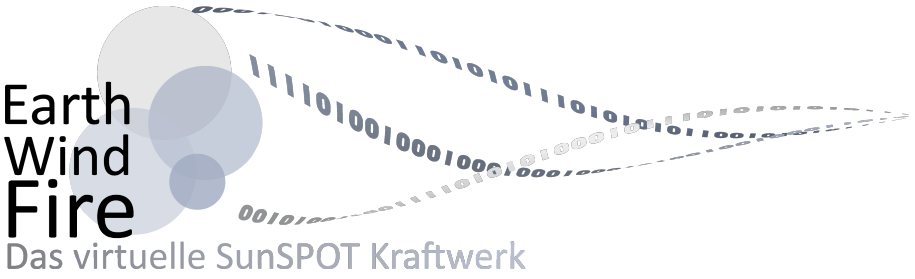
Projektgruppe Earth, Wind and Fire

Endbericht

Wintersemester 2008/2009 / Sommersemester 2009



Gruppe:
Jan-Ole Brodé, Simon Flandergan, Dennis Geesen,
Christian Hülsen, Wolfgang König, Kai-Andre Pancratz,
Gunnar Schulte-Loh, Marcel Sommerlatt, Ulf Truderung,
Jan Vornberger, Tobias Witt

Dieser Endbericht wurde im Rahmen der Projektgruppe Earth, Wind and Fire an der Carl-von-Ossietzky-Universität Oldenburg erstellt. Er behandelt die Entwicklung eines Demonstrators für Datenstromtechnologie mit Odysseus.

Inhaltsverzeichnis

1 Einleitung **8**

2 Hintergrund **9**
2.1 Das Datenstrommanagementsystem Odysseus . . . 9
2.2 Simulierte Energiequellen . . . 9
2.2.1 SunSpot-Sensoren . . . 9
2.3 Earth, Wind and Fire als Präsentator . . . 9

3 Anforderungen **10**
3.1 Einführung . . . 10
3.1.1 Odysseus . . . 10
3.1.2 Zweck . . . 10
3.1.3 Umfang des Systems . . . 10
3.1.4 Ziele und Erfolgskriterien des Projekts . . . 10
3.2 Beschreibung des Projektes . . . 11
3.2.1 Projektorganisation . . . 11
3.2.2 Beschreibung der Rollen . . . 11
3.2.3 Räumliche Gegebenheiten . . . 12
3.2.4 Standards und Richtlinien . . . 12
3.2.5 Lieferumfang . . . 12
3.3 Meilensteine . . . 12
3.3.1 Angebot . . . 13
3.3.2 Lastenheft . . . 13
3.3.3 Pflichtenheft . . . 13
3.3.4 Abschluss 1.Iteration . . . 13
3.3.5 Schülerinformationstag der Informatik . . . 13
3.3.6 Abschluss 2. Iteration . . . 13
3.3.7 Nacht der Wissenschaft . . . 13
3.3.8 Projektabschluss . . . 13
3.4 Vorhandenes System . . . 14
3.5 Vorgeschlagenes System . . . 14
3.5.1 Übersicht . . . 14
3.5.2 Funktionale Anforderungen . . . 14
3.5.3 Nicht Funktionale Anforderungen . . . 14
3.5.4 Qualitäts- und Dokumentationsanforderungen . . . 15
3.5.5 Produktstrukturplan . . . 16
3.5.6 Erste Iteration . . . 18
3.5.7 Zweite Iteration . . . 21
3.6 Anwendungsfälle . . . 22
3.6.1 SunSPOT . . . 22
3.6.2 Odysseus . . . 25
3.6.3 View . . . 27
3.7 Hinzugefügte und entfallene Anforderungen . . . 33
3.7.1 SunSPOT . . . 33
3.7.2 Odysseus . . . 34

3.7.3 View . . . 34

4 Entwurf **35**
4.1 Einführung . . . 35
4.1.1 Zweck des Systems . . . 35
4.1.2 Entwurfsziele . . . 35
4.1.3 Referenzen . . . 35
4.1.4 Überblick . . . 35
4.2 Aktuelle Softwarearchitektur . . . 35
4.3 Vorgeschlagene Softwarearchitektur . . . 36
4.3.1 Überblick . . . 36
4.3.2 Systemzerlegung . . . 36
4.3.3 SunSpots . . . 36
4.3.4 Odysseus . . . 41
4.3.5 View . . . 46
4.3.6 3D-View . . . 77
4.3.7 Management von persistenten Daten . . . 87
4.3.8 Globaler Kontrollfluss . . . 88

5 Umsetzung **91**
5.1 Kraftwerkssimulation . . . 91
5.1.1 Verwendete Technologien . . . 91
5.1.2 Architektur . . . 91
5.1.3 Challenge-Response Authentifikation . . . 92
5.1.4 Zeitsynchronisation . . . 92
5.2 View . . . 93
5.2.1 Verwendete Technologien . . . 93
5.2.2 Angewendete Konzepte . . . 93
5.3 3D-View . . . 94
5.3.1 Verwendete Technologien . . . 94
5.3.2 Verwendete Konzepte . . . 94
5.4 Odysseus . . . 97
5.4.1 Verwendete Technologien . . . 97
5.4.2 Angewendete Konzepte . . . 97

6 Test **99**
6.1 Einführung . . . 99
6.1.1 Zweck . . . 99
6.1.2 Umfang . . . 99
6.1.3 Beziehungen zu anderen Dokumenten . . . 99
6.1.4 Aufbau der Testbezeichner . . . 99
6.2 Merkmale . . . 100
6.2.1 Zu testende Merkmale . . . 100
6.2.2 Funktionale Anforderungen . . . 100
6.2.3 Nicht zu testende Merkmale . . . 100
6.3 Abnahme- und Testendekriterien . . . 100
6.4 Vorgehensweise . . . 100
6.4.1 Integrationstest . . . 100
6.4.2 Funktionstest . . . 101
6.4.3 Leistungstest . . . 101
6.5 Aufhebung und Wiederaufnahme . . . 101
6.6 Testfälle . . . 102
6.6.1 Integrationstests . . . 102
6.6.2 Funktionstests . . . 103
6.6.3 Leistungstests . . . 106
6.7 Durchgeführte Tests . . . 107
6.7.1 Integrationstest . . . 107

6.7.2 Funktionstest . . . 107
6.7.3 Leistungstest . . . 107

7 Ablauf 108
7.1 KickOff-Phase . . . 108
7.2 Erste Iteration . . . 108
7.2.1 Meilensteinbetrachtung . . . 108
7.2.2 Fortschritte weiterer Arbeitspakete . . . 109
7.2.3 Übergabe des Projekts an Christian . . . 110
7.3 Zweite Iteration . . . 113
7.3.1 Neue Gruppeneinteilung . . . 113
7.3.2 Meilensteinbetrachtung . . . 113
7.3.3 Fortschritte weiterer Arbeitspakete . . . 114
7.4 Test- und Dokumentenphase . . . 117
7.4.1 Meilensteinbetrachtung . . . 117
7.4.2 Fortschritte weiterer Arbeitspakete . . . 117

8 Fazit 119
8.1 Thematik Datenstrommanagement und regenerative Energien . . . 119
8.2 Ablauf der Projektgruppe . . . 119
8.3 Ausblick . . . 119

9 Installationshandbuch 121
9.1 Einführung . . . 121
9.2 Kraftwerkssimulation . . . 121
9.2.1 Installation der SunSPOTs . . . 121
9.2.2 Installation ohne SunSPOTs . . . 121
9.3 Odysseus . . . 121
9.3.1 Installation von Odysseus . . . 122
9.3.2 Archiving . . . 122
9.4 View . . . 122
9.4.1 Installation als Plugin . . . 122
9.4.2 Installation als Product . . . 122
9.5 3D View . . . 123
9.5.1 Hinzufügen neuer Modelle . . . 123
9.5.2 Meta-Daten . . . 123

10 Benutzerhandbuch 126
10.1 Einführung . . . 126
10.2 Kraftwerkssimulation . . . 126
10.2.1 SunSPOTs . . . 126
10.2.2 SPOTHost . . . 126
10.2.3 Virtual SPOTHost . . . 126
10.3 Odysseus . . . 127
10.4 View . . . 128
10.4.1 Anforderungen der RCP-View . . . 128
10.4.2 Starten der RCP-View . . . 129
10.4.3 Verbindung zu Odysseus herstellen . . . 129
10.4.4 Anfrageverwaltung . . . 130
10.4.5 Perspektiven Management . . . 132
10.4.6 Tabellen . . . 134
10.4.7 Graphen . . . 136
10.4.8 Topologie . . . 139
10.4.9 Kritische Werte . . . 140
10.4.10 Alarmleiste . . . 142
10.4.11 Ereignislog . . . 142
10.4.12 Rohdaten . . . 143

10.4.13 Kioskmodus . . . 143
10.4.14 Fehlerfall . . . 144
10.5 3D-View . . . 147
10.5.1 Anforderungen der 3D-View . . . 147
10.5.2 Starten der 3D-View . . . 148
10.5.3 Navigation . . . 148
10.5.4 Anpassen der Welt . . . 149
10.5.5 Anfragedarstellung . . . 150
10.5.6 Laden und Speichern . . . 151

11 Präsentationsdrehbuch **152**
11.1 Einführung . . . 152
11.2 Vorbereitung . . . 152
11.3 Präsentation . . . 152
11.3.1 Datenquelle . . . 152
11.3.2 Perspektive erstellen . . . 153
11.3.3 Stream und Statement anlegen, Tabelle erstellen . . . 153
11.3.4 Graph erstellen . . . 155
11.3.5 Kritische Werte . . . 157
11.3.6 Topologie . . . 158
11.3.7 3D-Welt . . . 159
11.4 Eingaben . . . 160

12 Messeflyer **161**
12.1 Daten . . . 161
12.2 Flyer . . . 161

13 Glossar **163**

Abbildungsverzeichnis

3.1 Produktstrukturplan . . . 16
3.2 Screenshot der momentanen View . . . 19

4.1 Entwurf des Gesamtsystems . . . 37
4.2 Kontrollflussdiagramm View . . . 89
4.3 Kontrollflussdiagramm 3D View . . . 90

5.1 Architecture . . . 91
5.2 Connection and Communication . . . 92
5.3 3D Software Komponenten . . . 94
5.4 3D Grundarchitektur . . . 95
5.5 Normierung der Attributwerte für eine Ausprägung in der 3D-Welt . . . 96

7.1 Netzplan der 1. Iteration . . . 111
7.2 Projektplan der 1. Iteration . . . 112
7.3 Projektplan der 2. Iteration . . . 116
7.4 Projektplan der Test- und Dokumentenphase . . . 118

9.1 Product Konfiguration, Abhängigkeiten . . . 123
9.2 Product export . . . 124
9.3 Ordnerstruktur der Modelle . . . 124
9.4 Erweitern des Enum StaticalTyp . . . 125

10.1 Virtual SPOTHost Screenshot . . . 127
10.2 Odysseus kann nicht erreicht werden . . . 129
10.3 Ändern der Odysseus Verbindung . . . 129
10.4 CQL-Anfragen verwalten . . . 130
10.5 Streams verwalten . . . 131
10.6 Aufgezeichnete Streams verwenden . . . 131
10.7 Perspektivenmanager ohne geöffnete Perspektive . . . 132
10.8 Neue Perspektive/ Ordner erstellen . . . 133
10.9 Leere Perspektive . . . 133
10.10 Element von der Anwendung abdocken . . . 134
10.11 Eine neue Tabelle erstellen . . . 135
10.12 Tabelle mit verschiedenen Hervorhebungen . . . 135
10.13 Dialog zur Erstellung eines Graphens . . . 136
10.14 Einstellungen für X,Y - Diagramm . . . 137
10.15 Zusätzliche Optionen für X,Y - Diagramm . . . 138
10.16 Kuchendiagramm inklusive Farbeinstellungen . . . 139
10.17 Balkendiagramm inklusive erweiterter Einstellungen . . . 139
10.18 Thermometerdiagramm . . . 140
10.19 2D Topologie . . . 141
10.20 Kritische Werte definieren . . . 141
10.21 Handlungsanweisungen vermerken . . . 142
10.22 Ereignislog mit teils bestätigen Ereignissen . . . 143
10.23 Rohdaten eines Tupels . . . 144

10.24Anwendung im Kioskmodus . . . 145
10.25Passwortabfrage zum Wechsel in den priviligierten Modus . . . 145
10.26Odysseus liefert einen Fehlercode beim Installieren einer Anfrage . . . 145
10.27Fehlerfall bei einem Webservice-Aufruf . . . 146
10.28Graph mit „verlorener“ Anfrage . . . 146
10.29Dialog zur Auswahl einer alternativen Anfrage . . . 147
10.30Gestartete 3D-Welt . . . 148
10.31Dynamisches 3D-Modell hinzufügen . . . 149
10.32Anfrage eines 3D-Modells bearbeiten . . . 150
10.33Anfrage eines 3D-Modells auswählen oder löschen . . . 151
10.34Speichern- und Ladendialog der 3D-Welt . . . 151

12.1 Vorderseite des Messeflyers . . . 162
12.2 Rückseite des Messeflyers . . . 162

Kapitel 1

Einleitung

Für die Präsentation des Datenstrommanagementsystems Odysseus ist es nötig, eine Oberfläche zur Bedienung sowie eine Datenquelle zu haben. Diese beiden Punkte erfüllt Earth, Wind and Fire (EWaF). Die Software liefert Datenströme mit Hilfe von SunSpot-Sensoren und bietet eine grafische Oberfläche zur Anfrage und Steuerung von Odysseus.

Dadurch ist es möglich, die Arbeit von Odysseus greifbar zu machen und optisch ansprechend darzustellen.

Als Szenario für die Präsentationssoftware wurden regenerative Energien ausgewählt. Hier wird der Fokus insbesondere auf die Überwachung der Energieerzeuger gelegt. Mittels der Sensoren werden Geothermal-, Solar- und Windkraftanlagen simuliert. Die gewonnenen Daten werden kontinuierlich als XML-Dokumente, welche bestehenden Standards im Energiebereich genügen, an Odysseus gesendet. Dort werden die Daten zerlegt und bearbeitet. Von der grafischen Oberfläche können diese Datenströme mittels CQL-Anfragen angefordert werden. Dabei ist es natürlich möglich, nur Teile des eingehenden Datenstroms auszugeben oder auch mehrere Datenströme zu einem ausgehenden Strom zu vereinen.

Die Software Earth, Wind and Fire kann im Zusammenspiel mit Odysseus auf Messen, Ausstellungen oder Präsentationen dazu genutzt werden, die Arbeitsweise und die Vorteile eines Datenstrommanagementsystems aufzuzeigen.

Dieser Endbericht beleuchtet den gesammten Ablauf der Projektgruppe. In Kapitel 2 wird auf die zugrundeliegenden Komponenten von EWaF, also Odysseus und die SunSpot-Sensoren, eingegangen. Der zu Projektbeginn festgelegte Funktionsumfang sowie ein grob geplanter Projektverlauf wird dann ich Kapitel 3 beschrieben. Kapitel 4 enthält den an das Endprodukt angepassten Softwareentwurf mit der Beschreibung sämtlicher Klassen sowie der wichtigsten Methoden. Auf Umsetzung dieses Entwurfs wird in Kapitel 5 eingegangen. Hier werden die Technologien und Konzepte, die bei der Entwicklung eingesetzt wurden, detailliert beschrieben. In Kapitel 6 werden die Bedingungen für ein erfolgreiches Testen aufgezeigt, sowie die Testfälle beschrieben. Darauf folgt in Kapitel 7 eine Beschreibung des Ablaufs der Projektgruppe, die auch den aktuellen Stand der Entwicklung sowie der Planung wiedergibt. Ein abschließendes Fazit wird dann in Kapitel 8 gegeben.

Im Anhang befinden sich dann die Dokumente, die zum Lieferumfang von Earth, Wind and Fire gehören. Dieses sind im Einzelnen das Installationshandbuch als Kapitel 9, welches die nötigen Schritte zur erfolgreichen Installation der Softwarekomponenten vermittelt. In Kapitel 10 ist das Benutzerhandbuch zu finden, dieses ist eine komplett bebilderte Bedienungsanleitung für die Software mit deren Hilfe jeder Benutzer die Funktionen von EWaF vollständig nutzen kann. Um bei Präsentationen durch einen Präsentator in einheitliches Vorgehen zu gewährleisten und sicherzustellen, das kein Feature ausgelassen wird folgt in Kapitel 11 ein Präsentationsdrehbuch. Als Mitgeber auf Messeständen ist in Kapitel 12 ein Messeflyer angehängt, der die Grundzüge von EWaF kurz beschreibt.

Kapitel 2

Hintergrund

2.1 Das Datenstrommanagementsystem Odysseus

Die Überwachung und Analyse kontinuierlicher (Sensor-)Datenströme findet klassischerweise mittels manuell erstellter Programme statt. Datenstrommanagementsysteme erlauben es diese Aufgaben stattdessen mit Hilfe von deklarativen Anfragen/Regeln – ähnlich wie Datenbankanfragen – zu erledigen.

Vorhandene Datenstrommanagementsysteme unterstützen zumeist nur das relationale Datenmodell und sind in ihren Kernkomponenten schlecht anpassbar. Manche Anwendungen erfordern aber genau solch eine Anpassung. Daher entwickelt die Abteilung Informationssysteme im Projekt ODYSSEUS ein Framework für Datenstrommanagementsysteme, dessen Architektur es erlaubt einzelne Komponenten anzupassen und/oder das Datenmodell auszutauschen.

Im EWaF-Softwarestack wird das ODYSSEUS-Framework eingesetzt, um die Daten der simulierten Energiequellen weiterzuverarbeiten und der Präsentationsschicht zur Verfügung zu stellen.

2.2 Simulierte Energiequellen

Für die Auswahl der Energiequellen wurden Dezentralität und Erneuerbarkeit als Kriterien gesetzt. Hieraus ergaben sich die Windenergie, Solarenergie (Photovoltaik) sowie die Geothermalenergie als zu simulierende Quellen. Die von den simulierten Quellen erzeugten Daten beschränken sich hierbei nicht auf die reinen Energieproduktionsmetriken, sondern simulieren viele weitere Eigenschaften der jeweiligen Kraftwerke. So enthält der Datenstrom bei einem Windkraftwerk die aktuelle Anzahl der Umdrehungen pro Minute, bei einer Solaranlage wird die aktuell gemessene Helligkeit mit übermittelt. Weiter werden auch die Zustände der einzelnen (simulierten) Komponenten sowie die jeweiligen Messzeitpunkte über den Datenstrom kommuniziert.

2.2.1 SunSpot-Sensoren

Auf den SunSPOTs befinden sich auf den DemoBoard bereits Sensoren für die Temperatur, die Helligkeit und ein Accelerometer. Weiter besteht die Möglichkeit Spannung von 0-3 Volt auf vier analogen Eingängen zu messen. Diese Werte werden dann gesamplet mit einer Auflösung von 10 Bit (1024 Werte).

2.3 Earth, Wind and Fire als Präsentator

Earth, Wind and Fire wurde als Präsentator für Odysseus entwickelt. Obwohl es ein wirklichkeitsnahes Szenario wiederspiegelt fehlen doch viele Elemente, um es so in einer Überwachung echter Kraftwerke einsetzen zu können. Es besteht Beispielsweise über die Software keine Möglichkeit, die Kraftwerke in irgendeiner Form zu steuern. Der Rückkanal ist schlichtweg nicht vorhanden.

Auch die Nutzersteuerung ist für einen reinen Messebetrieb ausgelegt. Hier existiert zwar der Kioskmodus, der viele Funktionen abschaltet, aber eine direkte Anmeldung am System mit Nutzernamen und Passwort, wie sie für eine nachvollziehbare Nutzersteuerung nötig ist, wurde ebenso vernachlässigt wie alle anderen sicherheitskritischen Aspekte.

Kapitel 3

Anforderungen

3.1 Einführung

Im Rahmen dieser Projektgruppe wird eine Visualisierung von Datenströmen realisiert. Es werden Kraftwerke durch SunSpot Sensoren simuliert, die Datenströme mit „Odysseus“ verarbeitet und auf verschiedene Arten visualisiert.

Die Anwendung setzt auf das bereits bestehende Forschungsprojekt „Odysseus“ auf und wird als Eclipse-PlugIn realisiert.

3.1.1 Odysseus

Die Überwachung und Analyse kontinuierlicher (Sensor-)Datenströme findet klassischerweise mittels manuell erstellter Programme statt. Datenstrommanagementsysteme (DSMS) erlauben es diese Aufgaben stattdessen mit Hilfe von deklarativen Anfragen/Regeln - ähnlich wie Datenbankanfragen - zu erledigen. Mit sogenannten Complex Event Processing (CEP) Systemen ist es möglich komplexe Ereignisabfolgemuster zu definieren und in Ereignisdatenströmen zu erkennen.

Ziel des Projektes Odysseus ist zum einen die Entwicklung eines Forschungsframeworks für Datenstrommanagement- und Complex Event Processing Technologien, sowie vorhandene Technologien in einem einheitlichen Framework zu evaluieren.

3.1.2 Zweck

Die zu entwickelnde Software kann als Demonstrator auf Messen und Vorführungen herangezogen werden. Sie dient der Visualisierungen von Datenströmen. Dabei wird die Möglichkeit einer Präsentation nach Drehbuch geboten aber auch die Möglichkeit gegeben von Interessenten eigenständig und selbsterklärend bedient zu werden.

