Kurveninterpolation mit einem finanzmathematischen Modell

Informatikprojekt

von

Nathanael Schärli

Frühling 1999

Durchführung: **Sherpa'x AG**, Glutz-Blotzheim-Str. 1, 4500 Solothurn

Betreuung: **Prof. Dr. Oscar Nierstrasz**
Institut für Informatik und angewandte Mathematik (IAM)
Universität Bern, Neubrückstr. 10, 3012 Bern

Dr. Beat Scherer
Sherpa'x AG, Glutz-Blotzheim-Str. 1, 4500 Solothurn

Dr. Xavier Fábregas
Sherpa'x AG, Glutz-Blotzheim-Str. 1, 4500 Solothurn

Zusammenfassung

Die Firma *Sherpa'x AG* (früher *GfAI*) erstellt seit mehreren Jahren verschiedene Applikationen, die in Grossbanken für den Wertschriftenhandel eingesetzt werden. *FORUMsystems* ist eine dieser Applikationen und dient als Handelsplattform für Börsenhändler. Im Rahmen der Produktpalette von *FORUMsystems* wurde eine finanzmathematische Bibliothek in *C++* geschaffen, welche die nötigen Bewertungsfunktionen bereitstellt.

In der Finanzmathematik werden Kurven verwendet um die aktuellen Marktverhältnisse zu modellieren. Sie dienen als Input für theoretische Bewertungen und nehmen daher eine zentrale Aufgabe in der Entscheidungsunterstützung im Handelsbereich einer Bank ein. In der finanzmathematischen Bibliothek von *FORUMsystems* stehen Kurven für die Bewertung zur Verfügung und werden auch für die Berechnung von theoretischen Kursen verwendet. Die Bestimmung (Interpolation) dieser Kurven erfolgt allerdings ausserhalb dieser Bibliothek, da die entsprechenden Algorithmen in *Mathematica* entwickelt wurden. Dazu wird ein direkter Link zwischen den *FORUM*-Clients und dem *Mathematica*-Kernel verwendet.

Ziel dieses Projektes ist nun die Bestimmung der Kurven ebenfalls in die finanzmathematische *C++*-Bibliothek zu integrieren. Damit wäre der direkte Link von den einzelnen Clients zum Mathematica-Kernel nicht mehr nötig, und es kann mit einem deutlichen Performancegewinn gerechnet werden.

Inhaltsverzeichnis

Kapitel 1: Einleitung 6

1.1 FORUMsystems 6
1.1.1 Finanzmathematische Bibliothek 6
1.1.2 User Interface 7

1.2 Aufgabenstellung und Ziel 7

Kapitel 2: Voraussetzungen 8

2.1 Finanzmathematische Bibliothek von FORUMsystems 8
2.1.1 Trading Instruments und Cashflows 8
2.1.2 Kurven 9
2.1.3 Weitere Komponenten 9

2.2 Hardware- und Softwarevoraussetzungen 9
2.2.1 Hardware und Betriebssystem 9
2.2.2 Entwicklungssoftware 10
2.2.3 Frameworks und Bibliotheken 10

Kapitel 3: Anforderungen 11

3.1 Anforderungsspezifikation 11
3.1.1 Erweiterung der DSS Library 11
3.1.2 Anpassung des OpenRoad-Interfaces 11
3.1.3 Geforderte Produkte und Dokumente 12

3.2 Mathematisches Modell 12
3.2.1 Grundsätzlicher Ablauf der Interpolation 12
3.2.2 Interpolationsarten 12
3.2.3 Stütztpunkttypen (Node-Typen) 14

Kapitel 4: Analyse 15

4.1 Use Cases 15

4.2 Erkenntnisse aus der Analyse 16

Kapitel 5: Design 18

5.1 Packages 18
5.1.1 Package Interpolator 19
5.1.2 Package Node 20
5.1.3 Package Cashflow 21
5.1.4 Package Curve 22
5.1.5 Package Math 23
5.1.6 Package Exception 24
5.1.7 Package RogueWave Tools.h++ 24

Kapitel 6: Ablauf einer Interpolation **25**
6.1 Diagramm 1: Create NodeVector **26**
6.2 Diagramm 2: Interpolate Curve **27**
6.3 Diagramm 3: Init UndetCurve **28**
6.4 Diagramm 4: Determine Curve **29**
6.4.1 Diagramm 4A: Determine LinearCurve 29
6.4.2 Diagramm 4B: Determine SplineCurve 30
6.5 Diagramm 5: Get NodeConditions **31**
6.5.1 Diagramm 5A: Get NodeFunctionRoot 31
6.5.2 Diagramm 5B: Calc NodeEquations 32
6.6 Diagramm 6: Get NodeConditions **33**
6.6.1 Diagramm 6A: Evaluate NodeFunction 33
6.6.2 Diagramm 6B: Get NodeEquation 34
6.7 Diagramm 7: Get NodeConditions **35**
6.7.1 Diagramm 7A: Evaluate LinearSpotFitFunction 35
6.7.2 Diagramm 7B: Get SplineSpotFitEquation 36

Kapitel 7: Projektablauf und Erfahrungen **37**
7.1 Projektablauf **37**
7.1.1 Einführung und Einarbeitung 37
7.1.2 Analyse und Design 38
7.1.3 Implementation 38
7.1.4 Integration und Test 38
7.2 Erfahrungen **39**
7.2.1 Mitarbeit bei einem grossen Softwareprojekt 39
7.2.2 Arbeit mit verschiedenen Softwaretools 39
7.2.3 Erfahrungen aus dem Entwicklungsprozess 40
7.3 Ausblick **40**
7.4 Danksagungen **40**

Anhang A: Screenshots **41**
A.1 Kurve mit Spot-Rate-Stützpunkten **41**
A.2 Kurve mit Forward-Rate-Stützpunkten **43**

Anhang B: Packages und Klassen **45**
B.1 Package Interpolator **45**
B.2 Package Node **47**
B.3 Package Cashflow **49**
B.4 Package Curve **50**
B.5 Package Math **52**
B.6 Package Exception **55**
B.7 Package RogueWave Tools.h++ **56**

Abbildungsverzeichnis

Abbildung 4-1: Use Cases ... 16
Abbildung 5-1: Package-Übersicht... 19
Abbildung 5-2: Package Interpolator... 19
Abbildung 5-3: Package Node ... 20
Abbildung 5-4: Package Cashflow ... 21
Abbildung 5-5: Package Curve ... 22
Abbildung 5-6: Package Math... 23
Abbildung 5-7: Package Exception ... 24
Abbildung 5-8: Package RogueWave Tools.h++ ... 24
Abbildung 6-1: Sequence Diagramm 1 - Create NodeVector ... 26
Abbildung 6-2: Sequence Diagramm 2 - Interpolate Curve ... 27
Abbildung 6-3: Sequence Diagramm 3 - Init UndetCurve ... 28
Abbildung 6-4: Sequence Diagramm 4A - Determine LinearCurve ... 29
Abbildung 6-5: Sequence Diagramm 4B - Determine SplineCurve ... 30
Abbildung 6-6: Sequence Diagramm 5A - Get NodeFunctionRoot... 31
Abbildung 6-7: Sequence Diagramm 5B - Calc NodeEquations... 32
Abbildung 6-8: Sequence Diagramm 6A - Evaluate NodeFunction ... 33
Abbildung 6-9: Sequence Diagramm 6B - Get NodeEquation ... 34
Abbildung 6-10: Sequence Diagramm 7A - Evaluate LinearSpotFitFunction ... 35
Abbildung 6-11: Sequence Diagramm 7B - Get SplineSpotFitEquation ... 36
Abbildung A-1: Screenshot einer glatten Interpolation (Spot-Rate-Nodes) ... 41
Abbildung A-2: Screenshot einer linearen Interpolation (Spot-Rate-Nodes) ... 42
Abbildung A-3: Screenshot einer glatten Interpolation (Forward-Rate-Nodes) ... 43
Abbildung A-4: Screenshot einer linearen Interpolation (Forward-Rate-Nodes) ... 44
Abbildung B-1: Klasse DSSTbCurveInterpolator... 45
Abbildung B-2: Klasse DSSTbCurveInterpolatorLinear... 46
Abbildung B-3: Klasse DSSTbCurveInterpolatorSpline... 46
Abbildung B-4: Klasse DSSTbCurveInterpolatorLinearSpot ... 46
Abbildung B-5: Klasse DSSTbCurveInterpolatorSplineSpot ... 47
Abbildung B-6: Klasse DSSTbCurveInterpolatorSplineStd ... 47
Abbildung B-7: Klasse DSSTermItem ... 47
Abbildung B-8: Klasse DSSSortedTermVector ... 47
Abbildung B-9: Klasse DSSNode... 48
Abbildung B-10: Alle von DSSNode abgeleiteten Klassen... 48
Abbildung B-11: Klasse CashFlowList... 49
Abbildung B-12: Klasse CashFlow ... 49
Abbildung B-13: Klassen FixedCashflow und RepeatedCashFlow... 49
Abbildung B-14: Klasse AbstractCurve ... 50
Abbildung B-15: Klasse DSSTbPoint ... 50
Abbildung B-16: Klasse DSSTbCurve... 51
Abbildung B-17: Klasse DSSInterestTbCurve... 51
Abbildung B-18: Klasse DSSDiscountTbCurve... 51
Abbildung B-19: Klasse DSSUndetDiscountTbCurve ... 52
Abbildung B-20: Klasse DSSMathVector ... 52
Abbildung B-21: Klasse DSSMathMatrix... 53
Abbildung B-22: Klassen DSSSpline und DSSLinEquation ... 53
Abbildung B-23: Klasse DSSLinSplineEquation... 54
Abbildung B-24: Klasse DSSLinEquationSystem... 54
Abbildung B-25: Klasse DoubleToDoubleFunction ... 54
Abbildung B-26: Utility DSSMathFunctions ... 55
Abbildung B-27: Alle Klassen des Packages Exception... 55
Abbildung B-28: Klasse RWTPtrSortedVector ... 56

Kapitel 1

Einleitung

Das hier vorgestellte Informatikprojekt habe ich bei der Firma *Sherpa'x AG* (früher *GfAI*) in Solothurn erarbeitet. Diese Firma erstellt seit mehreren Jahren Applikationen, die in Grossbanken für den Wertschriftenhandel eingesetzt werden. Meine Arbeiten beschränkten sich auf das Programm *FORUMsystems,* das als Handelsplattform für Börsenhändler dient.

Im Folgenden wird die bestehende Version von *FORUMsystems* kurz vorgestellt und dann wird die Aufgabenstellung und das Ziel der Projektarbeit erklärt. In diesem Einführungskapitel wird alles auf einem hohen Abstraktionsniveau behandelt und weitere Details sind den folgenden Kapiteln zu entnehmen.

1.1 FORUMsystems

FORUMsystems ist eine Handelsplattform für Börsenhändler und unterstützt alle wesentlichen Aspekte des Wertschriftenhandels. Die Applikation läuft auf verschiedenen Plattformen (*SUN Solaris*, *IBM AIX*, *MS Windows NT*), und deshalb ist ein besonderes Augenmerk auf eine plattformunabhängige Entwicklung gelegt worden. *FORUMsystems* besteht aus vielen verschiedenen Modulen, wobei für das vorliegende Projekt eigentlich nur die finanzmathematische Bibliothek sowie die Schnittstelle für Aufruf und Resultatrückgabe von Bedeutung sind.

1.1.1 Finanzmathematische Bibliothek

In der Finanzmathematik werden Kurven verwendet, um die aktuellen Marktverhältnisse zu modellieren. Sie dienen als Input für theoretische Bewertungen und nehmen daher eine zentrale Aufgabe in der Entscheidungsunterstützung im Handelsbereich einer Bank ein.

In *FORUMsystems* werden diese Berechnungen in der finanzmathematischen Bibliothek (*DSS Library*) ausgeführt. Dabei werden Kurven für die Bewertung von Instrumenten und auch für die Berechnung von theoretischen Kursen als Input verwendet.

Die *DSS Library* wurde in *C++* implementiert. Die Bestimmung (Interpolation) der Kurven erfolgt aber ausserhalb der Bibliothek, da die entsprechenden Algorithmen in *Mathematica* entwickelt wurden. Dazu wird bisher ein direkter Link zwischen den *FORUM*-Clients und dem *Mathematica*-Kernel verwendet. Das Ziel dieses Projektes ist nun die Bestimmung der Kurven ebenfalls in die finanzmathematische *C++*-Bibliothek zu integrieren.

1.1.2 User Interface

Das User Interface und andere Teile von *FORUMsystems* wurde mit *CA-OpenRoad (Windows 4GL*) realisiert. Dieses 4. Generation Framework wurde verwendet, da es für die verschiedenen Plattformen zur Verfügung steht. Vom User Interface werden die verschiedenen *C++*-Routinen der Bibliothek über ein spezielle *C++-OpenRoad*-Schnittstelle aufgerufen und auch die Resultatrückgabe erfolgt über diese Schnittstelle.

1.2 Aufgabenstellung und Ziel

Das Ziel dieser Arbeit ist die Bestimmung der Kurven ebenfalls in die finanzmathematische *C++*-Bibliothek zu integrieren. Damit wäre der direkte Link von den einzelnen Clients zum *Mathematica*-Kernel nicht mehr nötig, und es kann mit einem deutlichen Performancegewinn gerechnet werden.

Die bisherige *Mathematica*-Implementation beruht auf einer Sammlung von *Mathematica*-Funktionen. Durch den Mechanismus des *Mathematica*-Patternmatching konnte ein gewisses polymorphes Verhalten erreicht werden. Ausserdem wird von verschiedenen eingebauten *Mathematica*-Funktionen (insbesondere Matrizenoperationen) Gebrauch gemacht.

Weil die bisherige Implementation keine objektorientierte Struktur aufweist, ist für eine saubere OO-Umsetzung in *C++* ein komplettes Redesign notwendig. Dies erfordert ein recht tiefes Verständnis der *Mathematica*-Ausdrücke, der verwendeten finanzmathematischen Bewertungsmethoden und der linearen Algebra (insbesondere lineare Version der *Lagrange-Methode*).

Neben Analyse, Design und Implementation ist auch das Testen sowie die Integration ein Teil der Aufgabenstellung. Bei der Integration handelt es sich in erster Linie um die Modifikation der *C++-OpenRoad*-Schnittstelle, damit anstelle der *Mathematica*-Aufrufe nun die neu erstellten *C++*-Methoden aufgerufen werden.

Kapitel 2

Voraussetzungen

FORUMsystems liegt bereits in einer höheren Version vor und läuft erfolgreich auf verschiedenen Plattformen. Es ist klar, dass bei der Erweiterung eines solchen Systems sehr viele Randbedingungen und Voraussetzungen fest vorgegeben sind. Ausserdem muss man das System oder zumindest gewisse Komponenten davon recht gut verstehen, damit man die Erweiterungen in einer sauberen und homogenen Weise anbringen kann.

In diesem Kapitel wird nun die finanzmathematische Bibliothek von *FORUMsystems* etwas näher vorgestellt und danach wird noch auf andere Voraussetzungen und Randbedingungen eingegangen.

2.1 Finanzmathematische Bibliothek von FORUMsystems

Wie in Abschnitt *1.2* erläutert ist das Ziel dieses Projekts die Erweiterung der finanzmathematischen Bibliothek (*DSS Library*) um Funktionen zur Interpolation von Kurven. In diesem Abschnitt wird nun der Aufbau dieser Bibliothek erläutert. Dabei wird vor allem auf die für mein Projekt relevanten Komponenten eingegangen.

2.1.1 Trading Instruments und Cashflows

Damit man die Marktverhältnisse simulieren kann, wurden die verschiedenen *Handelsinstrumente* (*Trading Instruments*, *TI*) in dieser Bibliothek modelliert. Die verschiedenen *TI*-Klassen sind von einem funktionalen Standpunkt alle sehr ähnlich, unterscheiden sich aber in der konkreten Implementation teilweise beträchtlich. Diesem Umstand wurde mit einer Klassenhierarchie und abstrakten Basisklassen Rechnung getragen. Vielen *Trading Instruments* enthalten *Cashflows*, welche Geldzahlungen an den Besitzer repräsentieren. Wie die *Trading Instruments* bestehen solche *Cashflows* im Wesentlichen aus Wert-Datums-Paaren und auch sie wurden in einer Klassenhierarchie mit abstrakter Basisklasse modelliert. Da mit einem *TI* in der Regel mehrere *Cashflows* verbunden sind, wurde eine Listen-Datenstruktur, die sogenannte *Cashflow-List,* erstellt. Diese erlaubt eine komfortable Verwaltung mehrerer zusammengehöriger *Cashflows* und ist den verschiedenen *Trading Instruments* angehängt.

2.1.2 Kurven

In der *DSS Library* stehen Kurven für die Bewertung von Instrumenten zur Verfügung und werden auch für die Berechnung von theoretischen Kursen verwendet. Diese Kurven sind in einer kleinen Klassenhierarchie modelliert. Zuoberst steht eine abstrakte Kurvenklasse (*AbstractCurve*), welche gewisse Grundfunktionen und ein Interface bereitstellt. Die konkreten *Timeband-Kurven* (*DSSTbCurves*) werden von dieser Klasse abgeleitet. Sie bestehen aus einer beliebigen Anzahl Splines (polynomiale Kurven vom Grad 3), welche auf einem gewissen Intervall [a, b] definiert sind. Die einzelnen Splines werden in einer eigenen Klasse (*DSSTbPoint*) modelliert und innerhalb der Kurve in einem geordneten Vektor verwaltet.

Ist eine solche Kurve bekannt, so können verschiedene finanzmathematische Kennzahlen wie z. B. die *Interest-Rate* oder der *Discount-Factor* berechnet werden.

2.1.3 Weitere Komponenten

Neben *Trading Instruments*, *Cashflows* und *Kurven* sind in der finanzmathematischen Bibliothek noch zahlreiche andere Komponenten vorhanden. Dazu gehören einerseits eigenständige Teile, welche andere (mathematische) Aufgaben übernehmen. Auf der anderen Seite hat es aber auch noch verschiedene Hilfsklassen, welche mit den erwähnten Komponenten zusammenarbeiten (beispielsweise ein *Cashflow-Generator* zur Erzeugung einer *Cashflow-List*).

2.2 Hardware- und Softwarevoraussetzungen

Weil diese Projektarbeit aus der Erweiterung einer bestehenden Bibliothek besteht, sind natürlich auch verschiedene Voraussetzungen an die verwendeter Hard- und Software für den Entwicklungsprozess fest vorgegeben und werden hier kurz vorgestellt.

2.2.1 Hardware und Betriebssystem

Die Entwicklung des *C++*-Codes erfolgt unter *SUN Solaris 2.5* bzw. *2.7* und *IBM AIX 4.0*. Der Code muss schlussendlich auch unter *MS Windows NT 4.0* kompilierbar sein.

Für die Verwendung von *Rational Rose 98* und anderer Dokumentationssoftware stehen auch *MS Windows 95* bzw. *MS Windows NT 4.0* zur Verfügung.

2.2.2 Entwicklungssoftware

Für den Entwicklungsprozess stehen folgende Softwaretools zur Verfügung.

