderen Instanzen erstellt. Diese Aggregationen (siehe 5.4) sind dann innerhalb der Baumstruktur sichtbar.

Mit jeder Änderung des Modells erfolgt eine erneute Überprüfung der Gültigkeit gegenüber TradeKnowledge. Der Reiter „Info“ enthält den jeweils aktuellen Status dieser Überprüfung.

9 VOLLSTÄNDIGES BEISPIEL

Im Folgenden wird beispielhaft ein einfaches Handelssystem vollständig modelliert, in ProBuilder-Programmcode (für ProRealtime, https://www.prorealtime.com/de/) transferiert und in ProRealtime deployed.

Damit soll der Entwicklungsprozess eines Handelssystems mit TradeKnowledge verdeutlicht werden. Die Schritte innerhalb der Modellierung werden vereinfacht dargestellt.

Das Beispiel geht von den aktuellen Ständen der Knowledge-Sprachbeschreibung (1. Phase), der TradeKnowledge (2. Phase) und der prototypischen Umsetzung (siehe Kapitel 8) aus.

Das Beispiel soll auf Wochenbasis und auf, in Deutschland gehandelte, Aktien angewendet werden können. Der Anwender des Handelssystems ist ein voll Berufstätiger, der ggf. nur nach Feierabend oder am Wochenende die Gelegenheit hat, sich mit dem Handel an der Börse und sich mit seinem Aktienportfolio zu beschäftigen. Dieser Anwender hat einen kostenlosen Account für die Börsensoftware und Trading-Plattform ProRealTime (https://www.prorealtime.com/de/) und kennt die Möglichkeit, mit der in dieser Plattform enthaltenden Programmiersprache, ProBuilder (https://www.prorealtime.com/de/pdf/probuilder.pdf) Handelsstrategien zu testen.

Das beispielhafte Handelssystem soll auf den folgenden Ideen basieren:

1. Dieses Handelssystem verwendet zwei einfache gleitende Durchschnitte (SMA) mit unterschiedlichen Perioden:
    a. „slowSMA“: SMA mit der Periode 200
    b. „fastSMA“: SMA mit der Periode 50
2. Ein Einstieg in eine Position soll erfolgen, wenn der fastSMA den slowSMA von unten nach oben kreuzt.
3. Ein Ausstieg aus einer Position soll erfolgen, wenn der Schlusskurs der Woche den slowSMA durchbrochen hat.

Der Anwender hat ein Kapital von 20000 EUR für den Handel zur freien Verfügung; hat sich aber noch nie mit den Grundlagen des Risiko- und Money-Managements befasst.

9.1 MODELLIERUNG DES HANDELSSYSTEM (3. PHASE)

Der Trader kann sich nun auf das aktuell in TradeKnowledge gespeicherte Wissen (aus der 2. Phase) verlassen und folgt den gegebenen Meldungen und Hinweisen während der Modellierung.

Das Modellierungstool (siehe Kapitel 8) wird gestartet und der Handelssystementwickler (Modellierer genannt) beginnt mit der Modellierung des Handelssystems.

9.1.1 Instanz TradingSystem

Der erste Schritt ist stets die Erstellung einer Instanz der Klasse **TradingSystem** (siehe Arbeit, Kapitel 7.1.1). Der Modellierer setzt für diese Instanz die folgenden Attributwerte:

Attribut	**Gesetzter Wert**	**Kommentar**
@name	SimpleLong	Kurzer eindeutiger und aussagekräftiger Name des Handelssystems
@Description	Einfaches Handelssystem für Aktien, basierend auf zwei gleitende Durchschnitte.	Aussagekräftige und mehrzeilige Beschreibung des Handelssystems
@TradeDirection	LONG	Das System handelt nur LONG-Positionen
@SignalRecognition	temporary-SignalRecognition	Das Handelssystem wird nur zum Zwecke einer Signalermittlung ausgeführt. In unserem Fall geht der Trader am Wochenende explizit auf die Internetseite von ProRealtime und öffnet seinen Chart mit dem generierten Handelssystem-Programmcode. Hierbei werden dann vom Programmode neue Signale generiert.
@OrderRouting	indirectOrderRouting	Der Trader gibt bei Signalen die entsprechende Order in die Ordermaske seines Brokers (z.B. über die Internetseite von Flatex) ein. Die Orderabgabe an den Broker erfolgt also manuell und somit indirekt.
@onlyBacktest	False	Das Handelssystem soll nicht nur Backtests ausführen, sondern auch Signale für den realen Handel generieren.

Ein Abgleich mit dem Traderprofil eines Berufstätigen („Emploed“, siehe Arbeit, Kapitel 7.2.1.) ergab, dass die Werte für TradingSystem sinnvoll gesetzt wurden.

Das Modellierungstool zeigt nun die neue TradingSystem-Instanz mit dem Namen „SimpleLong“ und entsprechende Meldungen auf der rechten Seite (in rot):

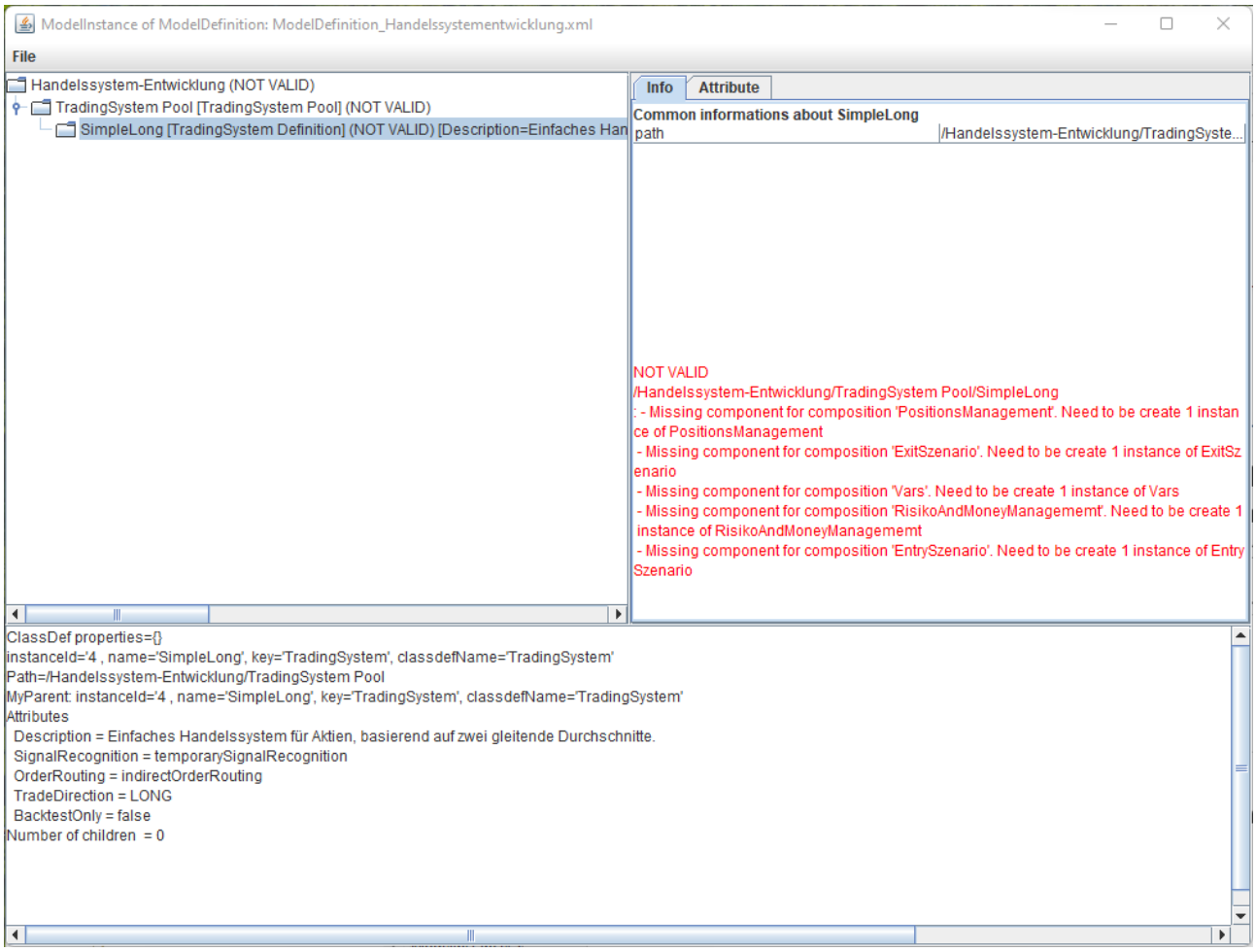


Abbildung 15: Modellierungstool: Instanz des Handelssystems

NOT VALID bedeutet, dass das Modell zu diesem Zeitpunkt noch nicht gültig ist und damit entsprechend der Beschreibung in TradeKnowledge noch keinen konsistenten Zustand aufweist. Das Modell ist also noch nicht anwendbar und kann noch nicht in Programmcode transformiert werden.

Die hier gezeigten Meldungen weisen auf noch fehlende und erwartete Instanzen hin. Diese werden in der weiteren Modellierung erzeugt.

9.1.2 Instanz PositionsManagement

Der Modellierer erstellt nun die erforderliche Instanz von PositionsManagement innerhalb der TradingSystem-Instanz und erhält für diese die folgende Meldung:

NOT VALID

/Handelssystem-Entwicklung/TradingSystem Pool/SimpleLong/PositionsManagement:

- Missing component for composition 'DetermineMarketState'. Need to be create 1 instance of Determine-MarketState

D.h. die Instanz PositionsManagement ist noch nicht konsistent und benötigt eine enthaltene Instanz von DetermineMarketState.

Der Modellierer erzeugt nun eine Instanz von DetermineMarketState innerhalb der Instanz von PositionsManagement. Mit dieser Standardfunktionalität kann das Handelssystem den Status erstellter Positionen ermitteln. Für die Instanz PositionsManagement meldet TradeKnowledge ein **VALID**. D.h. für diese Instanz gibt es entsprechend der Modellbeschreibung keine Regelverletzungen. So sind z.B. alle erforderlichen Attribute gesetzt und alle notwendigen untergeordneten Instanzen vorhanden (hier: DetermineMarketState):