3.1.3 Umfang des Systems

Der Schwerpunkt des Systems besteht in der Visualisierung von Datenströmen. Datenströme werden mit Hilfe der Software „Odysseus“, welche von der Universität, speziell die Abteilung IS, bereitgestellt wird, verarbeitet. Für die Simulation der Kraftwerke stehen 30 SunSpots zur Verfügung. Auf diesen Sensoren ist es möglich eine Datenvorverarbeitung durchzuführen, um die Datenströme IEC-konform weiterzuleiten.

Die Darstellung besteht aus unterschiedlichen Sichten. Zur ersten Sicht gehören tabellarische Darstellungen, sowie Histogramme und andere Graphentypen. Zur weiteren Veranschaulichung wird eine 3D-Oberfläche erstellt werden, auf der die Sensoren als Kraftwerke dargestellt werden. Dabei wird die Visualisierung der Kraftwerktypen abhängig von den unterschiedlichen Sensordaten erfolgen.

3.1.4 Ziele und Erfolgskriterien des Projekts

Das Ziel dieser Projektgruppe ist die Erstellung einer Software zur Technologiepräsentation. Dabei werden Datenströme von verschiedenen Sensoren gesendet und im Demonstrator visualisiert. Die Sensoren simulieren Kraftwerke, welche beispielsweise Windkrafträder, Solarzellen oder Erdwärmekraftwerke darstellen.

Das System wird lauffähig und sicher sein. Die Software wird während der Präsentation, sowie durch das eigenständige Nutzen der Interessenten sehr stark strapaziert. Die Stabilität wird daher während der gesamten Präsentationszeit und der Nutzung der User gegeben sein.

3.2 Beschreibung des Projektes

In diesem Abschnitt werden die Rahmenbedingungen zur Durchführung dieses Projektes beschreiben.

3.2.1 Projektorganisation

Die Mitarbeiter des Projektes sind:
Projektleiter: Marcel Sommerlatt
Qualitätsmanager und **stellvertretender Projektleiter**: Christian Hülsen
Schnittstelle: Dennis Geesen

Team View:

- Simon Flandergan (Teamleiter)
- Tobias Witt

Team Odysseus:

- Wolfgang König
- Kai-Andre Pancratz (Teamleiter)
- Jan Vornberger

Team SunSpot:

- Jan-Ole Brode
- Gunnar Schulte-Loh
- Ulf Truderung (Teamleiter)

3.2.2 Beschreibung der Rollen

Im Folgenden werden die vergebenen Rollen näher beschrieben.

Projektleiter

Der Projektleiter ist zuständig für die Koordination und Einteilung der Mitarbeiter und die Planung des Projektablaufs. Er erstellt die anfallenden Dokumente für den Kunden und ist für den Kontakt zu diesem zuständig.

Stellvertretender Projektleiter

Als Vertretung des Projektleiters kommt der stellvertretende Projektleiter zum Einsatz. Er unterstützt den Projektleiter bei anfallenden Tätigkeiten.

Qualitätsmanager

Der Qualitätsmanager ist für die Erstellung von Testplänen und die Einhaltung der Tests zuständig.

Schnittstelle

Aufgrund der besonderen Komplexität bei der Entwicklung von Schnittstellen zwischen den einzelnen Systemen wurde ein Schnittstellenbeauftragter bestimmt. Besonders zwischen den SunSpots und Odysseus erstellt er dazu passende Wrapper und sorgt für einen funktionierenden Datenaustausch.

Team View

Die Zuständigen für die View erstellen die komplette Ausgabe des Programms und bereiten die Daten visuell auf.

Team Odysseus

Das Datenstrommanagementwerkzeug Odysseus an die Bedingungen für das Projekt anzupassen ist Aufgabe des Odysseus-Teams.

Team SunSpot

Für das Team SunSpot erzeugt die Sensordaten mittels der SunSpots und überträgt diese in einem IEC-Standard entsprechenden Format an Odysseus.

3.2.3 Räumliche Gegebenheiten

Der Projektgruppe stehen Räume im Softwarelabor und der Universität für Teammeetings zur Verfügung. Die Entwicklungsarbeiten geschehen in Heimarbeit oder in den oben genannten Räumlichkeiten.

3.2.4 Standards und Richtlinien

Die Bedienbarkeit der Software wird durch die Einhaltung der „Java Look and Feel Design Guidelines“ [1] gewährleistet. Außerdem unterliegt sämtlicher Quellcode den „Code Conventions for the Java Programming Language“ [2].

In dem Projekt verwenden wir außerdem den IEC 61400-25 [IEC06]Standard für Windkraftanlagen. Dieser Standard legt fest, wie die Kommunikation für die Überwachung und Steuerung von Windkraftwerken ablaufen muss.

3.2.5 Lieferumfang

Die Auslieferung des Endproduktes erfolgt auf CD oder DVD. Folgende Auflistung wird im Lieferumfang enthalten sein:

- Projektplan
- Lastenheft
- Pflichtenheft
- Quellcode
- Benutzerhandbuch
- Präsentationsdrehbuch
- Messeflyer
- Installationshandbuch

Die angefertigten Dokumente werden als PDF im Studip in der Verantstalltung der Projektgruppe hochgeladen.

3.3 Meilensteine

Diese Termine stellen den ungefähr geplanten Ablauf des Projektes dar. Sie sind nicht endgültig und die Gruppenleitung behält sich vor, einzelne Meilensteine zu verschieben. Sollte es zu einer Verschiebung kommen, wird der Kunde umgehend darüber informiert.

- **Fr 16.01.09**: Angebot
- **Fr 16.01.09**: Lastenheft

[1] `http://java.sun.com/products/jlf/ed2/book/index.html`
[2] `http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html`

- **Fr 06.02.09**: Pflichtenheft
- **Mi 11.02.09**: Abschluss 1.Iteration
- **Do 12.02.09**: Schülerinformationstag der Informatik
- **Do 24.09.09**: Abschluss 2. Iteration
- **Fr 25.09.09**: Nacht der Wissenschaft
- **Mi 30.09.09**: Projektabschluss

3.3.1 Angebot

Ein ausgearbeites Angebot wird dem Kunden überreicht. Dieses Angebot enhält Abschnitte aus dem Lastenheft, sowie den Meilensteinen. Zusätzlich wird ein Kostenvoranschlag für die Software erstellt.

3.3.2 Lastenheft

Das Lastenheft beinhaltet das Angebot und baut darauf weiter auf. Es werden zusätzlich Anwendungsfälle mit den entsprechenden Beschreibungen dargelegt. Außerdem werden die Terminplanungen der Meilensteine im Lastenheft weiter ausgeführt. Das Lastenheft beschreibt das Projekt und die an die Software gestellten Anforderungen aus Sicht des Kunden. Es wird hier nicht besonders stark auf technische Details eingegangen.

3.3.3 Pflichtenheft

Das Pflichtenheft ist die Erweiterung des Lastenhefts um den Entwurf. Zum Entwurf gehören Diagramme wie Klassendiagramme oder Zustandsdiagramme. Auf diesem Dokument aufbauend wird die Software implementiert. Technische Details werden in diesem Dokument weiter vertieft und dargelegt. Weiterhin wird das Pflichtenheft zur Testplanung herangezogen um die Testpläne effizient gestalten zu können.

3.3.4 Abschluss 1.Iteration

Die erste Iteration ist fertiggestellt und getestet. Der Umfang der ersten Iteration wird in Kapitel 3.5.6 beschrieben.

3.3.5 Schülerinformationstag der Informatik

Der Schülerinformationstag der Informatik findet an der Universität Oldenburg statt. Die erste Iteration wird dort auf einem eigenen Stand allen Interessenten vorgeführt.

3.3.6 Abschluss 2. Iteration

Die zweite Iteration inklusive Modultest, Integrationstest und Ausarbeitung sämtlicher Dokumentation ist fertiggestellt und kann an den Kunden übergeben werden. Der genaue Umfang des endgültigen Produktes wird in Kapitel 3.5.7 aufgezeigt.

3.3.7 Nacht der Wissenschaft

Die erste Nacht der Wissenschaft (European Researchers' Night) findet an der Universität Oldenburg statt. Das fertige Projekt wird dort auf einem eigenen Stand allen Interessenten vorgeführt.

3.3.8 Projektabschluss

Der Abschluss des Projektes beinhaltet die Abgabe sämtlicher Dokumentation und der Software. Des Weiteren erfolgt an dieser Stelle die Entlassung der Projektmitarbeiter.

3.4 Vorhandenes System

Das Projekt ist eine Neuentwicklung, wobei auf bereits bestehende Systeme aufgesetzt wird. Bei diesen Systemen handelt es sich zum einen um das Datenstrommanagementsystem (DSMS) „Odysseus“ und zum anderen um die SunSpots, die als Sensoren fungieren. „Odysseus“ wird dabei auf unsere Bedürfnisse angepasst.

3.5 Vorgeschlagenes System

3.5.1 Übersicht

Das System lässt sich in mehrere Ebenen zerteilen. Bei der ersten Ebene handelt es sich um die Sensorenschicht. Hier werden die Daten mittels Sensoren ermittelt und vorverarbeitet, damit sie einen IEC-konformen Datenstrom liefern können. Die Vorverarbeitung kann in einem Zusammenschluss mehrerer Sensoren, einem so genannten Sensornetzwerk, erfolgen, um einen Kraftwerksverbund zu simulieren. Die Sensoren kommunizieren untereinander über Wireless Local Area Network (WLAN) oder über USB mit dem Hostrechner.

Das Datenstrommanagementsystem bildet die zweite Ebene. Das vorgegebene System Odysseus wird hierbei eingesetzt. Es dient zur Verarbeitung und Filterung der Datenströme. Für die Visualisierung werden bestimmte, angefragte Daten aus dem kontinuierlichen Datenstrom extrahiert und an die Visualisierung weitergeleitet. Zusätzlich werden die Datenströme an ein Datenhaltungssystem gesendet, um die Daten über einen längeren Zeitraum zur Verfügung stellen zu können.

Im Datenhaltungssystem, welches die dritte Ebene stellt, werden die verarbeiteten Datenströme gespeichert. Da die Datenmenge durch die Kontinuität der Datenströme exponenziell steigt, werden ältere Daten zu einem Durchschnittswert aggregiert.

Zu der dritten Ebene wird ebenfalls die Visualisierung instanziiert. Diese dient als Schnittstelle zum Benutzer, welcher Eingaben vornehmen kann. Dem Nutzer stehen verschiedene Ansichten zur Verfügung. Eine Ansicht beinhaltet die tabellarische und graphische Aufbereitung der Datenströme. Hier können die Skalierungen der Graphen und Tabellen in Umfang und Zeitrahmen auf die Bedürfnisse des Nutzers angepasst werden. Momentane Ansichten können für den Benutzer gespeichert werden. Die 3D-Ansicht ist eine fiktive Welt in der die Kraftwerke bildlich dargestellt werden können. Es besteht die Möglichkeit sich einzelne Daten durch anklicken der Kraftwerke anzeigen zu lassen. In der 3D-Welt kann der Benutzer frei navigieren. Neben dem Messemodus gibt es einen Administrationssmodus, in diesem ist es möglich neue Anfragen an das Datenstrommanagementsystem zu stellen um so neue Graphen und Tabellen zu erstellen und neue Kraftwerke zu erzeugen. Die Anfragen können sich sowohl auf aktuelle Daten, sowie historische Datensätze beziehen.

3.5.2 Funktionale Anforderungen

Für das zu entwickelte System wurden folgende funktionale Anforderungen identifiziert. Zur Laufzeit des Systems können weitere Sensoren dem System hinzugefügt werden oder aber aus dem System entfernt werden. Die Sensoren bieten die Möglichkeit Temperatur, Bewegung und Lichtintensität zu messen. Das DSMS bietet die verschiedenen, bereits beschriebenen Filter- und Selektionsmethoden.

Da Messebesucher eigenständig mit dem System arbeiten dürfen, muss eine versehentliche Abschaltung des Systems unterbunden werden. Weiter hat der Messebesucher die Möglichkeit sich aus den vorhandenen Anfragen Graphen und Tabellen zu erstellen. Der Administrator kann mit Hilfe eines Passworts in den Administrationsmodus wechseln. Dort stehen ihm Funktionen wie neue Anfragen Erstellen und das Hinzufügen oder Entfernen von Kraftwerken zur Verfügung. Nur der Administrator kann das System beenden.

3.5.3 Nicht Funktionale Anforderungen

Bedienbarkeit

Das System wird mit Maus und Tastatur bedient. Dabei wird die Bedienbarkeit intuitiv sein, d.h. es muss auch für Laien ohne große Erfahrung im Umgang mit Präsentationssoftware bedienbar sein, ohne dass diese vorher ein Handbuch lesen müssen. Alle möglichen Fehleingaben werden abgefangen.

Zuverlässigkeit

Das System wird während der Präsentationszeit stabil laufen. Einen möglichen Ausfall einzelner Sensoren wird dahingehend toleriert, dass eine Fehlermeldung an den Nutzer ausgegeben wird. Das Gesamtsystem darf dadurch nicht beeinträchtigt werden. Es kann dank der Datenhaltung immer noch auf die historischen Daten des ausgefallenen Kraftwerks zugegriffen werden.

Leistung

Das System muss mit einer Sensorlast von bis zu 30 Sensoren arbeiten können ohne das es zu Lasten der Stabilität geht. Die Visualisierung muss zu jedem Zeitpunkt über 26fps (min. Bilder pro Sekunde die das Menschliche Auge als flüssige Bewegung braucht) erreichen. Für die Vorverarbeitung der Datenströme werden bei zu großer Last durch die Datengewinnung und Datenverarbeitung Sensoren zu einem Netzwerk zusammengeschlossen. Hierbei übernimmt ein Sensor die komplette Verwaltung anderer Sensoren und sammelt selbst keine Daten mehr.

Unterstützung

Die Struktur des Systems wird modular gehalten, um die Möglichkeit der Erweiterung durch andere Projekte zu gewährleisten.

Implementierung

Die Entwickler schreiben den Quelltext nach den Richtlinien der Java Code Convention. Kommentare werden entsprechend der JavaDoc Convention erstellt. Auftretende Ausnahmebedingungen werden abgefangen und gesondert bearbeitet, so dass dem Benutzer keine undurchsichtgen Fehlermeldungen oder Programmabstürze angezeigt werden.

Die Software wird in Form eines Eclipse-PlugIn realisiert.

Schnittstellen

Durch den modularen Aufbaus des Systems werden verschiedene Schnittstellen zur leichteren Erweiterbarkeit angeboten. Da die Daten auf den SunSpots soweit vorverarbeitet werden, dass sie dem IEC-Standard entsprechen, besteht auch die Möglichkeit IEC-konforme Kraftwerke direkt an das System anzuschließen.

Betrieb

Der Betrieb des Systems wird von der Gruppe während der Laufzeit der Projektgruppe sichergestellt.

Verpackung

Vorerst wird keine Verpackung für unser Produkt angeboten.

Rechtliches

Der Quelltext unterliegt dem Urheberrecht und ist Eigentum der Projektgruppe und der Universität Oldenburg.

Benutzerschnittstelle

Eine detaillierte Entwurf der Benutzerschnittstelle ist noch nicht vorhanden.

3.5.4 Qualitäts- und Dokumentationsanforderungen

Qualitätsanforderung

Unsere Software wird in festgelegten Intervallen getestet. Zuständig für die Erstellung und Überwachung der Testpläne ist der Qualitätsmanager. Durch vielfältige Test werden auftretende Fehler gefunden und anschließend behoben. Vor jeder Abgabe eines Prototypen werden die Module vom ganzen Team geprüft.

Das Projekt wird in mehreren Iterationen geplant und durchgeführt. Dadurch kann eine Qualitätssteigerung durch den erzielten Lerneffekt erreicht werden.

Dokumentationsanforderungen

Das System umfasst eine vollständige Dokumentation. Dem Benutzer wird eine umfassende Hilfe und Dokumentation in englischer Sprache zur Verfügung gestellt, die ihm in allen Situationen begleitende Informationen liefern wird.

Die Dokumentation wird aus verschiedenen Teilen bestehen. Zum einen wird eine Installationsanleitung sowie ein Handbuch dem Administrator zur Verfügung gestellt. Für Präsentationen auf Messen wird ein umfassendes Drehbuch erstellt und eine Vorlage für Flyer für die Messebesucher angefertigt. Das Drehbuch wird für eine Präsentation über mehrere Minuten ausgelegt werden, die den vollen Leistungsumfang des Systems wiedergibt.

3.5.5 Produktstrukturplan

Der Produktstrukturplan stellt eine Spezifikation des zu entwickelnden Systems dar. Dazu werden die Funktionen und Objekte des Systems in einen hierarchischen Zusammenhang gebracht. Dies ermöglicht eine übersichtliche Darstellung des zu entwickelnden Systems.

In der Projektgruppe werden die Arbeitspakete anhand des Produktstrukturplans abgeleitet. Die einzelnen Elemente bilden also die Grundlage auf denen der Projektplan mit den Meilensteinen ausgearbeitet wurde.

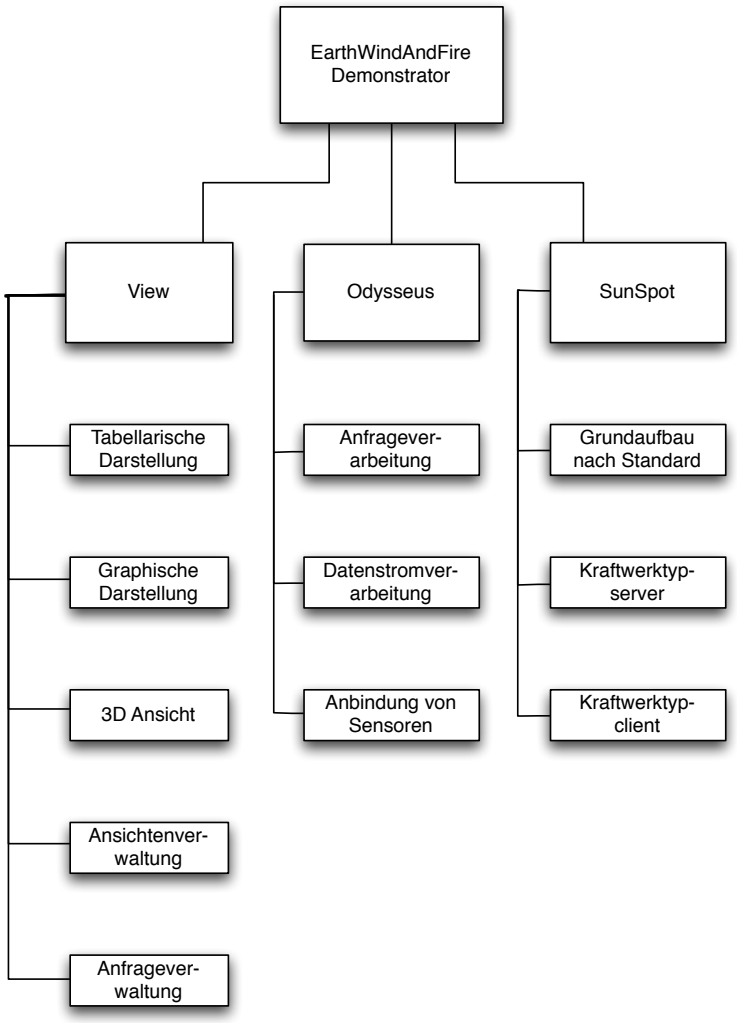


Abbildung 3.1: Produktstrukturplan

View - Tabellarische Darstellung

Die View besteht aus mehreren Modulen. Die tabellarische Darstellung dient dazu, die ankommenden Daten in einer sich ständig selbst aktualisierenden Tabelle anzeigen zu können. Es ist möglich mehrere Spalten in einer Tabelle anzuzeigen, damit verschiedene Attribute, bzw. ermittelte Werte angezeigt werden können. Die Tabellenbeizeichnungen entsprichen initial dem Namen des SunSPOTs, der zur Zeit die Daten schickt. Der User kann allerdings jederzeit einen eigenen Namen für die Tabellen erstellen.

View - Graphische Darstellung

Die graphische Darstellung dient dazu, die ankommenden Daten in einem sich ständig selbst aktualisierenden Graphen anzeigen zu können. Der User hat dabei die Möglichkeit den Graphentyp festzulegen, damit die Daten in einem dazu passenden Graphen angezeigt werden können. Für die Temperatur steht beispielsweise ein Thermometer-Graph bereit, der die Temperaturen in einem Thermometer darstellt. Es wird außerdem eine Funktion bereit stehen, in denen die Namen der Graphen angepasst werden können. Einige Graphentypen werden eine dynamische Anpassung des Graphenverlaufs vornehmen können, ohne dabei den gesamten Graphen neu zeichnen zu müssen.

View - Ansichtenverwaltung

In der Ansichtenverwaltung hat der User die Möglichkeit sich seine eigens erzeugten Ansichten/Perspektiven zu speichern. So hat der User jederzeit die Möglichkeit seine gespeicherte Ansicht wieder aufzurufen, um in seiner gewohnten Umgebung zu arbeiten. Ein Wechsel zwischen den verschiedenen Ansichten ist zu jeder Zeit möglich. Die Ansichten werden in einer Hierachie abgebildet, sodass Nutzer einfach ihre Ansichten wiederfinden können.

View - Anfrageverwaltung

Um Daten empfangen zu können, müssen im Datenstrommanagementsystem CQL-Anfragen installiert werden können. Die View bietet dazu einen Wizard an, um dynamisch Anfragen stellen zu können. Dies hat den Vorteil, dass der User Daten erfragen kann, die er für eine Präsentation benötigt.

View - 3D Ansicht

Zusätzlich zu den Graphen und den Tabellen wird es eine 3D-Animation gegeben. Diese Animation kann der User erkunden und so auf einer Karte sehen, welche Kraftwerke gerade Strom liefern und welche nicht. In dieser Welt wird ein Freiflugmodus angeboten, indem er frei durch die Welt fliegen kann.