- *Rational Rose 98 Professional C++-Edition* für die Erarbeitung eines *UML*-Modells und die Erstellung der Dokumentation.
- *GNU Emacs* bzw. *XEmacs* für das Erstellen und Bearbeiten des Sourcecodes.
- *Emake-Tools* unter *UNIX* für die Verwaltung des Entwicklungs-Workspace und des Sourcecodes.
- *C++*-Compiler und Debugger von *SUN*, *IBM* bzw. *Microsoft* für Entwicklung und Analyse des Codes.

2.2.3 Frameworks und Bibliotheken

FORUMsystems wurde unter Verwendung des Frameworks *CA-OpenRoad* (*Windows 4GL*) und einer *Ingres* Datenbank realisiert. Von *OpenRoad* aus werden die verschiedenen *C++*-Routinen über eine spezielle *4GL/3GL*-Schnittstelle aufgerufen, und auch die Resultatrückgabe erfolgt über diese Schnittstelle.

Ausserdem wird die Bibliothek *Tools.h++* von *RogueWave* verwendet. Diese stellt vor allem sehr stabile und effiziente Datenstrukturen (verschiedene Listen, Vektoren, Hashtables, usw.) in *C++* zur Verfügung.

Kapitel 3
Anforderungen

In diesem Kapitel werden die Anforderungen an meine Projektarbeit aufgeführt. Zuerst wird auf die Anforderungsspezifikation und die geforderten Produkte und Dokumente eingegangen. Danach folgt eine kurze Erklärung des zugrundeliegenden mathematischen Modells.

3.1 Anforderungsspezifikation

In Abschnitt *1.2* wurde das wesentliche Ziel dieses Informatikprojekts bereits auf einem höheren Abstraktionsniveau formuliert. Hier folgt nun eine genauere Spezifikation.

3.1.1 Erweiterung der DSS Library

Die Bestimmung (Interpolation) der Kurven für die finanzmathematischen Berechnungen erfolgt bisher in *Mathematica*. Dieser Teil ist nun direkt in die *DSS Library* zu integrieren. Da die bisherige *Mathematica*-Implementation rein funktional aufgebaut ist, muss für die geforderte objektorientierte Realisierung in *C++* ein komplettes Redesign vorgenommen werden. Dabei darf das abstrakte Verhalten des Interpolationsalgorithmus aber nicht verändert werden: Bei gleichen Input-Daten muss mit der neuen Version genau dieselbe Kurve als Output resultieren.

Die Erweiterungen sollen möglichst homogen in die *DSS Library* integriert werden, und wo immer möglich sollen bereits bestehende Komponenten und Hierarchien genutzt oder erweitert werden. Da die Bibliothek auf den Plattformen *SUN Solaris*, *IBM AIX* und *MS Windows NT* kompiliert werden muss, darf bei der Implementation nur Code gemäss dem *ANSI-C++-Standard* geschrieben werden. Damit nicht alle Datenstrukturen selbst erstellt und getestet werden müssen, kann und soll auf die Bibliothek *Tools.h++* von *RogueWave* zurückgegriffen werden, welche unter allen Plattformen zur Verfügung steht.

3.1.2 Anpassung des OpenRoad-Interfaces

Mit der *C++-Openroad*-Schnittstelle können beliebige Methodenaufrufe mit Parameterübergabe und Resultatrückgabe zwischen *OpenRoad* und *C++* realisiert werden. Im Moment wird eine Schnittstellen-Implementation verwendet, welche die *OpenRoad-Aufrufe* für die Kurveninterpolation direkt an den *Mathematica*-Kernel weiterleitet und die Resultate an *OpenRoad* zurückgibt.

Dies soll nun so modifiziert werden, dass anstelle des *Mathematica*-Aufrufs die neu geschriebenen Interpolationsmethoden der *DSS Library* aufgerufen werden. Dazu werden aus den Parametern von *OpenRoad* die geeignete Datenstrukturen generiert und an die neuen

Methoden übergeben. Die interpolierte Kurve wird dann wieder ausgelesen, um die wesentlichen Daten an *OpenRoad* zurückgegeben.

3.1.3 Geforderte Produkte und Dokumente

Damit die Interpolationsmethoden vernünftig und zuverlässig gebraucht, angepasst oder erweitert werden können, sollen folgende Produkte und Dokumente erstellt werden.

- *UML*-Dokumentation von Analyse, Design und Laufzeitverhalten. Dies beinhaltet insbesondere ein OO-Modell der Bibliothekserweiterungen.
- Implementation der Erweiterungen in *C++* und Dokumentation des Sourcecodes.
- Anpassung der Generierungsskripte für die verschiedenen Plattformen (*Makefiles*, usw.).
- Integration und Modifikation der Schnittstelle zu *Openroad*.
- Tests auf verschiedenen Plattformen.

3.2 Mathematisches Modell

Hier wird das mathematische Modell für die Interpolation der Kurven kurz erläutert. Eine detaillierte Erklärung kann aber aus verschiedenen Gründen nicht gegeben werden und würde wahrscheinlich auch zu weit führen.

3.2.1 Grundsätzlicher Ablauf der Interpolation

Bei jeder Kurveninterpolation gibt es eine Menge von Stützpunkten (*Nodes*) welche als Randbedingungen verwendet werden. Die Interpolationsart bestimmt, wie die vorhandenen *Nodes* durch die Kurve approximiert werden sollen und welcher Kurventyp dabei entstehen soll. Neben der verschiedenen Interpolationsarten gibt es aber auch verschiedene *Node*-Typen, welche unterschiedliche (mathematische) Bedingungen an die approximierende Kurve stellen.

3.2.2 Interpolationsarten

Im Folgenden werden die verschiedenen Arten der Kurveninterpolation von *FORUMsystems* kurz erläutert.

3.2.2.1 Glatte Spot-Rate-Interpolation (InterpolationSplineSpot)

Bei dieser Interpolationsart wird eine glatte (mindestens 2 mal stetig differenzierbare) Kurve mit minimaler globaler Krümmung erzeugt. Die Kurve besteht aus einzelnen Splines, welche je auf einem Intervall zwischen zwei Stützpunkten (*Nodes*) definiert sind.

Es wird nun ein erstes Gleichungssystem für die Koeffizienten dieser Splines aufgestellt, in welches die Gleichungen für die Glattheit eingefügt werden. Ausserdem wird für jeden *Node* eine (*Node*-spezifische) Gleichung für die gewünschte *Spot-Rate*-Approximation zugefügt. Bei der Startstelle (in der Regel bei 0) wird dabei eine spezielle Gleichung generiert. Die minimale Krümmung der Kurve wird mit Hilfe einer geeigneten Qualitätsfunktion erzielt, welche dann mit der *Methode von Lagrange* minimiert wird. Dazu wird sichergestellt, dass der Gra-

dient der Qualitätsfunktion als Linearkombination der Gradienten aller linearen Gleichungen darstellbar ist. Das entstehende Gleichungssystem wird nun mit dem ersten Gleichungssystem kombiniert, und dies liefert die gesuchte Lösung.

3.2.2.2 Glatte Spot-Rate-Interpolation mit approximativen Stützpunkten (InterpolationSplineSpotApp)

Die Eigenschaften dieser Interpolationsart sind fast identisch mit denjenigen der nicht-approximativen Version. Der Unterschied besteht nur darin, dass die einzelnen Stützpunkte (*Nodes*) jetzt noch mit einem Gewicht bewertet sind. Bei der Interpolation wird versucht, das Quadrat der Abweichung von theoretischer Bewertung und aktuellem Kurs unter Berücksichtigung des Gewichtes zu minimieren.

Bemerkung: Die Implementation dieser Interpolationsart ist nicht Teil der Aufgabe dieses Informatikprojekts!

3.2.2.3 Gewöhnliche glatte Interpolation (InterpolationSplineStd)

Dieses Verfahren ist ziemlich ähnlich wie dasjenige der glatten *Spot-Rate*-Interpolation. Es wird ebenfalls eine glatte Kurve mit minimaler globaler Krümmung erzeugt, welche aus einzelnen Splines aufgebaut ist.

Die Gleichungen für die Glattheit und die minimale Krümmung sowie das Lagrange Verfahren sind vollkommen identisch wie bei der glatten *Spot-Rate*-Interpolation. Die Gleichungen für die einzelnen *Nodes* sehen aber anders aus, da hier eine gewöhnliche Interpolation ausgeführt werden soll. Ausserdem wird keine zusätzliche Gleichung für die Startstelle berechnet. Die Kurve wird aber nach ihrer Berechnung möglichst glatt bis zum Anfang ihres Definitionsbereichs (in der Regel 0) fortgesetzt.

3.2.2.4 Lineare Spot-Rate-Interpolation (InterpolationLinearSpot)

Diese Interpolationsart bestimmt eine stetige, stückweise affine Kurve, welche die gegebenen *Nodes* approximiert. Dazu wird das Verfahren des *Bootstrappings* angewendet, um zwischen je zwei benachbarten *Nodes* eine geeignete affine Kurve zu bestimmen.

Beim *Bootstrapping* ist der Endpunkt der Vorgängerkurve bekannt, und wegen der Stetigkeit ist somit jeweils nur die Steigung der nächsten Strecke zu bestimmen. Dazu wird für den nächsten *Node* eine (*Node*-spezifische) Funktion für die gewünschte *Spot-Rate*-Approximation bestimmt und mittels *Brent-Methode* eine Nullstelle davon berechnet. Diese liefert dann die Steigung des nächsten affinen Abschnitts. Bei der Startstelle (in der Regel 0) wird eine horizontale Kurve angenommen, deren Höhe ebenfalls durch Iteration berechnet wird.

3.2.3 Stütztpunkttypen (Node-Typen)

Bei der Kurveninterpolation gibt es viele verschiedene *Node*-Typen, welche eine Bedingung für die gesuchte Kurve darstellen. Die einzelnen Stützpukttypen werden für die unterschiedlichen Finanzinstrumente und Bewertungsmethoden verwendet: Beispielsweise *Swap-Rate*, *Spot-Rate*, *Money-Market-Rate*, *Bond-Future* (mit *Cashflows*) oder Forward-Rate.

Im einfachsten Fall besteht ein *Node* aus einer Laufzeit (*Term*) und einem Wert (*Value*). In der Anwendung kann aber anstatt der Laufzeit auch ein Datum verwendet werden. Bei gewissen Typen sind zusätzlich noch weitere Attribute wie z.B. *Cashflow*-Listen angehängt.

Das Wesentliche ist nun, dass aus dem *Node*-Typ und den vorhandenen Daten eine Gleichung (für die glatte Interpolation) bzw. eine Steigungsfunktion (für die lineare Interpolation) berechnet werden kann. Dies wird nämlich bei der Interpolationsmethode als Bedingung für die gesuchte Kurve verwendet.

Kapitel 4

Analyse

Nachdem in den vorangehenden Kapiteln die Aufgabe und das Ziel des Projekts genauer erläutert wurden, stelle ich nachfolgend die wesentlichen Erkenntnisse aus meiner Analyse vor.

4.1 Use Cases

Ein kleiner aber wesentlicher Punkt meiner Analyse sind die verschiedenen Use Cases, welche die möglichen Benutzungsszenarien der geplanten Erweiterungen darstellen.

Das grobe Verfahren ist bei jeder Interpolationsmethode gleich: Ein externer Benutzer generiert zuerst die verschiedenen *Nodes* gemäss den konkret vorliegenden Randbedingungen (z.B. *Trading Instruments*). Diese werden dann in einer geeigneten Kollektion (z.B. einem sortierten Vektor) gespeichert. Damit kann der externe Benutzer eine beliebige Interpolationsmethode aufrufen und die eben generierte *Node*-Datenstruktur als Parameter übergeben. Neben diesem Parameter sind der Interpolationsmethode noch ein paar weitere Argumente (wie z.B. aktuelles Datum) zu übergeben. Diese Objekte stehen dem externen Benutzer aber direkt zur Verfügung oder können mit bereits bestehenden Klassen sehr einfach erzeugt werden.

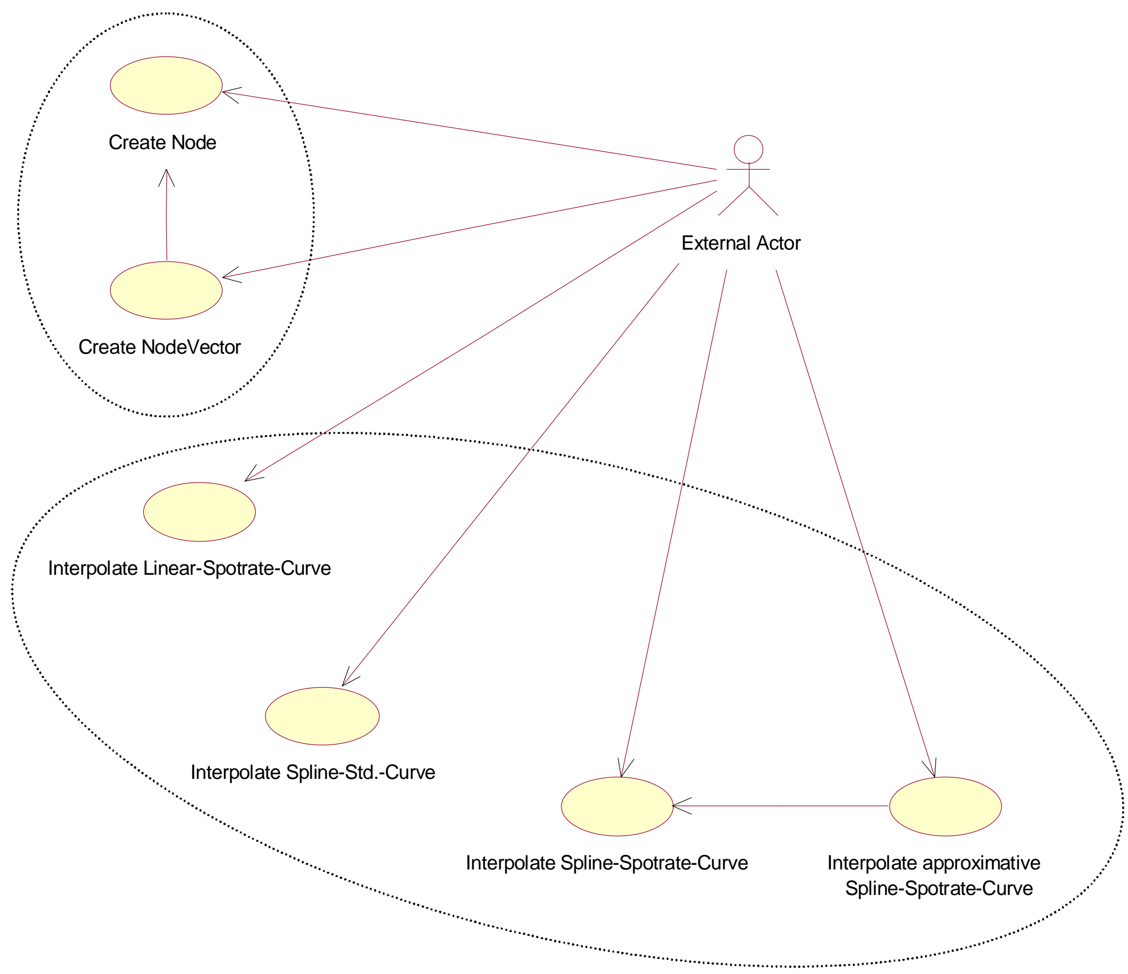

Abbildung 4-1: Use Cases

4.2 Erkenntnisse aus der Analyse

- Sowohl die verschiedenen Interpolationsmethoden als auch die verschiedenen Stützpunkte (*Nodes*) haben von einem äusseren Standpunkt dieselben Aufgaben und hängen auch von denselben Parametern ab. Dies legt sowohl für die verschiedenen *Interpolatoren* als auch für die *Nodes* eine Klassenhierarchie mit einer abstrakten Basisklasse nahe. Das gemeinsame Interface sollte dabei bereits in der Basisklasse soweit festgelegt werden, dass man einen beliebigen Erben unabhängig von seinen konkreten Eigenschaften in einer polymorphen Weise verwenden kann.
- Sowohl die Interpolationsvorgänge als auch die verschiedene Berechnungen in den *Nodes* sind auf einem hohen Abstraktionsniveau identisch. Erst auf einer tieferen Ebene unterscheiden sich die angewandten (mathematischen) Verfahren in verschiedenen Details. Deshalb habe ich mich für eine recht intensive Verwendung des Designpatterns *„Template Method“* entschieden. Dabei wird der grundlegende Ablauf bereits in einer Methode der abstrakten Basisklasse festgelegt. Darin werden aber verschiedene Methoden (sogenannte primitive Methoden) aufgerufen, die in der abstrakten Klasse noch nicht implementiert sind. In einem konkreten Erben müssen dann bloss noch die verschiede-

nen primitiven Methoden ausgefüllt werden, ohne sich um den globalen Ablauf zu kümmern, der ja bereits in der Basisklasse festgelegt ist.

- Die verwendeten Algorithmen machen teilweise sehr intensiven Gebrauch von linearer Algebra. Deshalb ist es sinnvoll, die verschiedenen Objekte, welche in der linearen Algebra verwendet werden (Vektoren, Matrizen, lineare Gleichungssysteme, usw.), in unabhängigen und komfortablen Klassen zur Verfügung zu haben. Dadurch werden die verwendeten mathematischen Formeln viel übersichtlicher, und der Code ist weniger Komplex. Da keine geeignete Bibliothek mit diesen Fähigkeiten zur Verfügung steht, müssen die benötigten Klassen im Rahmen dieses Projekts ebenfalls manuell erstellt werden.
- In der *DSS Library* besteht schon eine Klassenhierarchie für Kurven. Insbesondere gibt es dort auch eine Klasse, welche Kurven, die stückweise aus Splines zusammengesetzt sind, modellieren kann. Es ist deshalb sinnvoll, diese Hierarchie zu verwenden und gegebenenfalls zu erweitern. (Vgl. *Abschnitt 2.1.2*)
- Bei verschiedenen *Nodes* wird eine Liste von *Cashflows* verwendet, welche gewisse Eigenschaften des *Trading Instruments* darstellen. Da in der *DSS Library* bereits die nötigen *Cashflows* sowie eine *Cashflow-List* vorhanden sind, können diese verwendet und erweitert werden. (Vgl. *Abschnitt 2.1.1*)
- In der *DSS Library* wurde der *C++-Exception*-Mechanismus bisher nicht verwendet. Trotzdem werde ich in meiner Erweiterung *Exceptions* verwenden und dazu eine eigene kleine *Exception*-Klassen-Hierarchie aufbauen. (Gerade bei mathematischen Operationen ist z. B. das konsequente Prüfen von Vorbedingungen für mich sehr wichtig).
- Da bei der bestehenden *DSS Library* bereits von der *RogueWave*-Bibliothek *Tools.h++* Gebrauch gemacht wird, ist es sinnvoll, diese Hilfsmittel ebenfalls zu verwenden. (Vgl. *Abschnitt 2.2.3*)

Kapitel 5

Design

In diesem Kapitel stelle ich meinen Lösungsansatz vor. In erster Linie werden die verschiedenen Klassen und ihre Beziehungen dargestellt und erläutert. Die konkrete Verwendung der Klassen und der Informationsfluss wird im nächsten Kapitel behandelt. Im Weiteren sind in *Anhang B* alle Klassen mit ihren Attributen und Methoden (inklusive Signaturen) aufgelistet.