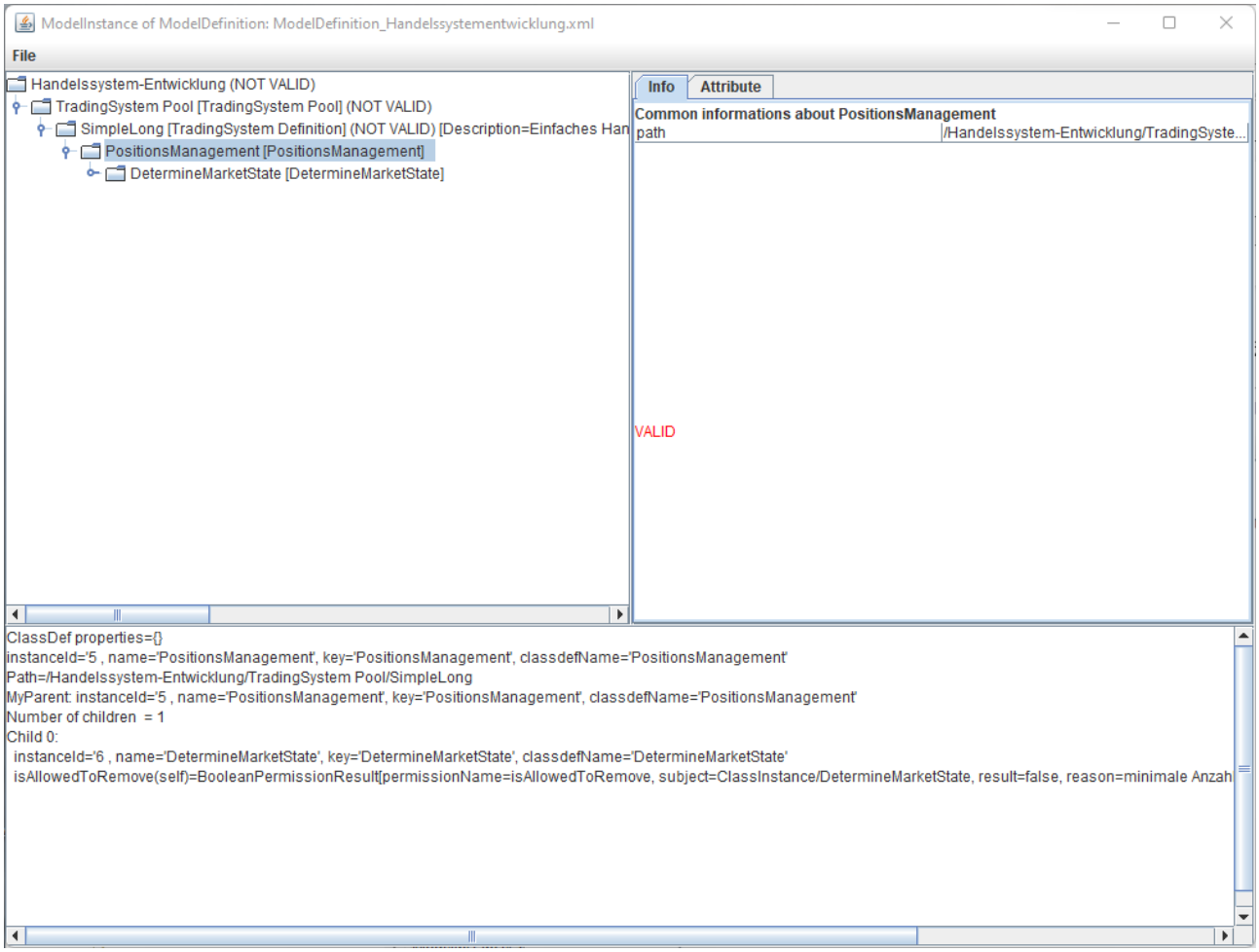


Abbildung 16: Modellierungstool: Positionsmanagement

Der Modellierer wird also anhand der Meldungen von TradeKnowledge durch den Prozess der Modellierung geführt.

9.1.3 Instanz RisikoAndMoneyManagement

Entsprechend der Meldung für die geforderte Komposition „RisikoAndMoneyManagement“ erstellt der Modellierer die geforderte Instanz der Klasse RisikoAndMoneyManagement. TradeKnowledge liefert für diese Instanz die Meldung:

NOT VALID

/Handelssystem-Entwicklung/TradingSystem Pool/SimpleLong/RisikoAndMoneyMananagememt

- Missing component for composition 'DetermineStockAmount'. Need to be create 1 instance of DetermineStockAmount

- Missing releation for aggregation 'LossLimitExitSzenario'. Need to be create 1 releation to ExitSzenario

Der Modellierer ändert den vorgeschlagenen Wert (50000EUR) des Attributes InitialEquityForBacktest auf die vorhandenen 20000 EUR. Nun startet das Handelssystem mit einem Kapital von 20000 EUR für einen Backtest. Der vorgeschlagene Wert (5%) für das Attribut 'MinimumDistanceFromInitialStopPriceAtEntry' bleibt bestehen:

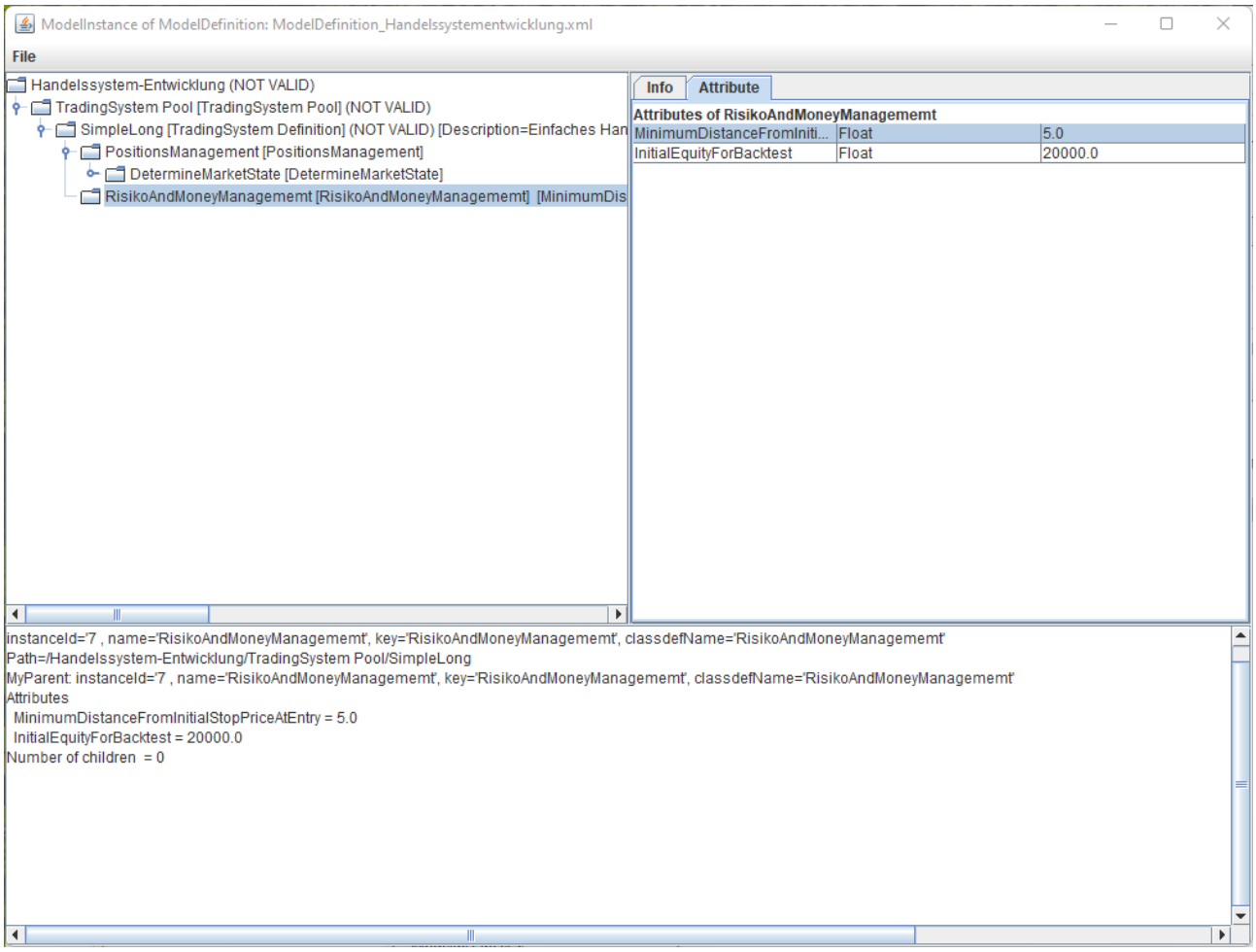


Abbildung 17: Modellierungstool: Attribute für das Risko- und Moneymanagement

Entsprechend der Meldung für die geforderte Komposition „DetermineStockAmount“ erstellt der Modellierer die geforderte Instanz der Klasse DefaultDetermineStockAmount. Das Handelssystem erhält somit ein, von TradeKnowledge bereitgestelltes, Standardverhalten für die Bestimmung der Größe für neue einzugehende Positionen auf Basis des 1%-Risikos des jeweiligen Portfoliokapitals.

Die Meldung **Missing releation for aggregation 'LossLimitExitSzenario'** bedeutet, dass für das Risiko- und Moneymanagement ein Exit-Szenario für eine Risikobegrenzung notwendig ist. Dieses Exit-Szenario existiert z.Zt. im Modell noch nicht. Es wird im weiteren Modellierungsverlauf erstellt (siehe 9.1.8) und dann mit dem RisikoAndMoneyManagement verknüpft. Bis dahin bleibt das RisikoAndMoneyManagement im Status NOT VALID:

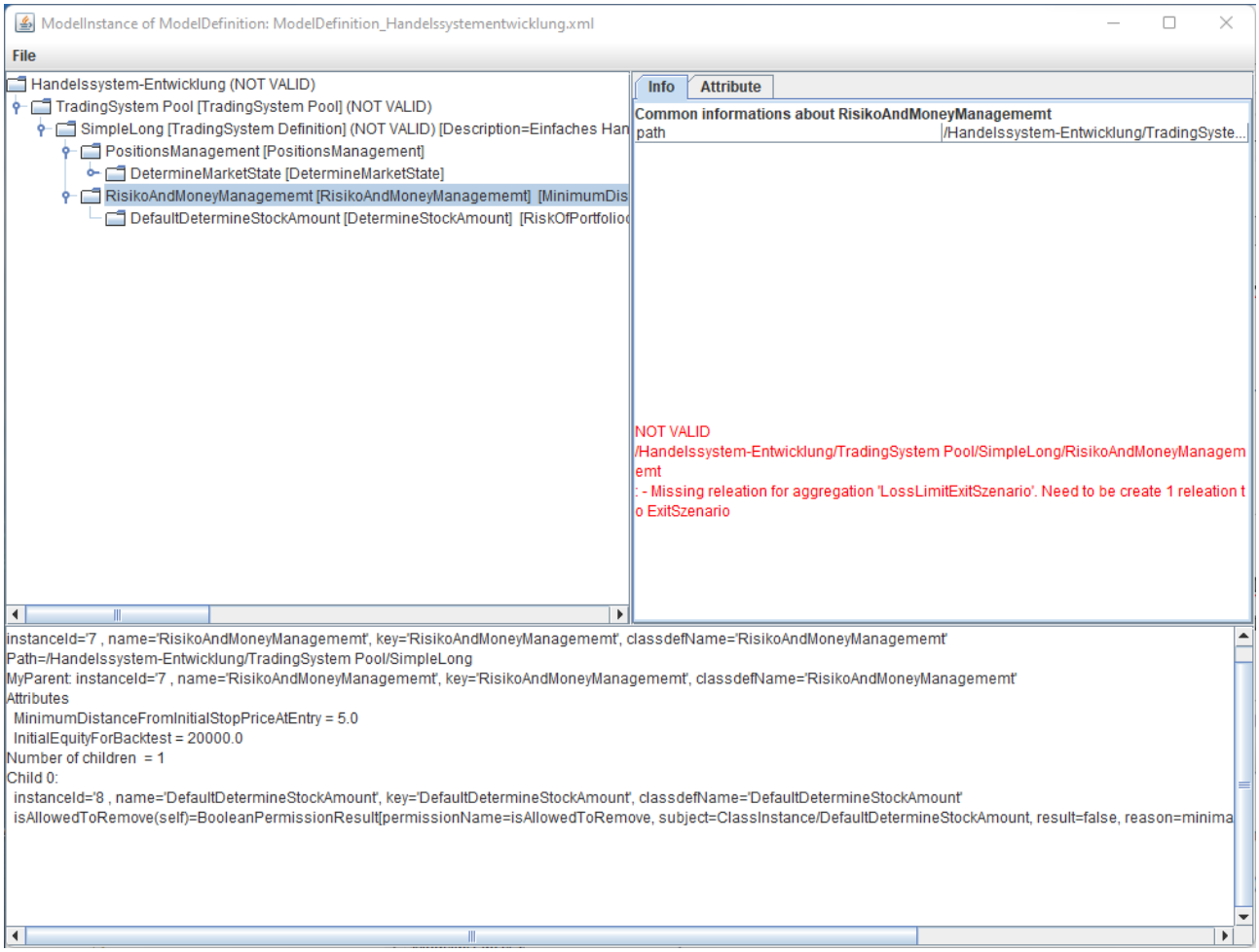


Abbildung 18: Modellierungstool: fehlender Verweis auf ein Ausstiegsszenario für die Riskobegrenzung