Odysseus - Anfrageverwaltung

Werden dynamische Anfragen an Odysseus gestellt, so wird Odysseus diese Anfragen verarbeiten und die entsprechenden Daten an die View weiterleiten. Odysseus nutzt verschiedene Operationen, um ankommende Daten so verarbeiten zu können, damit diese in einem für die View kompatiblen Format vorliegen.

Odysseus - Anbindung von Sensoren

Die Daten die angefragt werden, werden von verschiedenen Sensoren geliefert. Um eine Vielzahl von Anfragen verarbeiten zu können, müssen die Sensoren an das System angeschlossen werden. Die Anzahl der Sensoren wird prinzipiell beliebig sein.

Odysseus - Datenstromverarbeitung

Zur Datenstromverarbeitung zählt zum einen die verschiedene Operationen die unterschiedliche Daten kombinieren, um beispielsweise die gefühlten Temperaturen zu erhalten, und zum anderen die Verwaltung von mehreren Datenströmen die von den Sensoren (SunSPOTs) geschickt werden. Dabei ist es wichtig zu einer Anfrage der View die richtigen Daten aus den richtigen Datenströmen zu filtern.

SunSPOT - Grundaufbau nach Standard

Mittel der SunSPOT Sensoren werden Kraftwerke simuliert, indem damit verschiedene Daten erfasst werden, die auch in einem realen Kraftwerk zu finden sind. Um den Demonstrator später theoretisch durch eine reale Windkraftanlage ersetzen zu können, werden als Struktur für die Datenübertragung mit Odysseus verschiedene IEC-Standards herangezogen, die diese Struktur reglementieren. Die Datenübertragung findet dabei mittels XML statt und läuft über das TCP/IP Protokoll.

SunSPOT - Kraftwerkserver

Für die erste Iteration sind zentrale Kraftwerkserver vorgesehen, welche die Daten der SunSPOTs annehmen und Standard-konform weiterleiten. In der zweiten Iteration wird die Standard-konforme Datenübermittlung direkt von den SunSPOTs erfolgen.

SunSPOT - Kraftwerkclient

Für jeden Kraftwerkstyp werden spezifische Clients durch die SunSPOTs instanziiert. Diese ermitteln die für den jeweiligen Kraftwerkstyp relevanten Daten und leiten diese an den Kraftwerksserver weiter.

3.5.6 Erste Iteration

Der Abgabetermin für die erste Iteration ist der 12.02.2009. Die Abgabe wird mit dem Schülerinformationstag (SIT) verbunden. Die PG wird einen Vortrag und eine Livedemonstration halten, um die erste Iteration vorzustellen.

Die erste Iteration beinhaltet einen vertikalen Prototypen mit entsprechender Dokumentation. Der Prototyp wird einen Datenstrom von den SunSPOTs kommend und über Odysseus laufend in der View tabelarisch und graphisch darstellen. Die beigefügte Dokumentation besteht aus dem Angebot, dem Lastenheft und dem Pflichtenheft.

Für die erste Iteration sind folgende Festlegungen gesetzt worden.

View

In der ersten Iteration ist die View fähig die ankommenden Daten zu visualisieren. Dies geschieht auf zwei unterschiedliche Wege. Zum einen werden die Werte tabellarisch und zum anderen graphisch dargestellt.

Tabellen stellen die Daten nur als eine Liste von Werten dar. Diese sind schwierig auszuwerten. Aus diesem Grund werden diese Werte zusätzlich graphisch dargestellt, um einen Überblick über die Daten zu schaffen. Dazu kann sich der User über einen Wizard einen einfachen Graphen zu den entsprechenden Werten erstellen. In der ersten Iteration wird die View prinzipiell alle Wertetypen (Temperatur-, Licht-, Windwerte) ausgeben können, allerdings wird von den SunSPOTs nur ein bestimmter Wertetyp geschickt, dazu aber später mehr.

Um die Tabellen und die Graphen überhaupt erstellen zu können, müssen die Daten aus dem Datenstrom herausgefiltert werden, dazu kann die View eine Anfrage an Odysseus stellen, um die gewünschten Daten zu erhalten. Für die erste Iteration werden die Anfragen nur textuell gestellt, d.h. um eine Anfrage stellen zu können, muss die CQL-Query direkt eingegeben werden. Das System bietet aber eine kleine Vorauswahl an CQL-Queries, um die Erstellung von Anfragen zu erleichtern.

Die Kommunikation zwischen der View und Odysseus findet dabei über Remote Method Invocation statt. Dieses Konzept erlaubt es von der View aus Methoden im Odysseus aufzurufen, um Anfragen zu installieren. Umgekehrt findet auch ein entfernter Methoden Aufruf statt, um die angefragten Daten an die View zu schicken.

Alle angezeigten Elemente lassen sich einzeln vom User in der Position und der dargestellten Größe ändern, so dass sich jeder User eine personalisierte View erstellen und speichern kann.

Die View speichert die persönlichen Einstellungen aller Nutzer persistent. So können die Nutzer bei jedem Aufruf der Anwendung ihre erstellten Perspektiven direkt wiederherstellen und betrachten. Die Einstellungen umfassen dabei jedoch nur den Aufbau der Graphen/Tabellen, sowie deren Anordnung. Die Inhalte werden nicht gespeichert.

Eine weitere Funktion, die das System bietet, ist die Freeze-Funktion. Die Elemente beziehen ihre Daten aus einem Datenstrom, d.h. es werden laufend neue Daten geliefert, die auch angezeigt werden. Dadurch ändern sich ständig die Tabellen und Graphen. Damit sich der User ein Bild über die Daten verschaffen kann, wird ihm die Möglichkeit gegeben, die Daten zum aktuellen Zeitpunkt einzufrieren, sodass er die Daten in aller Ruhe analysieren kann.

Abbildung 3.2 zeigt das momentane Design der View.

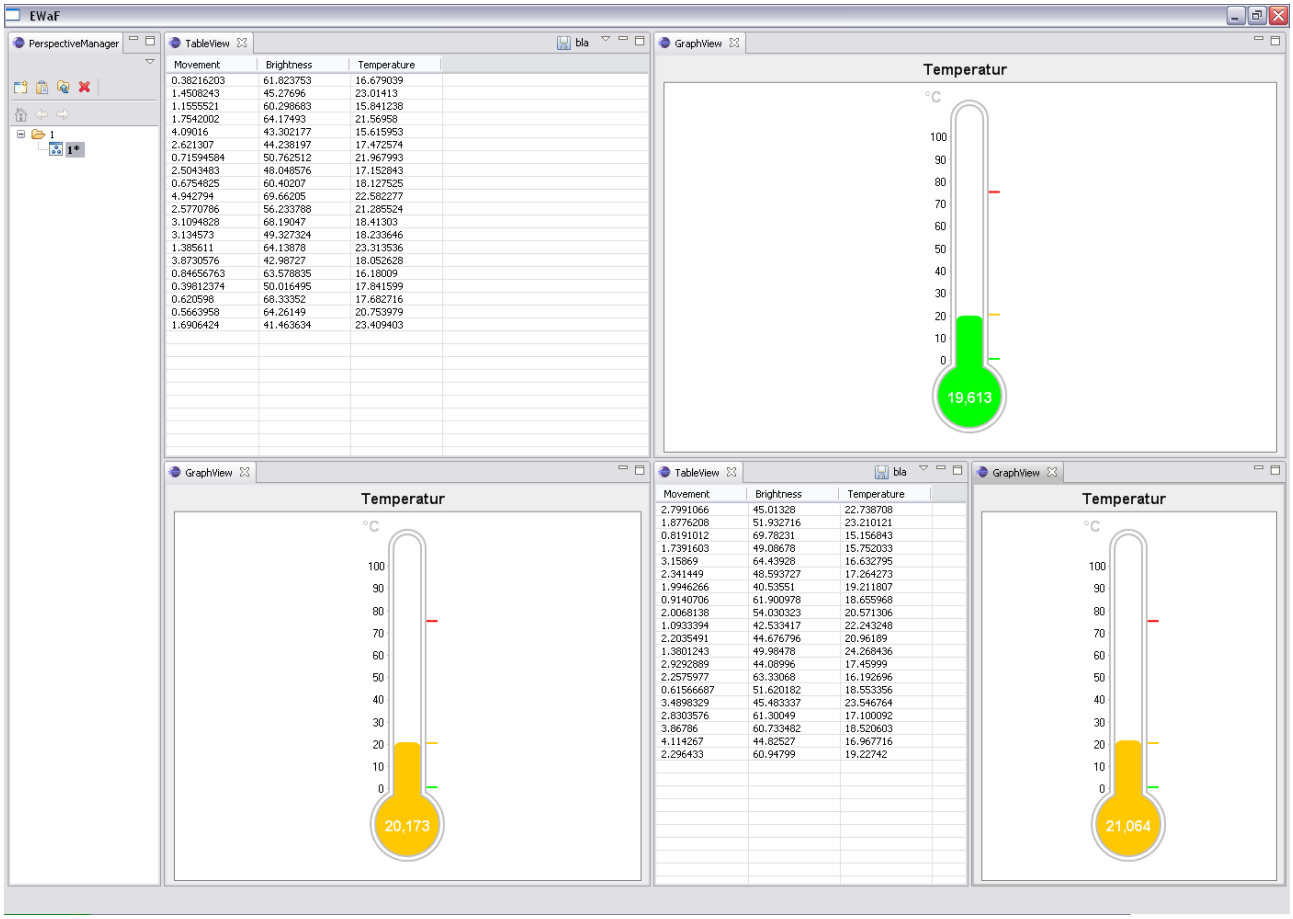


Abbildung 3.2: Screenshot der momentanen View

Odysseus

Odysseus wird in dem Projekt als Datenstrommanagementsystem verwendet. Dieses System hilft die ankommenden Daten der SunSPOTs zu verarbeiten, zu filtern und weiterzuleiten.

Die Ziele, die für die erste Iteration festgelegt wurden, beziehen sich zum einen auf die Daten, die weitergeleitet werden und zum anderen auf die Anfragen, die gestellt werden können.

Odysseus wird nach Abschluss der ersten Iteration in der Lage sein, Daten vom Kraftwerktyp 'Windkraft' weiterzuleiten. Dazu werden XML-Daten ausgewertet, die von den SunSPOTs an Odysseus geschickt worden sind. Diese Daten werden in eine bestimmte Struktur gebracht, damit diese von Odysseus besser verarbeiten werden können.

Um beliebig viele SunSPOTs verwalten zu können, muss Odysseus das Datenformat der verschickten Daten Odysseus kennen. Auf diese Art und Weise wird es möglich sein, dass die Daten von den SunSPOTs über nur einen Datenstrom verschickt werden können. Dazu wird in Odysseus eine statische Create-Anweisung bereits vorinstalliert sein. Diese Anfrage legt ein Schema für die Daten an, die durch den Datenstrom laufen werden, so dass Select-Anfragen gestellt werden können. So wird beispielsweise das Attribut W von der Klasse WTUR abfragt werden können. Die Select-Anfragen, die gestellt werden können, sind vom Typ: „SELECT * FROM test:wind WHERE w > 25“. Der Wert 25 kann hierbei variiert werden. Weitere dynamische Anfragen - vorallem um neue Ströme zu erzeugen - sind in der ersten Iteration nicht vorgesehen.

Odysseus wird außerdem bereit sein, Daten zu verarbeiten, die an den IEC-Standard angelehnt sind.

SunSPOT

Zentral für die erste Iteration ist der zu entwickelnde Demonstrator. An diesem werden mittels SunSPOTs relevante Daten erfasst und durch die Kraftwerksserver anhand von IEC-Normen in Datenpakete zusammengestellt. Da es sich beim Demonstrator um ein Windkraftwerk handelt und insbesondere für Windkraftanlagen IEC-Normen vorliegen

wird der Schwerpunkt bei der Entwicklung auch hier liegen. Die Simulation von Solarkraftwerken und Geothermalkraftwerken wird grundlegend ermöglicht, jedoch in dieser Iteration nicht forciert. Ein besonderes Augenmerk kommt der Dokumentation aller durch die IEC-Standards vorgegebenen Datenstrukturen zu, um das Anwenden jener zu erleichtern.

Eine vollständige Abdeckung der verschiedenen IEC-Standards wird in Iteration 1 noch nicht angestrebt, u.a. weil noch nicht alle dazu benötigten Standards vorliegen. Die Datenübertragung ist hier daher weitestgehend am Standard angelehnt, erfüllt diesen jedoch noch nicht voll. Für Solar- wie Geothermalkraftwerke liegen noch keine fertigen Standards vor, so dass hier Datenstrukturen auf Basis der IEC 61400-25-Norm für Windkraftwerke, wo sinnvoll, angenähert werden.

Die SunSPOTs werden so implementiert, dass sie erfasste Daten an einen virtuellen Kraftwerkserver leiten, den sog. SpotHost. Dieser verarbeitet die ermittelten Daten von den verschiedenen Kraftwerken vor und leitet diese mittels einer TCP-Verbindung an Odysseus weiter.

Für den SIT werden wir eine Demo-Konfiguration unter Verwendung eines Demo-Windrads aufbauen. Dabei wird u.a. die Rotationsenergie per Dynamo in eine elektrische Spannung gewandelt und von den SunSPOTs abgenommen. Diese Daten werden dann im XML-Fromat zur Weiterverarbeitung an Odysseus weitergeleitet.

3.5.7 Zweite Iteration

Die Zweite Iteration wird am 24.09.2009 ihren Abschluss finden und die Projektgruppe beenden. Der fertiggestellte Demonstrator wird auf der Nacht der Wissenschaft am 25.09.2009 vorgestellt. Dazu werden wir den Besuchern die Möglichkeit geben, die Software zu testen und mit dem Versuchsaufbau verschiedene Szenarien durchzuführen.

Zum Umfang der zweiten Iteration gehört zum einen der Demonstrator und zum anderen eine Umfangreiche Dokumentation. Die Dokumentation besteht dabei aus dem Handbuch, der Projektdokumentation (Pflichtenheft, Lastenheft), einem Drehbuch zur Vorstellung des Demonstrators und einem Flyer für die Messebesucher.

View

Die Zweite Iteration erweitert die View vor allem um eine 3D-Ansicht. In dieser wird sich der Nutzer frei bewegen und den verschiedenen Kraftwerkstypen bei der Arbeit zusehen können. Die Funktionalitäten erweitern sich dahingehend, dass die View zusätzlich zu aktuellen Daten Langzeitdaten darstellen kann.

Der Demonstrator soll zur Vorstellung auf Messen benutzt werden können. Dazu werden wir in der zweiten Iteration einen Kiosk-Modus hinzufügen. Dieser Modus erlaubt es Nutzern, die das System nicht genau kennen, das System auszuprobieren. Anders als im Admin-Modus ist die Anfrageverwaltung ausgeschaltet, sodass der Nutzer nicht (un)beabsichtigt das System zum Absturz bringen kann. Für die Moderation und Vorstellung des Systems kann der Moderator dann wieder in den Admin-Modus wechseln. Die Authentifizierung funktioniert über eine Passwortabfrage. Im Admin-Modus kann kann der Moderator neue Tabellen und Graphen hinzufügen. Weiterhin kann er neue Anfragen stellen um andere bzw. aggregierte Daten zu erhalten. In der 3D-Ansicht hat der Moderator zusätzlich die Möglichkeit mehrere Kraftwerktypen (beispielsweise mehrere Windräder) zu einem Komplex (anderes Symbol) zusammenzuschließen.

Außerdem erhält die View eine Funktion in der angezeigt werden kann, wie viel Energie alle Kraftwerkelemente (SunSPOTs) zusammen erzeugen, um immer einen Überblick darüber zu haben, wie viel Energie das virtuelle Kraftwerk erzeugt.

Odysseus

Das Team Odysseus wird in der zweiten Iteration fähig sein, mehrere Anfragen parallel zu verarbeiten. Dabei kann die View Anfragen stellen, um diese in Odysseus zu integrieren. Es wird also möglich sein dynamische Anfragen zu verarbeiten, um bestimmte Datensätze zu erhalten. Des Weiteren wird eine Datenbank integriert. Diese hat die Aufgabe, die Daten vorzuhalten, so dass Anfragen gestellt werden können, die Daten aus einem größeren Zeitraum zusammenfassen und an die View weiterleiten. Daraus resultiert die Verarbeitung von mehreren Datenströmen.

Da in der ersten Iteration nur Windkraftdaten weitergeleitet werden können, werden in der zweiten Iteration sämtliche Kraftwerkstypdaten weitergeleitet werden können. Dazu wird der Wrapper angepasst werden.

Odysseus soll so angepasst werden, dass eine automatische Fehlerbehebung bei einem Verbindungsverlust zu den SunSPOTs implementiert wird. Eine Lösung dazu kann beispielsweise so aussehen, dass versucht wird die Verbindung neu aufzubauen und erst nach mehrmaligen Fehlversuchen eine Fehlerbehandlung angestoßen wird.

SunSPOT

Nach Abschluss der zweiten Iteration werden die SunSPOTs in der Lage sein die Daten IEC-Standard-konform zu verschicken. Dazu gehört auch, dass die Datentypen dem Standard entsprechen. Anpassungen im Bereich der XML-Kommunikation sind dazu notwendig. Dieser Aufwand ist notwendig, um eine bisher virtuelle Kraftwerksanlage durch eine reale ersetzen zu können.

Statt die Kommunikation über Kraftwerksserver abzuwickeln sollen die SunSPOTs in der zweiten Iteration in der Lage sein direkt mit Odysseus Standard-konforme Datenströme auszutauschen. Kraftwerke werden dabei mittels Sensornetzwerke simuliert. Durch Sensornetzwerke ist es außerdem möglich Werte verschiedene SunSPOTs miteinander zu verbinden, um z.B. die gefühlte Temperatur zu ermitteln.

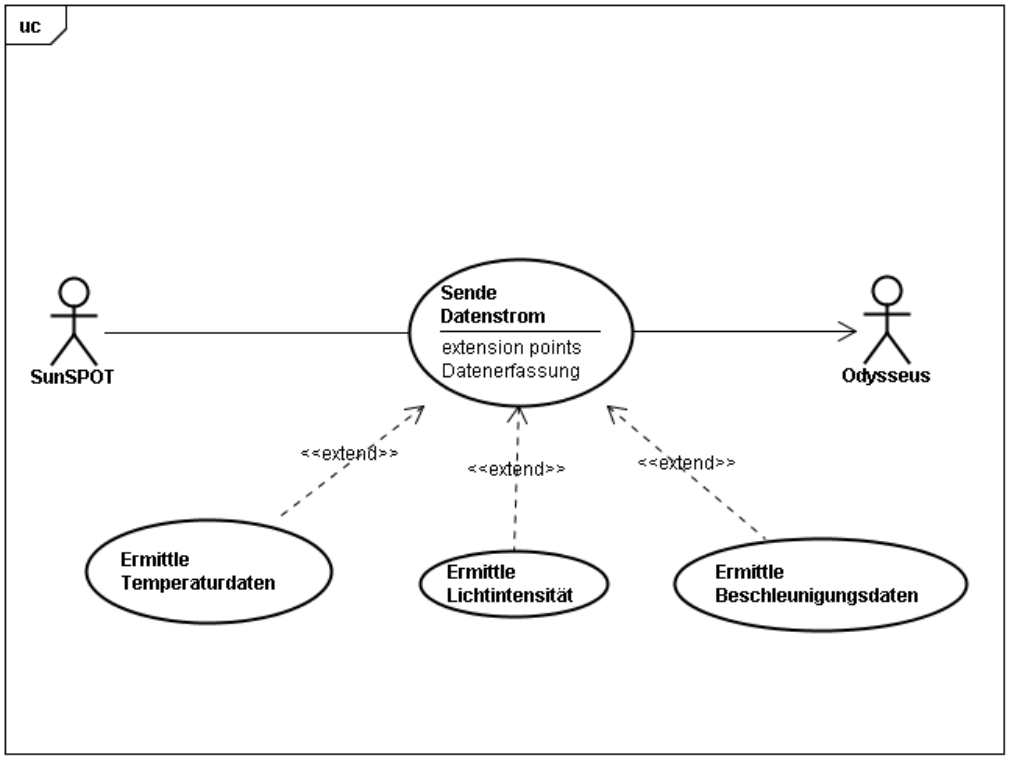
Zusätzlich können Ereignisse auftreten, die bestimmte Aktionen erforderlich machen. Im einfachsten Fall könnte es auch sein, dass ein Akku eines SunSPOT Sensors zur Neige geht. Damit nicht davon ausgegangen werden muss, dass ein ganzes Kraftwerk ausgefallen ist, könnte eine Statusmeldung bei Eintritt in einen kritischen Ladebereich für einen Austausch des Akkus sorgen. Somit wird eine frühzeitige Reaktion auf mögliche Fehler erleichtert.

3.6 Anwendungsfälle

In den folgenden Anwendungsfällen sind diese Akteure vertreten:

- SunSPOT: Der Akteur SunSpot gibt die Sensoren selber wieder.
- Odysseus: Der Akteur Odysseus ist das DSMS.
- SPOTHost: Der Akteur SPOTHost ist der Prozess, der die Daten an Odysseus sendet.
- View: Der Akteur View ist die datenempfangende Anzeigeplattform.
- Admin: Ein Admin hat vollen Zugriff auf alle Funktionen der View.
- Professioneller Anwender: Ein professioneller Anwender kann ein Präsentierender sein, er kennt sich mit der Bedienung der Software aus. Zudem hat er vollen Zugriff auf alle Fnktionen.
- Anwender / Besucher: Besucher bekommen keinen vollen Zugriff auf alle Funktionen, so dass sie die Softwarekomponenten nicht unbeabsichtigt beschädigen können.