Es ist klar, dass ich nicht gleich beim ersten Ansatz auf diese Lösung gestossen bin. Sie ist viel mehr das Ergebnis einer stetigen und iterativen Weiterentwicklung und Verbesserung. Die Informationen über meine Erfahrungen bei dieser Arbeit können im *Kapitel 7* nachgelesen werden.

5.1 Packages

Ich habe die in dieser Projektarbeit relevanten Klassen in 7 logische Packages unterteilt. Diese werden in diesem Abschnitt auf einem relativ hohen Abstraktionsniveau vorgestellt.

Für jedes Package ist ein Klassendiagramm aus *Rational Rose 98* vorhanden. Im Allgemeinen sind aus Platzgründen nur die Klassennamen aber keine Methoden und Attribute aufgeführt. In den einzelnen Diagrammen sind teilweise auch Klassen dargestellt, welche aus einem anderen Package stammen. Bei diesen Klassen ist der Packagename in der Form „(*from Packagename*)“ angegeben.

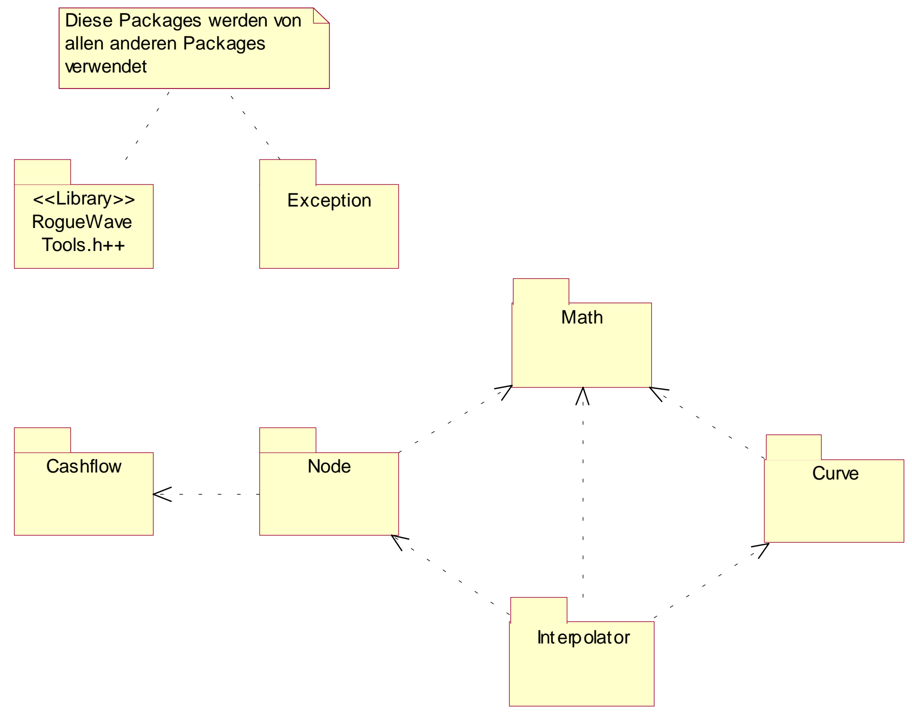

Abbildung 5-1: Package-Übersicht

5.1.1 Package Interpolator

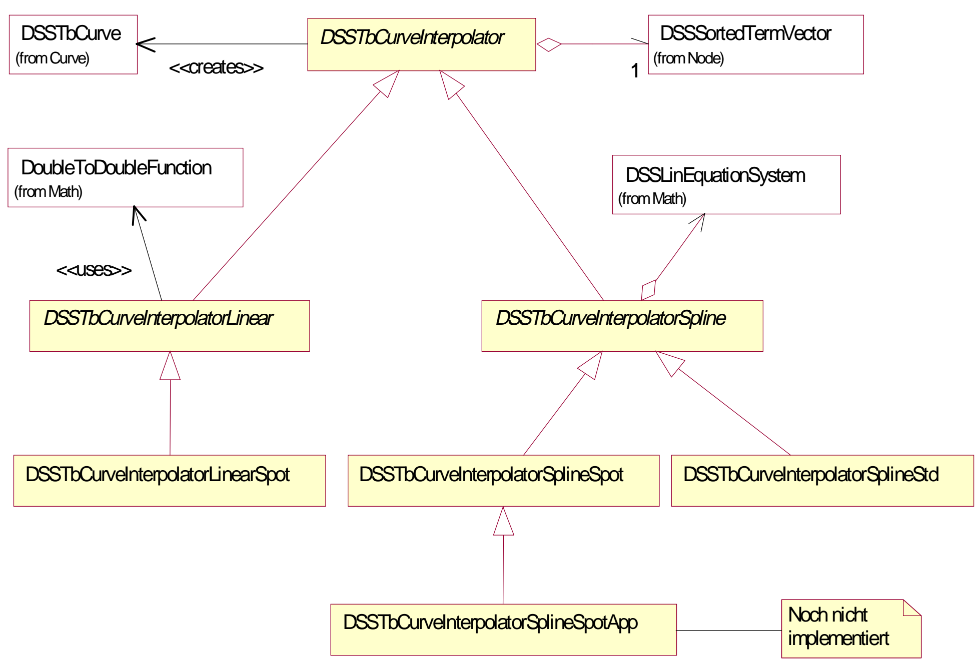

Abbildung 5-2: Package Interpolator

In diesem Package sind die verschiedenen Klassen, welche eine Kurve interpolieren können, enthalten. Wie in Abschnitt *4.2* erläutert, bilden diese Klassen eine Hierarchie mit einer abstrakten Basisklasse, die das wesentliche Interface festlegt.

Die abstrakte Basisklasse *DSSTbCurveInterpolator* verwendet ein Objekt der Klasse *DSSSortedTermVector.* Das ist ein sortierter Vektor, der die als Randbedingung verwendeten Stützpunkte enthält (vgl. *Package Node*). Am Ende wird eine Instanz von *DSSTbCurve erzeugt*, welche die berechnete (interpolierte) Kurve darstellt.

Grundsätzlich gibt es zwei Interpolationsverfahren: Lineare Interpolation und glatte Interpolation (Spline-Interpolation). Dies ist mit den beiden abstrakten *Klassen DSSTbCurveInterpolatorLinear* und *DSSTbCurveInterpolatorSpline* modelliert. Bei der linearen Interpolation wird die *Brent-Methode* beim Bootstrapping verwendet, und diese ist in der Klasse *DoubleToDoubleFunction* realisiert. Bei der Spline-Interpolation wird dagegen ein lineares Gleichungssystem (*DSSLinEquationSystem*) verwendet.

Wie in Abschnitt *4.2* erwähnt, wird bei dieser Klassenhierarchie mehrfach vom Designpattern *„Template Method“* Gebrauch gemacht.

5.1.2 Package Node

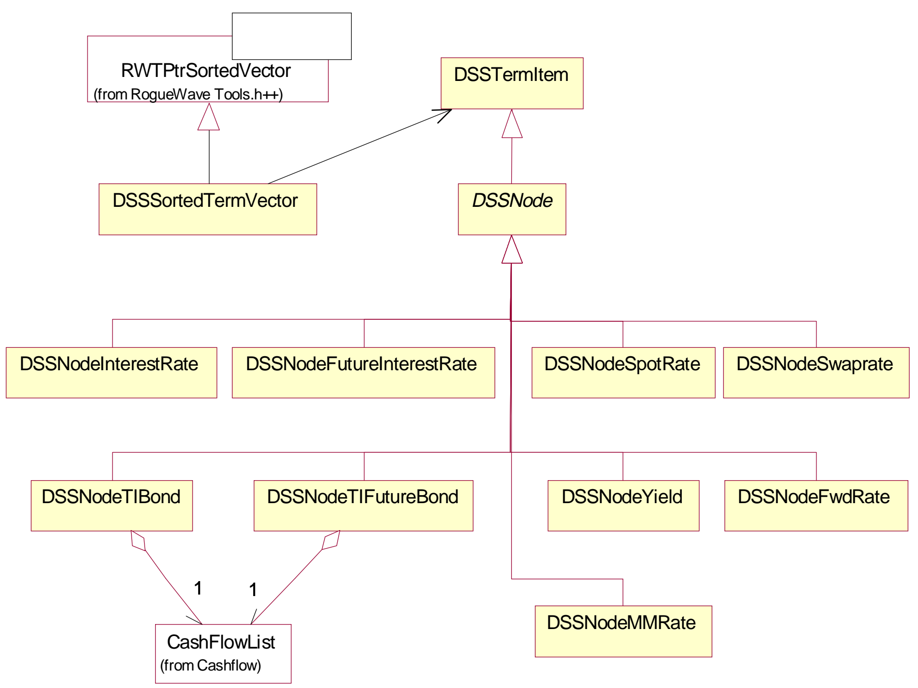

Abbildung 5-3: Package Node

Das *Package Node* enthält die verschiedenen Stützpunkttypen und die Klasse *DSSSortedTermVector*, welche mehrere Stützpunkte verwalten kann.

Die verschiedenen Stützpunkt-Typen sind alle von der abstrakten Basisklasse *DSSNode* abgeleitet, die das Interface und ein gewisses allgemeines Verhalten festlegt. Insbesondere wird auch hier wieder das Designpattern *„Template Method“* verwendet (vgl. *Abschnitt 4.2*).

Wie in der mathematischen Beschreibung (*Abschnitt 3.2.3*) erläutert, enthält jeder Stützpunkttyp mindestens eine Laufzeit (*Term*) und einen Wert (*Value*). Weil die bei der Interpolation verwendeten Stützpunkte in einer Datenstruktur nach dem Attribut *Term* sortiert

werden sollen, wurde noch die Klasse *DSSTermItem* erstellt. Diese enthält im Wesentlichen bloss einen *Term* und einen Vergleichsoperator. Die Klasse *DSSSortedTermVector* ist eine Vektor-Datenstruktur welche auf Basis eines *RogueWave*-Vektors aufgebaut ist. Sie kann beliebige Objekte vom Typ *DSSTermItem* verwalten. Da *DSSNode* von *DSSTermItem* abgeleitet ist, können somit auch Stützpunkte mit diesem Vektor verwaltet werden.

Als besonderes Merkmal enthalten die Stützpunkte *DSSNodeTIBond* und *DSSNodeTIFutureBond* eine Instanz der Klasse *CashFlowList*, welche ihrerseits mehrere *Cashflows* enthält. Diese Stützpunkte erhalten alle für die Interpolation relevanten Daten aber direkt aus *CashFlowList* und müssen nicht auf die einzelnen *Cashflows* zugreifen. (Vgl. *Package Cashflow*).

5.1.3 Package Cashflow

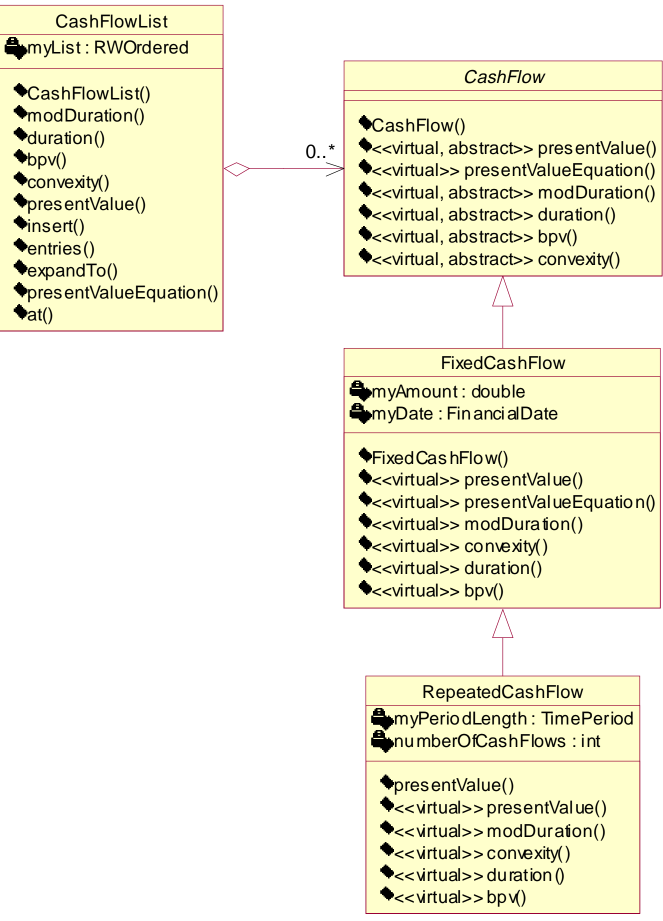

Abbildung 5-4: Package Cashflow

In diesem Package befinden sich die verschiedenen *Cashflow*-Klassen und auch eine Datenstruktur, welche *Cashflows* in einer Liste verwalten kann (*CashFlowList*).

Die *Cashflows* werden bei verschieden Stützpunkttypen verwendet, um die benötigten Informationen für die Kurveninterpolation zu speichern. Diese Stützpunkttypen enthalten dann jeweils eine Instanz der Klasse *CashFlowList* und müssen nie direkt auf die darin enthalte-

nen Cashflows zugreifen. In der *CashFlowList* sind nämlich bereits alle benötigten Methoden enthalten, welche durch die *Cashflows* iterieren und dabei gewisse Operationen ausführen.

Bemerkung: Diese Klassen waren bereits in der *DSS Library* enthalten. Im Rahmen dieses Projektes habe ich bloss ein paar neue Methoden zugefügt.

5.1.4 Package Curve

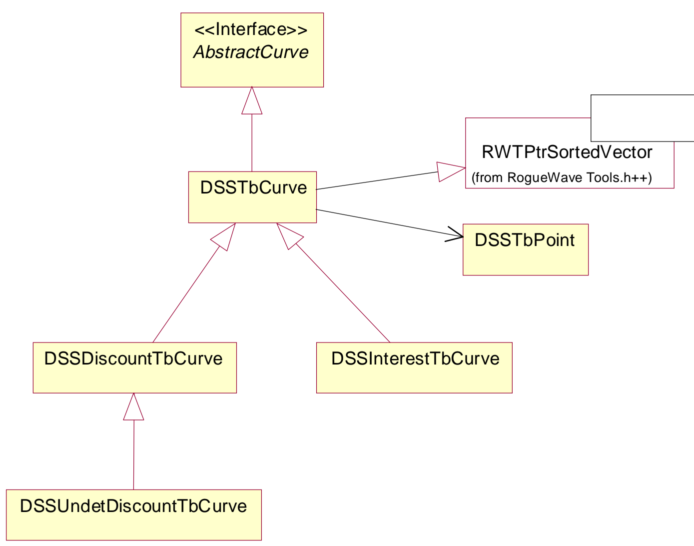

Abbildung 5-5: Package Curve

In diesem Package befinden sich alle Klassen, welche eine finanzmathematische Kurve repräsentieren. Ausserdem ist noch die Klasse *DSSTbPoint* enthalten, die ein einzelnes Spline einer Kurve darstellt.

In der Klasse *AbstractCurve* ist ein abstraktes Interface für eine Kurve definiert. Ausserdem befinden sich darin verschiedene Hilfsmethoden, die in diesem Zusammenhang praktisch sind. *DSSTbCurve* stellt eine aus Splines aufgebaute Kurve dar. Die einzelnen Splines (Klasse *DSSTbPoint*) werden im *RogueWave*-Vektor *RWTPtrSortedVector* verwaltet.

Die im obigen Absatz erwähnten Klassen waren bereits in der Bibliothek enthalten. Bei der Kurveninterpolation gibt es nun aber je nach Interpolationsart verschiedene Modelle einer Kurve. Deshalb wurden die Klassen *DSSDiscountTbCurve*, *DSSInterestTbCurve* und *DSSUndetDiscountTbCurve* zugefügt, welche diese Unterschiede berücksichtigen. Bei der glatten Interpolation (Spline-Interpolation) wird eine Kurve vom Typ *DSSUndetDiscountTbCurve* verwendet, während bei der linearen Interpolation ein Objekt der Klasse *DSSInterestTbCurve* erstellt wird.

5.1.5 Package Math

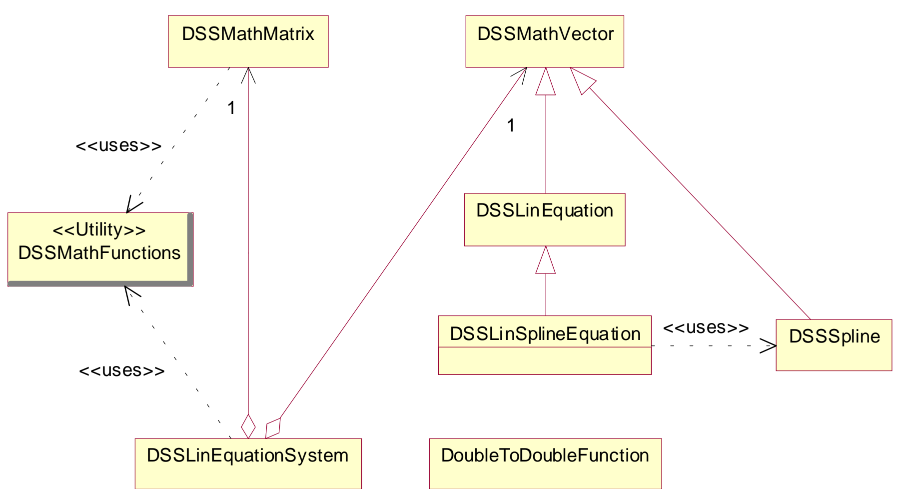

Abbildung 5-6: Package Math

Wie der Name schon sagt, sind in diesem Package alle verwendeten mathematischen Klassen und Utilities enthalten.

Die Klasse *DoubleToDoubleFunction* war schon früher in *FORUMsystems* enthalten und ist für die *Brent-Methode* zur Nullstellensuche verantwortlich.

Die anderen Klassen implementieren die nötigen Operationen aus der linearen Algebra. *DSSMathVector* und *DSSMathMatrix* repräsentieren beliebig grosse mathematische Vektoren bzw. Matrizen mit allen nötigen Operationen. Insbesondere ist bei der Matrix-Klasse auch ein Verfahren zur Matrix-Inversion mittels *LU-Dekomposition* enthalten. Dazu verwendet diese Klasse das Utility *DSSMathFunctions*, welches am Ende dieses Abschnitts vorgestellt wird.

Die Klasse *DSSSpline* ist von der Vektor-Klasse abgeleitet und stellt die Koeffizienten eines Splines dar. (Bei der Interpolation müssen ja gerade die Spline-Koeffizienten der Kurvenstücke berechnet werden). *DSSLinEquation* ist ebenfalls von der Vektor-Klasse abgeleitet und stellt eine lineare Gleichung dar. Auch *DSSLinSplineEquation* stellt eine lineare Gleichung dar. Hier kann aber zusätzlich auf mehrere benachbarte Koeffizienten, die als Splines gruppiert sind, zugegriffen werden.

Die Klasse *DSSLinEquationSystem* repräsentiert ein lineares Gleichungssystem und verwendet dazu je eine Instanz von *DSSMathVector* und *DSSMathMatrix*. Für die Lösung dieses Gleichungssystem mittels *LU-Dekomposition* wird ebenfalls das Utility *DSSMathFunctions* verwendet. Darin befinden sich verschiedene mathematische Algorithmen, welche in der linearen Algebra verwendet werden. Diese wurden dem Buch *„Numerical Recipes in C“* entnommen und direkt als *C*-Funktionen implementiert.