9.1.4 Instanz Var

Entsprechend der Meldung von TradingSystem **Missing component for composition 'Vars'. Need to be create 1 instance of Vars** wird die verpflichtende Komponente für die Speicherung von Variablen erzeugt:

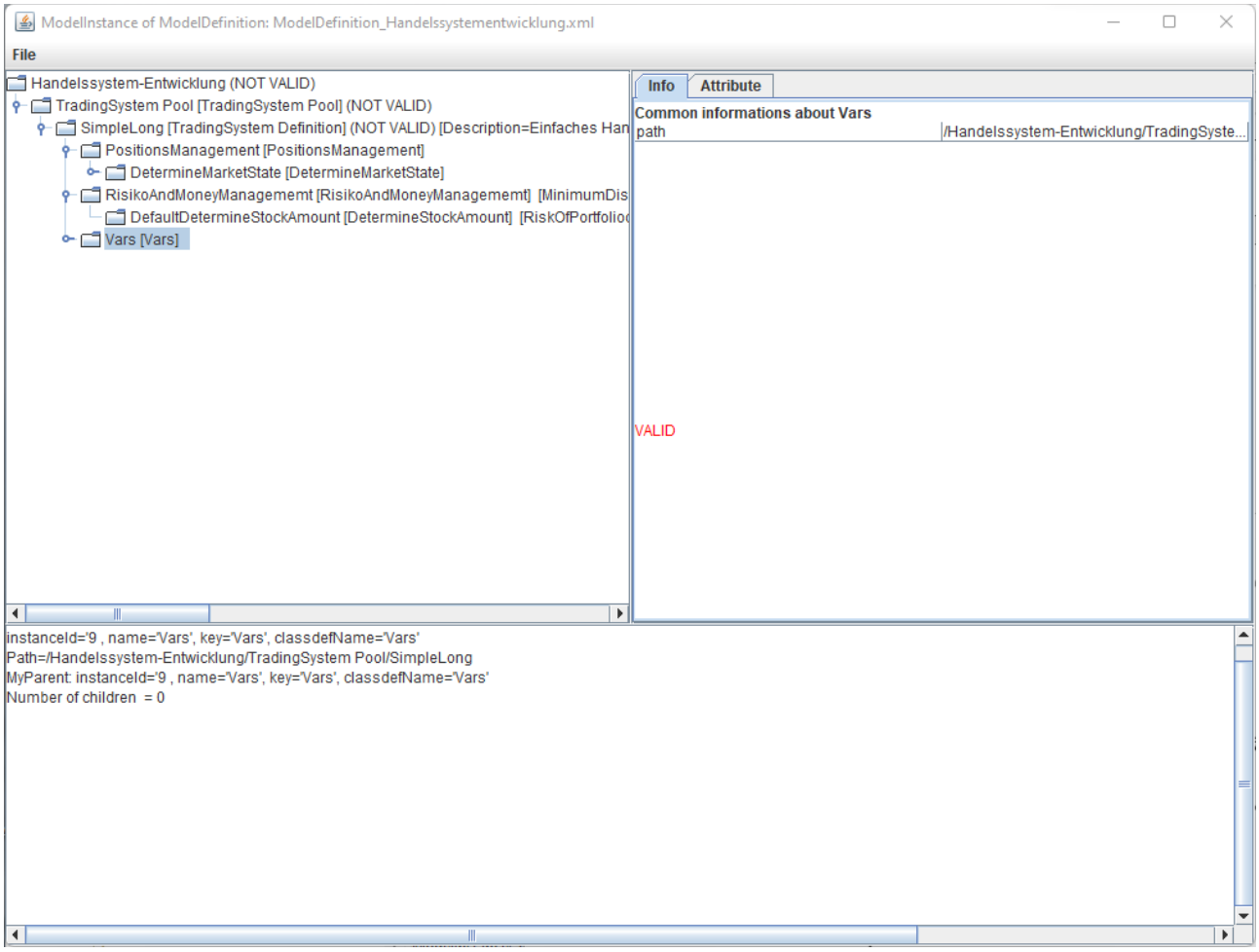


Abbildung 19: Modellierungstool: Var-Instanz

9.1.5 Indikatoren

Innerhalb der Instanz „Var“ werden nun die zwei SMA-Indikatoren definiert, die von den Einstiegs- und Ausstiegsszenarien verwendet werden sollen.

Jeder SMA-Indikator benötigt zur Berechnung die Parameter: Schlusskurs der Woche und die Periode.

Dafür erzeugt der Modellierer nun die folgenden Instanzen:

- „Close“ (vordefiniert), für den Schlusskurs der Woche. Es wird als Intervall „Week“ angegeben.
- „periodeSlow“ als IntegerInput für die Periode des langsamen SMA. Es wird der initiale Wert von 200 angegeben.
- „periodeFast“ als IntegerInput für die Periode des schnellen SMA. Es wird der initiale Wert von 50 angegeben.

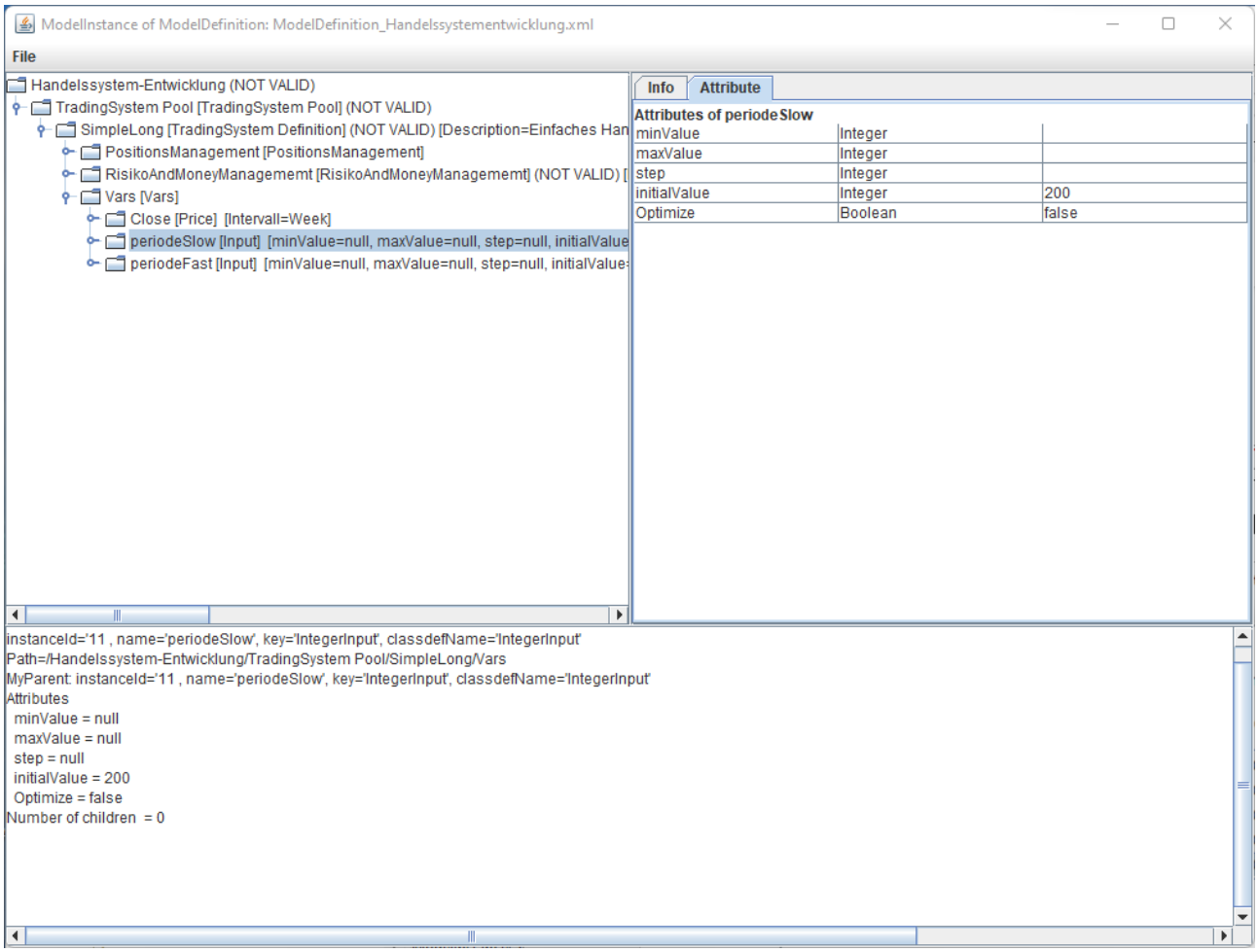


Abbildung 20: Modellierungstool: Parameter für die Indikatoren

Mit dem Attribut „Optimize“=true würden dem generierten Code entsprechend Anweisungen für eine Parameteroptimierung hinzugefügt. Mit „minValue“, „maxValue“ und „step“ können zusätzliche Werte für eine Steuerung der Parameteroptimierung angegeben werden.

Über einen entsprechenden Menüpunkt ist der, in TradeKnowledge vordefinierte, SMA-Indikator auswählbar:

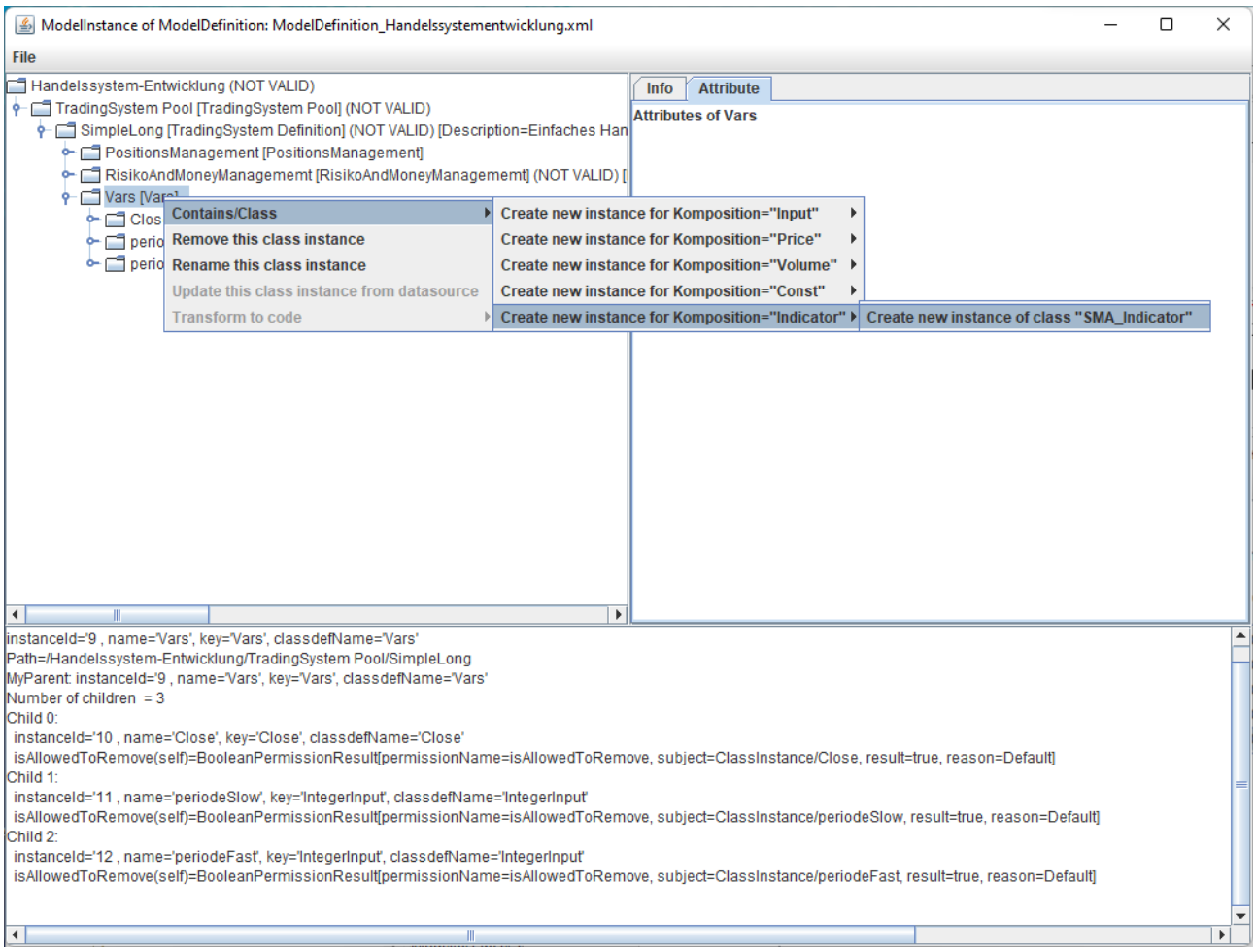


Abbildung 21: Modellierungstool: Menüpunkt zur Erstellung eines SMA-Indikators

Der Modellierer erstellt nun die zwei SMA-Indikatoren mit den Namen „slowSMA“ und „fastSMA“ und verknüpft die Parameter jeweils zum Schlusskurs der Woche und mit den entsprechenden Eingabewerten für die Perioden:

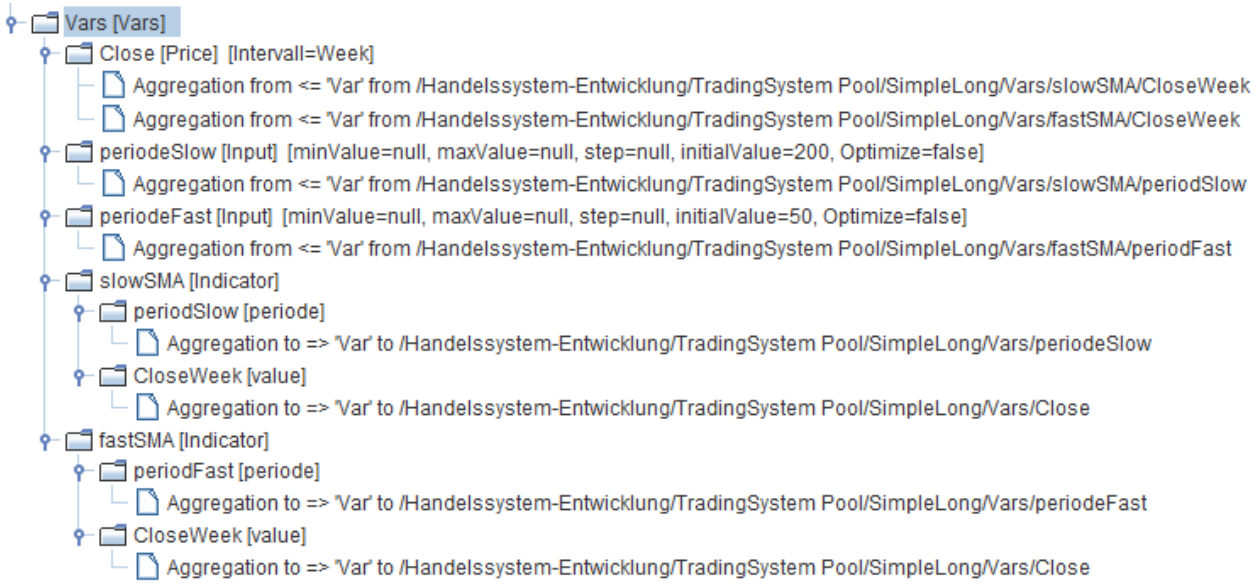


Abbildung 22: Modellierungstool: modellierte Indikatoren

9.1.6 Szenarien

Für die Instanz TradingSystem werden noch die Meldungen angezeigt, dass noch mindestens ein Einstiegs- und mindestens ein Ausstiegsszenario hinzugefügt werden müssen. Außerdem fehlt der Instanz RisikoAndMoneyManagement noch ein Verweis auf ein Ausstiegsszenario für eine Risikobegrenzung (siehe 9.1.3):

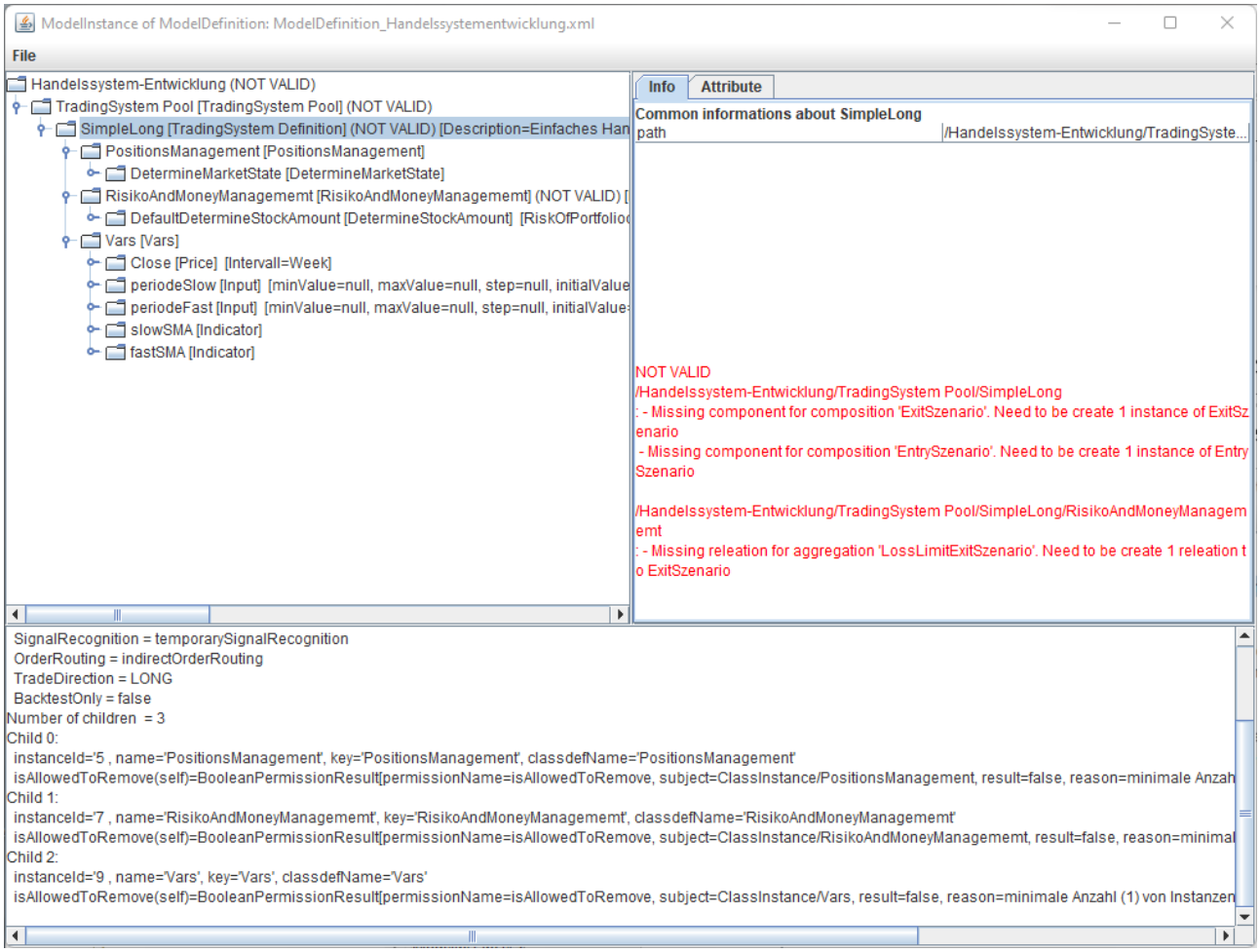


Abbildung 23: Modellierungstool: Modellierungsmeldungen für fehlende Szenarien

Im Folgenden werden diese Szenarien erstellt:

9.1.7 Einstiegsszenario

Den definierten Handelssystemregeln folgend soll ein Einstieg in eine Position erfolgen, wenn der fastSMA den slowSMA von unten nach oben kreuzt.

Der Modellierer erstellt eine Instanz von EntrySzenario mit dem Namen ‚EntryWithSMA' und setzt die folgenden Attribute bzw. Werte:

Attribut	Gesetzter Wert	Kommentar
@name	EntryWithSMA	Kurzer eindeutiger und aussagekräftiger Name des Einstiegs-Szenarios
@Description	Ein Einstieg in eine Position soll erfolgen, wenn der fastSMA den slowSMA von unten nach oben kreuzt.	Aussagekräftige und mehrzeilige Beschreibung des Einstiegs-Szenarios

@SignalTermUsage	AtCloseThisBar	Mit einem Handelssignal soll ein Handel zum Schlusskurs des aktuellen Intervalls (Week) erfolgen. Siehe Arbeit, Kapitel 7.1.2.1.4. und 7.4. Fall D1.
@OrderIntervall	Week	Es sollten auf Wochenbasis Handelssignale und Orders generiert werden. Siehe Arbeit, Kapitel 7.1.2.1.
@OrderTimeFrame	OutsideTradingHours	Die Orderaufgabe erfolgt außerhalb der regulären Börsenhandelszeiten bzw. im außerbörslichen Handel. Siehe Arbeit, Kapitel 7.1.2.1.4.

Für das Einstiegsszenario werden nun Meldungen bzgl. noch fehlender Order, SignalPrice und SignalTerm ausgegeben:

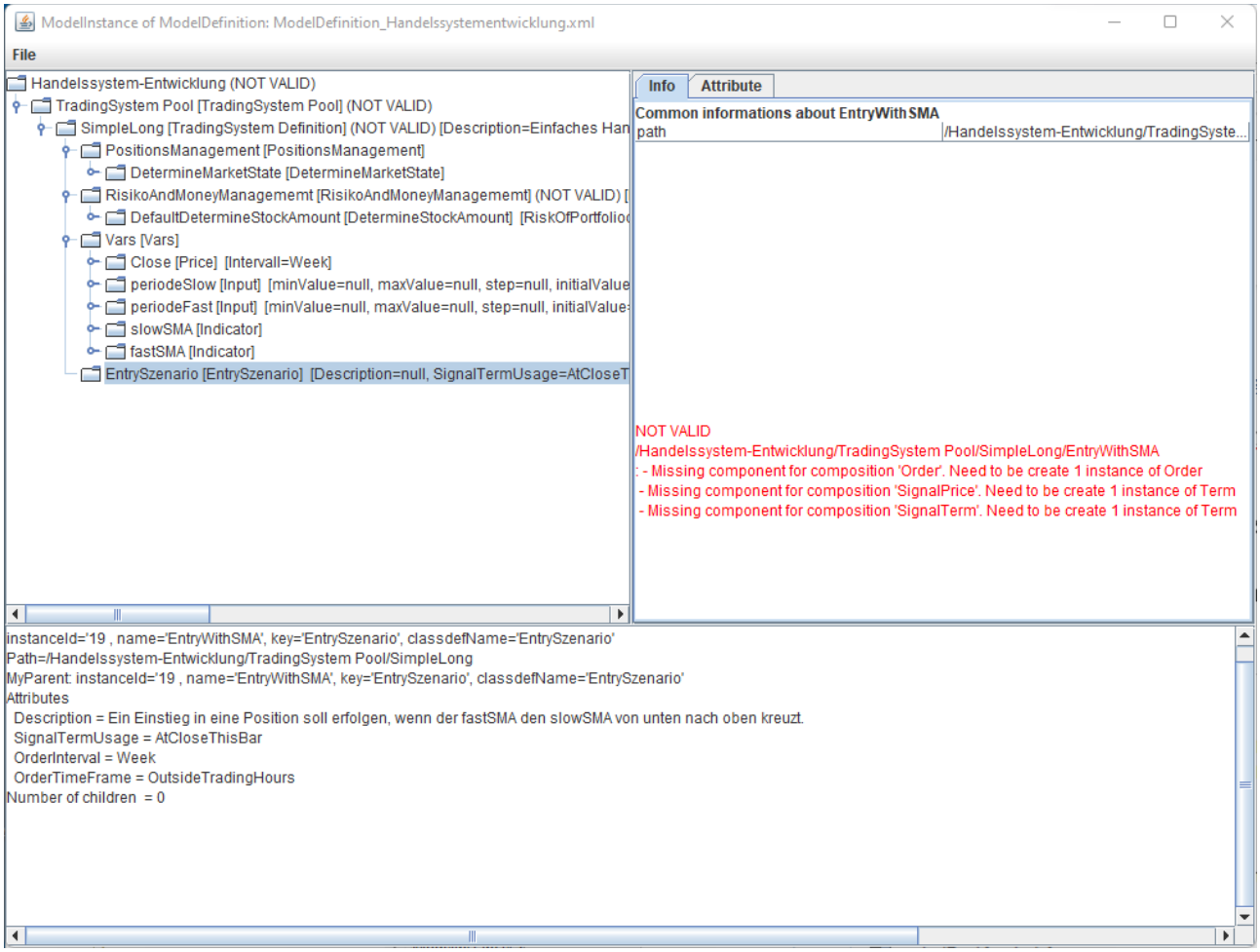


Abbildung 24: Modellierungstool: noch unvollständiges Einstiegsszenario

Der Modellierer erstellt einen SignalTerm mit dem in TradeKnowledge vorgefertigte Term „CrossOver“ (siehe 6.2.2).

Mit dem CrossOver-Term soll ein Handelssignal erzeugt werden, wenn der schnelle SMA den langsamen SMA von unten nach oben kreuzt (der Wert des linken numerischen Terms kreuzt den Wert des rechten numerischen Terms). Dem CrossOver-Term wird als linker Term der fastSMA und als rechter Term der slowSMA zugewiesen:

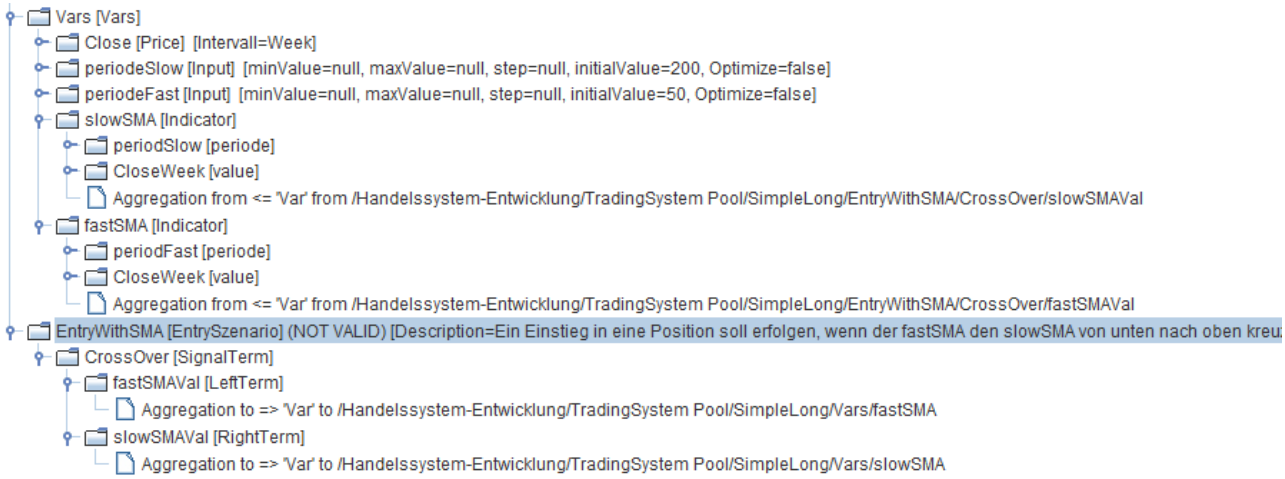


Abbildung 25: Modellierungstool: CrossOver-Term im Einstiegsszenario

Den gesetzten Werten von @SignalTermUsage=AtCloseThisBar und @OrderIntervall=Week entsprechend nimmt TradeKnowledge für den SignalPreis den Close-Preis der aktuellen Woche an (siehe Arbeit, Hinweis im Kapitel 7.1.2.1.4.):

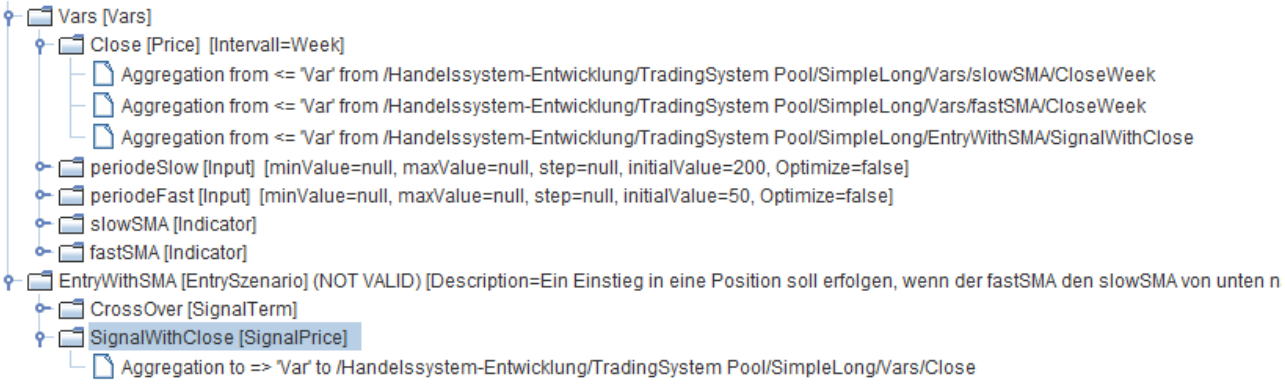


Abbildung 26: Modellierungstool: vorgegebener SignalPreis (Close der Woche) im Einstiegsszenario

Entsprechend den gesetzten Werten von TradingSystem@TradeDirection=LONG und @SignalTermUsage=AtCloseThisBar nimmt TradeKnowledge für die SignalOrder innerhalb eines Szenarios für eine Positionseröffnung eine BuyMarketOrder an (siehe Arbeit, Kapitel 7.1.3.).

Das Einstiegsszenario ist somit fertig modelliert.

9.1.8 Ausstiegsszenario für die Verlustbegrenzung

TradeKnowlege fordert mindestens ein Ausstiegsszenario für eine Verlustbegrenzung. Dazu kann, das von TradeKnowledge vordefiniertes Szenario „**DefaultExitLimitSzenario**“ verwendet werden (siehe Arbeit, Kapitel 7.1.2.1.5).

Dieses generiert eine entsprechende Stopp-Loss-Order zusammen mit bzw. kurz nach einem Einstieg zum angegebenen Ausstiegspreis.

Der Modellierer erstellt eine Instanz von DefaultExitLimitSzenario. Für dieses Szenario gelten die folgenden vorgegebenen Einstellungen:

- Für @ExitSzenarioType ist LossLimitExit gesetzt.
- Für @SignalTermUsage wird AtStopPrice verwendet
- Als SignalTerm wird der vorgefertigte Term **AtEntry** verwendet.

- Als Order wird eine Stopp-Order für einen Verkauf (da TradingSystem@TradeDirection=LONG) zum SignalPrice verwendet

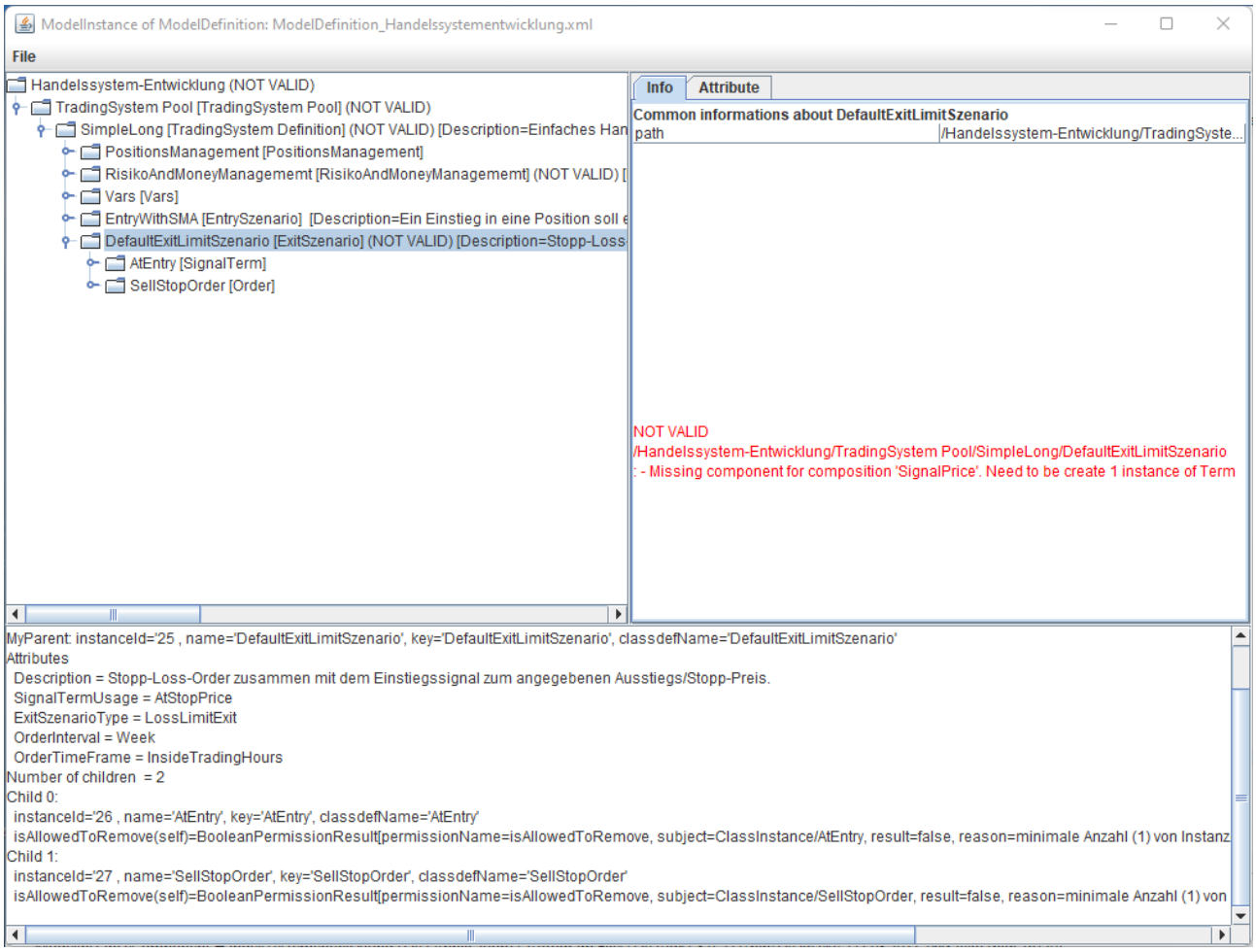


Abbildung 27: Modellierungstool: vordefinierter DefaultExitLimitSzenario

Der Modellierer muss entsprechend der Meldung bzgl. des noch fehlenden SignalPrices den Preis für Ausführungspreis der Stopp-Order angeben. In unserem System bietet sich hier der slowSMA an:

Abbildung 28: Modellierungstool: DefaultExitLimitSzenario mit Stopp-Loss-Preis

Dem RisikoAndMoneyManagement wird noch als Verweis auf ein Ausstiegsszenario für eine Risikobegrenzung (siehe 9.1.3) das eben erstelle Szenario zugewiesen:

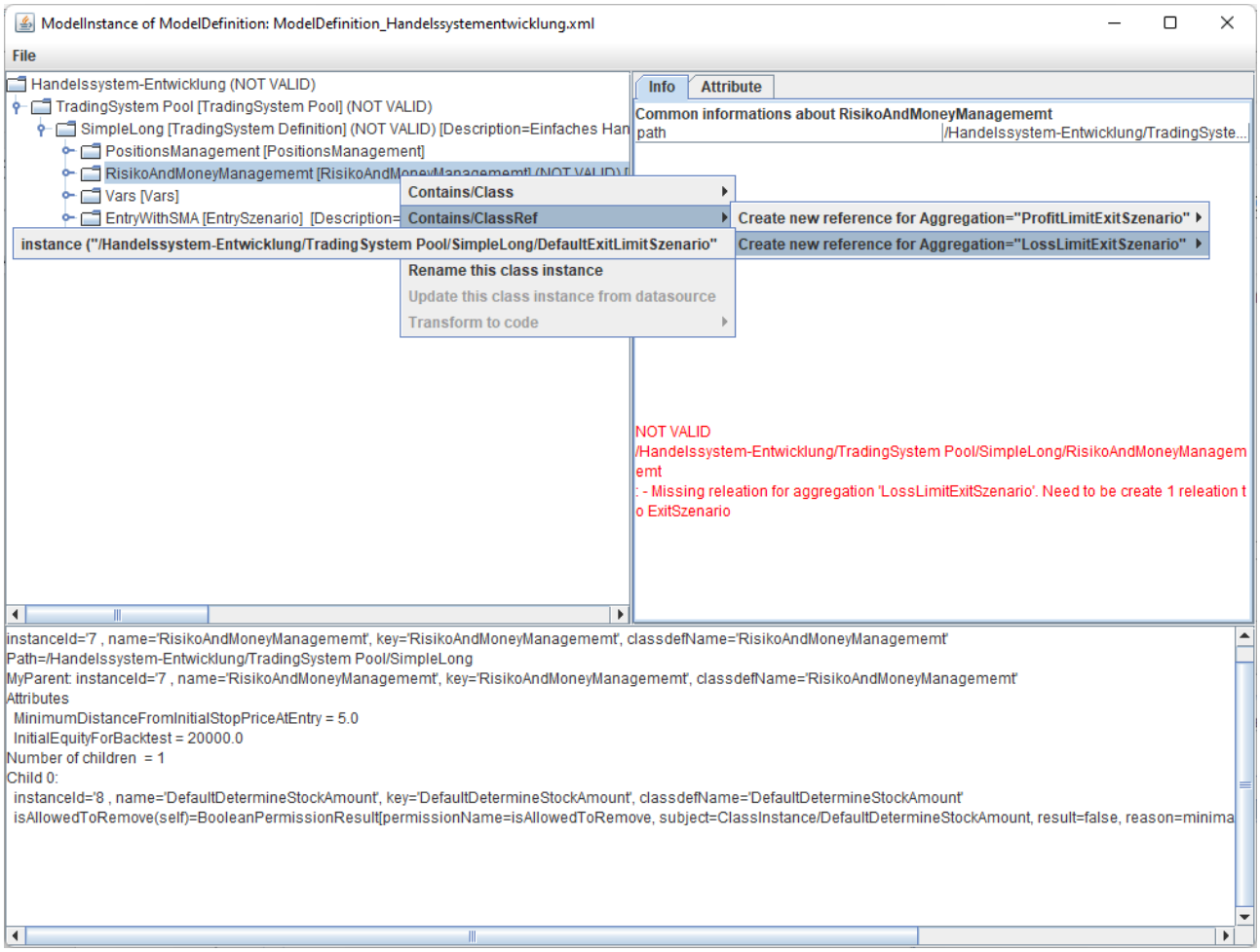


Abbildung 29: Modellierungstool: Zuweisung zum RisikoAndMoneyManagement

9.1.9 Alternatives Ausstiegsszenario

Entsprechend der definierten Handelssystemregeln soll ein Ausstieg aus einer Position erfolgen, wenn der Schlusskurs der Woche den slowSMA durchbrochen hat.

Der Modellierer erstellt eine Instanz von ExitSzenario mit dem Namen ‚ExitWithCloseBelowSMA' und setzt die folgenden Attribute bzw. Werte:

Attribut	Gesetzter Wert	Kommentar
@name	ExitWithCloseBelowSMA	Kurzer eindeutiger und aussagekräftiger Name des Ausstiegszenarios
@Description	Ein Ausstieg aus einer Position soll erfolgen, wenn der Schlusskurs der Woche den slowSMA durchbrochen hat.	Aussagekräftige und mehrzeilige Beschreibung des Ausstiegszenarios
@SignalTermUsage	AtCloseThisBar	Mit einem Handelssignal soll ein Handel zum Schlusskurs des aktuellen Intervalls (Week) erfolgen. Siehe Arbeit, Kapitel 7.1.2.1.4. und 7.4. Fall D1.
@OrderIntervall	Week	Es sollten auf Wochenbasis Handelssignale und Orders generiert werden. Siehe Arbeit, Kapitel 7.1.2.1.

@OrderTimeFrame	OutsideTradingHours	Die Orderaufgabe erfolgt außerhalb der regulären Börsenhandelszeiten bzw. im außerbörslichen Handel. Siehe Arbeit, Kapitel 7.1.2.1.4.
@ExitSzenarioType	AlternativeExit	Typ des Ausstiegs; hier als ein alternatives Ausstiegsszenario.

Wie beim Einstiegsszenario (9.1.7) gibt es Meldungen bzgl. noch fehlender Order, SignalPrice und SignalTerm.

Der Modellierer erstellt einen SignalTerm mit dem, in TradeKnowledge vorgefertigte Term „CrossUnder“ (siehe 6.2.2).

Mit dem Term CrossUnder soll ein Handelssignal erzeugt werden, wenn der Schlusskurs der Woche den langsamen SMA von oben nach unten kreuzt (der Wert des linken numerischen Terms kreuzt den Wert des rechten numerischen Terms). Dem CrossUnder-Term wird nun als linker Term der Close und als rechter Term der slowSMA zugewiesen:

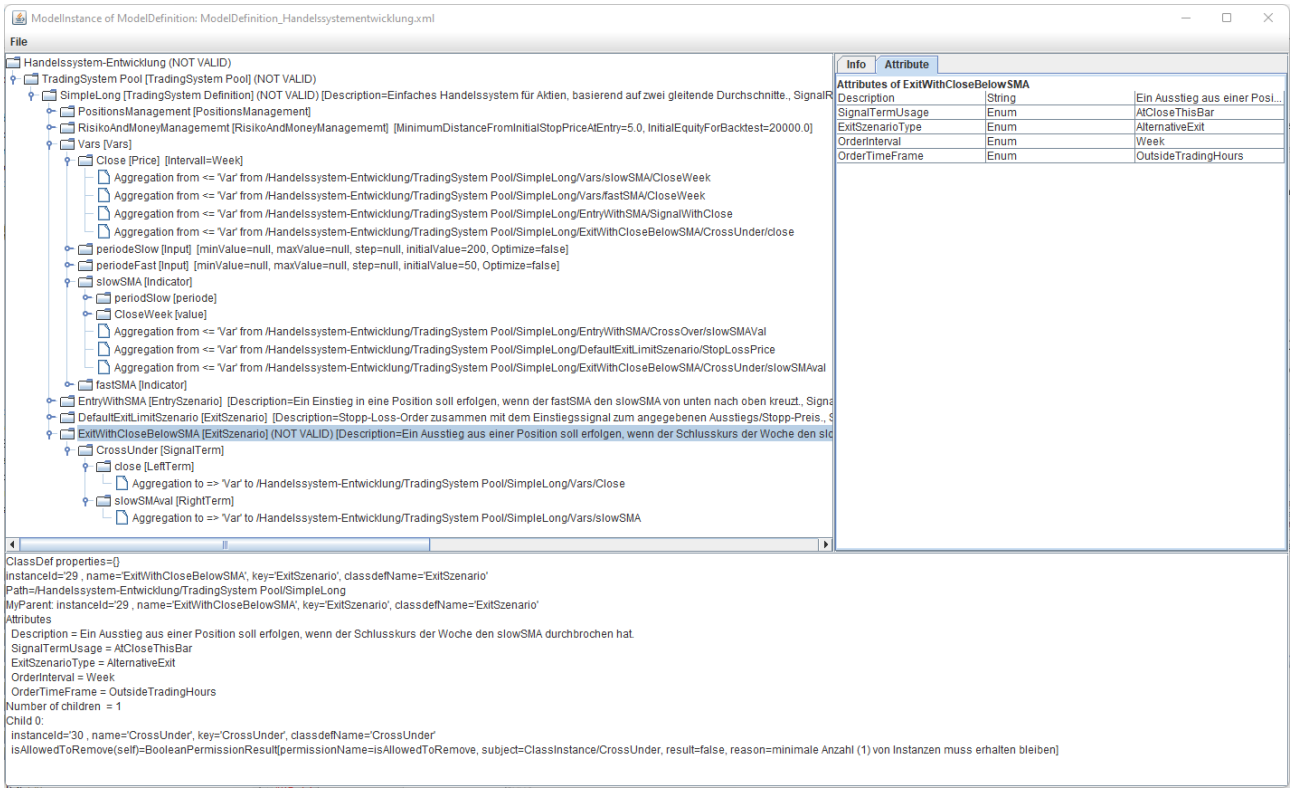


Abbildung 30: Modellierungstool: Attribute des Exitszenarios und Signalterm

Den gesetzten Werten von @SignalTermUsage=AtCloseThisBar und @OrderIntervall=Week entsprechend nimmt TradeKnowledge für den SignalPreis den Close-Preis der aktuellen Woche an (siehe Arbeit, Hinweis im Kapitel 7.1.2.1.4.):

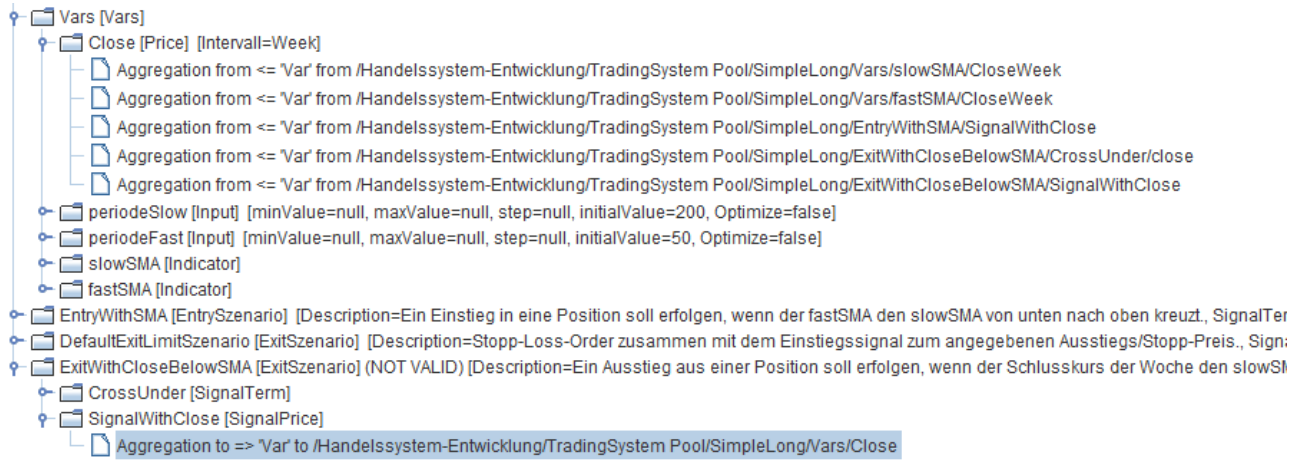


Abbildung 31: Modellierungstool: vorgegebener SignalPreis (Close der Woche) im Ausstiegsszenario

Den gesetzten Werten von TradingSystem@TradeDirection=LONG und @SignalTermUsage=AtCloseThisBar entsprechend nimmt TradeKnowledge für die SignalOrder innerhalb eines Szenarios für eine Positionsschließung eine SellMarketOrder an (siehe Arbeit, Kapitel 7.1.3.).