3.6.1 SunSPOT



Anwendungs-fallname	**SP01 - Simuliere Kraftwerk**
Akteure	SunSPOT, Odysseus

Ereignisfluss	1. SunSPOT wird konfiguriert Daten zu erfassen. 2. Es können alle, einige oder keine Datenquellen ausgewählt werden. Als Datenquellen stehen zur Verfügung: • Temperatur (s. SP-02) • Lichtintensität (s. SP-03) • Beschleunigung (s. SP-04) 3. SunSPOT sendet gewünschten Datenstrom oder nichts, wenn keine Datenquelle angegeben wurde.
Anfangsbedingungen	SunSPOT ist betriebsbereit
Abschlussbedingungen	SunSPOT zum Senden eines Datenstroms bereit oder am Senden
Qualitätsanforderungen	-

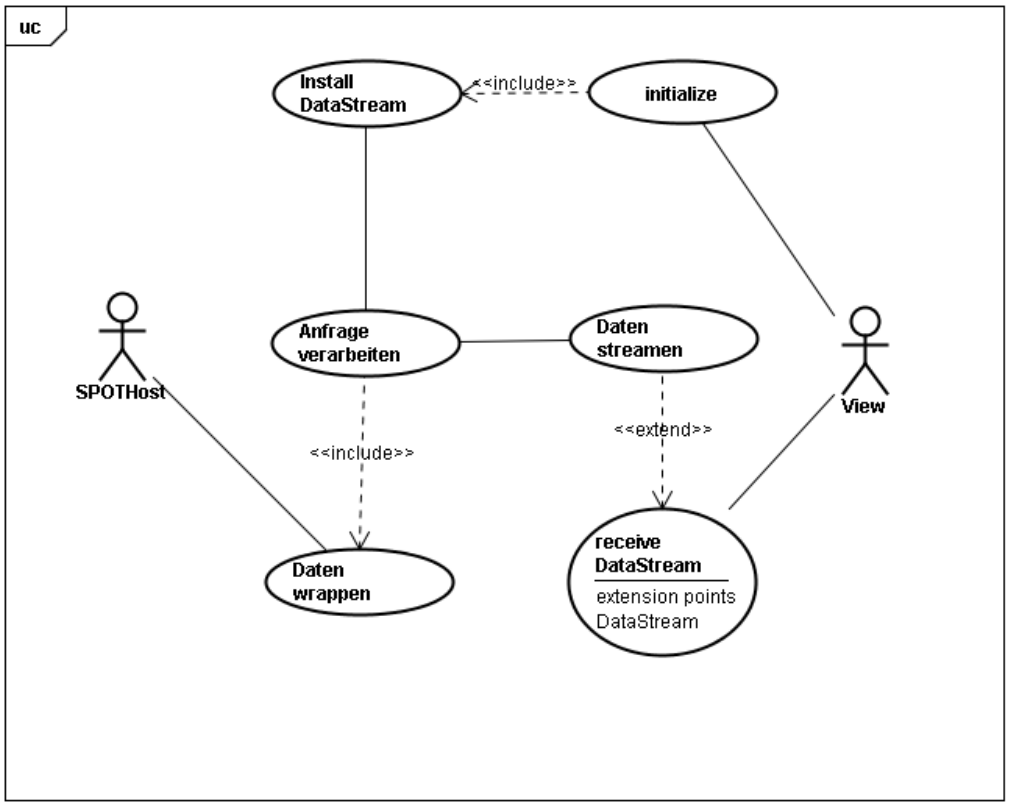
Anwendungsfallname	**SP02 - Ermittle Temperaturdaten**
Akteure	SunSPOT
Ereignisfluss	1. SunSPOT ermittelt Umgebungstemperatur 2. SunSPOT erzeugt XML-Nachricht. 3. XML wird an SPOTHost gesendet.
Anfangsbedingungen	Datenquelle Temperatur angefordert
Abschlussbedingungen	Temperaturdaten können bereitgestellt werden
Qualitätsanforderungen	-

Anwendungsfallname	**SP03 - Ermittle Lichtintensität**

Akteure	SunSPOT
Ereignisfluss	1. SunSPOT ermittelt Lichtintensität des auf ihn von oben auftreffenden Lichts. 2. SunSPOT erzeugt XML-Nachricht. 3. XML wird an SPOTHost gesendet.
Anfangsbedingungen	Datenquelle Lichtintensität angefordert
Abschlussbedingungen	Lichtintensitätsdaten können bereitgestellt werden
Qualitätsanforderungen	-

Anwendungsfallname	**SP04 - Ermittle Beschleunigungsdaten**
Akteure	SunSPOT
Ereignisfluss	1. SunSPOT ermittelt die eigene Beschleunigung. 2. SunSPOT erzeugt XML-Nachricht. 3. XML wird an SPOTHost gesendet.
Anfangsbedingungen	Datenquelle Beschleunigung angefordert
Abschlussbedingungen	Beschleunigungsdaten können bereitgestellt werden
Qualitätsanforderungen	-

3.6.2 Odysseus

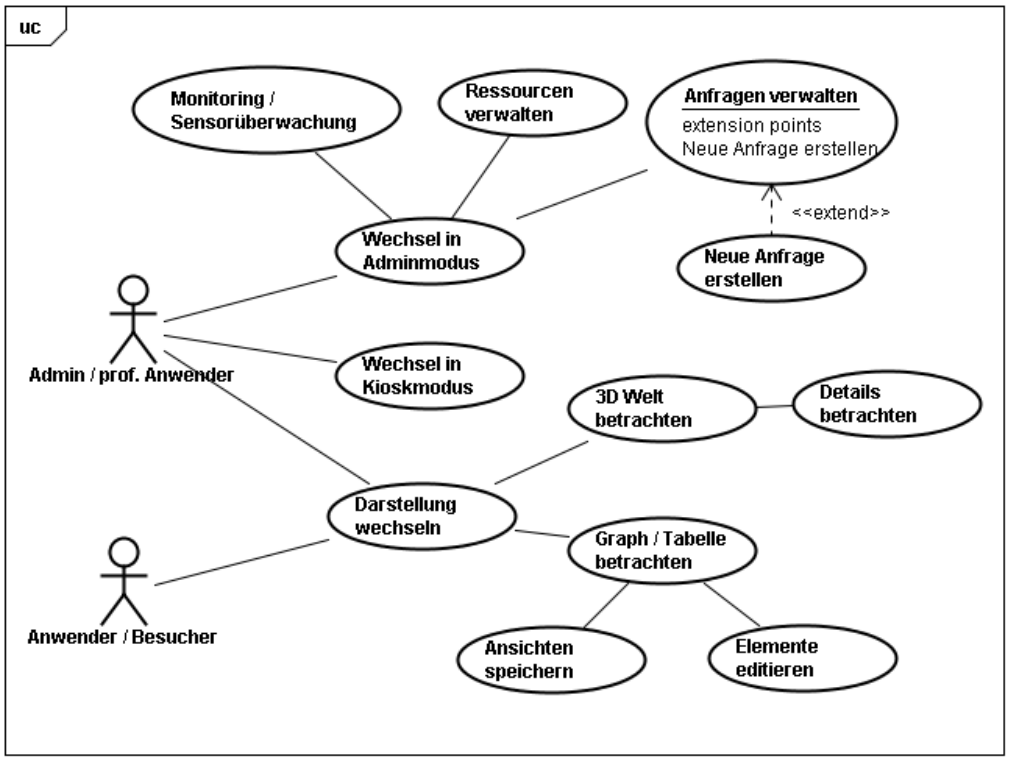


Anwendungsfallname	**Datenstrominitialisierung**
Akteure	View, Odysseus
Ereignisfluss	1. View initialisiert den Datenstrom mittels einer Query
Anfangsbedingungen	• Odysseus und die View sind betriebsbereit • View befindet sich im Administratormodus
Abschlussbedingungen	Damit die Query in Odysseus installiert werden kann, muß diese Query ein korrektes Format aufweisen
Qualitätsanforderungen	-

Anwendungsfallname	**Empfang der Messdaten**
Akteure	SPOTHost, Odysseus
Ereignisfluss	1. SPOTHost sendet Messdaten in Form eines oder mehrerer Datenströme an den Wrapper 2. Wrapper formatiert die Daten, damit Odysseus diese verarbeiten kann
Anfangsbedingungen	Der Wrapper muß Ein- und Ausgangsformat des Datenstroms kennen
Abschlussbedingungen	Messdaten können vom SPOTHost bereitgestellt werden
Qualitätsanforderungen	Wrapper soll den Datenstrom möglichst ohne Verzögerung verarbeiten und formatiert an Odysseus weiterleiten

Anwendungsfallname	**Odysseus Anfrageverarbeitung**
Akteure	SPOTHost, View, Odysseus
Ereignisfluss	1. View initialisiert den Datenstrom mittels einer Query 2. Die Daten der SunSPOTs werden über den SPOTHost an an einen Wrapper geschickt 3. Der Wrapper verarbeitet die Daten so, dass diese von Odysseus verarbeitet werden können 4. Odysseus verarbeitet die Daten 5. Die verarbeiteten Daten können nun von der View empfangen werden
Anfangsbedingungen	SPOTHost, View und Odysseus müssen betriebsbereit sein
Abschlussbedingungen	Damit die View Datenströme empfangen kann, muß der SPOTHost Daten senden.
Qualitätsanforderungen	Odysseus soll den Datenstrom möglichst ohne Verzögerung verarbeiten

3.6.3 View



Anwendungs-fallname	**Wechsel in Administrationsmodus**
Akteure	Initiiert von Administrator oder professioneller Anwender (Initiator).
Ereignisfluss	1. Der Initiator klickt auf den Menüpunkt zum Wechsel in den Administrationsmodus. 2. Das System fordert den Benutzer zur Eingabe eines Passworts auf, um Unberechtigten den Zugriff zu verweigern. Nutzer kann diesen Prozess abbrechen und verlässt den Anwendungsfall ohne das in den Administratormodus gewechselt wird. 3. Der Initiator gibt das Passwort ein. 4. Das System überprüft die Korrektheit des Passworts. Falls dieses korrekt ist, wird die Oberfläche um die Monitoring, Ressourcenverwaltung und Anfrageverwaltung erweitert. Andernfalls wird Schritt 2 wiederholt.
Anfangsbe-dingungen	System ist im Kioskmodus.
Abschlussbe-dingungen	System ist in den Administrationsmodus gewechselt.

Qualitätsanforderungen	• Die zusätzlichen Bedienelemente müssen ins Schema der restlichen Bedienoberfläche passen. • Dem Nutzer muss aufgezeigt werden, was sich an neuen Bedienmöglichkeiten ergeben hat. • Die Passwortabfrage muss zuverlässig sein und darf nicht umgangen werden können.

Anwendungsfallname	**Monitoring/Sensorüberwachung**
Akteure	Initiiert von Administrator oder professioneller Anwender (Initiator).
Ereignisfluss	1. Der Nutzer beobachtet die Sensoren. 2. Bei Defekten oder anderen Unregelmäßigkeiten werden diese Angezeigt. 3. Der Nutzer kann entsprechend auf die Fehler reagieren.
Anfangsbedingungen	Das System befindet sich im Administrationsmodus.
Abschlussbedingungen	Fehler bei den Sensoren werden erkannt und können behoben werden.
Qualitätsanforderungen	• Fehler müssen eindeutig beschrieben werden.

Anwendungsfallname	**Ressourcen verwalten**
Akteure	Initiiert von Administrator oder professioneller Anwender (Initiator).
Ereignisfluss	1. Neue Sensoren werden aktiviert oder 2. Sensoren werden aus den Netzwerk entfernt oder 3. Sensoren werden ausgetauscht. 4. Diese Änderungen werden in das laufende System integriert.
Anfangsbedingungen	Das System befindet sich im Administrationsmodus.
Abschlussbedingungen	Das System reagiert auf die Änderungen bei den Sensoren und zeigt diese Änderungen entsprechend an.

Qualitätsanforderungen	• Neue Sensoren müssen korrekt integriert werden, nicht mehr vorhandene Sensoren dürfen nicht mehr abgefragt werden.

Anwendungsfallname	**Anfrage verwalten**
Akteure	Initiiert von Administrator oder professioneller Anwender (Initiator)
Ereignisfluss	1. Der Initiator klickt auf die Schaltfläche zur Verwaltung der Anfragen. 2. Das System zeigt dem Nutzer die aktiven Anfragen an. Dabei besteht sowohl die Möglichkeit aktive Anfragen zu entfernen oder neue Anfragen zu gestalten.
Anfangsbedingungen	System ist im Administrationsmodus.
Abschlussbedingungen	Anfragen wurden aktualisiert. Der Nutzer wurde an „Neue Anfrage“ Definieren weitergeleitet, falls er eine neue Anfrage erstellen wollte.
Qualitätsanforderungen	• Gewünschte Änderungen werden korrekt an das DSMS weitergeleitet.

Anwendungsfallname	**Wechsel in Kioskmodus**
Akteure	Initiiert von Administrator oder professioneller Anwender (Initiator).
Ereignisfluss	1. Der Initiator klickt auf den Menüpunkt zum Wechsel in den Kiosmodus. 2. Das System entfernt Monitoring, Ressourcenverwaltung und Anfrageverwaltung aus der Benutzeroberfläche.
Anfangsbedingungen	System ist im Administrationsmodus.
Abschlussbedingungen	System ist in den Kioskmodus gewechselt.

Qualitätsanforderungen	• Geöffnete Adminbedienelemente müssen geschlossen werden.

Anwendungsfallname	**Darstellung wechseln**
Akteure	Initiiert von Administratoren, professioneller Anwender, Anwendern oder Besuchern
Ereignisfluss	1. Der Initiator klickt auf den Menüpunkt zum Wechsel der Darstellung. 2. Die Darstellung wechselt zwischen 3D Welt und Graphen und Tabellen
Anfangsbedingungen	System ist im Kioskmodus.
Abschlussbedingungen	Darstellung wurde gewechselt.
Qualitätsanforderungen	• Die Darstellung muss komplett gewechselt werden.

Anwendungsfallname	**3D-Welt (Zusammenfassung mehrere Anwendungsfälle)**
Akteure	Initiiert von Administrator oder professioneller Anwender oder Anwender bzw. Besucher (Initiator)

Ereignisfluss	1. Der Initiator klickt auf die Schaltfläche zum Wechsel in den 3D-Modus. 2. Das System stellt eine dreidimensionale Übersicht einer Landschaft bereit, in der das Netzwerk der Kraftwerke, Sensoren und Verbraucher dargestellt wird. Die Kamera ist in einem 45° Winkel auf die Landschaftsoberfläche geneigt. Je nach Anzahl an Objekten, die darzustellen sind, werden die Kraftwerke in unterschiedlichen Abstraktionsgraden animiert, als Symbol oder als Liste angezeigt. Linien stellen die Verbindungen von Kraftwerken und Sensoren bzw. Verbrauchern dar. Über den Kraftwerken, Sensoren und Verbrauchern werden zusammengefasst Statusinformationen des jeweiligen Objekten angezeigt. Besondere Ereignisse wie z.B. ein kritscher Batteriezustand eines Sensors werden durch Animationen hervorgehoben. 3. Der Initiator hat folgende Interaktionssmöglichkeiten: • Der Initiator kann frei durch die 3D-Welt navigieren und zoomen. • b) Der Initiator kann sich durch Auswählen eines Kraftwerks, Sensors oder Verbrauchers Details anzeigen lassen. (siehe Anwendungsfall „Details betrachten“) • Der Initiator kann, sofern er im Administrationsmodus ist, die 3D-Welt bearbeiten. Dazu kann er neue Kraftwerke, Sensoren und Verbraucher erstellen, diese in der 3D-Welt platzieren und sie mit anderen Objekten verknüpfen. Vorhandene Objekte, sowie die Landschaft an sich können ebenfalls bearbeitet werden. 4. Der Initiator verlässt den 3D-Modus über eine Schaltfläche.
Anfangsbedingungen	Das Grundsystem ist gestartet. Eine konfigurierte Version der 3D-Welt liegt vor.
Abschlussbedingungen	Änderungen an der 3D-Welt werden auf Wunsch gesichert.
Qualitätsanforderungen	• Die Darstellung der 3D-Welt läuft ruckelfrei auf durchschnittlicher Hardware. • Die 3D-Welt ist ein grafisch ansprechender Bestandteil des Demonstrators, der spielerisch auf das Projekt aufmerksam macht. • Da die 3D-Welt im Kioskmodus verfügbar ist, geht das System schlichtend mit Fehleingaben des Benutzers um.

Anwendungsfallname	**Details betrachten**
Akteure	Initiiert von Administrator oder professioneller Anwender oder Anwender bzw. Besucher (Initiator)
Ereignisfluss	1. Der Initiator klickt auf ein Kraftwerk, einen Sensor oder einen Verbraucher. 2. Das System zeigt abstrahierte Übersichtsgraphen eingeblendet an und bietet weitere Graphen und Tabellen an. 3. Der Initiator wählt einen weiteren Graphen aus, der angezeigt wird. 4. Der Initiator schließt das Detailfenster über eine Schaltfläche.

Anfangsbedingungen	Der Initiator ist im 3D-Modus. Das gewählte Objekt bietet Detailinformationen an.
Abschlussbedingungen	Das Detailfenster wurde geschlossen
Qualitätsanforderungen	Die Graphen und Tabellen passen visuell in die 3D-Welt und sind lesbar skaliert.

Anwendungsfallname	**Graph/Tabelle betrachten**
Akteure	Initiiert von Administrator oder professioneller Anwender oder Anwender bzw. Besucher (Initiator)
Ereignisfluss	1. Der Initiator klickt auf die Schaltfläche zur statistischen Auswertung 2. System bietet Möglichkeit Graphen und/oder Tabellen anzuzeigen 3. Graphen und/oder Tabellen werden angezeigt
Anfangsbedingungen	Das Grundsystem ist gestartet.
Abschlussbedingungen	Keine.
Qualitätsanforderungen	• Graphen und/oder Tabellen werden exakt und an richtiger Position angezeigt.

Anwendungsfallname	**Ansichten speichern**
Akteure	Initiiert von Administrator oder professioneller Anwender oder Anwender bzw. Besucher (Initiator)
Ereignisfluss	1. Der Initiator hat sich eine Ansicht in der gestaltet und klickt auf den Button „Ansicht speichern“. 2. System speichert die Ansicht für den Nutzer.

Anfangsbedingungen	Eine Ansicht muss vorher gestaltet werden.
Abschlussbedingungen	Die Ansicht wird gespeichert und beim nächsten Aufruf wieder angezeigt.
Qualitätsanforderungen	• Graphen und/oder Tabellen werden exakt und an richtiger Position gespeichert.

Anwendungsfallname	**Elemente editieren (Zusammenfassung mehrere Anwendungsfälle)**
Akteure	Initiiert von Administrator oder professioneller Anwender oder Anwender bzw. Besucher (Initiator)
Ereignisfluss	1. Der Initiator klickt auf die Tabelle/Graph. 2. Die Skalierung des Graphen lässt sich ändern / Spalten können bei den Tabellen hinzugefügt bzw. gelöscht werden. 3. Tabelle/Graph können aus der View entfernt werden bzw. hinzugefügt werden. 4. Verschiedene Parameter können für die Tabellen/Graphen gesetzt werden.
Anfangsbedingungen	Keine.
Abschlussbedingungen	Tabelle und Graphen können angezeigt werden.
Qualitätsanforderungen	• Graphen und/oder Tabellen werden exakt und an richtiger Position angezeigt.

3.7 Hinzugefügte und entfallene Anforderungen

Im Laufe des Projektes wurden einige zusätzliche Anforderungen gestellt, zudem sind einige Anforderungen entfallen.

3.7.1 SunSPOT

Da nicht genügend Sensoren zur Verfügung stehen um sämtliche angezeigten Kraftwerke mit unterschiedlichen Daten zu beliefern müssen diese Sensoren simuliert werden. Zudem würde es platztechnisch den Rahmen einer jeden Präsentation sprengen, jedes Kraftwerk mit Sensoren wiederzugeben.

Zu diesem Zweck wurde eine Software erstellt, die Sensordaten mit Zufallswerten simuliert. Diese Werte müssen dabei im Rahmen der möglichen echten Werte liegen und dürfen auch nur im Bereich der normalen Schwankungen steigen oder fallen.

3.7.2 Odysseus

Archiving

Um Datenströme zu einem späteren Zeitpunk wieder ausgeben zu können werden diese direkt an der ersten Senke abgegriffen und in einer Datenbank gespeichert. So können die Daten aus der Datenbank zu einem späteren Zeitpunk abgefragt werden und genau so behandelt werden wie Datenströme von Sensoren.

3.7.3 View

Kritische Werte

Um besondere Ereignisse deutlicher zu machen wurden in Anlehnung an das SCADA-Systeme [FF05] kritische Werte eingeführt. Diese legen obere und untere Grenzen für bestimmte Wertebereiche fest. Im Falle eines Über- oder Unterschreitens dieser Werte wird je nach Einstellung das Ereigniss in einem Event Viewer festgehalten und zusätzlich in einer Alarm Bar angezeigt.

Im Event Viewer können die Ereignisse genauer betrachtet und quittiert werden. Durch das Quittieren verschwindet das Event aus der Alarm Bar und der genaue Zeitpunkt wird im Event Viewer festgehalten.

Darstellung wechseln

Die 3D-Welt ist ein Bestandteil der View und kann genau so wie Tabellen und Graphen gestartet und beendet werden. Beide Ansichten können also gleichzeitig betrachtet werden, was ein Wechseln zwischen den Ansichten überflüssig macht.