5.1.6 Package Exception

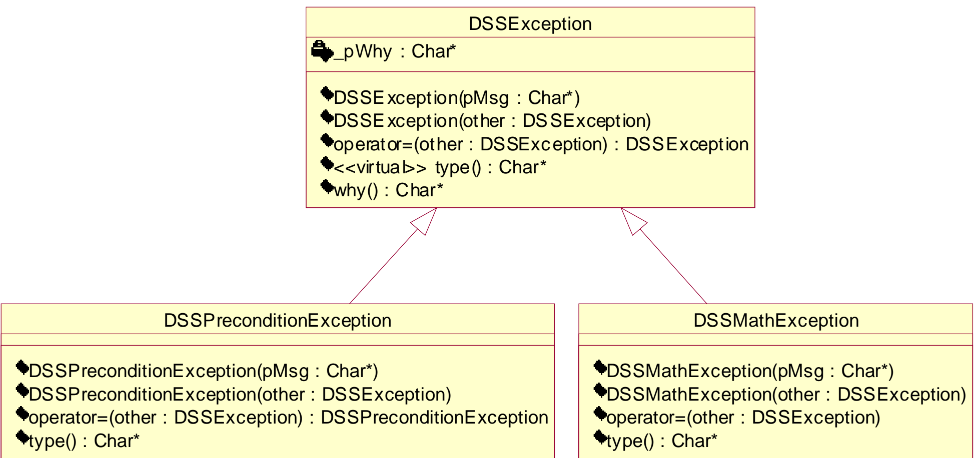

Abbildung 5-7: Package Exception

Dieses Package enthält Klassen, die für das *Exception-Handling* verwendet werden. *DSSException* wird bei einer allgemeine Ausnahmesituation geworfen und die beiden Erben *DSSPreconditionException und DSSMathException* werden bei einer Verletzung der Vorbedingung bzw. bei einem mathematischen Fehler verwendet.

5.1.7 Package RogueWave Tools.h++

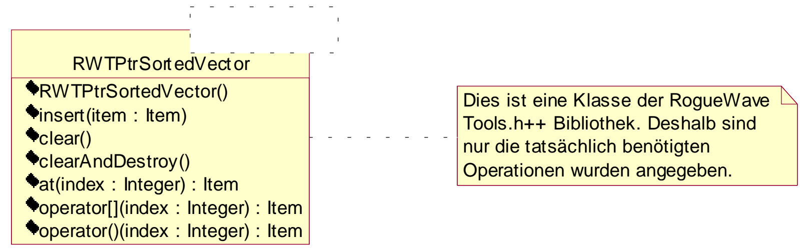

Abbildung 5-8: Package RogueWave Tools.h++

In diesem Package befindet sich die einzige in meinem Projekt verwendete Klasse der *RogueWave*-Bibliothek *Tools.h++*. Es ist die Klasse *RWTPtrSortedVector*, und sie stellt einen sortierten Vektor zur Verfügung. Alle in einem solchen Vektor enthaltenen Objekte müssen mindestens die Operatoren „gleich“ und „kleiner“ implementiert haben.

Kapitel 6
Ablauf einer Interpolation

In diesem Kapitel werden exemplarisch zwei verschiedene Interpolationsabläufe mit Sequence Diagrammen dokumentiert. Dadurch sollten auch die Aufgaben und Eigenschaften der wichtigsten Klassen und Methoden klar werden. Für weitere Angaben zu den einzelnen Klassen mit ihren Attributen und Methoden (inklusive Signaturen) sei hier auf *Anhang B* verwiesen.

Der Interpolationsvorgang wird hier für die folgende beiden Fälle dokumentiert:

A) Lineare Spot-Rate-Interpolation (*InterpolationLinearSpot*)
B) Glatte Spot-Rate-Interpolation (*InterpolationSplineSpot*)

Wie in den vorangehenden Kapiteln erwähnt, wurde bei der Implementation der verschiedenen Interpolationsverfahren sehr oft das Designpattern *„Template Method“* verwendet. Deshalb sind die beiden hier dargestellten Interpolationsbeispiele vom Aufbau her sehr ähnlich und konnten beide in 7 Diagramme eingeteilt werden. Diese Diagramme erklären die Teile des Interpolationsvorgangs auf verschiedenen Detailstufen. In der Regel wird bei einem ersten Diagramm ein gewisser Vorgang auf einem hohen Abstraktionsniveau dargestellt, und bei den folgenden Diagrammen werden dann die einzelnen Details berücksichtigt.

Im Folgenden werden die 7 verschiedenen Phasen für beide Fälle dargestellt und erklärt. Dabei wird in jedem Abschnitt zuerst der Fall A) und dann der Fall B) betrachtet. Da die Diagramme 1, 2 und 3 für beide Fälle identisch sind, erübrigt sich hier diese Fallunterscheidung.

Zusätzlich zu den Sequence Diagrammen ist jeweils noch angegeben, welches Package bzw. welche Klassen für die Steuerung dieses Ablaufs verantwortlich sind. Darüber hinaus werden die Diagramme auch kurz erklärt.

6.1 Diagramm 1: Create NodeVector

Package: *Node*

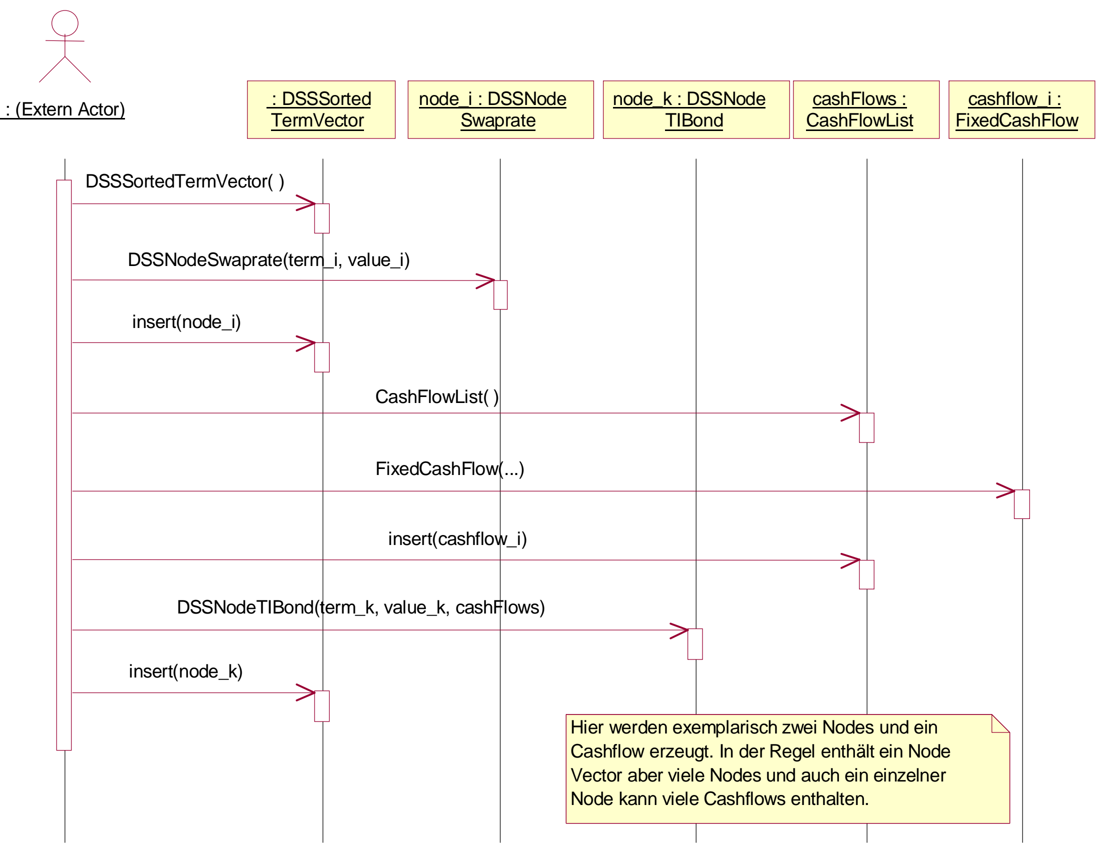

Abbildung 6-1: Sequence Diagramm 1 - Create NodeVector

Hier wird dargestellt, wie der externe Benutzer (z.B. eine Routine im *OpenRoad*-Interface) einen Vektor mit den Stützpunkten (*Nodes*) erzeugt. Wie im nächsten Abschnitt gezeigt, muss ein solcher Vektor der Interpolationsklasse als Argument übergeben werden.

6.2 Diagramm 2: Interpolate Curve

Package: *Interpolator*

Klasse: *DSSTbCurveInterpolator (abstract)*

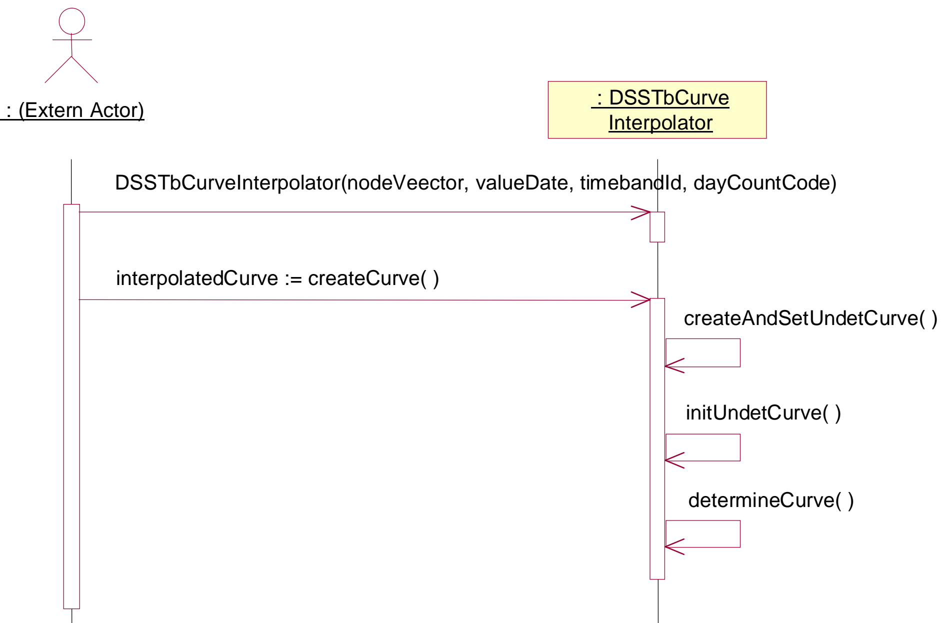

Abbildung 6-2: Sequence Diagramm 2 - Interpolate Curve

Hier wird der gesamte Interpolationsvorgang auf einem sehr hohen Abstraktionsniveau dargestellt. Da auf dieser Ebene alle Interpolationsverfahren noch gleich aussehen, konnte dieser Vorgang in Anlehnung an das „Template Method“-Designpattern bereits in der abstrakten Klasse *DSSTbCurveInterpolator* implementiert werden. In den folgenden Abschnitten werden die einzelnen Schritte detaillierter erläutert.

6.3 Diagramm 3: Init UndetCurve

Package: *Interpolator*

Klasse: *DSSTbCurveInterpolator (abstract)*

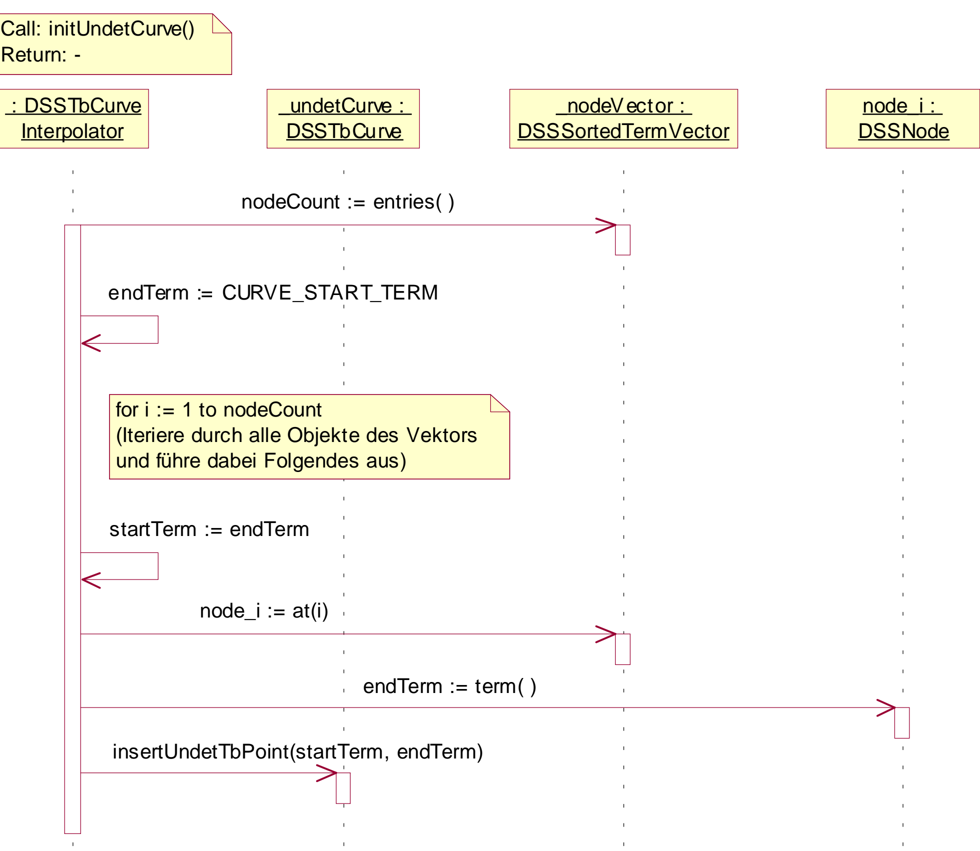

Abbildung 6-3: Sequence Diagramm 3 - Init UndetCurve

In diesem Diagramm wird gezeigt, wie die zunächst noch unbestimmte Kurve initialisiert wird. Im Wesentlichen wird einfach durch die Stützpunkte (*DSSNode*) iteriert und dabei für je zwei benachbarte *Nodes* ein neues Spline (*DSSTbPoint*) in die Kurve eingefügt. Die Koeffizienten dieses Splines sind zunächst noch unbestimmt, und es wird bloss der Anfang (*startTerm*) und das Ende (*endTerm*) des Definitionsintervalls eingetragen.

6.4 Diagramm 4: Determine Curve

Package: *Interpolator*

6.4.1 Diagramm 4A: Determine LinearCurve

Klasse: *DSSTbCurveInterpolatorLinear (abstract)*

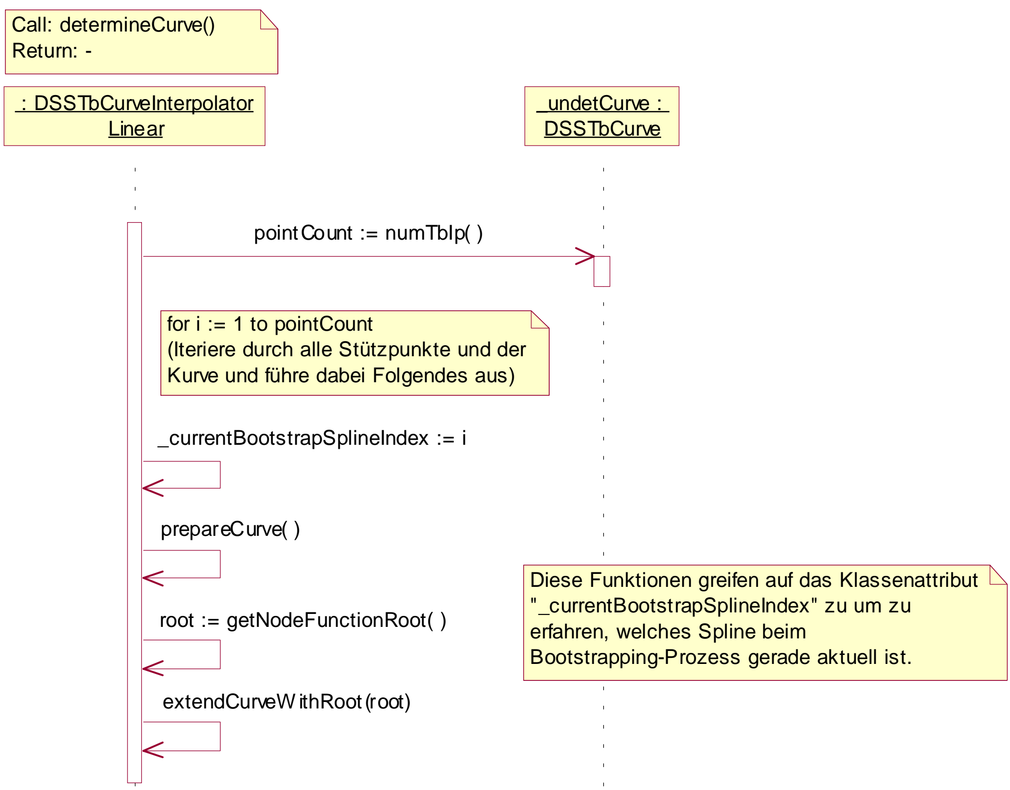

Abbildung 6-4: Sequence Diagramm 4A - Determine LinearCurve

Mit diesem Sequence Diagramm soll die Bestimmung der Kurve im linearen Fall dargestellt werden. Wie in Abschnitt *3.2.2* erklärt, basiert diese Bestimmung auf einem *Bootstrapping*-Verfahren. Es wird durch alle Splines der Kurve iteriert und dabei jeweils die Steigung berechnet. (Damit ist das Spline bestimmt, da die Kurve stückweise affin und stetig ist). Mit der Methode *prepareCurve()* wird das aktuelle Spline vorbereitet und mit *extendCurveWithRoot(root)* wird die gefundene Steigung in das Spline eingetragen. Die Methode *getNodeFunctionRoot()* bestimmt die vom momentanen Stützpunkt abhängige Steigung und wird in Abschnitt *6.5.1* erklärt.