Das Ausstiegsszenario ist somit fertig modelliert.

9.1.10 Zusammenfassung

Das Modell wurde nach den Ideen für die Handelsanweisungen erstellt und besitzt entsprechend TradeKnowledge einen gültigen Status und ist damit gültig:

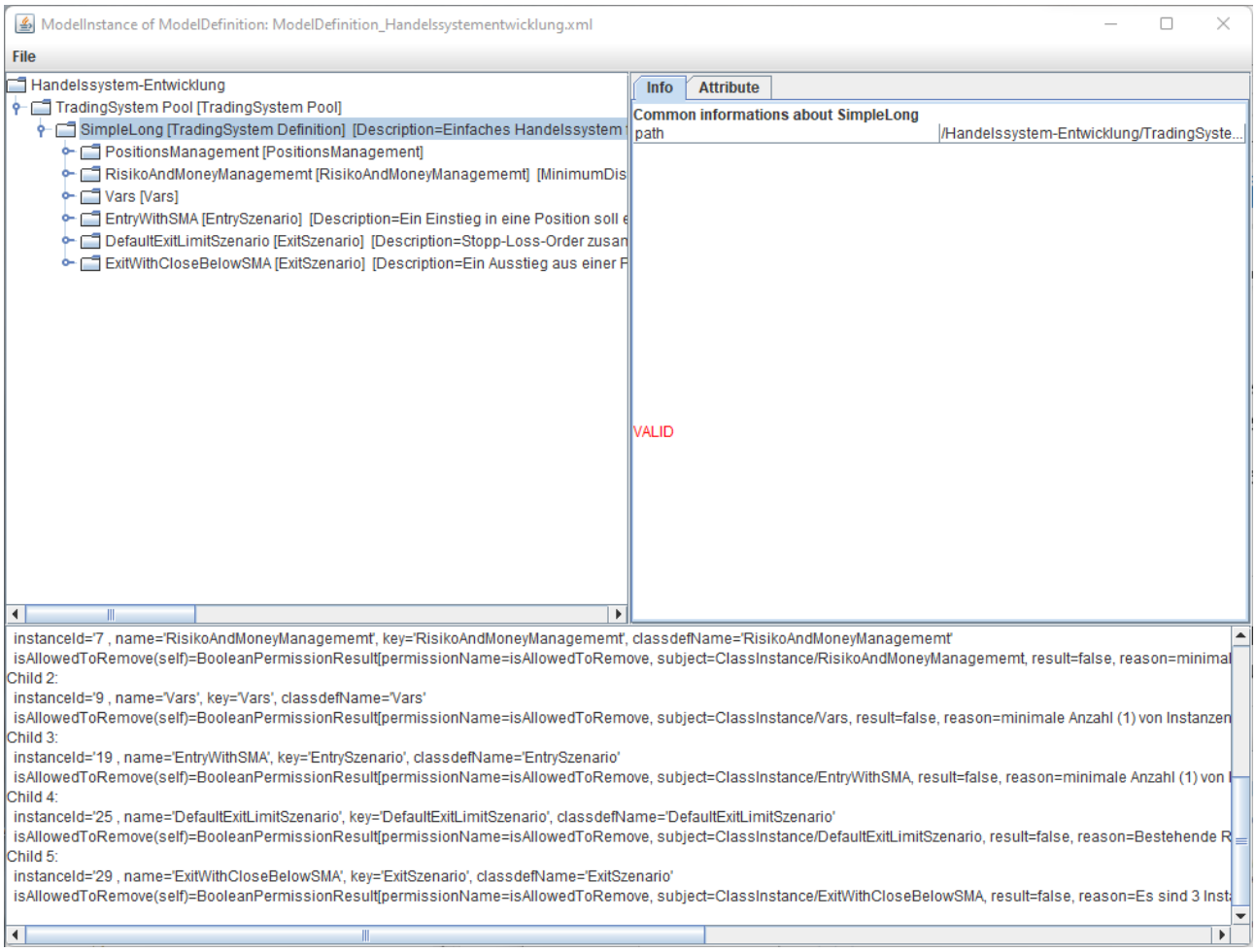


Abbildung 32: Modellierungstool: Gegenüber TradeKnowledge gültiges Modell

Das Modell kann nun in ein Programmcode für eine Ausführung transformiert werden.

9.1.11 Optionale Schritte

Mit einem gültigen Modell können die folgenden weiteren Schritte durchgeführt werden:

- Durchführung von Backtests mit dem Modell-Prozessor (siehe 7.3)
- Definition und Ausführung von Funktionstests (siehe 7.4)
- Generierung einer Dokumentation (siehe 7.5)
- Evaluierung des Modells bzgl. möglicher Zukunftsblicke (siehe 7.7)
- Generierung von Tool zur Unterstützung des praktischen Tradings (siehe 7.6)

9.2 TRANSFORMATION IN PROBUILDER-CODE FÜR PROREALTIME (4. PHASE)

Aus unserem Modell wird nun Programmcode generiert, mit dem das Handelssystem innerhalb von ProRealtime ausgeführt werden kann:

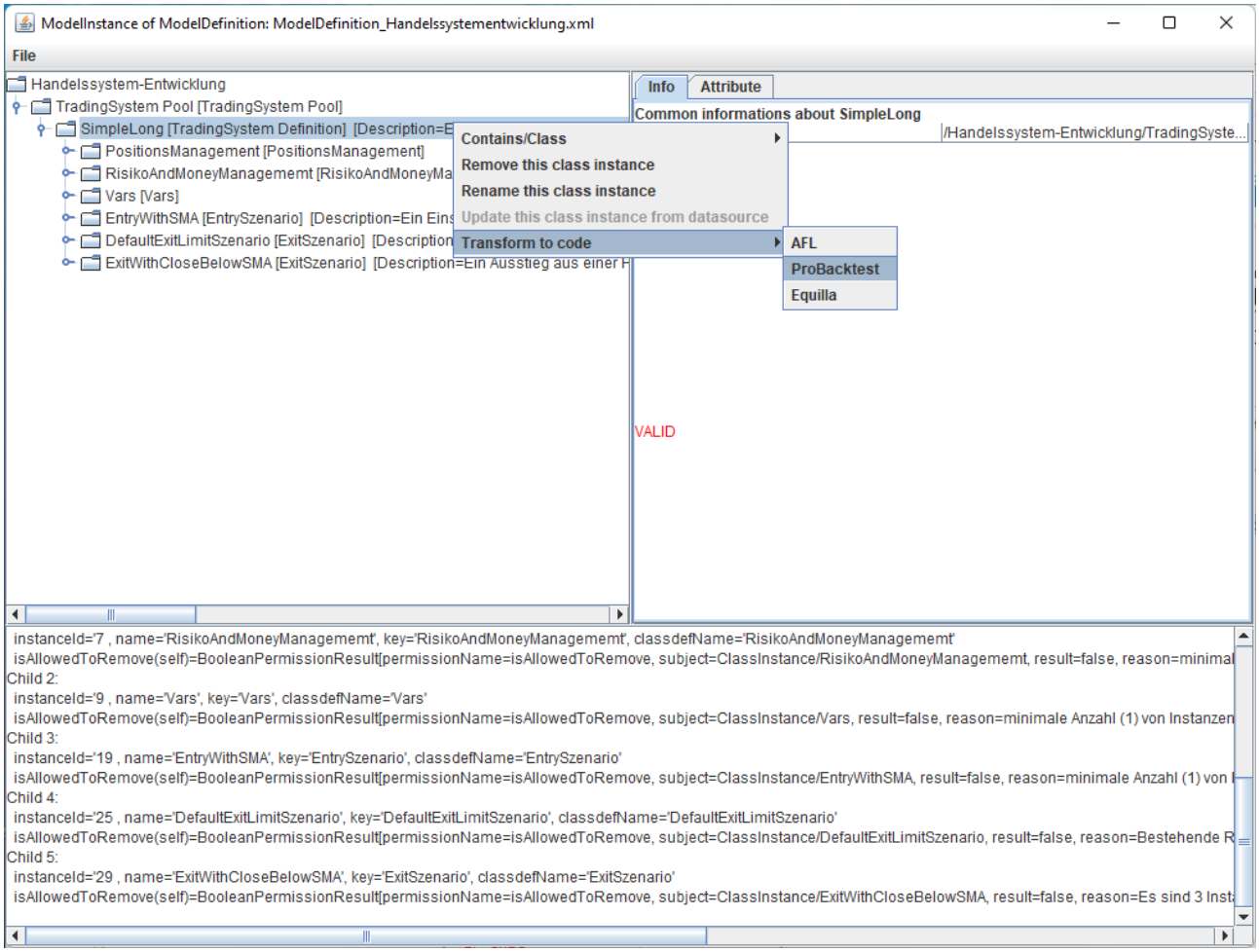


Abbildung 33: Modellierungstool: Transformation in Programmcode

Nach Ausführung der Transformation durch Auswahl von „ProBacktest“ befindet sich der Code in der Zwischenablage und kann mit einem beliebigen Texteditor geöffnet werden:

```
/*
DO NOT EDIT!
Generated by de.modelconfiguration.codetemplates.probacktest.ProBacktestTradingSystemCodeTemplate (at 12.08.22 07:23)



Name: "SimpleLong"

Description:
Einfaches Handelssystem für Aktien, basierend auf zwei gleitende Durchschnitte.

Backtest only: No
Trade direction: LONG
Signal recognition: temporary
Order routing: indirect

Entry-Szenario: 'EntryWithSMA' (Intervall: Week)
Ein Einstieg in eine Position soll erfolgen, wenn der fastSMA den slowSMA von unten nach oben kreuzt.

Exit-Szenario: 'DefaultExitLimitSzenario' (Intervall: Week)
Stopp-Loss-Order zusammen mit dem Einstiegssignal zum angegebenen Ausstiegs/Stopp-Preis.

Exit-Szenario: 'ExitWithCloseBelowSMA' (Intervall: Week)
Ein Ausstieg aus einer Position soll erfolgen, wenn der Schlusskurs der Woche den slowSMA durchbrochen hat.



Risiko & Moneymanagement:
- verwendete Strategie zur Bestimmung der Positionsgröße: Risikospanne entspricht 1% vom Portfoliokapital pro Position

Anweisungen zur Konfiguration:
- Zeiteinheit für den Chart: 1 Woche
- Startkapital für Backtest: 20000.0

*/

REM       do not accumulate positions
defparam cumulateorders = false

REM       used timeframe
TIMEFRAME(weekly)

REM       initiales Kapital für ein Backtest [EUR]
once initialCapital = 20000.0

REM       Risiko pro Trade [%]
once riskPercent = 1.0

REM       minimaler Abstand des Einstiegspreises zum initalen Stopp-Preis [%]
once minDistanceAtEntryPercent = 5.0


REM       ######### USED PRICES #########

closePriceWeek = Close

REM       ######### USED INPUTS #########

inputPeriodeslow = 200
inputPeriodefast = 50

REM       ######### USED INDICATORS #########


REM       INDICATOR: 'slowSMA'
indicatorValueSlowsma = Average[inputPeriodeslow](closePriceWeek)
GRAPHONPRICE indicatorValueSlowsma COLOURED (0,0,255) as "indicatorValueSlowsma"

REM       INDICATOR: 'fastSMA'
indicatorValueFastsma = Average[inputPeriodefast](closePriceWeek)
GRAPHONPRICE indicatorValueFastsma COLOURED (0,255,0) as "indicatorValueFastsma"


REM       ######### USED ENTRY SZENARIOS #########


REM       ENTRY SZENARIO 'EntryWithSMA' :
REM       Ein Einstieg in eine Position soll erfolgen, wenn der fastSMA den slowSMA von unten nach oben kreuzt.

entrysignalEntrywithsma = indicatorValueFastsma CROSSES OVER indicatorValueSlowsma
REM       Behaviour-Code: minimum distance to initial stop price
entrysignalEntrywithsma = entrysignalEntrywithsma AND closePriceWeek >= indicatorValueSlowsma * (1 + minDistanceAtEntryPercent/100)
IF entrysignalEntrywithsma AND NOT LONGONMARKET THEN
          kapital = initialCapital + STRATEGYPROFIT
          positionRisk = kapital * riskPercent / 100
          initialStopLineExit = indicatorValueSlowsma
          GRAPHONPRICE initialStopLineExit COLOURED (255,0,0, 100) as "initialStopLineExit"
          numberOfShares = ROUND(positionRisk / (closePriceWeek - initialStopLineExit))
          barAtEntry = BarIndex
          BUY numberOfShares shares AT MARKET
```

```
ENDIF


REM       ######### USED EXIT SZENARIOS #########


REM       EXIT SZENARIO 'DefaultExitLimitSzenario' :
REM       Stopp-Loss-Order zusammen mit dem Einstiegssignal zum angegebenen Ausstiegs/Stopp-Preis.

exitsignalDefaultexitlimitszenario = (entrysignalEntrywithsma)[BarIndex-barAtEntry]
IF exitsignalDefaultexitlimitszenario AND LONGONMARKET THEN
          SELL numberOfShares shares AT indicatorValueSlowsma[BarIndex-barAtEntry] STOP
ENDIF


REM       EXIT SZENARIO 'ExitWithCloseBelowSMA' :
REM       Ein Ausstieg aus einer Position soll erfolgen, wenn der Schlusskurs der Woche den slowSMA durchbrochen hat.

exitsignalExitwithclosebelowsma = closePriceWeek CROSSES UNDER indicatorValueSlowsma
IF exitsignalExitwithclosebelowsma AND LONGONMARKET THEN
          SELL numberOfShares shares AT MARKET
ENDIF
```

Der Code beginnt mit einem großen Kommentar, der

- die Eigenschaften des Handelssystems,
- Kurzbeschreibungen der verwendeten Szenarien und
- Anweisungen zur Konfiguration enthält.

Jeder Indikator und jedes Szenario erhält einen geordneten Codeblock. Dieser enthält dann die entsprechenden Terme, Kommentare und Beschreibungen aus dem Modell. Ggf. werden Prüfmethoden für Meldungen zur Laufzeit hinzugefügt (siehe Arbeit, Kapitel 7.3.2).

Für den SMA wird die, in ProBuilder enthaltene, Methode „Average“ verwendet. D.h. für die Verwendung einer Modell-Prozessierung (siehe 7.3) müsste der Algorithmus zur Berechnung eines SMAs als Term modelliert oder als wiederverwendbarer Algorithmus dem Modell-Prozessor implementiert werden.

Weiterhin enthält der Code Anweisungen zur Visualisierung der Indikatoren und des jeweils aktuellen Stopp-Preises (so aus dem Exit-Szenario für die Risikobegrenzung).

ProRealtime generiert bereits visuelle Ausgaben von ausgeführten bzw. simulierten Orders. Somit ist hier kein entsprechender Programmcode zur Visualisierung notwendig.

9.3 DEPLOYMENT DES HANDELSSYSTEMS (5. PHASE)

Hiermit wird der Programmcode in die Handelsumgebung ProRealtime geladen:

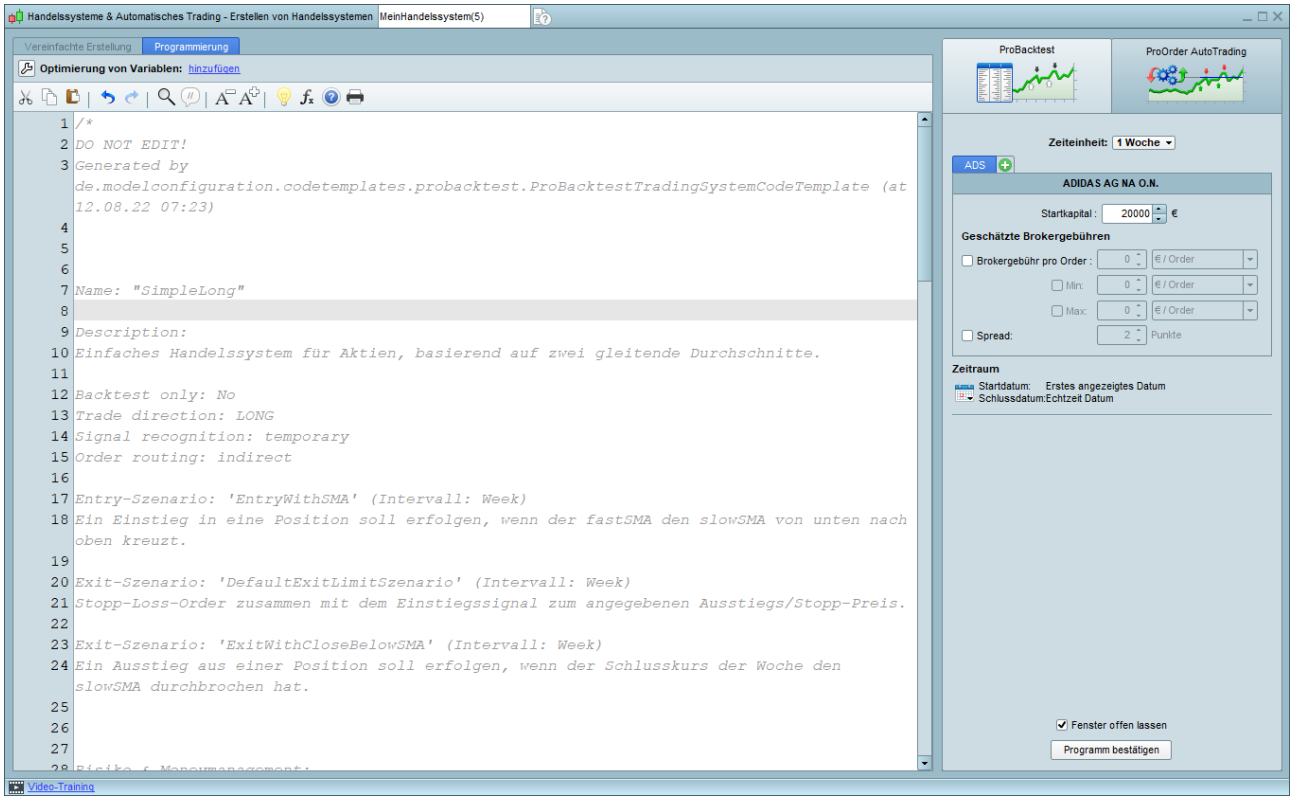


Abbildung 34: Der generierte Programmcode des modellierten Handelssystems in ProRealtime deployed

Entsprechend den, als Kommentar genannten, Anweisungen zur Konfiguration (Zeilen 31 bis 33) sind diese Werte von Startkapital und Zeiteinheit im rechten Teil des Fensters zu setzen.

Das Handelssystem kann nun im ProRealtime verwendet werden und ist innerhalb eines Charts aktiv und sichtbar:

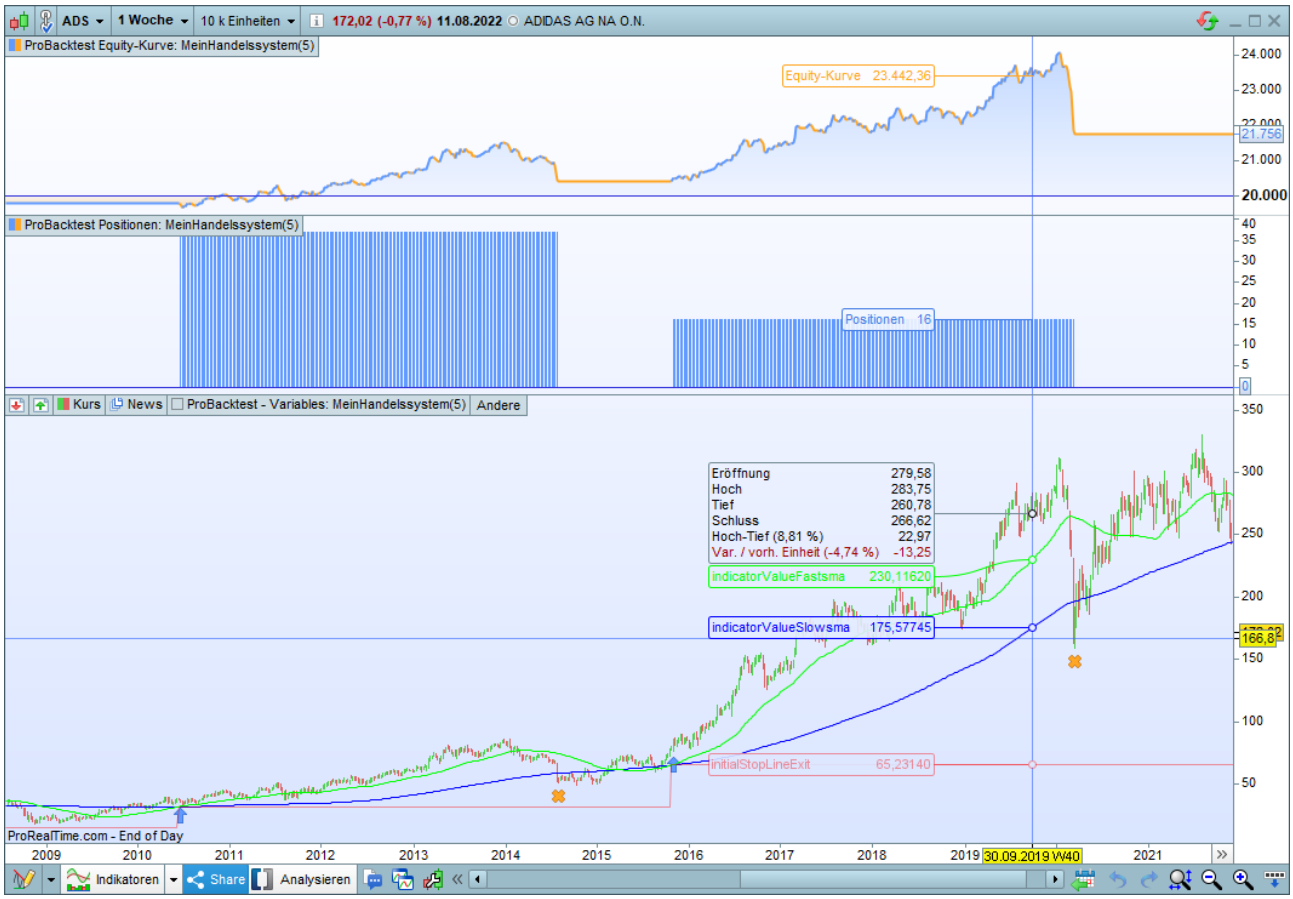


Abbildung 35: Handelssystem innerhalb eines Charts