Details betrachten

Da die 3D-Welt ein fester Bestandteil der View und keine eigenständige Umgebung geworden ist entfällt eine Anzeige von Tabelln oder Graphen innerhalb der 3D-Welt. Die Tabellen und Graphen können stattdessen direkt neben der 3D-Welt angezeigt werden.

Monitoring/Sensorüberwachung

Eine Anzeige der Sensorwerte wie Batteriestand erfolgt nicht,da mehrere Sensoren für einen Stream eingesetzt werden und eine genaue Identifizierung der Sensoren so nicht möglich ist.

Anfrage verwalten

Da es in Odysseus nicht möglich ist, vorhandene Anfragen zu deinstallieren, besteht auch in der View diese Möglichkeit ncht. Einmal installierte Anfragen bleiben also bis zu einem Neustart von Odysseus vorhanden.

Kapitel 4

Entwurf

4.1 Einführung

Dieser Entwurf der Projektgruppe Earth, Wind and Fire gibt einen Überblick über Konzepte und Design unserer geplanten Implementierung eines Demonstrators für das Datenstrommanagementsystem (DSMS) Odysseus. Er dient als Referenz und Orientierung für die Entwickler. Zunächst bieten wir einen Überblick über die Systemarchitektur. Mit Hilfe von Klassendiagrammen und anschaulichen Texten verdeutlichen wir den Aufbau und die Abläufe in unserem Programm. Die Klassenbeschreibungen gehen dabei noch weiter ins Detail, um die Schnittstellen und Funktionsweisen der einzelnen Klassen genau zu definieren. Eine Übersicht der Pakete und Klassen verdeutlichen zudem noch die Verbindungen untereinander. Jedoch ist es nicht ausgeschlossen, dass weitere Klassen, Methoden oder Attribute während der Implementierung hinzugefügt, geändert oder weggelassen werden.

4.1.1 Zweck des Systems

Das System soll die Demonstration der Arbeitsweise des DSMS Odysseus ermöglichen. Dazu werden Sensordaten mittels SunSpots erzeugt, durch Odysseus geleitet und in Tabellen, Grafiken oder einer 3D-Welt angezeigt.

4.1.2 Entwurfsziele

Ziel des Projektes ist es, einen lauffähigen und aussagekräftigen Demonstrator für Odysseus bereitzustellen. Dabei sollen so viele Features von Odysseus wie möglich visualisiert werden. Der Demonstrator soll nach der Fertigstellung auf Messen und sonstigen Veranstaltungen eingesetzt werden, um die Funktionen von Odysseus zu präsentieren.

4.1.3 Referenzen

Das Projekt baut auf dem DSMS Odysseus auf, welches uns von der Universität Oldenburg zur Verfügung gestellt wird. Alle Modifikationen an Odysseus, die durch die Projektgruppe durchgeführt werden, werden dokumentiert.

4.1.4 Überblick

Dieses Dokument dient dem Projektleiter, den Systemarchitekten und den Entwicklern als gemeinsame Grundlage auf Architekturebene zur Klärung ggf. später auftretender Unklarheiten. Es werden die einzelnen Subsysteme und die Datenverwaltung erklärt und Kontrollflussmechanismen erläutert. Außerdem wird die aktuelle Softwarearchitektur erläutert, auf der das System basiert.

4.2 Aktuelle Softwarearchitektur

Das gesamte Projekt baut grundlegend auf dem DSMS Odysseus auf. Dieses wird derzeit von Mitarbeitern der Universität Oldenburg stetig weiterentwickelt. Für das Projekt wird auf einer eigenen Version von Odysseus gearbeitet. Alle Änderungen werden später, soweit möglich, in die aktuelle Entwicklungsversion eingegliedert.

4.3 Vorgeschlagene Softwarearchitektur

Als Hintergrund der Demonstration wird ein Szenario erstellt, welches die Datensammlung aus verschiedenen Kraftwerken für regenerative Energien vorsieht. Hierzu werden Windkraftanlagen, Sonnenkollektoren und Geothermalkraftwerke simuliert.

4.3.1 Überblick

Das System ist - wie in Abbildung 4.1 dargestellt - in vier Subsysteme unterteilt:

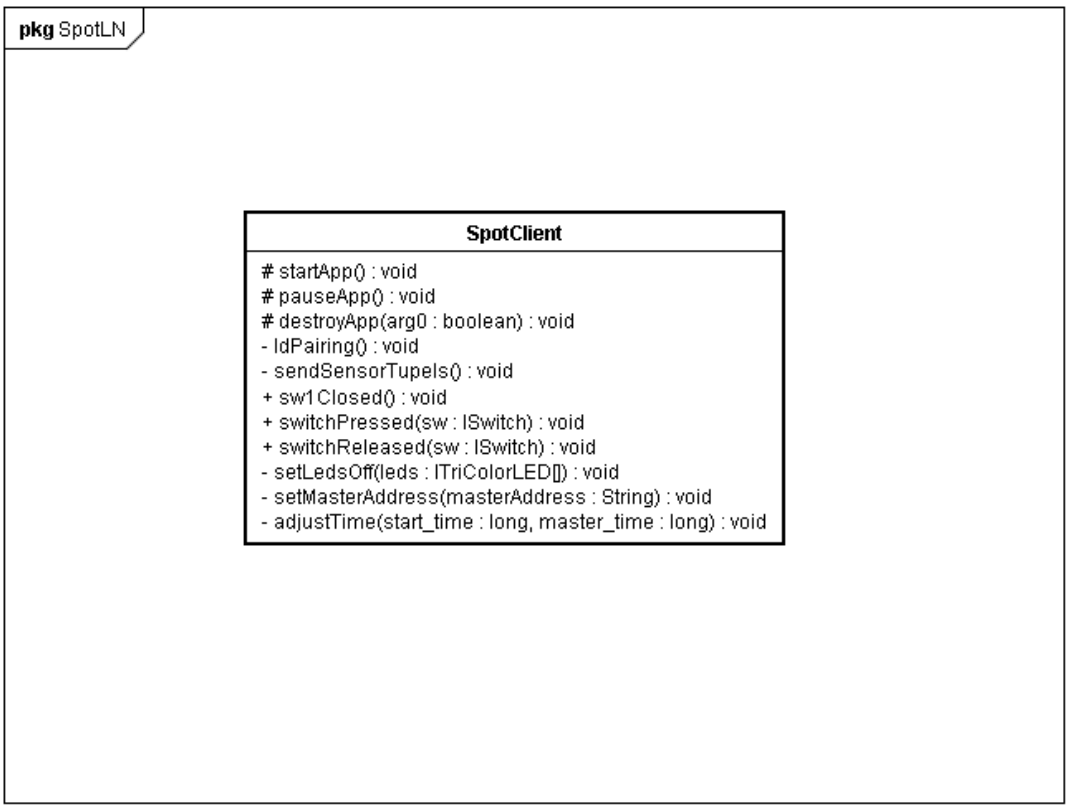
- Das Subsystem SunSpot befasst sich mit der Ermittlung von Daten aus Sensoren und dem Weiterleiten dieser Datenpackete an das DSMS.
- Das Subsystem Odysseus realisiert die Annahme, die Weiterverarbeitung und das Versenden der Daten. Zusätzlich können hier Datenströme archiviert und zu einem späteren Zeitpunkt wieder abgefragt werden.
- Das Subsystem View erstellt eine Schatzentrale für das gesammte Projekt. Hier werden neue Datenströme erstellt und Anfragen auf diese Ströme gestellt. Die Daten dieser Ströme werden dann in Tabellen oder Grafiken angezeigt.
- Das Subsystem 3D-Welt erstellt weiterführend auf dem Subsystem View eine dreidimensionale Welt, in der die einzelnen Kraftwerke als Modelle angezeigt werden können.

4.3.2 Systemzerlegung

An dieser Stelle werden die einzelnen Pakete, ihre Zusammenhänge und die dazugehörigen Klassen vorgestellt. Anhand der Klassenbeschreibungen wird die Verwendung der jeweiligen Klasse mit ihren Methoden und Attributen verdeutlicht.

4.3.3 SunSpots

Zur Sensordatenerfassung mit Hilfe der SunSPOTs werden drei Anwendungen benötigt.



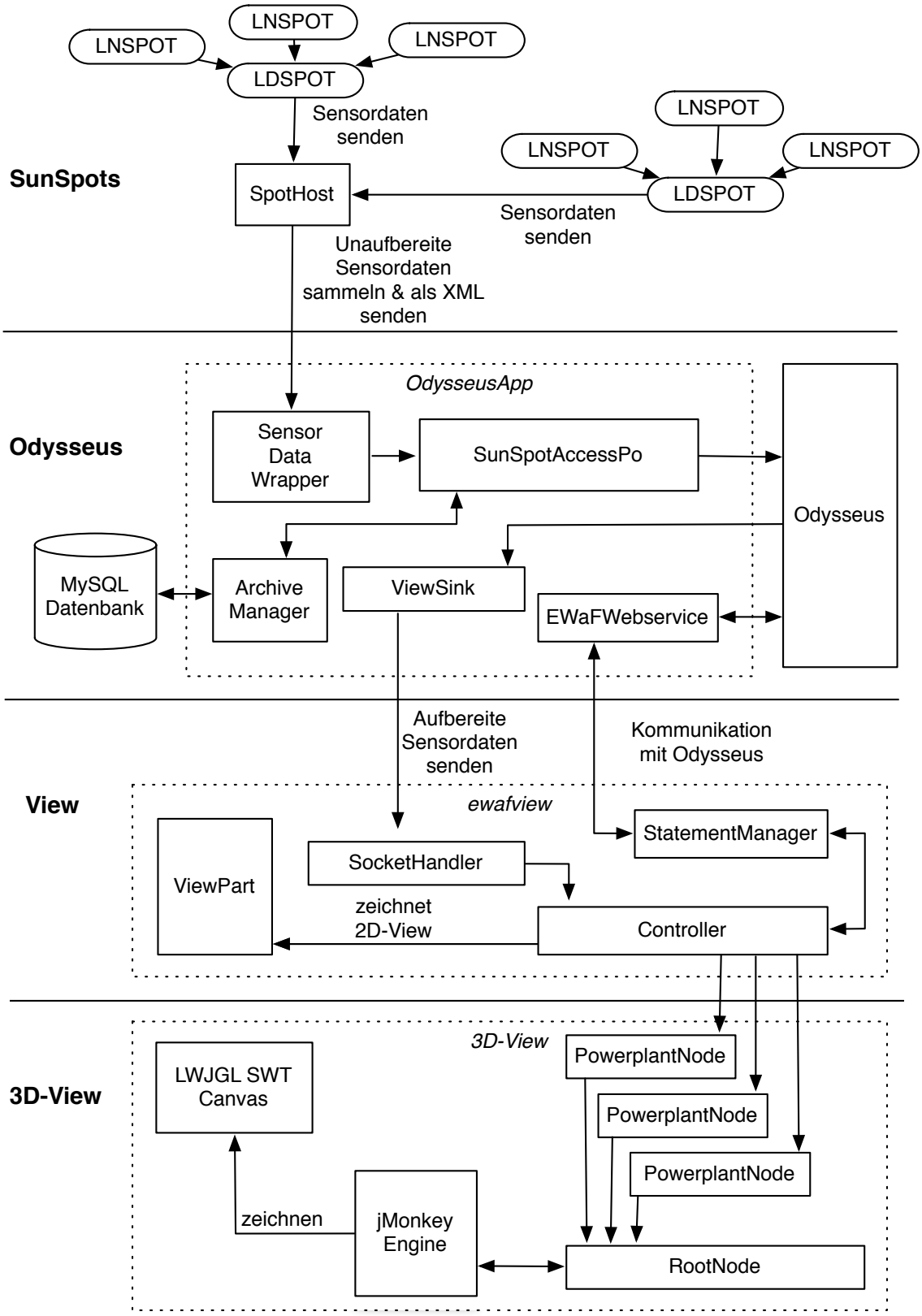


Abbildung 4.1: Entwurf des Gesamtsystems

SpotClientLN

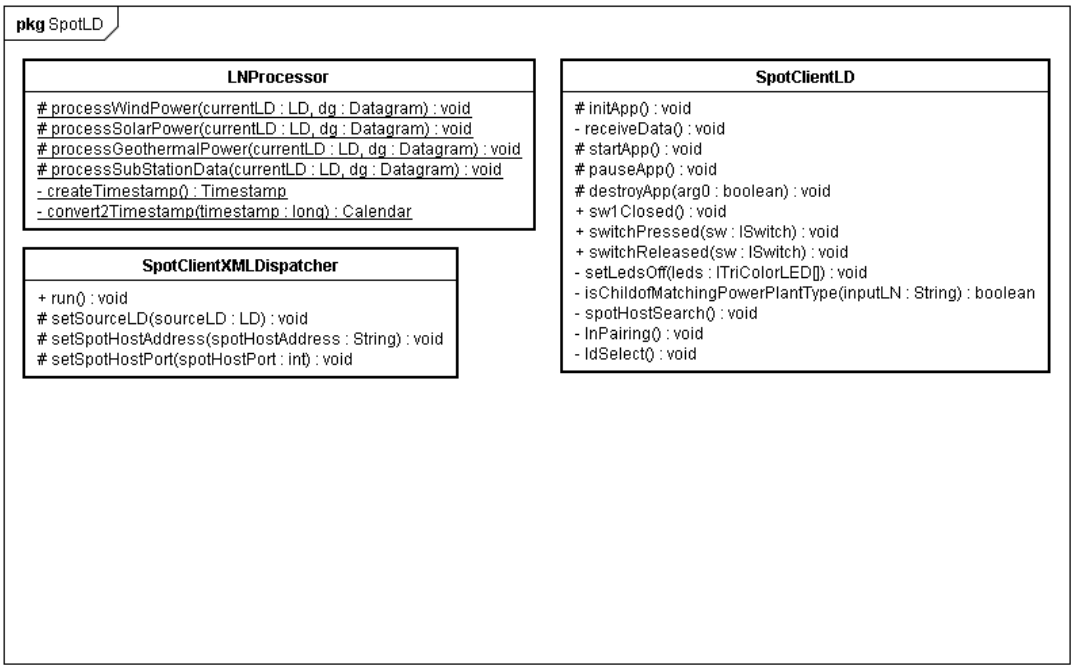
Die Instanzen des SpotClientLN simulieren jeweils einen logischen Knoten (LN) im Sinne der Struktur, wie sie in der IEC Norm 61400-25 definiert wird. Sie senden die auf dem eDemoBoard des SunSPOTS gemessenen Sensorwerte an den übergeordneten Knoten.

ldPairing()

Sendet per Broadcast Anfragen an LD Knoten und führt bei erfolgter Antwort eine Zeitsynchronisation durch.

sendSensorTupels()

Sendet periodisch Messwerte an den übergeordneten LD Knoten.



SpotClientLD

Die Instanzen des SpotClientLD simulieren jeweils einen logisches Gerät (LD) im Sinne der Struktur, wie sie in der IEC Norm 61400-25 definiert wird.

SpotClientLD.spotHostSearch

Sendet per Broadcast Anfragen um einen SpotHost zu finden und führt bei erfolgter Antwort eine Zeitsynchronisation mit diesem durch.

SpotClientLD.lnPairing

Nimmt Anfragen von LN Knoten entgegen, damit diese sich dem LN Knoten unterordnen können.

SpotClientLD.receiveData

Nimmt Sensordaten in einem eigenen Thread von den LN Knoten entgegen und übergibt diese an die LNProcessor

LNProcessor.processWindPower(LD currentLD, Datagram dg)

Aktualisiert den Zustand des Datenmodells eines Windkraftwerkes mit den empfangenen Sensordaten.

LNProcessor.processSolarPower(LD currentLD, Datagram dg)

Aktualisiert den Zustand des Datenmodells eines Solarkraftwerkes mit den empfangenen Sensordaten.

LNProcessor.processGeothermalPower(LD currentLD, Datagram dg)

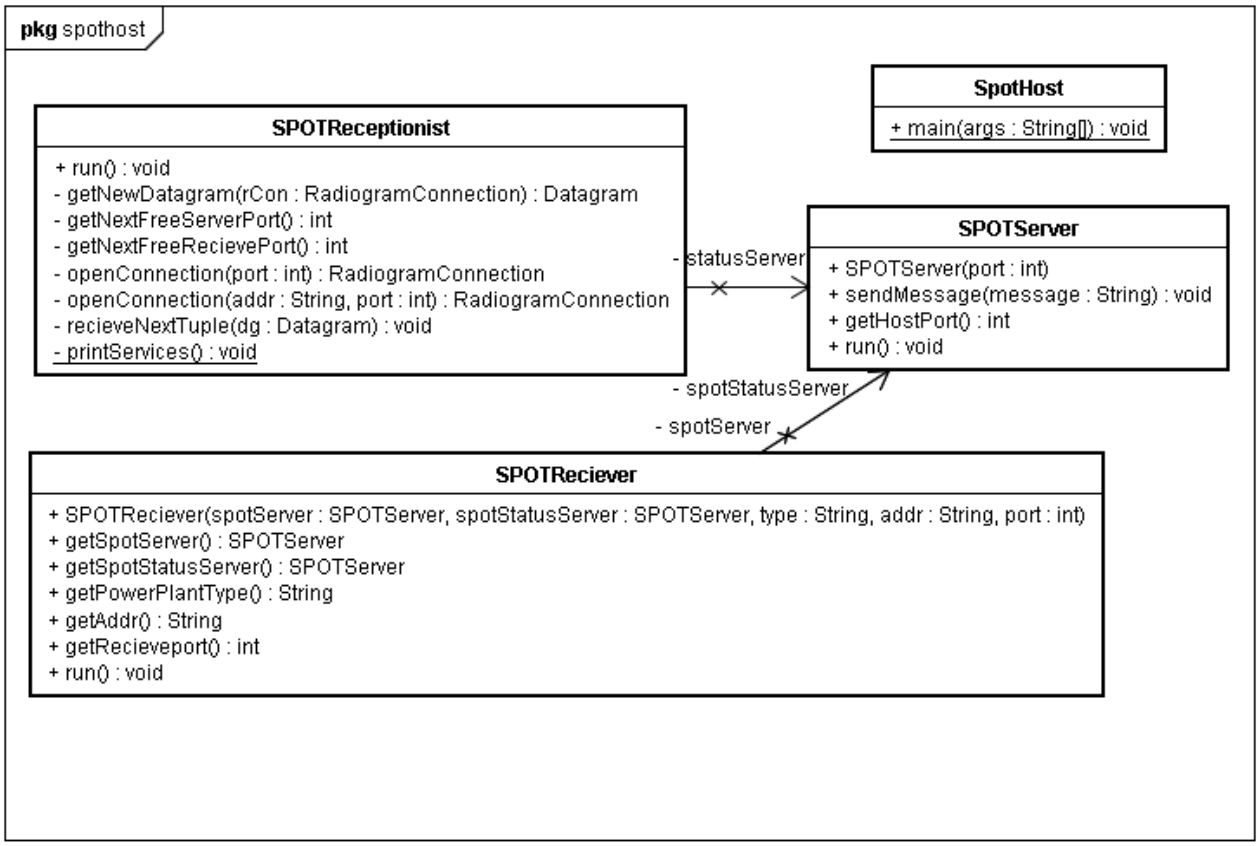
Aktualisiert den Zustand des Datenmodells eines Geothermikraftwerkes mit den empfangenen Sensordaten.

LNProcessor.processSubStationData(LD currentLD, Datagram dg)

Aktualisiert den Zustand des Datenmodells eines Umspannwerkes mit den empfangenen Sensordaten.

SpotClientXMLDispatcher.run()

Erzeugt periodisch eine XML-Struktur, die dem gegenwärtigen Zustand des LD Knotens entspricht. Diese XML-Struktur wird dann über eine com.sun.spot.io.j2me.radiostream.RadiostreamConnection an den SpotHost übertragen.



SPOTHost

SPOT Host zur Verwaltung von SunSPOTs und Bereitstellung deren Daten als Serverdienste.

SPOTReceptionist Der SPOTReceptionist ist die erste Anlaufstelle für neue SunSPOTs. Sie werden in Empfang genommen und bekommen einen eigenen Empfänger-Thread (SPOTReciever) zugewiesen. Hat eine Übersicht über alle laufenden Recieverprozesse.

run()

Wartet auf eingehende Verbindungen neuer SunSPOTs, validiert ihre Verbindungsversuche und erstellt ggf. neue Empfänger- und Serverthreads zur Bereitstellung der Daten oder weist sie ihren jeweiligen alten zu.

SPOTReciever Lauscht für eigehende Daten und sendet diese an die wartenden Clients.

SPOTReciever(SPOTServer spotServer, SPOTServer spotStatusServer, String type, String addr, int port)

Startet einen neuen Lauschclient für neue SunSPOTs.

getSpotServer()

Gibt den korespondierenden SPOTServer zurück.

getSpotStatusServer()

Gibt den jeweiligen SPOTStatusServer zurück. (Der SPOTStatusServer liefert allgemeine Daten z.B. über den Batteriestand der SunSPOTs)

getPowerPlantType())

Gibt den jeweiligen Kraftwerkstyp des SunSPOTs wieder (Geothermal, Solar oder Wind).

getAddr()

Die MAC-Adresse des SunSPOTs.

getRecieveport()

Gibt den Port zurück, auf dem nach den Daten des SunSPOTs gelauscht wird.

run()

Lauscht auf eingehende Daten des jeweiligen SunSPOTs.

SPOTServer Ein Serverdienst, der die Verbindungen von Sun SPOTs verwaltet.