6.4.2 Diagramm 4B: Determine SplineCurve

Klasse: *DSSTbCurveInterpolatorSpline (abstract)*

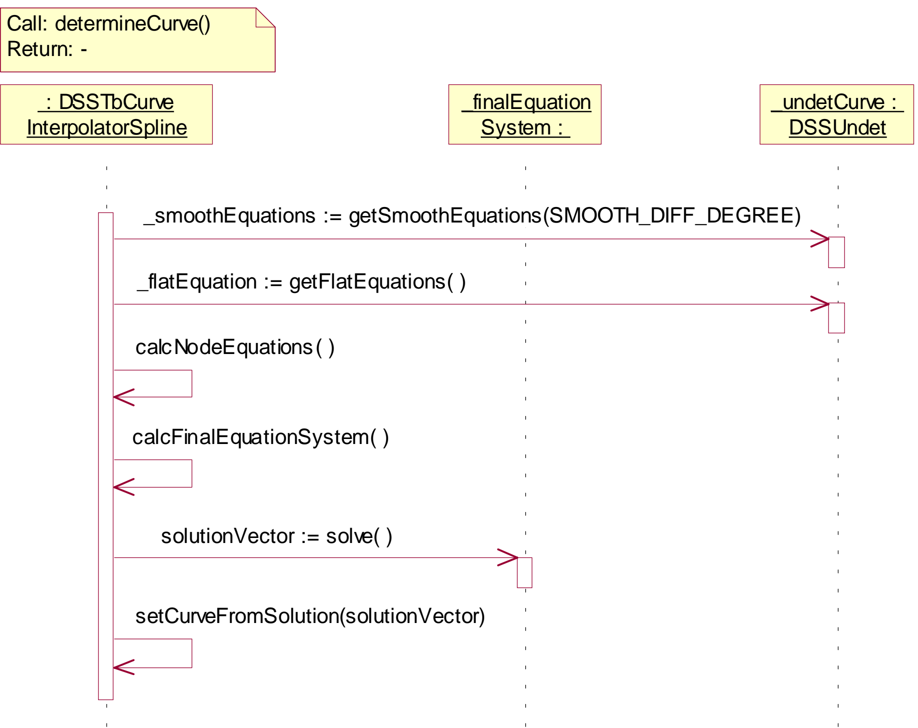

Abbildung 6-5: Sequence Diagramm 4B - Determine SplineCurve

Hier wird die Bestimmung der Kurve im glatten Fall dargestellt. Wie in Abschnitt *3.2.2* erläutert werden dazu ein Gleichungssystem für die Glattheit (*_smoothEquations*), ein Gleichungssystem für die minimale globale Krümmung (*_flatEquations*) und ein Gleichungssystem für die geeignete Approximation der Stützpunkte erzeugt. Der letzte Schritt ist von den auftretenden *Node*-Typen abhängig und wird in den folgenden Abschnitten dargestellt.

Am Ende werden diese 3 Gleichungssysteme mittels *calcFinalEquationSystem()* kombiniert, und dann wird das endgültige System gelöst. Die Methode *setCurveFromSolution(solution)* trägt die Lösungen dann in die Kurve bzw. in die Splines der Kurve ein.

6.5 Diagramm 5: Get NodeConditions

Package: *Interpolator*

6.5.1 Diagramm 5A: Get NodeFunctionRoot

Klasse: *DSSTbCurveInterpolatorLinear (abstract)*

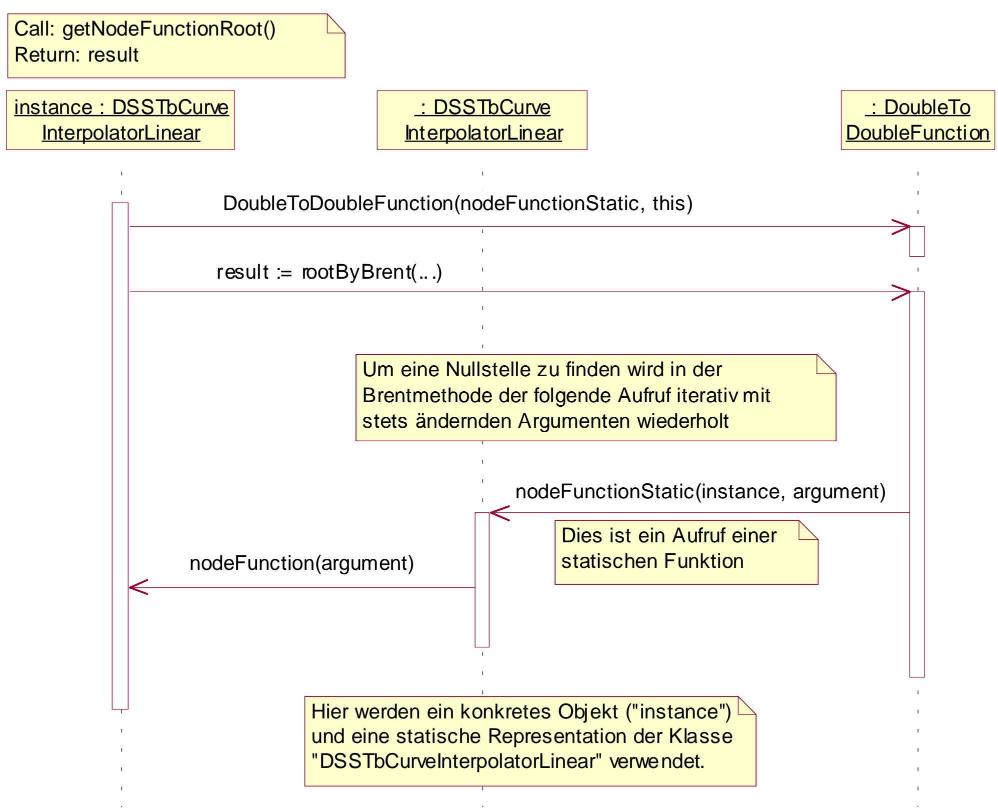

Abbildung 6-6: Sequence Diagramm 5A - Get NodeFunctionRoot

Hier wird dargestellt, wie beim *Bootstrapping* die Steigung für den nächsten Kurvenabschnitt bestimmt wird. Beim Aufruf des Konstruktors der Klasse *DoubleToDoubleFunction* wird zuerst ein Pointer auf die statische Funktion *nodeFunctionStatic()* und ein Pointer auf die aktuelle Interpolatorinstanz übergeben. Dann wird die Methode *rootByBrent()* des neu erstellten Objekts aufgerufen. Diese versucht, eine Nullstelle der beim Konstruktoraufruf registrierten statischen Funktion *nodeFunctionStatic(instance, argument)* zu finden und gibt diese zurück. (Der Parameter *argument* wird dabei gemäss der *Brent-Methode* variiert).

Die statische Funktion *nodeFunctionStatic()* ruft hier bloss die nicht-statische (!) Funktion *nodeFunction()* auf und gibt deren Rückgabewert zurück. Dies ist deshalb möglich, weil beim Aufruf der statischen Methode ein Pointer (*instance*) auf die zu verwendende Interpolatorinstanz übergeben wird.

Die Funktion *nodeFunction(argument)* wird in Abschnitt *6.6.1* genauer behandelt.

6.5.2 Diagramm 5B: Calc NodeEquations

Klasse: *DSSTbCurveInterpolatorSpline (abstract)*

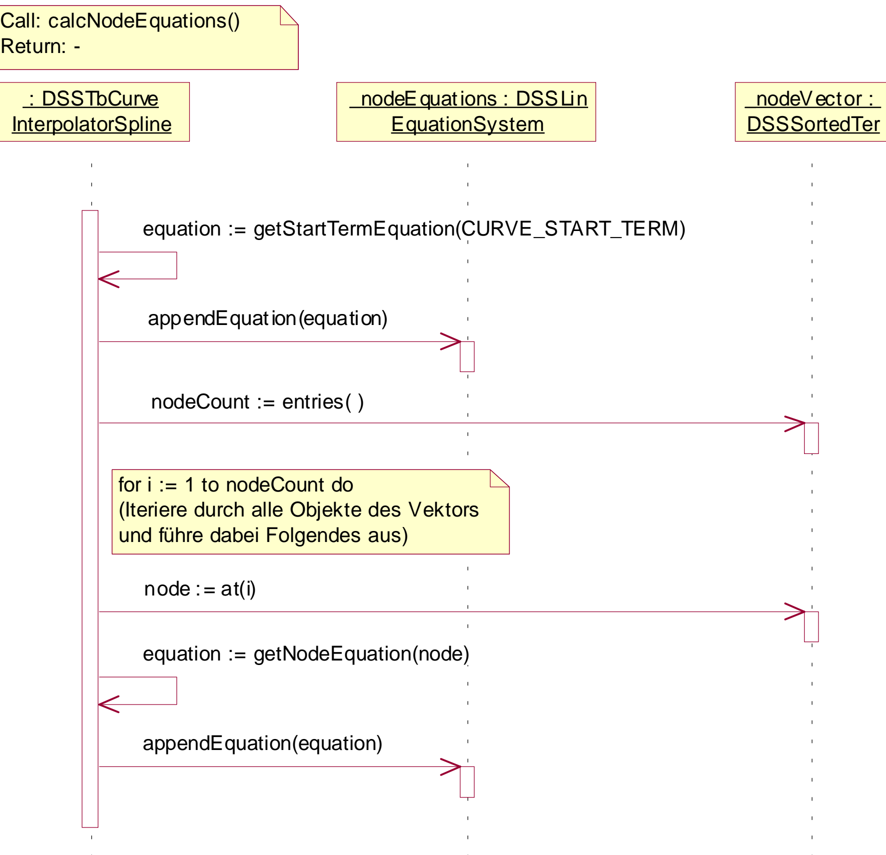

Abbildung 6-7: Sequence Diagramm 5B - Calc NodeEquations

Die Stützpunktgleichungen werden wie folgt berechnet: Für den „künstlichen Stützpunkt“ an der Stelle *CURVE_START_TERM* (in der Regel 0) wird eine spezielle Gleichung mittels *getStartTermEquation(CURVE_START_TERM)* berechnet und an das Gleichungssystem angefügt. Danach wird durch alle Stützpunkte iteriert, und bei jedem Stützpunkt *node* wird eine spezielle Gleichung mit der Methode *getNodeEquation(node)* berechnet und dem Gleichungssystem zugefügt. Dieser Vorgang wird in Abschnitt *6.6.2* etwas detaillierter dargestellt.

6.6 Diagramm 6: Get NodeConditions

Package: *Interpolator*

6.6.1 Diagramm 6A: Evaluate NodeFunction

Klasse: *DSSTbCurveInterpolatorLinearSpot*

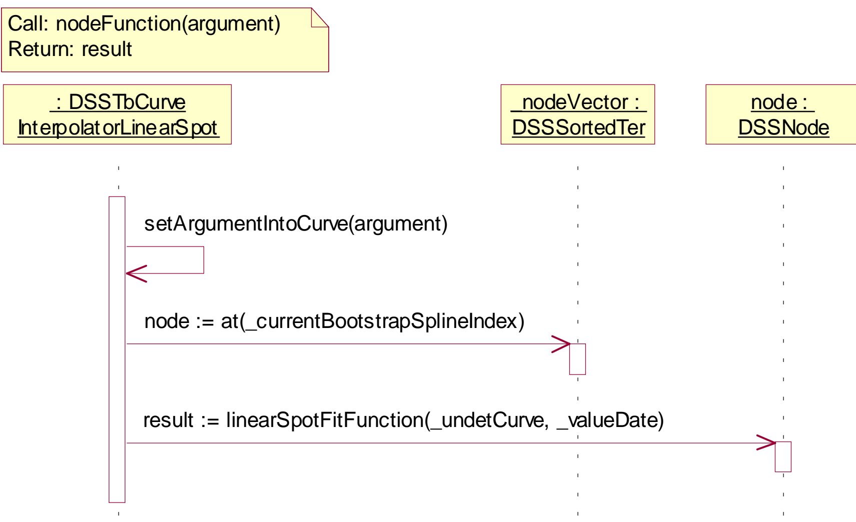

Abbildung 6-8: Sequence Diagramm 6A - Evaluate NodeFunction

Hier wird die Methode *nodeFunction(argument)* dargestellt. Zuerst wird der übergebene Parameter *argument* mittels *setArguemntIntoCurve(arguement)* an geeigneter Stelle in die Kurve eingefügt. Dann wird die Funktion *linearSpotFitFunction(_undetCurve, _valueDate)* des beim *Bootstrapping*-Prozess gerade akuellen *Nodes* (Index *_currentBootstrapSplineIndex*) aufgerufen. Diese vom *Node*-Typ abhängige Funktion liefert genau dann Null, wenn der vorher in der Kurve eingetragene Wert (*argument*) gleich der idealen Steigung für das betrachtete Kurvenstück ist. Ein Ablaufdiagramm dieser Funktion ist in Abschnitt *6.7.1* vorhanden.

Die Parameter *_undetCurve* (die unbestimmte Kurve) und *_valueDate* (das Valutadatum) sind Attribute der Klasse *DSSTbCurveInterpolator* und wurden zu Beginn generiert bzw. bereits beim Konstruktoraufruf übergeben.

6.6.2 Diagramm 6B: Get NodeEquation

Klasse: *DSSTbCurveInterpolatorSplineSpot*

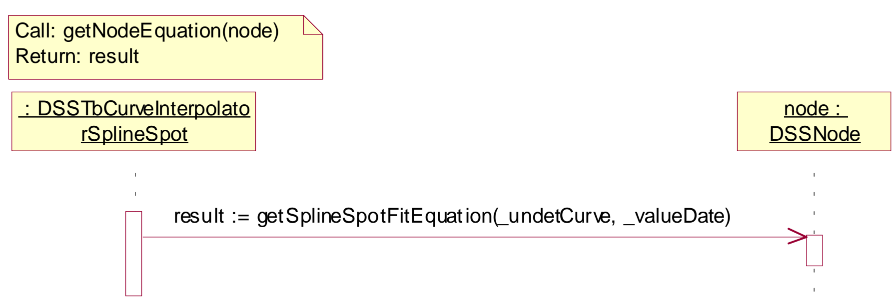

Abbildung 6-9: Sequence Diagramm 6B - Get NodeEquation

Bei der Berechnung der Gleichung für die Approximation des Stützpunktes *node* wird nur die Funktion *getSplineSpotFitEquation(_undetCurve, _valueDate)* des Objekts *node* aufgerufen (vgl. *Abschnitt 6.7.2*).

Die Parameter *_undetCurve* (die unbestimmte Kurve) und *_valueDate* (das Valutadatum) sind Attribute der Klasse *DSSTbCurveInterpolator* und wurden zu Beginn generiert bzw. bereits beim Konstruktoraufruf übergeben.

6.7 Diagramm 7: Get NodeConditions

Package: *Node*

Bei den Folgenden Diagrammen werden Funktionen dargestellt, welche bereits in der abstrakten Klasse *DSSNode* deklariert sind. Um etwas mehr Details zeigen zu können wird hier aber die konkrete Implementation der Klasse *DSSNodeTIBond* dargestellt.

6.7.1 Diagramm 7A: Evaluate LinearSpotFitFunction

Klasse: *DSSNode* (bzw. *DSSNodeTIBond*)

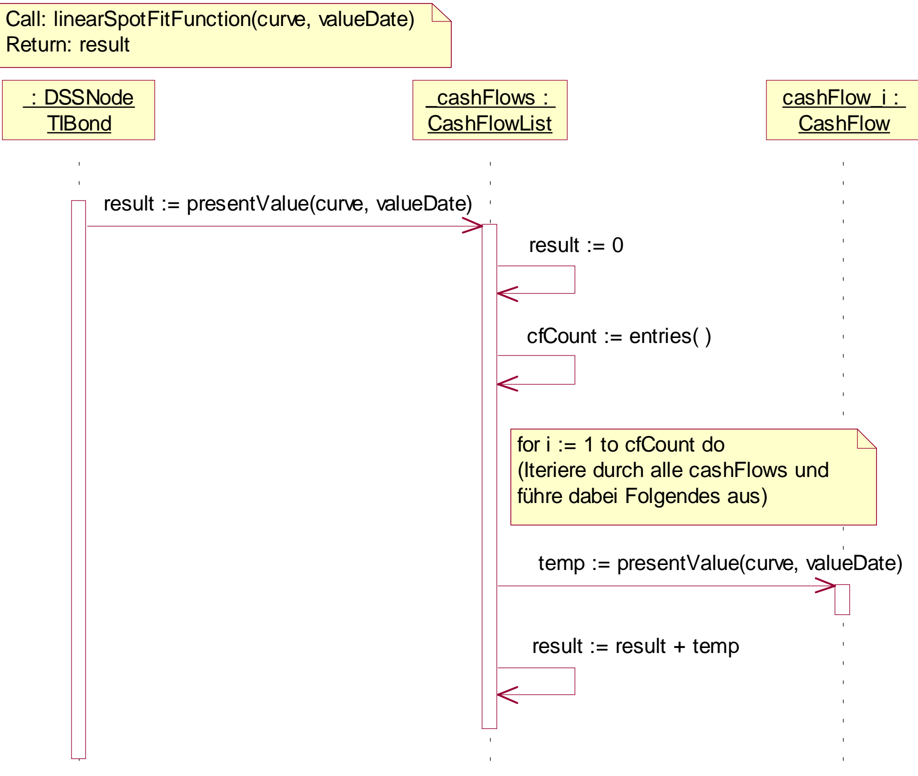

Abbildung 6-10: Sequence Diagramm 7A - Evaluate LinearSpotFitFunction

Dieses Diagramm stellt die Berechnung der „Steigungs-Funktion“ im Falle des Stützpunkts *DSSNodeTIBond* dar. Dabei wird unter anderem die Funktion *presentValue(curve, valueDate)* der angehängten *Cashflow-List* aufgerufen. Diese Funktion war schon vorher in der *DSS Library* vorhanden!

6.7.2 Diagramm 7B: Get SplineSpotFitEquation

Klasse: *DSSNode* (bzw. *DSSNodeTIBond*)

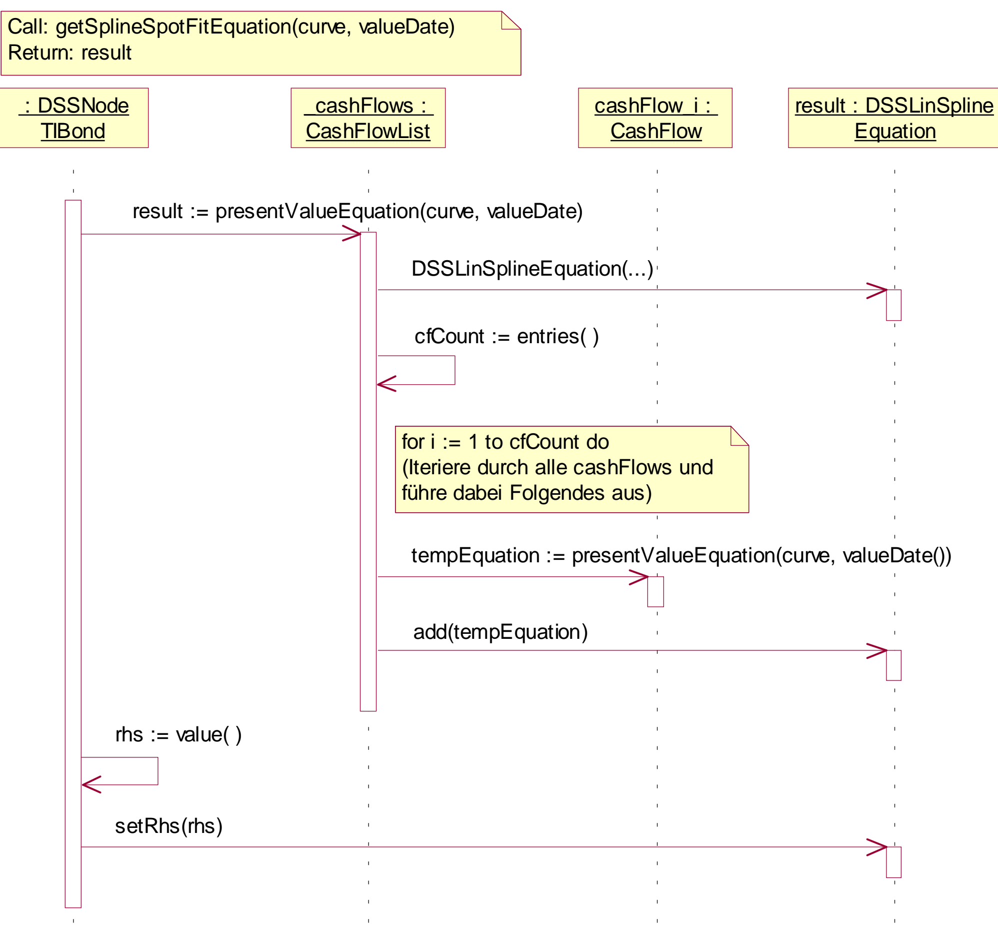

Abbildung 6-11: Sequence Diagramm 7B - Get SplineSpotFitEquation

Hier wird die Gleichung für eine geeignete Approximation des Stützpunktes *DSSNodeTIBond* berechnet. Dabei wird die Funktion *presentValueEquation(curve, valueDate)* der angehängten *Cashflow-List* aufgerufen. Diese Funktion wurde im Rahmen dieses Projekts in die Klasse *CashFlow* integriert und ist das Analogon zur Funktion *presentValue(curve, valueDate)* aus dem vorangehenden Abschnitt. Anstatt mit skalaren Werten wird hier aber mit Vektoren operiert.