VSPOTServer(int port)

Erstellt einen neuen Serverthread auf dem angegebenen Port, der auf Verbindungen von Clients zum Abruf der Daten wartet.

getHostPort()

Gibt den Port zurück, auf dem nach neuen Verbindungen gelauscht wird.

run()

Lauscht auf dem jeweiligen Host-Port nach eingehenden Verbindungen und erstellt einen neuen Thread für jeden verbundenen Client.

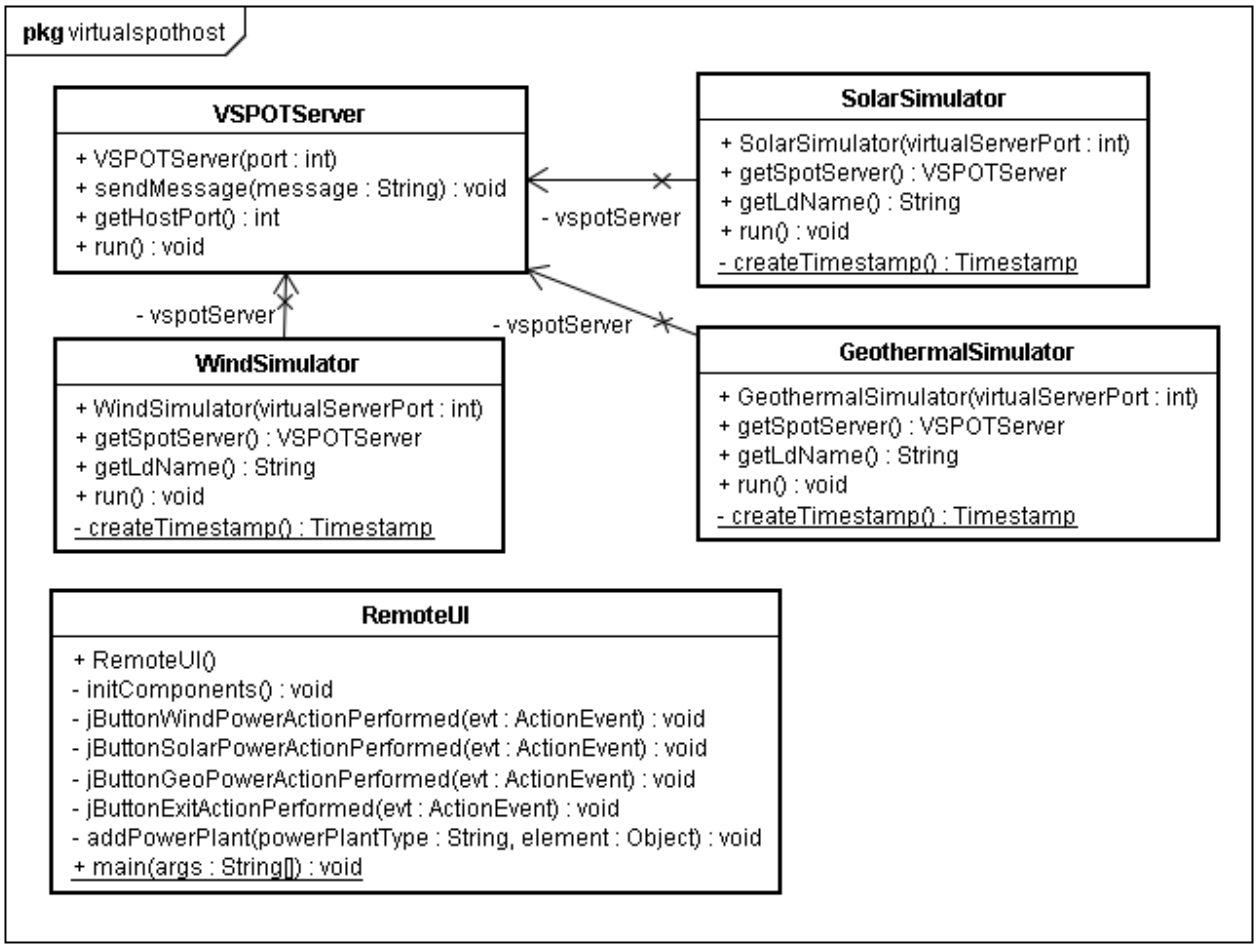
sendMessage(String message)

Sendet eine Nachricht an alle lauschenden Clients.

SPOTHost Kapselt den Start des SPOTReceptionist in eine eigene Klasse.

main()

Startet den SPOTReceptionist



VirtualSPOTHost

Virtueller SPOT Host zur Simulation von Sun SPOTs über die Anzahl physisch verfügbarer SPOTs hinaus und zu Testzwecken.

RemoteUI RemoteUI ist die grafische Oberfläche zur Kontrolle der Simulation verschiedener Kraftwerkstypen.

main()

Startet eine neue Instanz der grafischen Oberfläche.

VSPOTServer Ein Serverdienst, der die Simulation von Sun SPOTs verwaltet.

VSPOTServer(int port)
Erstellt einen neuen Server-Thread zur Simulation von Sun SPOTs auf dem angegebenen Port.

getHostPort()
Gibt den Port zurück, auf dem nach neuen Verbindungen gelauscht wird.

run()
Lauscht auf dem jeweiligen Host-Port nach eingehenden Verbindungen und erstellt einen neuen Thread für jeden verbundenen Client.

sendMessage(String message)
Sendet eine Nachricht an alle lauschenden Clients.

GeothermalSimulator Simuliert einen Geothermal-SunSPOT.

GeothermalSimulator(int virtualServerPort)
Erstellt einen neuen VSPOTServer-Thread auf dem angegebenen Port.

getSpotServer()
Gibt das korespondierende VSPOTServer-Objekt zurück.

getLdName()
Gibt den Namen des simulierten LDs zurück.

run()
Simuliert Sensordaten.

SolarSimulator Simuliert einen Solar-SunSPOT.

GeothermalSimulator(int virtualServerPort)
Erstellt einen neuen VSPOTServer-Thread auf dem angegebenen Port.

getSpotServer()
Gibt das korespondierende VSPOTServer-Objekt zurück.

getLdName()
Gibt den Namen des simulierten LDs zurück.

run()
Simuliert Sensordaten.

WindSimulator Simuliert einen Wind-SunSPOT.

GeothermalSimulator(int virtualServerPort)
Erstellt einen neuen VSPOTServer-Thread auf dem angegebenen Port.

getSpotServer()
Gibt das korespondierende VSPOTServer-Objekt zurück.

getLdName()
Gibt den Namen des simulierten LDs zurück.

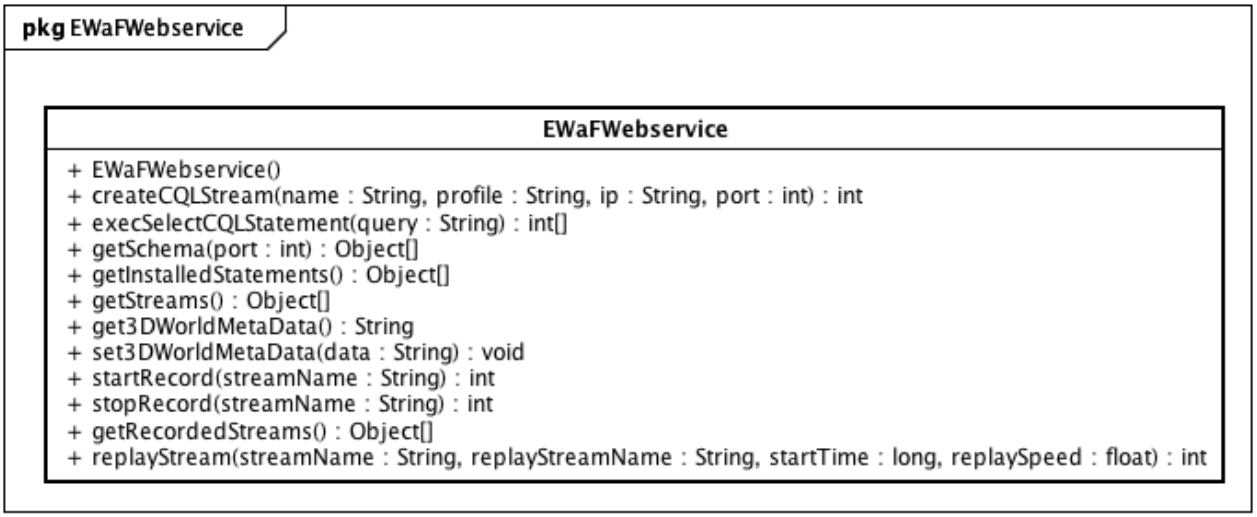
run()
Simuliert Sensordaten.

Weitere Klassen Für alle weiteren Klassen siehe SpotClient LD.

4.3.4 Odysseus

EWaFWebservice

Der EWaFWebservice stellt den Dienst bereit, mit der Odysseus von View Komponenten gesteuert und angesprochen wird.



EWaFWebservice createCQLStream(String name, String profile, String ip, int port)
Dieser Befehl sagt Odysseus unter welcher IP und unter welchen Port ein CQL Stream (eines SunSpots) zu finden ist. Dieser CQLStream hat zudem ein Profil, welches die Art des CQLStreams identifiziert (beispielsweise WindPowerPlant). Diesem Stream muss ein Name gegeben werden.

execSelectCQLStatement(String query)
Führt die übergebene Query auf dem Odysseus-System aus.

getSchema(int statementID)
Gibt die Schemainformationen von der übergebenen StatementID wieder.

getInstalledStatements()
Gibt eine Liste der auf dem Odysseus-System installierten Statements wieder.

getStreams()
Gibt eine Liste über die auf dem Odysseus-System installierten Streams wieder.

startRecord(String streamName)
Weist das Odysseus-System an, den Stream mit den übergebenen Namen aufzuzeichnen.

stopRecord(String streamName)
Weist das Odysseus-System an, die Aufzeichnung des Streams mit den übergebenen Namen zu stoppen.

getRecordedStreams()
Gibt eine Liste der Streams wieder, die aufgezeichnet worden sind.

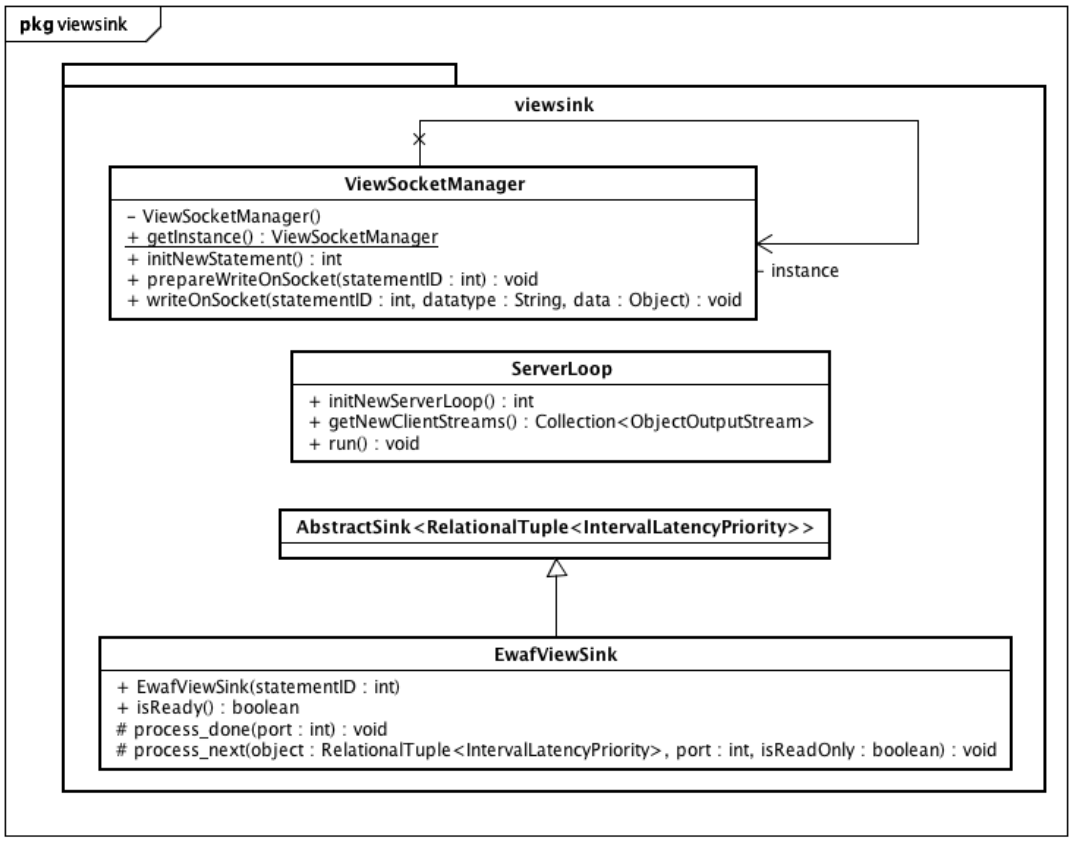
replayStream(String streamName, String replayStreamName, long startTime, float replaySpeed)
Weist das Odysseus-System an, einen Stream aus der Archiving-Datenbank wiederzugeben. Der Startzeitpunkt und die Wiedergabegeschwindigkeit können eingestellt werden.

get3DWorldMetaData()
Gibt die Metadaten der 3D-Visualisierung zurück.

set3DWorldMetaData(String data)
Setzt die Metadaten der 3D-Visualisierung.

viewsink

Die ViewSink ist die Senke, die alle SunSpot-Daten, die von Odysseus verarbeitet wurden, empfängt und diese an die entsprechende View weiterleitet.



ViewSocketManager Der ViewSocketManager verwaltet die Netzwerksockets für alle Statements.

initNewStatement()

Eine neue ServerLoop (siehe unten) wird für ein Statement angelegt.

prepareWriteOnSocket(int statementID)

Das Übertragen von Daten wird vorbereitet, indem die ServerLoop (siehe unten) gefragt wird, ob sich neue Clients verbunden haben. Falls ja, werden diese in die Liste aller Clients für das Statement mit aufgenommen und beim Aufruf von writeOnSocket mit Daten versorgt.

writeOnSocket(int statementID, String datatype, Object data)

Ein Attribut wird an alle Views übertragen, die zur Zeit mit diesem Statement verbunden sind.

ServerLoop Die ServerLoop verwaltet einen Serverport für genau ein Statement. Neue Verbindungen zu diesem Port werden akzeptiert und vorgehalten, bis der ViewSocketManager sie zur Nutzung anfordert.

getNewClientStreams()

Liefert neue Verbindungen, die seit dem letzten Aufrufen geöffnet wurden.

run()

Startet die ServerLoop.

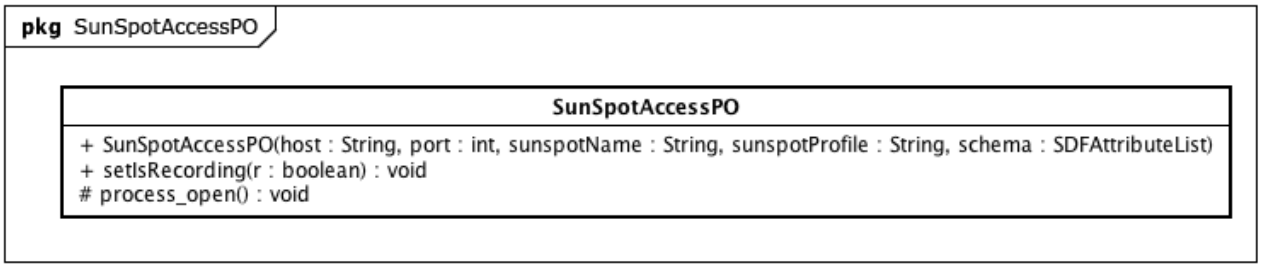
EWaFViewSink Die EWaFViewSink wird als Senke im ODYSSEUS-Framework konfiguriert.

process_next(RelationalTuple<IntervalLatencyPriority> object, int port, boolean isReadOnly)

Bereitet die Daten vor und übertragt sie mit Hilfe des ViewSocketManagers an alle Views, die mit diesem Statement verbunden sind.

SunSpotAccessPO

Der SunSpotAccessPO wird instanziert, wenn eine „SELECT“ Query auf einer SunSpot Quelle ausgeführt wird. Der SunSpotAccessPO bindet Odysseus und den SunSpotDataWrapper aneinander.

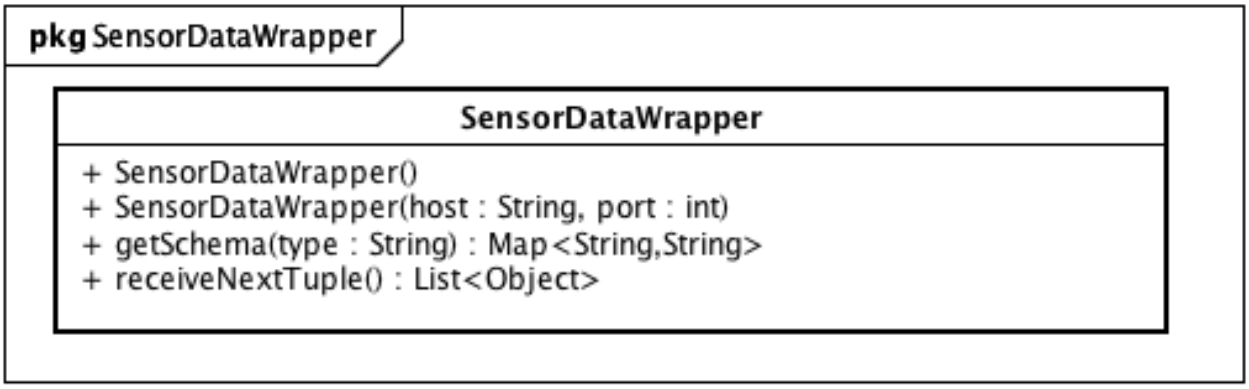


setIsRecording(boolean r)
Schaltet das Recording für den SunSpotAccessPO an oder aus.

process_open()
Startet einen Thread, der dafür verantwortlich ist, dass die Streaming Daten des jeweiligen SunSpots korrekt verarbeitet werden.

SensorDataWrapper

Diese Klasse kapselt die Logik, die gebraucht wird, damit die SunSpots und Odysseus miteinander interagieren können.

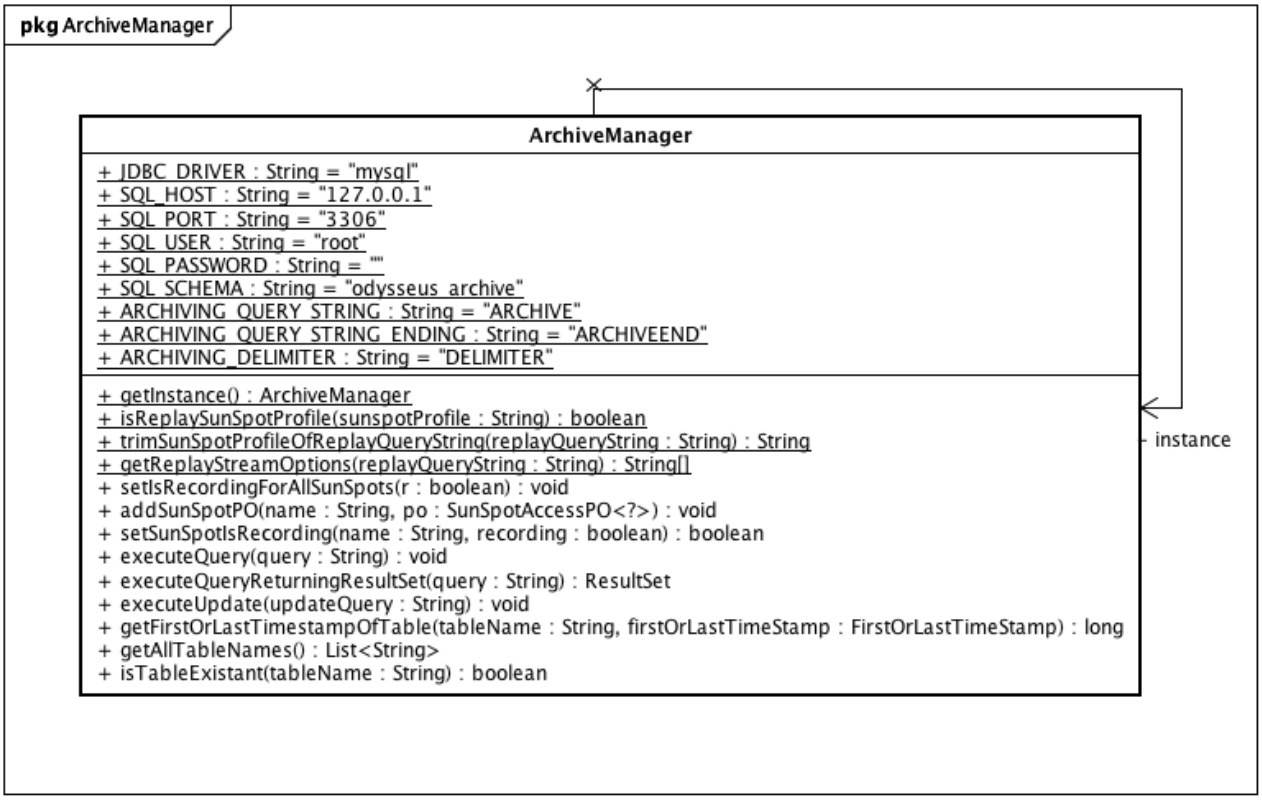


getSchema(String type)
Liefert das jeweilige Schema des übergebenen Strings zurück.

receiveNexTuple()
Verarbeitet die ankommenden Tuplen, die von den jeweiligen SunSpot geschickt werden.

ArchiveManager

Der ArchiveManager ist für die permanente Speicherung von Datenströmen zuständig. Des Weiteren stellt er Methoden bereit, um Zugriff auf die Datenbank zu erhalten.



getInstance()
Stellt den Singleton ArchiveManager zur Verfügung. Stellt sicher, dass die genutzte Datenbank in der MySQL Instanz vorhanden. Wenn dies nicht der Fall ist, wird diese erstellt.

isReplaySunSpotProfile(String sunspotProfile)
Prüft, ob das übergebene SunSpot-Profil ein SunSpot-Profil für das Replaying ist. Wenn es das ist, dann enthält das SunSpot-Profil auch noch Informationen, die das Replaying betreffen.

trimSunSpotProfileOfReplayQueryString(String replayQueryString)
Wenn das SunSpot-Profil ein SunSpot-Profil für das Replaying ist, dann werden mit dieser Methode die Informationen die das Replaying betreffen gekürzt, so dass das SunSpot-Profil für Odysseus wieder als solches erkennbar ist.

getReplayStreamOptions(String replayQueryString)
Wenn das SunSpot-Profil ein SunSpot-Profil für das Replaying ist, dann können mit dieser Methode die Information für das Replaying aus dem SunSpot-Profil extrahiert werden.

setIsRecordingForAllSunSpots(boolean r)
Für alle SunSpot-Streams wird das Recording der Daten an- bzw. ausgeschaltet.

addSunSpotPO(String name, SunSpotAccessPO<?> po)
Wenn ein SunSpotAccessPO an das Odysseus-System angeschlossen wird, so er wird er mit der Hilfe dieser Methode auch beim ArchiveManager registriert.

setSunSpotIsRecording(String name, boolean recording)
Schaltet das Recording für einen bestimmten SunSpot an- bzw. aus.

executeQuery(String query)
Führt die übergebene Query auf der Archiving-Datenbank aus.

executeQueryReturningResultSet(String query)
Führt die übergebene Query auf der Archiving-Datenbank aus und gibt das resultierende ResultSet zurück.

executeUpdate(String updateQuery)
Führt eine Update-Query auf der Archiving-Datenbank aus.

getFirstOrLastTimestampOfTable(String tableName, FirstOrLastTimeStamp firstOrLastTimeStamp)
Holt den ersten oder letzten Timestamp aus einer Tabelle aus der Archiving-Datenbank.

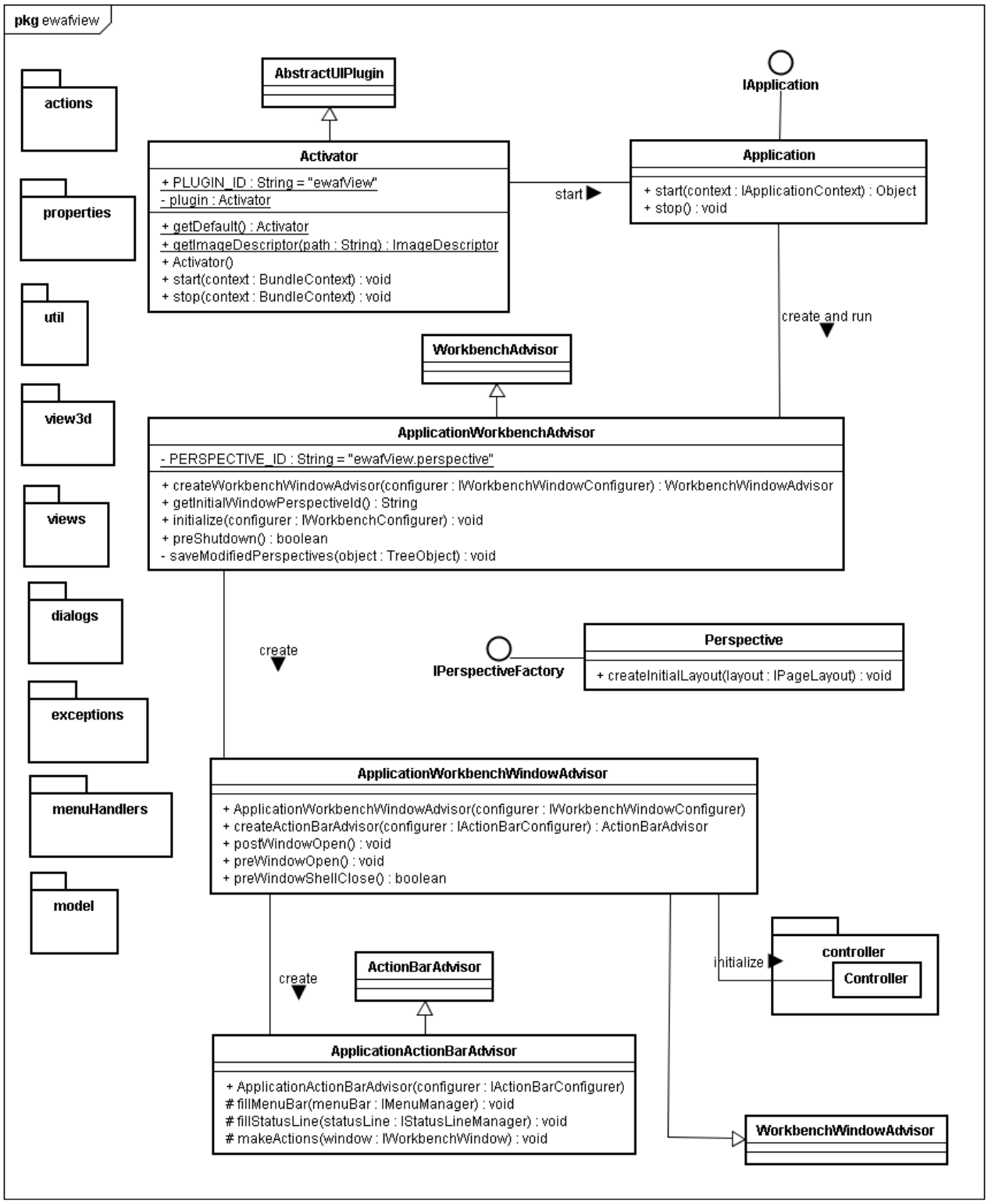
getAllTableNames()
Gibt alle Tabellennamen der Archiving-Datenbank aus.

isTableExistant(String tableName)
Prüft, ob der übergebene Tabellenname schon existiert.

Archiving

4.3.5 View

ewafview



Activator Der Activator steuert den Lebenszyklus des Plugins.

start(org.osgi.framework.BundleContext context)
Initialisiert den BundleListener und registriert ihn beim context
stop(org.osgi.framework.BundleContext context)
Entfernt den BundleListener und speichert die Positionen der GUI-Elemente.

Application Die Application ist verantwortlich für die Initialisierung, sowie der korrekten Beendigung der Anwendung.
start(org.eclipse.equinox.app.IApplicationContext context)
Erzeugt und startet den Workbench.
stop()
Beendet den Workbench.

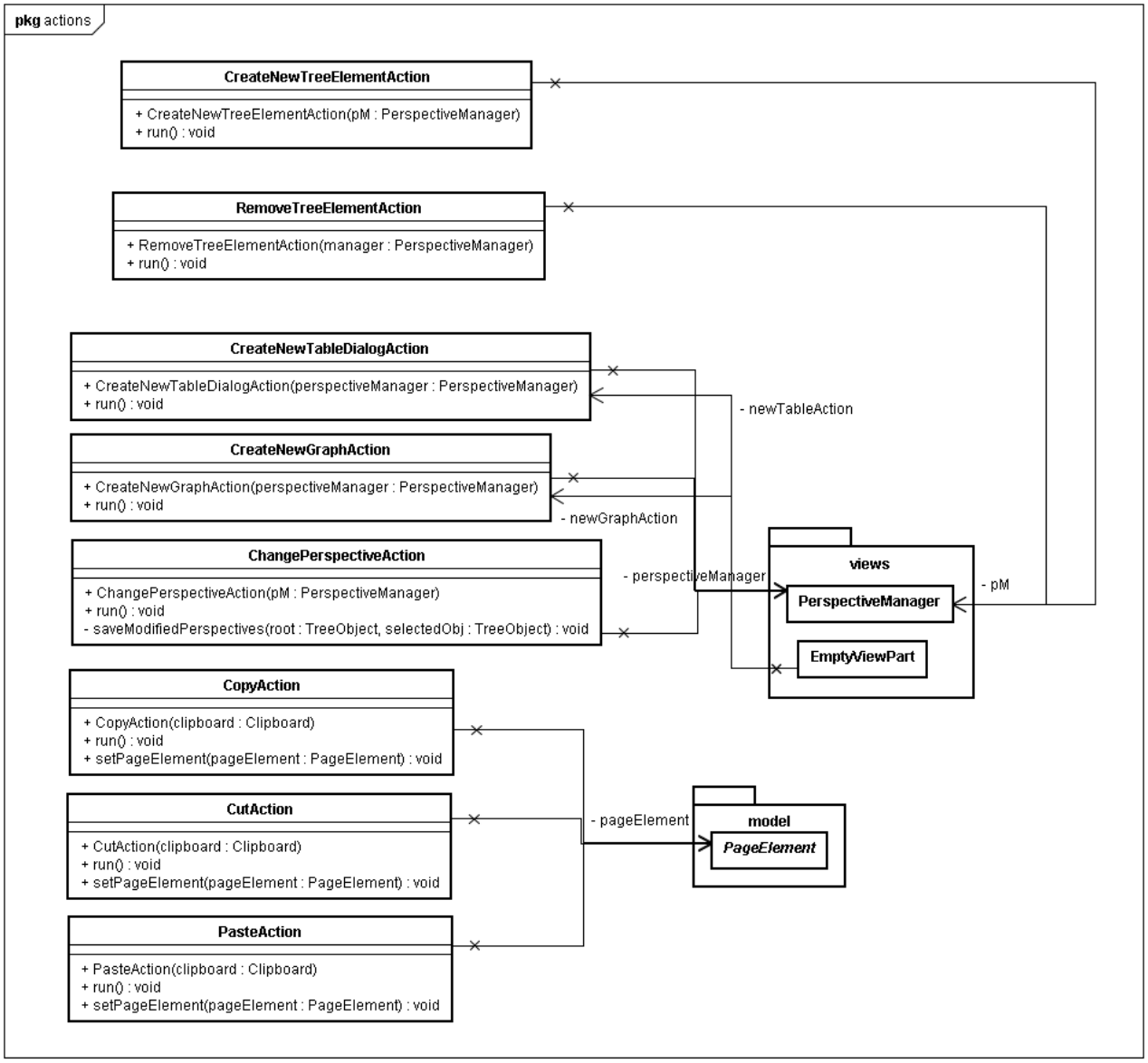
ApplicationActionBarAdvisor Der ApplicationActionBarAdvisor fügt die Odysseus Konnektivitätsinfo und die lokale Zeit zur Statusleiste hinzu.

ApplicationWorkbenchAdvisor Der ApplicationWorkbenchAdvisor speichert Änderungen an der Perspektive beim Beenden der Anwendung.

ApplicationWorkbenchWindowAdvisor Der ApplicationWorkbenchWindowAdvisor initialisiert die Workbench und die Odysseus-Verbindung.

Perspective Die Perspective Klasse erstellt ein initiales Layout, wenn die Anwendung zum ersten Mal gestartet wird.

actions



ChangePerspectiveAction Action zum Wechsel der aktuellen Perspektive

CopyAction Action zum Kopieren eines ViewPart2s aus einer Perspektive

CreateNewGraphAction Action zum Öffnen eines CreateNewGraph Dialogs

CreateNewTableDialogAction Action zum Öffnen eines CreateNewTable Dialogs

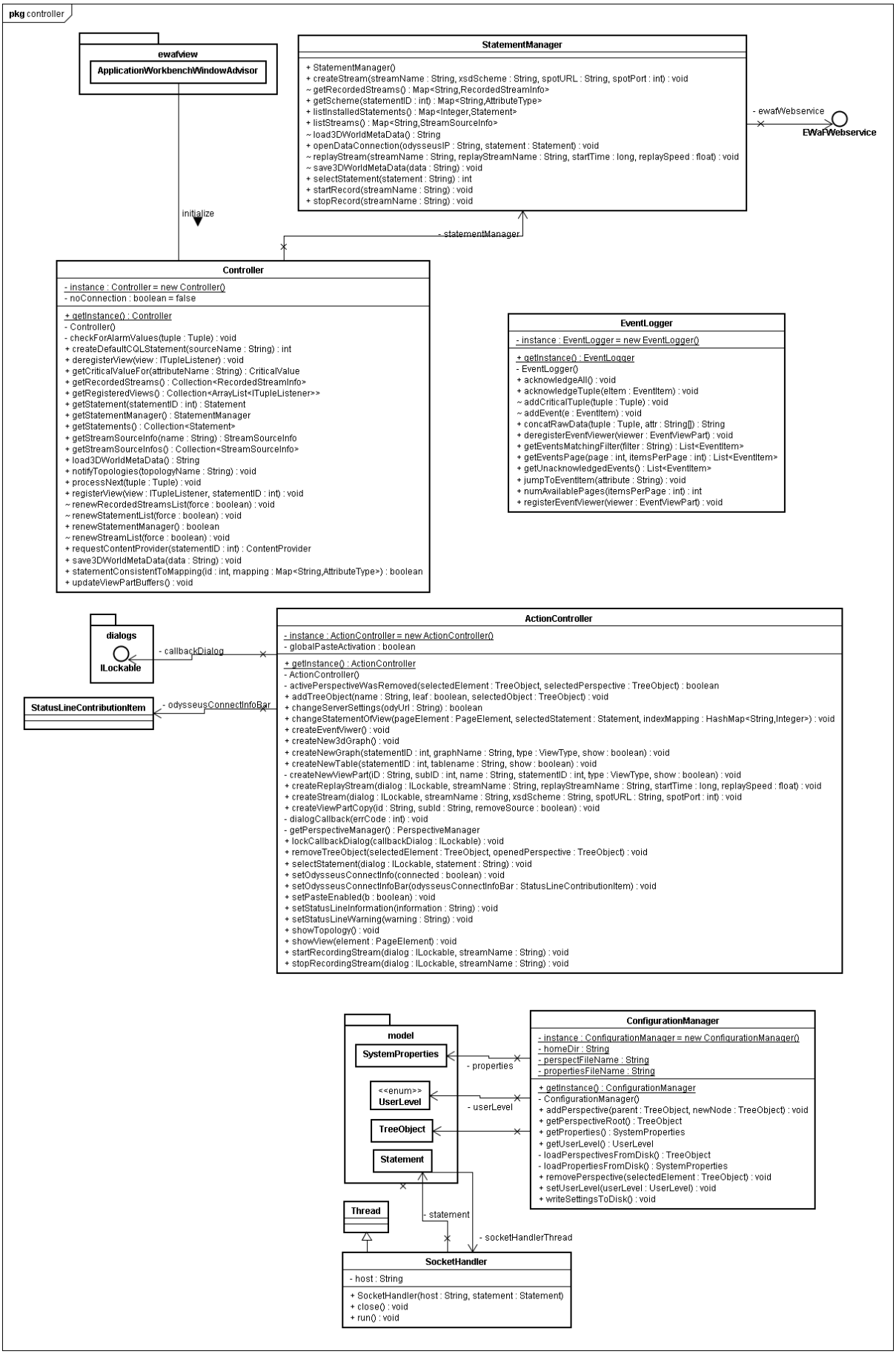
CreateNewTreeElementAction Action zum Öffnen eines CreateNewTreeElement Dialogs

CutAction Action zum Ausschneiden eines ViewPart2s aus einer Perspektive

PasteAction Action zum Einfügen eines ViewPart2s in eine Perspektive

RemoveTreeElementAction Action zum Entfernen eines TreeElements

controller


pkg controller
ewafview
ApplicationWorkbenchWindowAdvisor
StatementManager
+ StatementManager()
+ createStream(streamName : String, xsdScheme : String, spotURL : String, spotPort : int) : void
~ getRecordedStreams() : Map<String,RecordedStreamInfo>
+ getScheme(statementID : int) : Map<String,AttributeType>
+ listInstalledStatements() : Map<Integer,Statement>
+ listStreams() : Map<String,StreamSourceInfo>
~ load3DWorldMetaData() : String
+ openDataConnection(odysseusIP : String, statement : Statement) : void
~ replayStream(streamName : String, replayStreamName : String, startTime : long, replaySpeed : float) : void
~ save3DWorldMetaData(data : String) : void
+ selectStatement(statement : String) : int
+ startRecord(streamName : String) : void
+ stopRecord(streamName : String) : void
- ewafWebservice
EWafWebservice
initialize
- statementManager
Controller
- instance : Controller = new Controller()
- noConnection : boolean = false
+ getInstance() : Controller
- Controller()
- checkForAlarmValues(tuple : Tuple) : void
+ createDefaultCQLStatement(sourceName : String) : int
+ deregisterView(view : ITupleListener) : void
+ getCriticalValueFor(attributeName : String) : CriticalValue
+ getRecordedStreams() : Collection<RecordedStreamInfo>
+ getRegisteredViews() : Collection<ArrayList<ITupleListener>>
+ getStatement(statementID : int) : Statement
+ getStatementManager() : StatementManager
+ getStatements() : Collection<Statement>
+ getStreamSourceInfo(name : String) : StreamSourceInfo
+ getStreamSourceInfos() : Collection<StreamSourceInfo>
+ load3DWorldMetaData() : String
+ notifyTopologies(topologyName : String) : void
+ processNext(tuple : Tuple) : void
+ registerView(view : ITupleListener, statementID : int) : void
~ renewRecordedStreamsList(force : boolean) : void
~ renewStatementList(force : boolean) : void
+ renewStatementManager() : boolean
~ renewStreamList(force : boolean) : void
+ requestContentProvider(statementID : int) : ContentProvider
+ save3DWorldMetaData(data : String) : void
+ statementConsistentToMapping(id : int, mapping : Map<String,AttributeType>) : boolean
+ updateViewPartBuffers() : void
EventLogger
- instance : EventLogger = new EventLogger()
+ getInstance() : EventLogger
- EventLogger()
+ acknowledgeAll() : void
+ acknowledgeTuple(eItem : EventItem) : void
~ addCriticalTuple(tuple : Tuple) : void
~ addEvent(e : EventItem) : void
+ concatRawData(tuple : Tuple, attr : String[]) : String
+ deregisterEventViewer(viewer : EventViewPart) : void
+ getEventsMatchingFilter(filter : String) : List<EventItem>
+ getEventsPage(page : int, itemsPerPage : int) : List<EventItem>
+ getUnacknowledgedEvents() : List<EventItem>
+ jumpToEventItem(attribute : String) : void
+ numAvailablePages(itemsPerPage : int) : int
+ registerEventViewer(viewer : EventViewPart) : void
dialogs
ILockable
- callbackDialog
StatusLineContributionItem
- odysseusConnectInfoBar
ActionController
- instance : ActionController = new ActionController()
- globalPasteActivation : boolean
+ getInstance() : ActionController
- ActionController()
- activePerspectiveWasRemoved(selectedElement : TreeObject, selectedPerspective : TreeObject) : boolean
+ addTreeObject(name : String, leaf : boolean, selectedObject : TreeObject) : void
+ changeServerSettings(odyUrl : String) : boolean
+ changeStatementOfView(pageElement : PageElement, selectedStatement : Statement, indexMapping : HashMap<String,Integer>) : void
+ createEventViwer() : void
+ createNew3dGraph() : void
+ createNewGraph(statementID : int, graphName : String, type : ViewType, show : boolean) : void
+ createNewTable(statementID : int, tablename : String, show : boolean) : void
- createNewViewPart(iD : String, subID : int, name : String, statementID : int, type : ViewType, show : boolean) : void
+ createReplayStream(dialog : ILockable, streamName : String, replayStreamName : String, startTime : long, replaySpeed : float) : void
+ createStream(dialog : ILockable, streamName : String, xsdScheme : String, spotURL : String, spotPort : int) : void
+ createViewPartCopy(id : String, subId : String, removeSource : boolean) : void
- dialogCallback(errCode : int) : void
- getPerspectiveManager() : PerspectiveManager
+ lockCallbackDialog(callbackDialog : ILockable) : void
+ removeTreeObject(selectedElement : TreeObject, openedPerspective : TreeObject) : void
+ selectStatement(dialog : ILockable, statement : String) : void
+ setOdysseusConnectInfo(connected : boolean) : void
+ setOdysseusConnectInfoBar(odysseusConnectInfoBar : StatusLineContributionItem) : void
+ setPasteEnabled(b : boolean) : void
+ setStatusLineInformation(information : String) : void
+ setStatusLineWarning(warning : String) : void
+ showTopology() : void
+ showView(element : PageElement) : void
+ startRecordingStream(dialog : ILockable, streamName : String) : void
+ stopRecordingStream(dialog : ILockable, streamName : String) : void
model
SystemProperties
<<enum>>
UserLevel
TreeObject
Statement
- properties
- userLevel
ConfigurationManager
- instance : ConfigurationManager = new ConfigurationManager()
- homeDir : String
- perspectFileName : String
- propertiesFileName : String
+ getInstance() : ConfigurationManager
- ConfigurationManager()
+ addPerspective(parent : TreeObject, newNode : TreeObject) : void
+ getPerspectiveRoot() : TreeObject
+ getProperties() : SystemProperties
+ getUserLevel() : UserLevel
- loadPerspectivesFromDisk() : TreeObject
- loadPropertiesFromDisk() : SystemProperties
+ removePerspective(selectedElement : TreeObject) : void
+ setUserLevel(userLevel : UserLevel) : void
+ writeSettingsToDisk() : void
Thread
- statement
- socketHandlerThread
SocketHandler
- host : String
+ SocketHandler(host : String, statement : Statement)
+ close() : void
+ run() : void

ActionController Der ActionController dient als Schnittstelle zwischen Dialogen und dem Modell/ Controller.

addTreeObject(String name, boolean leaf, TreeObject selectedObject)
Fügt ein neues TreeObject unter dem selektierten Object hinzu. Leaf = true: Perspektive, false: Ordner. Zudem wird der PerspectiveManager aktualisiert.

changeServerSettings(String odyUrl)
Verbindet mit einem neuen Odysseus mit der angegebenen URL.

changeStatementOfView(PageElement pageElement, Statement selectedStatement, HashMap<String, Integer> indexMapping)
Assoziiert ein Alternativ-Statement mit der gegebenen View.

createEventViewer()
Erzeugt einen neuen EventViewer und zeigt ihn an.

createNew3dGraph()
Erzeugt eine 3D-Welt und zeigt sie an.

createNewGraph(int statementID, String graphName, PageElement.ViewType type, boolean show)
Erzeugt einen neuen Graphen und zeigt ihn an.

createNewTable(int statementID, java.lang.String tablename, boolean show)
Erzeugt eine neue Tabelle und zeigt sie an.

createReplayStream(ILockable dialog, String streamName, String replayStreamName, long startTime, float replaySpeed)
Erzeugt die Wiedergabe eines zuvor aufgezeichneten Streams. Asynchron zur GUI.

createStream(ILockable dialog,String streamName, String xsdScheme, String spotURL, int spotPort)
Erzeugt einen neuen Stream. Asynchron zur GUI.

createViewPartCopy(String id, String subId, boolean removeSource)
Erzeugt eine Kopie eines ViewParts und löscht bei Bedarf die Quelle (verschieben). Die Kopie wird am Ende angezeigt.

lockCallbackDialog(ILockable callbackDialog)
Sperrt den gegebenen Dialog, da auf asynchrone Ergebnisse gewartet werden soll.

removeTreeObject(TreeObject selectedElement, TreeObject openedPerspective)
Entfernt das gegebene TreeObject, mit allen Kindknoten, und aktualisiert den PerspektivenManager.

selectStatement(ILockable dialog, String statement)
Erzeugt ein neues Statement. Die Daten des Statements können über Graphen oder Tabellen angezeigt werden.

showTopology()
Zeigt ein neues TopologyFenster an.

showView(PageElement element)
Zeigt einen zuvor erzeugten ViewPart wieder an.

startRecordingStream(ILockable dialog, String streamName)
Startet die Aufzeichnung eines Streams. Asynchron zur GUI.