Allgemein kann man sagen, dass die Funktion *getSplineSpotFitEquation()* und *linearSpotFitFunction()* absolut analog aufgebaut sind. Der Unterschied besteht nur darin, dass bei der ersten Funktion mit skalaren Werten und bei der zweiten mit Vektoren gerechnet wird.

Kapitel 7

Projektablauf und Erfahrungen

Bisher wurde eigentlich nur der Stand vor und nach meiner Projektarbeit gezeigt. In diesem Kapitel werden nun auch der Projektablauf und die dabei gewonnenen Erfahrungen berücksichtigt.

7.1 Projektablauf

Ich habe diese Projektarbeit im März 1999 begonnen und habe insgesamt mehr als 200 Stunden dafür aufgewendet. Alle Arbeiten habe ich bei der Firma *Sherpa'x AG* in Solothurn ausgeführt. Hier nun einen Überblick über den (zeitlichen) Ablauf der ausgeführten Tätigkeiten.

Es ist wichtig zu bemerken, dass dieses Projekt natürlich nicht so linear verlaufen ist, wie das nachfolgend geschildert wird. (Zwischen den angeführten Schritten wurde vielmehr vor- und wieder zurückgesprungen). So war es manchmal nötig, das Wissen in einem bestimmten Gebiet zu vertiefen oder eine bereits gefällte Entscheidung zu modifizieren. Ausserdem war ich während des gesamten Ablaufs mit meinen beiden Betreuern Dr. Beat Scherer (Mathematik) und Dr. Xavier Fábregas (Informatik) oder anderen Mitarbeiter der Firma *Sherpa'x AG* in Kontakt.

7.1.1 Einführung und Einarbeitung

- Allgemeine Einführung in das geplante Projekt.
- Einführung in die mathematischen Abläufe der finanzmathematischen Bibliothek (*DSS Library*) von *FORUMsystems*.
- Einführung in die bestehenden Kurveninterpolations-Algorithmen von *Mathematica*.
- Einführung in die Arbeitsumgebung und die verwendete Hard- und Software (Compiler, Debugger, *Emake-Tools*, *RCS*, usw.).
- Einarbeitung in *Mathematica* anhand eines Buches.
- Einarbeitung in die Interpolationsverfahren und Analyse der *Mathematica*-Algorithmen.
- Einarbeitung in *DSS Library*.

7.1.2 Analyse und Design

- Überlegungen, wie man die bestehenden Algorithmen möglichst sauber, flexibel und homogen in die bestehende Bibliothek integrieren kann.
- Erarbeitung der Anaylsedokumente. Im Zentrum steht dabei ein Modell, bei dem die geplanten Klassen mit ihren Aufgaben auf einem hohen Abstraktionsniveau dargestellt sind. Bei den einzelnen Klassen sind bloss die wichtigsten Methoden (ohne detaillierte Signaturen) aufgeführt.
  Diese Dokumente wurden mit meinen Betreuern besprochen und mehrfach modifiziert.
- Implementation kleiner Prototypen um verschiedene Vorgänge zu simulieren und besser zu verstehen.
- Erarbeitung der Designdokumente. Dazu gehören vor allem Klassendiagramme, *CRC*-Karten und Sequence Diagramme, die mit *Rational Rose 98* erstellt wurden.
  Diese Dokumente wurden ebenfalls mit meinen Betreuern besprochen und mehrfach modifiziert.
- Offizieller Review, wo meine Analyse- und Designdokumente von zwei unbeteiligten Mitarbeitern begutachtet wurden. Anschliessend Review-Meeting und Besprechung der Resultate.

7.1.3 Implementation

Während der Implementationsphase habe ich darauf geachtet, alle Module möglichst früh zu testen und ständig eine lauffähige Test-Applikation zu pflegen. Selbstverständlich habe ich während dieser Phase auch noch kleinere Änderungen am Design vorgenommen. Dies waren vor allem *C++*-spezifische Modifikationen.

- Einarbeitung in die bei *Sherpa'x* verwendeten Entwicklungstools. Dabei war vor allem die Sourcecode-Verwaltung mit verschiedenen Workspaces und Generierungsskripten (*Makefiles*, usw.) sehr interessant.
- Implementation und Test der Packages *Exception* und *Math*.
- Implementation und Test der anderen Packages. Dabei mussten auch bestehende Klassen modifiziert und erweitert werden.
- Testen der neuen und erweiterten Klassen in möglichst realistischen Situationen, aber ohne Integration in *FORUMsystems*. Vergleich der Resultate mit den originalen *Mathematica*-Berechnungen und Beseitigung der Fehler.
- Anpassung der Analyse- und Designdokumente an die Implementation.

7.1.4 Integration und Test

- Integration der neuen Bibliotheksmodule in *FORUMsystems*. Dazu musste vor allem das C++-Openroad-Interface angepasst werden (vgl. *Abschnitt 3.1.2*).
- Testen der Erweiterungen direkt in der *FORUM-Applikation.* Dazu wurden die verschiedenen Kombinationen von Interpolationsarten und Stützpunkttypen im neuen System mit einer *FORUM-Applikation* auf *Mathematica*-Basis verglichen. (Dies entspricht der Idee des „Path Testing“ auf einem höheren Abstraktionsniveau). Die ausgeführten Tests wurden mit Screenshots dokumentiert.
- Korrektur der aufgetauchten Fehler und erneutes Testen.

7.2 Erfahrungen

Bei meiner Projektarbeit habe ich viele Erfahrungen in ganz verschiedenen Bereichen sammeln können. Das Wichtigste ist in diesem Abschnitt aufgeführt.

7.2.1 Mitarbeit bei einem grossen Softwareprojekt

Bisher habe ich meistens für mich alleine oder in kleinen Gruppen (maximal 5 Personen) an eher bescheidenen Projekten gearbeitet. Deshalb war es eine ganz neue Erfahrung in einem so grossen Softwareprojekt mitzuarbeiten. Folgende Punkte waren besonders interessant:

- Einarbeitung in ein grosses Projekt, bei dem man nicht von Anfang an dabei war. Dabei ist es wichtig das richtige Mass zu finden, denn man muss gewisse Teile ziemlich genau verstehen und andere braucht man nur ganz oberflächlich zu kennen.
- Änderung und Erweiterung an Bibliotheken einer bereits funktionierenden Applikation ohne die bestehenden Komponenten zu beeinträchtigen.
- Mit anderen Mitarbeitern über Ideen, Risiken und Lösungsalternativen zu diskutieren. Es war vor allem interessant, die Kenntnisse der Mitarbeiter mit langjähriger Erfahrung in der professionellen Softwareentwicklung mit meinem Wissen von der Universität zu vergleichen.

7.2.2 Arbeit mit verschiedenen Softwaretools

Sehr interessant und lehrreich war für mich auch die Arbeit mit verschiedenen Softwaretools. Ich habe zwar die meisten Tools bereits gekannt, habe aber meistens nur wenig Erfahrung in der alltäglichen Benutzung dieser Werkzeuge gehabt.

- Zum ersten mal habe ich für den Analyse- und Designprozess ein professionelles Softwaretool, *Rational Rose 98 Professional C++-Edition,* verwendet. Dabei habe ich die zahlreichen Vorteile gegenüber dem Verfahren mit Papier und Bleistift schnell schätzen gelernt. Allerdings muss ich sagen, dass auch gewisse Nachteile, vor allem in der Flexibilität, vorhanden sind.
- Wenn so viele Leute an einer Applikation arbeiten, muss sehr viel Wert auf eine saubere Codeverwaltung gelegt werden. Bei *Sherpa'x AG* wird der Sourcecode mit *RCS* und den *Emake-Tools* verwaltet, und noch nicht definitiv getestete Teile der Applikation werden in verschiedenen Workspaces kompiliert. Ausserdem werden Generierungsskripte für die Kompilierung und Installation der Applikation auf den verschiedenen Plattformen verwendet.
- Es war lehrreich, aber oftmals auch ziemlich nervenaufreibend, die Eigenheiten der verschiedenen *C++*-Compiler auf den Plattformen *SUN Solaris*, *IBM AIX* und *Microsoft Windows* kennenzulernen. Meistens war nämlich Abstimmungsarbeit erforderlich, damit derselbe Sourcecode auch auf allen Plattformen erfolgreich kompiliert werden konnte.

7.2.3 Erfahrungen aus dem Entwicklungsprozess

Die eigentliche Entwicklung der Bibliothekserweiterungen war für mich ebenfalls eine sehr interessante Komponente meines Informatikprojekts. Ich habe während des gesamten Entwicklungsprozesses versucht, die theoretischen Kenntnisse aus dem Informatikstudium möglichst gut in die Praxis umzusetzen. Viele Dinge konnte ich dabei relativ problemlos übernehmen, während es bei anderen zu gewissen Problemen kam. In diesen Fällen war es immer eine grosse Herausforderung, aus den theoretischen Kenntnissen und den konkreten praktischen Gegebenheiten eine geeignete Lösung zu finden.

7.3 Ausblick

Mittlerweile sind die in diesem Projekt vorgestellten Bibliothekserweiterungen und deren Integration in *FORUMsystems* abgeschlossen. Bei recht umfangreichen Tests und Vergleichen mit der ursprünglichen Applikation konnte festgestellt werden, dass die auf den neuen Modulen basierende Applikation ihre Aufgabe erfüllt. (Zumindest waren alle ausgeführten Testfälle erfolgreich). Anfang August 1999 wird ein neuer Release von *FORUMsystems* erstellt, bei welchem diese Erweiterungen bereits integriert sein werden.

7.4 Danksagungen

An dieser Stelle möchte ich mich ganz herzlich bei den verschiedenen Personen bedanken, welche dieses Informatikprojekt ermöglicht haben. Das sind vor allem Prof. Dr. Oscar Nierstrasz von der Universität Bern sowie Dr. Beat Scherer und Dr. Xavier Fábregas von *Sherpa'x AG*.

Anhang A

Screenshots

In diesem Kapitel sind verschiedene Screenshots direkt aus der Applikation *FORUMsystems* abgebildet.

A.1 Kurve mit Spot-Rate-Stützpunkten

Bei den folgenden Abbildungen wurden Spot-Rate-Stützpunkte festgelegt und durch eine glatte bzw. lineare Kurve interpoliert.

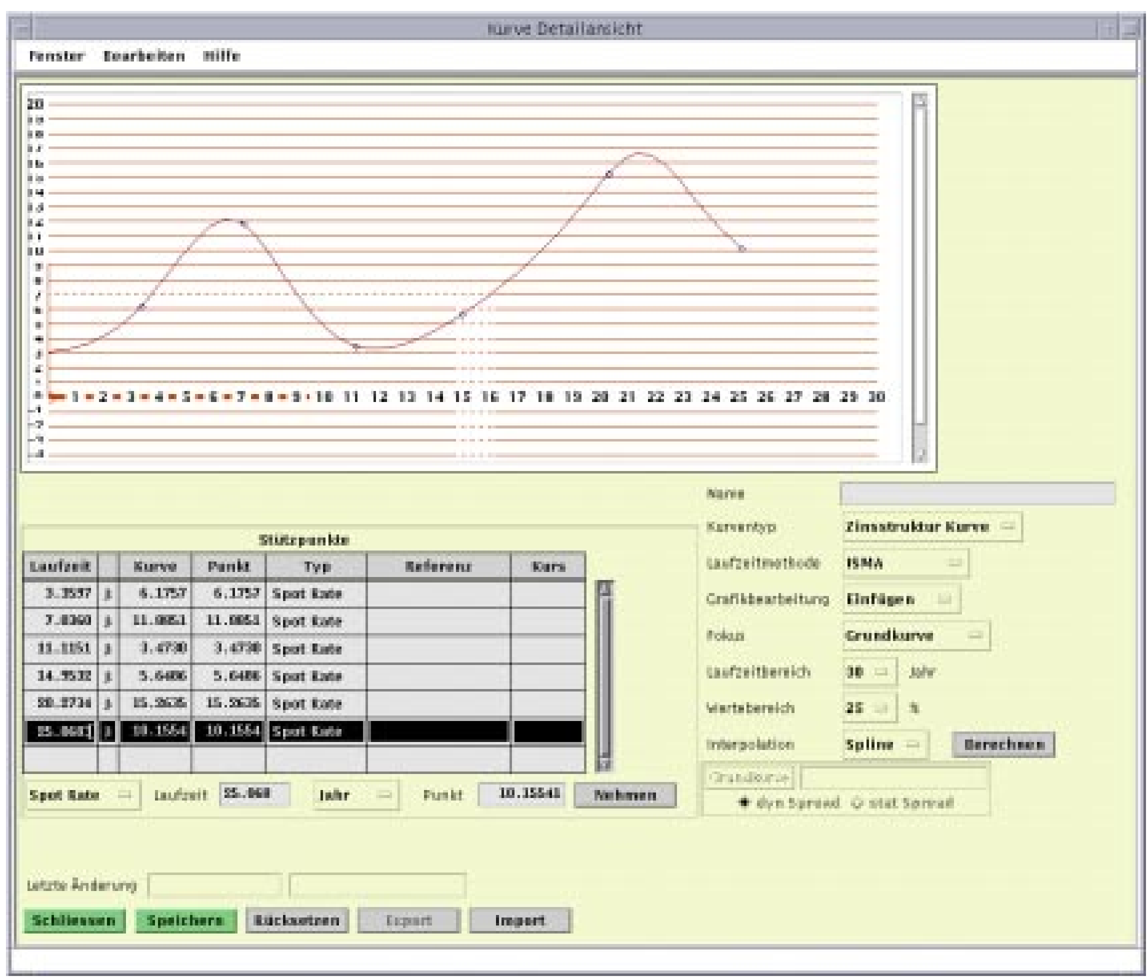

Abbildung A-1: Screenshot einer glatten Interpolation (Spot-Rate-Nodes)

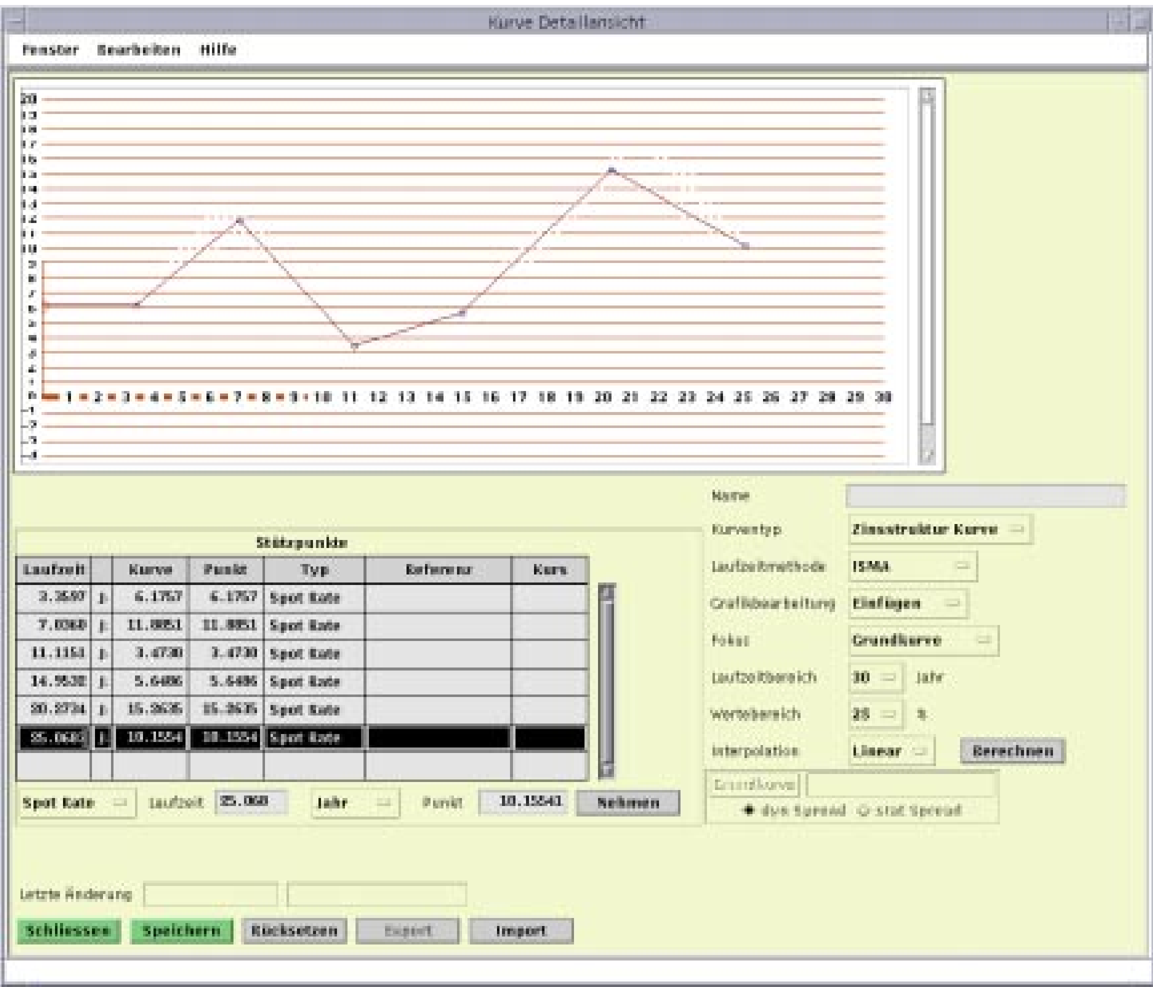

Abbildung A-2: Screenshot einer linearen Interpolation (Spot-Rate-Nodes)

A.2 Kurve mit Forward-Rate-Stützpunkten

Bei den folgenden Abbildungen wurden Forward-Rate-Stützpunkte festgelegt und durch eine glatte bzw. lineare Kurve interpoliert.

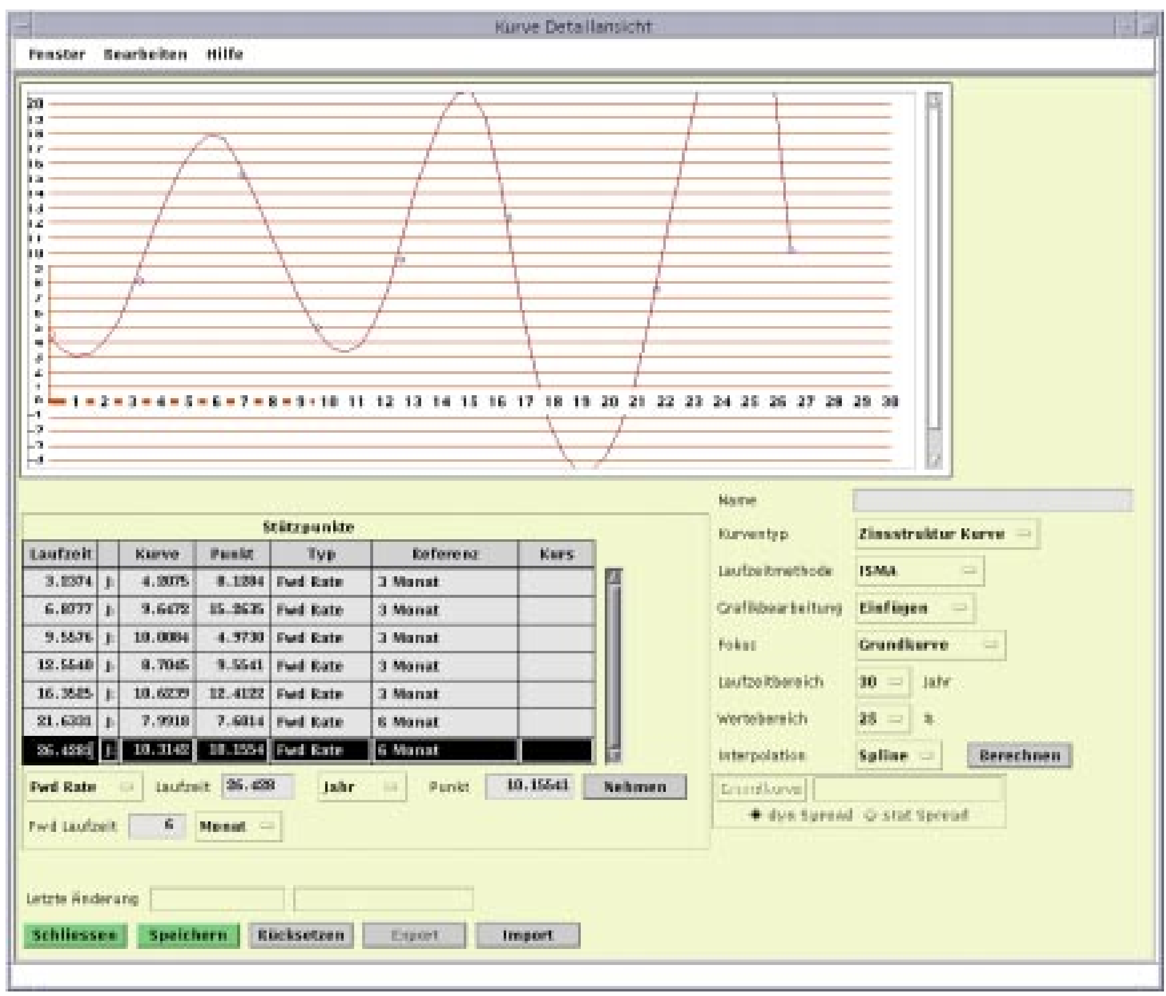

Abbildung A-3: Screenshot einer glatten Interpolation (Forward-Rate-Nodes)

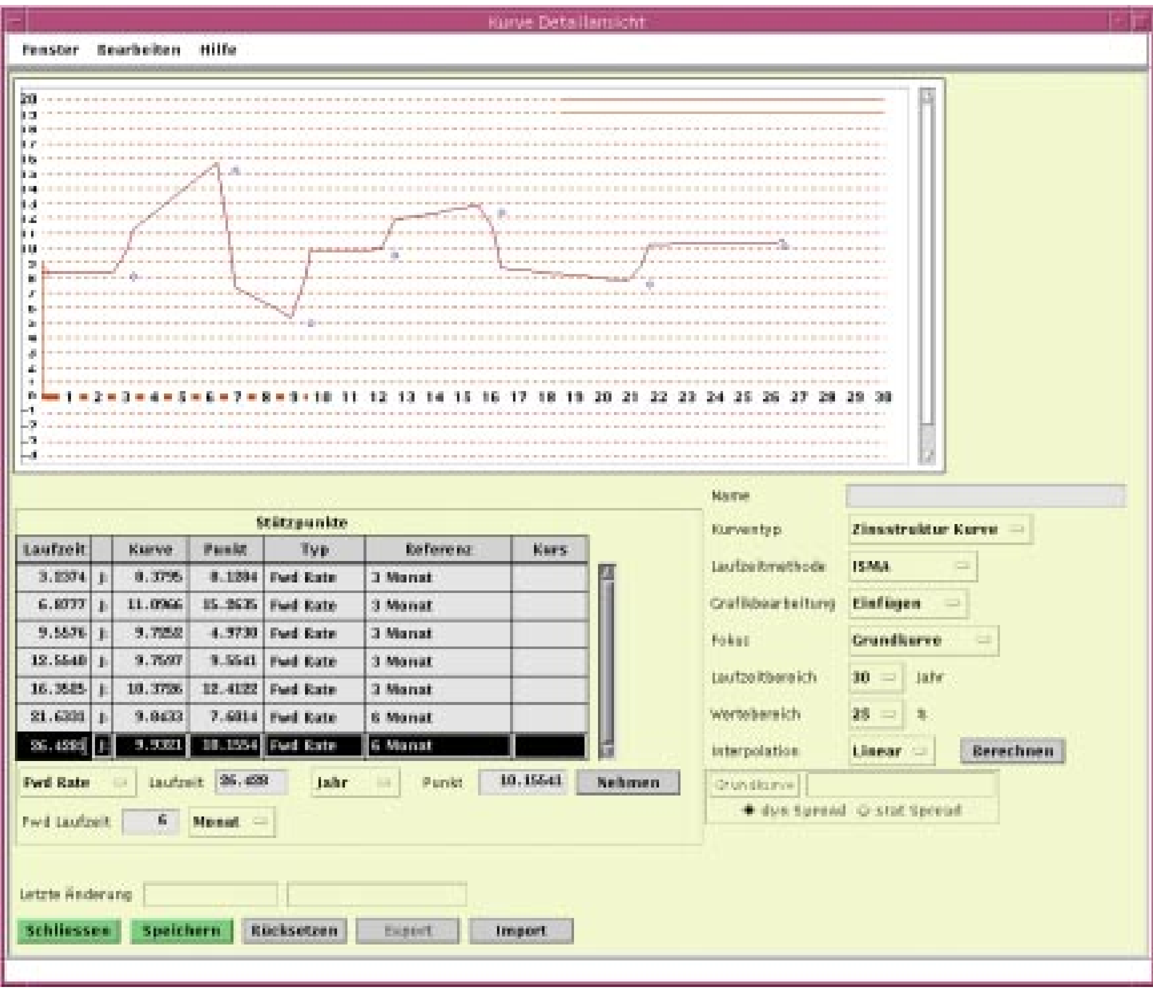

Abbildung A-4: Screenshot einer linearen Interpolation (Forward-Rate-Nodes)

Anhang B

Packages und Klassen

In diesem Anhang werden alle bei diesem Projekt beteiligten Packages mit ihren Klassen dargestellt. Dabei sind jeweils die (wesentlichen) Attribute und Methoden inklusive Signaturen berücksichtigt worden. Für eine Übersicht der Packages sei hier auf *Abschnitt 5.1* und *Abbildung 5-1* verwiesen. Ausserdem werden im *Kapitel 6* die wichtigsten Methoden erklärt und deren Aufrufe in Sequence Diagrammen dargestellt.

B.1 Package Interpolator

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.1*

Klassendiagramm dieses Packages: *Abbildung 5-2*

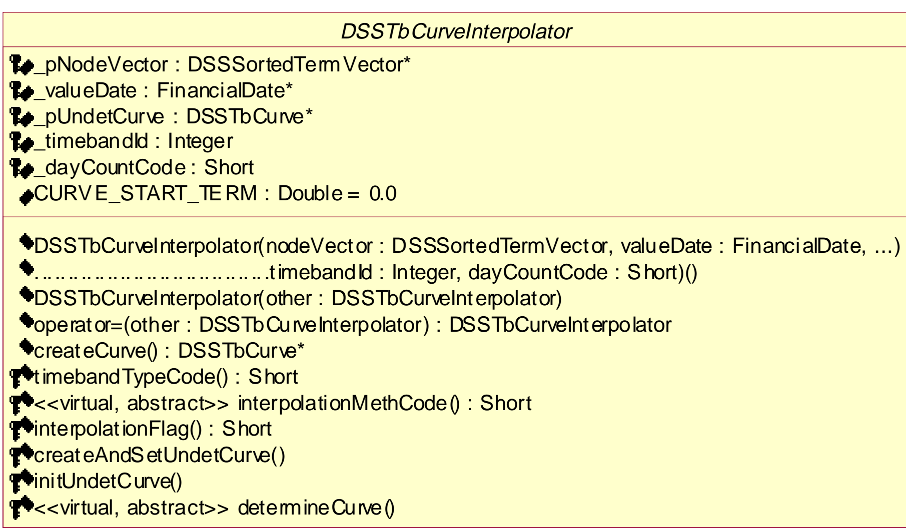

Abbildung B-1: Klasse DSSTbCurveInterpolator

DSSTbCurveInterpolatorLinear

_currentBootstrapSplineIndex : Integer

DSSTbCurveInterpolatorLinear(nodeVector : DSSSortedTermVector, valueDate : FinancialDate, ...)
....timebandId : Integer, dayCountCode : Short)()
DSSTbCurveInterpolatorLinear(other : DSSTbCurveInterpolatorLinear)
operator=(other : DSSTbCurveInterpolatorLinear) : DSSTbCurveInterpolatorLinear
<<static>> nodeFunctionStatic(pCalcObject : Object*, argument : Double)
<<virtual>> interpolationMethCode() : Short
<<virtual>> determineCurve()
bootstrap()
getNodeFunctionRoot()
<<virtual, abstract>> prepareCurve()
<<virtual, abstract>> nodeFunction(argument : Double)
<<virtual, abstract>> extendCurveWithRoot(root : Double)
getTbPointStartTerm() : Double
getTbPointStartValue() : Double

Abbildung B-2: Klasse DSSTbCurveInterpolatorLinear

DSSTbCurveInterpolatorSpline

_smoothEquations : DSSLinEquationSystem
_nodeEquations : DSSLinEquationSystem
_flatEquations : DSSLinEquationSystem
_finalEquations : DSSLinEquationSystem
SMOOTH_DIFF_DEGREE : Integer = 2

DSSTbCurveInterpolatorSpline(nodeVector : DSSSortedTermVector, valueDate : FinancialDate, ...)
....timebandId : Integer, dayCountCode : Short)()
DSSTbCurveInterpolatorSpline(other : DSSTbCurveInterpolatorSpline)
operator=(other : DSSTbCurveInterpolatorSpline)
<<virtual>> interpolationMethCode() : Short
<<virtual>> determineCurve()
initEquationSystems()
calcNodeEquations()
<<virtual, abstract>> getNodeEquation(targetEquation : DSSLinSplineEquation, node : DSSNode) : DSSLinSplineEquatioin
<<virtual, abstract>> getStartTermEquation(targetEquation : DSSLinSplineEquation, term : Double)
calcFinalEquationSystem()
setCurveFromSolution(solutionVector : DSSMathVector)
getSmoothEquationCount() : Integer
getNodeEquationCount() : Integer
getFlatEquationCount() : Integer
getSmoothUnknownCount() : Integer
getFlatUnknownCount() : Integer
getNodeUnknownCount() : Integer

Abbildung B-3: Klasse DSSTbCurveInterpolatorSpline

DSSTbCurveInterpolatorLinearSpot

DSSTbCurveInterpolatorLinearSpot(nodeVector : DSSSortedTermVector, valueDate : FinancialDate, ...)
....timebandId : Integer, dayCountCode : Short)()
<<virtual>> DSSTbCurveInterpolatorLinearSpot(other : DSSTbCurveInterpolatorLinearSpot) : DSSTbCurveInterpolatoeLinearSpot
operator=(other : DSSTbCurveInterpolatorLinear) : DSSTbCurveInterpolatorLinearSpot
<<virtual>> prepareCurve()
<<virtual>> nodeFunction(argument : Double)
<<virtual>> extendCurveWithRoot(root : Double)
setArgumentIntoCurve(argument : Double)

Abbildung B-4: Klasse DSSTbCurveInterpolatorLinearSpot

DSSTbCurveInterpolatorSplineSpot

DSSTbCurveInterpolatorSplineSpot(nodeVector : DSSSortedTermVector, valueDate : FinancialDate, ...)
....timebandId : Integer, dayCountCode : Short)()
DSSTbCurveInterpolatorSplineSpot(other : DSSTbCurveInterpolatorSplineSpot)
operator=(other : DSSTbCurveInterpolatorSplineSpot) : DSSTbCurveInterpolatorSplineSpot
<<virtual>> getNodeEquation(targetEquation : DSSLinSplineEquation, node : DSSNode) : DSSLinSplineEquation
<<virtual>> getStartTermEquation(targetEquation : DSSLinSplineEquation, term : Double) : DSSLinSplineEquation

Abbildung B-5: Klasse DSSTbCurveInterpolatorSplineSpot

DSSTbCurveInterpolatorSplineStd

DSSTbCurveInterpolatorSplineStd(nodeVector : DSSSortedTermVector, valueDate : FinancialDate, ...)
....timebandId : Integer, dayCountCode : Short)()
DSSTbCurveInterpolatorSplineStd(other : DSSTbCurveInterpolatorSplineStd)
operator=(other : DSSTbCurveInterpolatorSplineStd) : DSSTbCurveInterpolatorSplineStd
<<virtual>> getNodeEquation(targetEquation : DSSLinSplineEquation, node : DSSNode) : DSSLinSplineEquation
<<virtual>> getStartTermEquation(targetEquation : DSSLinSplineEquation, term : Double) : DSSLinSplineEquation

Abbildung B-6: Klasse DSSTbCurveInterpolatorSplineStd

B.2 Package Node

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.2*

Klassendiagramm dieses Packages: *Abbildung 5-3*

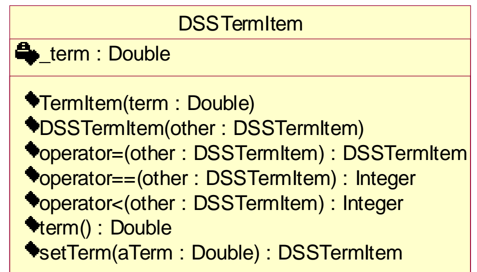

Abbildung B-7: Klasse DSSTermItem

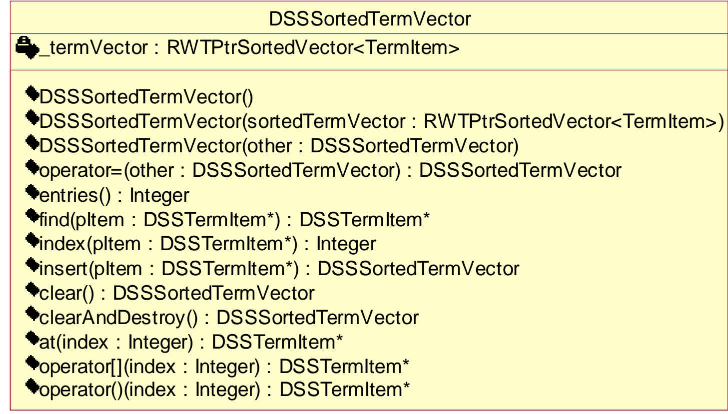

Abbildung B-8: Klasse DSSSortedTermVector

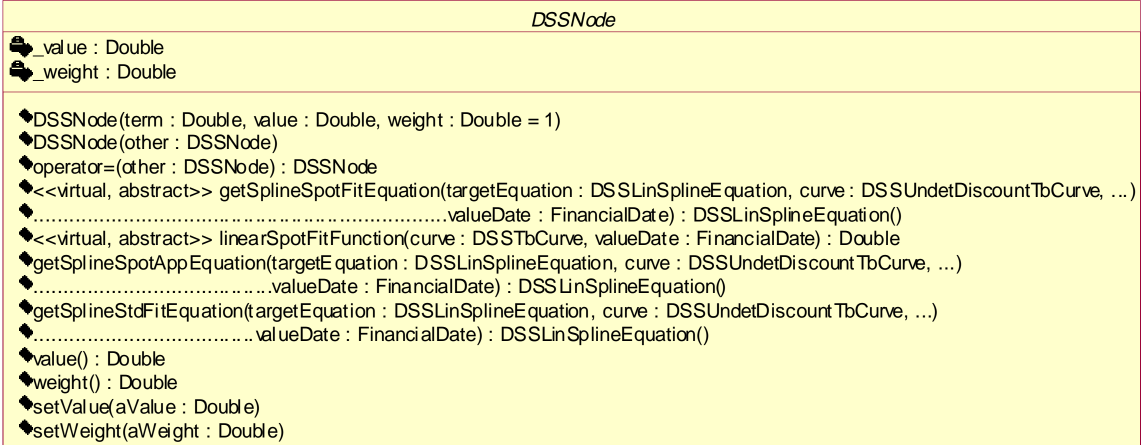

Abbildung B-9: Klasse DSSNode

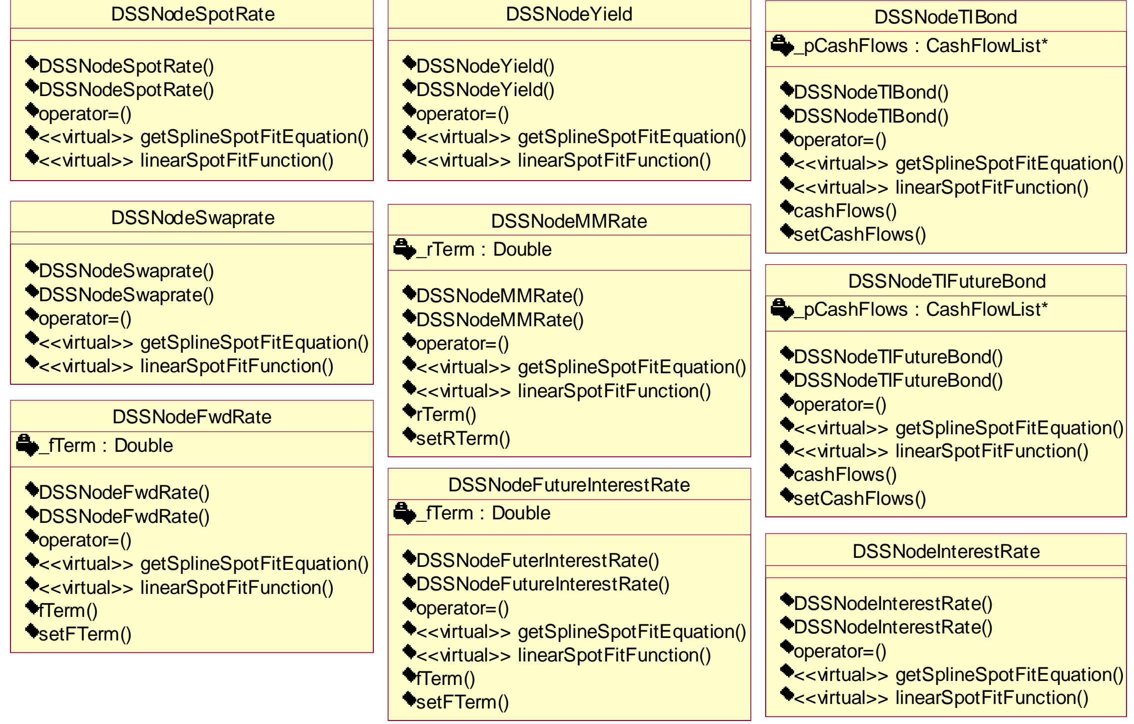

Abbildung B-10: Alle von DSSNode abgeleiteten Klassen

B.3 Package Cashflow

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.3*

Klassendiagramm dieses Packages: *Abbildung 5-4*

CashFlowList

myList : RWOrdered

CashFlowList()
modDuration(interestRate : AbstractCurve = default)
duration()
bpv()
convexity()
presentValue(interestRate : AbstractCurve = default, valueDate : FinancialDate = default, simulationTime : double = 0.0)
insert(cashFlow : CashFlow)
entries()
expandTo()
presentValueEquation(interestRate : AbstractCurve = default, valueDate : FinancialDate = default, ...)
.................................simulationTime : double = 0.0) : LinSplineEquation()
at(index : Integer) : CashFlow

Abbildung B-11: Klasse CashFlowList

CashFlow

CashFlow()
<<virtual, abstract>> presentValue(curve . AbstractCurve, valueDate : FinancialDate*, simulationTime : Double = 0.0) : Double
<<virtual>> presentValueEquation(targetEquation : DSSLinSplineEquation, curve : DSSUndetDiscountTbCurve, ...)
....valueDate : FinancialDate, simulationTime : Double = 0.0) : DSSLinSplineEquation()
<<virtual, abstract>> modDuration(...) : Double
<<virtual, abstract>> duration(...) : Double
<<virtual, abstract>> bpv(...) : Double
<<virtual, abstract>> convexity(...) : Double

Abbildung B-12: Klasse CashFlow

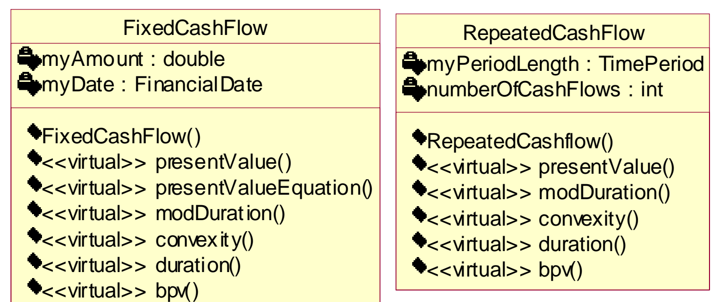

Abbildung B-13: Klassen FixedCashflow und RepeatedCashFlow

B.4 Package Curve

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.4*

Klassendiagramm dieses Packages: *Abbildung 5-5*

<<Interface>> *AbstractCurve*
myDayCountCode : int
AbstractCurve() *AbstractCurve(dayCountCode : Integer)* *calculatePeriod(startDate : Financialdate*, endDate : FinancialDate*) : Double* *calculatePeriod(period TimePeriod, valueDate : FinancialDate) : Double* *<<virtual, abstract>> modDuration(time : double) : double* *modDuration(startDate : FinancialDate = default, endDate : FinancialDate = default) : double* *modDuration(period : TimePeriod, valueDate : FinancialDate) : double* *<<virtual, abstract>> duration(time : double) : double* *duration(startDate : FinancialDate = default, endDate : FinancialDate = default) : double* *duration(period : TimePeriod, valueDate : FinancialDate) : double* *<<virtual, abstract>> convexity(time : double) : double* *convexity(startDate : FinancialDate = default, endDate : FinancialDate = default) : double* *convexity(period : TimePeriod, valueDate : FinancialDate) : double* *<<virtual, abstract>> bpv(time : double) : double* *bpv(startDate : FinancialDate = default, endDate : FinancialDate = default) : double* *bpv(period : TimePeriod, valueDate : FinancialDate) : double* *<<virtual, abstract>> discountFactor(time : double) : double* *discountFactor(startDate : FinancialDate = default, endDate : FinancialDate = default, simulationTime : double = 0.0) : double* *discountFactor(period : TimePeriod, valueDate : FinancialDate) : double* *<<virtual, abstract>> interestRate(time : double) : double* *interestRate(startDate : FinancialDate = default, endDate : FinancialDate = default) : double* *interestRate(period : TimePeriod, valueDate : FinancialDate) : double* *<<virtual, abstract>> presentValueRepeatedCashFlow(cashFlow : RepeatedCashFlow, numberOfCashFlows : Integer, ...)* *..simulationTime : Double) : double()* *getDayCountCode() : Integer* *setDayCountCode(aCode : Integer)*

Abbildung B-14: Klasse AbstractCurve

DSSTbPoint
_periodStart : Double _periodEnd : Double _coeffA : Double _coeffB : Double _coeffC : Double _coeffD : Double
DSSTbPoint(periodStart : Double, periodEnd : Double, coeffA : Double, coeffB : Double, coeffC : Double, coeffD : Double) periodStart() : Double periodEnd() : Double value(term : Double) operator==(otherPoint : TbPoint) : Integer operator<(otherPoint : TbPoint) : Integer

Abbildung B-15: Klasse DSSTbPoint

DSSTbCurve

- _tbPoints : RWTPtrSortedVector<DSSTbPoint>
- _timebandID : Integer
- _timebandTypeCode : Short
- _interpolationMethCode : Short
- _interpolationFlag : Short

- DSSTbCurve()
- DSSTbCurve(timebandId : Integer, timebandTypeCode : Short, dayCountCode : Short, interpolationMethCode : Short, ...)
- ....interpolationFlag : Short)()
- <<static>> create(timebandId : Integer, timebandTypeCode : Short, dayCountCode : Short, interpolationMethCode : Short, ...)
- ....interpolationFlag : Short, determined : Integer = TRUE) : DSSTbCurve()
- numTbIp() : Integer
- growthFactor(term : Double) : Double
- <<virtual>> discountFactor(laufzeit : Double) : Double
- <<virtual>> interestRate(laufzeit : Double) : Double
- volatility(term : Double) : Double
- yield(term : Double) : Double
- effModDuration(term : Double) : Double
- <<virtual>> modDuration(term : Double) : Double
- <<virtual>> duration(term : Double)
- <<virtual>> convexity(term : Double) : Double
- <<virtual>> bpv(term : Double) : Double
- presentValueRepeatedCashFlow(cashFlow : RepeatedCashFlow, numberOfCashFlows : Integer, simulationTime : Double) : double
- tbId() : Integer
- getTbPoints() : RWTPtrSortedVector<DSSTbPoint>
- getTimeBandId() : Integer
- setTimeBandId(anId : Integer)
- getTimeBandTypeCode() : Short
- setTimeBandTypeCode(aCode : Short)
- getInterpolationMethCode() : Short
- setInterpolationMethCode(aCode : Short)
- getInterpolationFlag() : Short
- setInterpolationFlag(aFlag : Short)
- insertUndetTbPoint(periodStart : Double, periodEnd : Double)
- setTbPointCoeffs(splineIndex : Integer, coeffA : Double, coeffB : Double, coeffC : Double, coeffD : Double)
- searchIpIndex(term : Double) : Integer
- searchIp(double term) : TbPoint

Abbildung B-16: Klasse DSSTbCurve

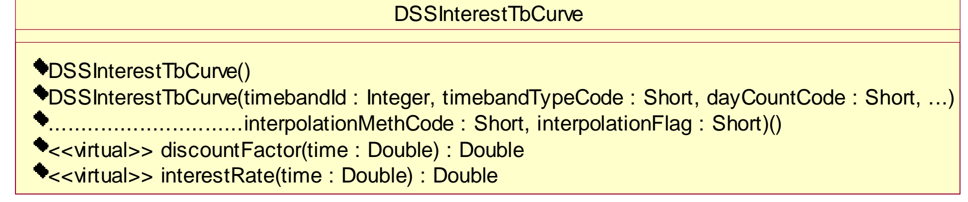

Abbildung B-17: Klasse DSSInterestTbCurve

DSSDiscountTbCurve

- DSSDiscounTbCurve(timebandId : Integer, timebandTypeCode : Short, dayCountCode : Short, ...)
- ....interpolationMethCode : Short, interpolationFlag : Short)()
- DSSDiscountTbCurve()
- <<virtual>> discountFactor(term : Double) : Double
- <<virtual>> interestRate(term : Double) : Double

Abbildung B-18: Klasse DSSDiscountTbCurve

DSSUndetDiscountTbCurve

DSSUndetDiscountTbCurve()
DSSUndetDiscountTbCurve(timebandId : Integer, timebandTypeCode : Short, dayCountCode : Short, ...)
....interpolationMethCode : Short, interpolationFlag : Short)()
getEmptyEquation(targetEquation : DSSLinSplineEquation) : DSSLinSplineEquation
getDiscountFactorEquation(targetEquation : DSSLinSplineEquation, term : Double) : DSSLinSplineEquation
getDiscountFactorEquation(targetEquation : DSSLinSplineEquation, pStartDate : FinancialDate*, ...)
....pEndDate : FinancialDate*) : DSSLinSplineEquation()
getDiscountFactorEquation(targetEquation : DSSLinSplineEquation, pPeriod : TimePeriod*, ...)
....pValueDate : FinancialDate*) : DSSLKinSplineEquation()
getSmoothEquations(targetEquations : DSSLinEquationSystem, diffDegree : Integer) : DSSLinEquationSystem
getFlatEquations(targetEquations : DSSLinEquationSystem) : DSSLinEquationSystem
getSmoothEquation(targetEquation : DSSLinSplineEquation, splineIndex : Integer, diffDegree : Integer) : DSSLinSplineEquation
getFlatEquation(targetEquation : DSSLinSplineEquation, splineIndex : Integer, diffCoeffIndex : Integer) : DSSLinSplineEquation
getQualityGradient(targetSpline : DSSSpline, term : Double, diffCoeffIndex : Integer) : DSSSpline
unknownCount() : Integer

Abbildung B-19: Klasse DSSUndetDiscountTbCurve

B.5 Package Math

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.5*

Klassendiagramm dieses Packages: *Abbildung 5-6*

DSSMathVector

_dimension : Integer
_pArray : Double*

DSSMathVector(dimension : Integer, pInitArray : Double* = NULL)
DSSMathVector(other : DSSMathVector)
operator=(other : DSSMathVector) : DSSMathVector
operator==(other : DSSMathVector) : Integer
add(other : DSSMathVector) : DSSMathVector
sub(other : DSSMathVector) : DSSMathVector
scalarProd(other : DSSMathVector) : Double
mult(factor : Double) : DSSMathVector
negate() : DSSMathVector
dimension() : Integer
clear(value : Double = 0.0) : DSSMathVector
clear(startIndex : Integer, endIndex : Integer, value : Integer = 0.0) : DSSMathVector
reset(aDimension : Integer, pInitArray : Double* = NULL) : DSSMathVector
resetDimension(aDimension : Integer) : DSSMathVector
set(pArray : Double*, arrayLen : Integer, startIndex : Integer = 1, arrayStartIndex : Integer = 0) : DSSMathVector
set(other : DSSMathVector, startIndex : Integer = 1, otherStartIndex : Integer = 1) : DSSMathVector
setCoeff(value : Double, index : Integer) : DSSMathVector
get(targetVector : DSSMathVector, startIndex : Integer = 1)
getArrayRef() : Double*
at(index : Integer) : Double
operator()(index : Integer) : Double
operator[](index : Integer) : Double

Abbildung B-20: Klasse DSSMathVector

Abbildung B-21: Klasse DSSMathMatrix

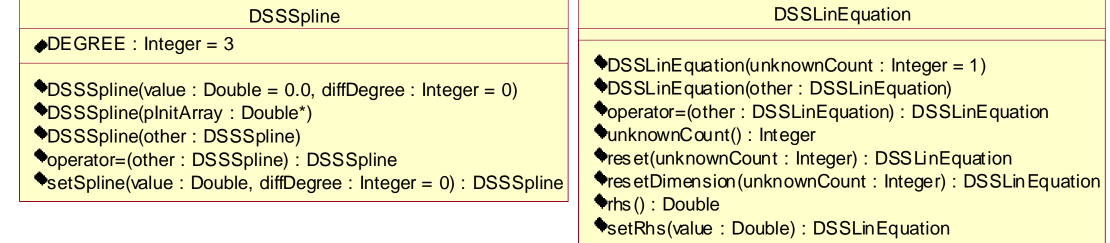

Abbildung B-22: Klassen DSSSpline und DSSLinEquation

DSSLinSplineEquation

DSSLinSplineEquation(splineCount : Integer = 1)
DSSLinSplineEquation(other : DSSLinSplineEquation)
operator=(other : DSSLinSplineEquation) : DSSLinSplineEquation
splineCount() : Integer
setSpline(term : Double, splineIndex : Integer, diffDegree : Integer = 0) : DSSLinSplineEquation
setSpline(spline : DSSSpline, splineIndex : Integer) : DSSLinSplineEquation
setCoeff(value : Double, splineIndex : Integer, coeffIndex : Integer) : DSSLinSplineEquation
getSpline(targetSpline : Spline, splineIndex : Integer) : DSSSpline
at(splineIndex : Integer, coeffIndex : Integer) : Double
operator()(splineIndex : Integer, coeffIndex : Integer) : Double

Abbildung B-23: Klasse DSSLinSplineEquation

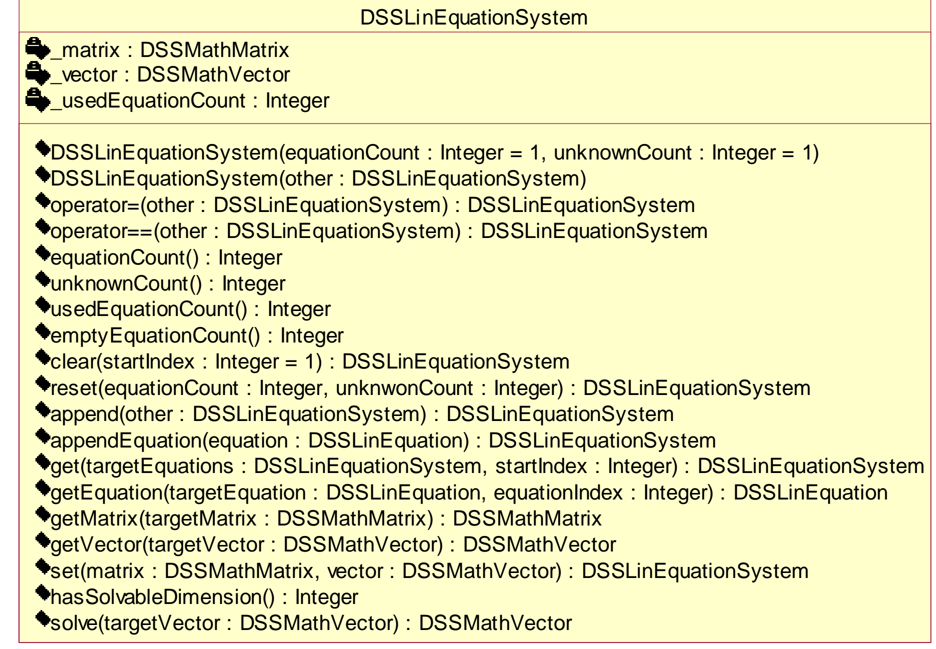

Abbildung B-24: Klasse DSSLinEquationSystem

DoubleToDoubleFunction

myFunction : FunctionPointer
myObject : Object*
myStretch : Double

DoubleToDoubleFunction(f : functionPointer, anObject : Object*)
rootByBrent(y : Double, x_min : Double, x_max : Double, y_min : Double, y_max : Double, x1 : Double, x2 : Double) : Double
rootByBrent(y : Double, x_min : Double, x_max : Double, y_min : Double, y_max : Double, x_guess : Double) : Double
rootByBrent(y : Double, x_min : Double, x_max : Double, x_guess : Double) : Double

Abbildung B-25: Klasse DoubleToDoubleFunction

<<Utility>>
DSSMathFunctions

max(n : Double, m : Double) : Double
max(n : Integer, m : Integer) : Integer
min(n : Double, m : Double) : Double
min(n : Integer, m : Integer) : Integer
dvector(nl : Long, nh : Long) : Double*
free_dvector(v : Double*, nl : Long, nh : Long)
dmatrix(nrl : Long, nrh : Long, ncl : Long, nch : Long) : Double**
free_dmatrix(m : Double**, nrl : Long, nrh : Long, ncl : Long, nch : Long)
ludcmp(a : Double**, n : Integer, indx : Integer*, d : Double*)
lubksb(a : Double**, n : Integer, indx : Integer*, b : Double*)

Abbildung B-26: Utility DSSMathFunctions

B.6 Package Exception

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.6*

Klassendiagramm dieses Packages: *Abbildung 5-7*

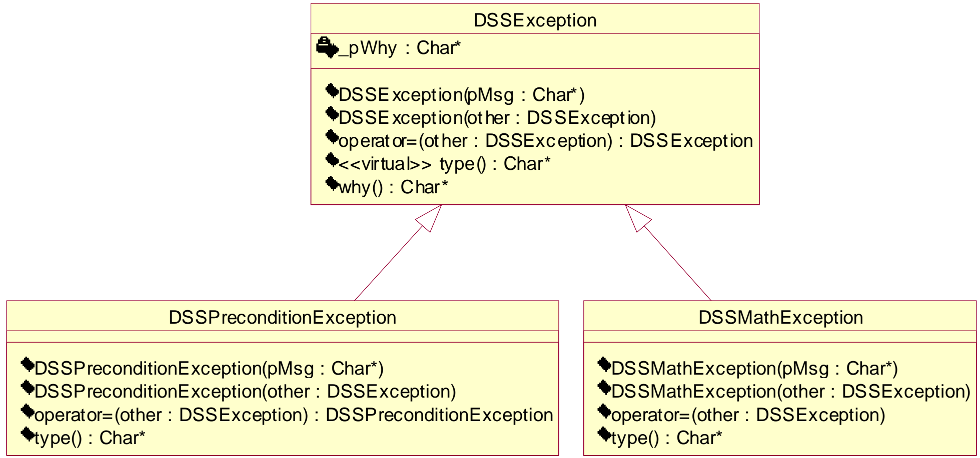

Abbildung B-27: Alle Klassen des Packages Exception

B.7 Package RogueWave Tools.h++

Weitere Erklärungen zu diesem Package: *Abschnitt 5.1.7*

Klassendiagramm dieses Packages: *Abbildung 5-8*

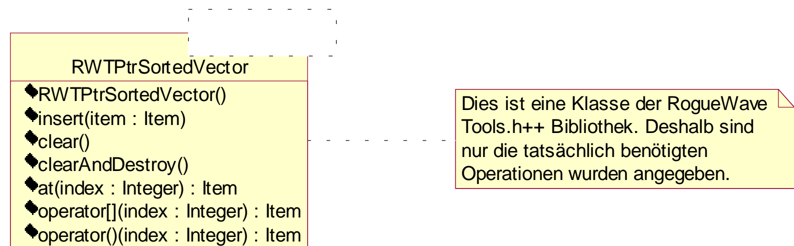

Abbildung B-28: Klasse RWTPtrSortedVector