ConfigurationManager Der ConfigurationManager ist zuständig für das Laden/Speichern der Konfiguration und der Perspektiven.

addPerspective(TreeObject parent, TreeObject newNode)
Hängt eine Perspektive/ Ordner an das parent TreeObject an.

getPerspectiveRoot()
Liefert oberstes Element der TreeObject Hierarchie.

removePerspective(TreeObject selectedElement)
Entfernt das TreeObject aus der Hierarchie der Objekte.

writeSettingsToDisk()
Schreibt die Konfiguration auf die Festplatte. Zu finden unter *home*/.ewaf.

Controller Der Controller implementiert das Observer-Muster. ViewParts können sich beim Controller registrieren und werden dann fortan aktualisiert, sobald neue Daten ankommen.

createDefaultCQLStatement(String sourceName)
Installiert eine SELECT * FROM sourceName Statement, falls es noch nicht existiert.

deregistersView(ITupleListener view)
Entfernt view vom Controller, sodass dieser nicht mehr aktualisiert wird.

load3DWorldMetaData()
Fordert die XML mit den gespeicherten 3D-Welten von Odysseus an.

notifyTopologies(String topologyName)
Benachrichtigt geöffnete Topologien, dass die Topologie topologyName verändert wurde.

processNext(Tuple tuple)
Verarbeitet das Tupel und übergibt es an die zugehörigen ContentProvider. Zusätzlich werden betroffene ViewParts aktualisiert.

registerView(ITupleListener view, int statementID)
Registriert die View, sodass diese bei neuen Daten des entsprechenden Statements aktualisiert wird.

renewStatementManager()
Erneuert die Verbindung zum Odysseus Webservice

requestContentProvider(int statementID)
Fordert den ContentProvider zur statementID an. Existiert dieser noch nicht wird ein neuer erstellt.

save3DWorldMetaData(String data)
Speichert die 3D-Welten als XML in Odysseus ab.

statementConsistentToMapping(int id, Map<String, Tuple.AttributeType> mapping)
Prüft ob das statement mit der id konsistent zum angegebenen mapping ist.

updateViewPartBuffers()
Aktualisiert die Anzeige der Buffergröße für alle ViewParts.

EventLogger Der EventLogger erstellt EventItems für das Ereignislog und die AlarmBar. Er hält alle Events und kann diese nach Bestätigung, Anfrage-ID oder Attributsnamen filtern. Ebenfalls kann der EventLogger Ereignisse als bestätigt markieren.

acknowledgeAll()
Markiert alle Ereignisse als bestätigt.

acknowledgeTuple(EventItem eItem)
Markiert das gegebene Ereignis als bestätigt.

addCriticalTuple(Tuple tuple)
Erzeugt Ereignisse aus einem Tupel mit kritschen Werten.

concatRawData(Tuple tuple, String[] attr)
Erzeugt einen String mit den Rohdaten des Tupels.

getUnacknowledgedEvents()
Liefert eine Liste mit unbestätigten Ereignissen, die in der AlarmBar angezeigt werden sollen.

getEventsPage(int page, int itemsPerPage)
Liefert eine Teilmenge der Ereignisse, so dass diese im Ereignislog angezeigt werden können.

SocketHandler Ein SocketHandler verbindet sich mit Odysseus und empfängt die Tupel einer installierten Anfrage. Dafür wird ein ObjectInputStream benutzt und die entsprechenden Methoden für die Basisdatentypen aufgerufen, so wie sie in dem Attribut-Mapping der Anfrage angegeben sind. Der Controller kann die Socketverbindung schließen, wenn diese nicht mehr benötigt wird.

StatementManager Der StatementManager kapselt die Webserviceaufrufe zu Odysseus und wandelt die Ergebnisse in die Datenstruktur der View um.

createStream(String streamName, String xsdScheme, String spotURL, int spotPort)
Erzeugt eine Quelle in Odysseus (vgl. CREATE STREAM Anfrage).

getRecordedStreams()
Liefert eine Map von aufgezeichneten Streams.

getScheme(int statementID)
Liefert die Schemainformation für eine Anfrage.

listInstalledStatements()
Liefert eine Liste von aktuell in Odysseus installierten Anfragen.

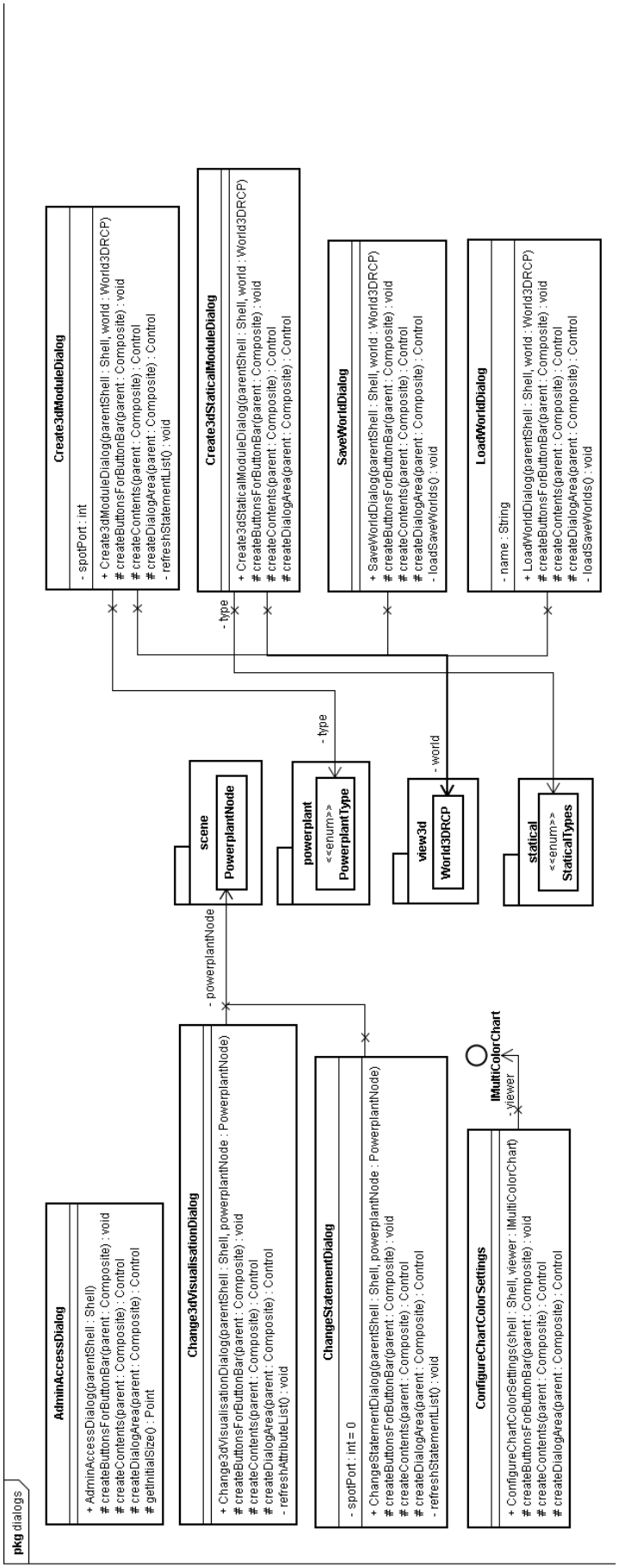
listStreams()
Liefert eine Liste von aktuell in Odysseus installierten Quellen.

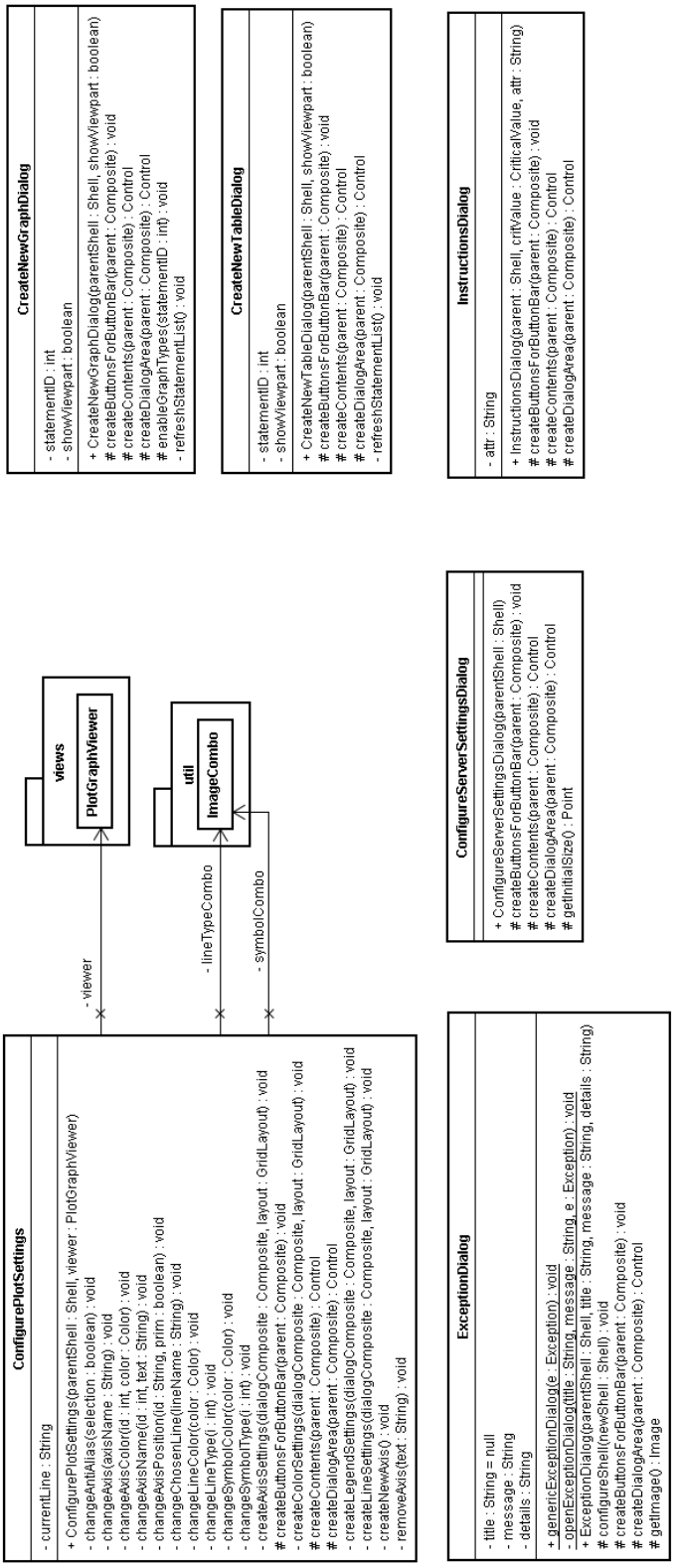
openDataConnection(String odysseusIP, Statement statement)
Erzeugt und startet einen SocketHandler Thread, der Tupel einer Anfrage empfangen soll.

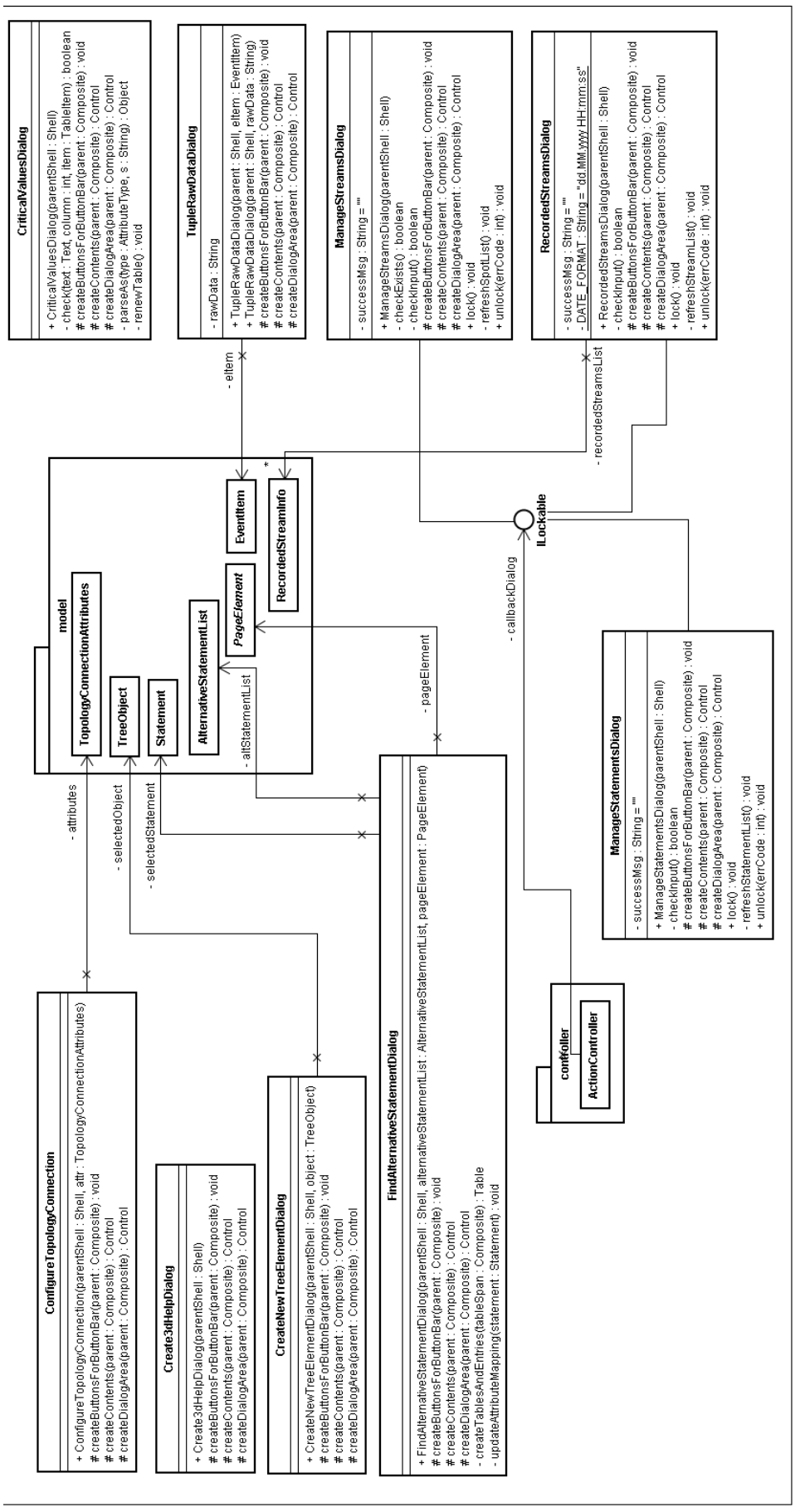
replayStream(String streamName, String replayStreamName, long startTime, float replaySpeed)
Erzeugt einen neuen Wiedergabe-Stream in Odysseus, der aufgezeichnete Daten wiedergibt.

selectStatement(String statement)
Installiert eine Select-CQL-Anfrage in Odysseus (vgl. SELECT ? FROM Anfrage).

dialogs


pkg dialogs
AdminAccessDialog
+ AdminAccessDialog(parentShell : Shell)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
# getInitialSize() : Point
Change3dVisualisationDialog
+ Change3dVisualisationDialog(parentShell : Shell, powerplantNode : PowerplantNode)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- refreshAttributeList() : void
ChangeStatementDialog
- spotPort : int = 0
+ ChangeStatementDialog(parentShell : Shell, powerplantNode : PowerplantNode)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- refreshStatementList() : void
ConfigureChartColorSettings
+ ConfigureChartColorSettings(shell : Shell, viewer : IMultiColorChart)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
IMultiColorChart
- viewer
Create3dModuleDialog
- spotPort : int
+ Create3dModuleDialog(parentShell : Shell, world : World3DRCP)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- refreshStatementList() : void
Create3dStaticalModuleDialog
+ Create3dStaticalModuleDialog(parentShell : Shell, world : World3DRCP)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
SaveWorldDialog
+ SaveWorldDialog(parentShell : Shell, world : World3DRCP)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- loadSaveWorlds() : void
LoadWorldDialog
- name : String
+ LoadWorldDialog(parentShell : Shell, world : World3DRCP)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- loadSaveWorlds() : void
- type
- type
- world
- powerplantNode
scene
PowerplantNode
powerplant
«enum»
PowerplantType
view3d
World3DRCP
statical
«enum»
StaticalTypes


CreateNewGraphDialog
- statementID : int
- showViewpart : boolean
+ CreateNewGraphDialog(parentShell : Shell, showViewpart : boolean)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
# enableGraphTypes(statementID : int) : void
- refreshStatementList() : void
CreateNewTableDialog
- statementID : int
- showViewpart : boolean
+ CreateNewTableDialog(parentShell : Shell, showViewpart : boolean)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- refreshStatementList() : void
InstructionsDialog
- attr : String
+ InstructionsDialog(parent : Shell, critValue : CriticalValue, attr : String)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
views
PlotGraphViewer
util
ImageCombo
- viewer
- lineTypeCombo
- symbolCombo
ConfigurePlotSettings
- currentLine : String
+ ConfigurePlotSettings(parentShell : Shell, viewer : PlotGraphViewer)
- changeAntiAlias(selection : boolean) : void
- changeAxis(axisName : String) : void
- changeAxisColor(id : int, color : Color) : void
- changeAxisName(id : int, text : String) : void
- changeAxisPosition(id : String, prim : boolean) : void
- changeChosenLine(lineName : String) : void
- changeLineColor(color : Color) : void
- changeLineType(i : int) : void
- changeSymbolColor(color : Color) : void
- changeSymbolType(i : int) : void
- createAxisSettings(dialogComposite : Composite, layout : GridLayout) : void
createButtonsForButtonBar(parent : Composite) : void
- createColorSettings(dialogComposite : Composite, layout : GridLayout) : void
createContents(parent : Composite) : Control
createDialogArea(parent : Composite) : Control
- createLegendSettings(dialogComposite : Composite, layout : GridLayout) : void
- createLineSettings(dialogComposite : Composite, layout : GridLayout) : void
- createNewAxis() : void
- removeAxis(text : String) : void
ConfigureServerSettingsDialog
+ ConfigureServerSettingsDialog(parentShell : Shell)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
# getInitialSize() : Point
ExceptionDialog
- title : String = null
- message : String
- details : String
+ genericExceptionDialog(e : Exception) : void
- openExceptionDialog(title : String, message : String, e : Exception) : void
+ ExceptionDialog(parentShell : Shell, title : String, message : String, details : String)
# configureShell(newShell : Shell) : void
# createButtonsForButtonBar(parent : Composite) : void
# createDialogArea(parent : Composite) : Control
# getImage() : Image


CriticalValuesDialog
+ CriticalValuesDialog(parentShell : Shell)
- check(text : Text, column : int, item : TableItem) : boolean
createButtonsForButtonBar(parent : Composite) : void
createContents(parent : Composite) : Control
createDialogArea(parent : Composite) : Control
- parseAs(type : AttributeType, s : String) : Object
- renewTable() : void
TupleRawDataDialog
- rawData : String
+ TupleRawDataDialog(parent : Shell, eItem : EventItem)
+ TupleRawDataDialog(parent : Shell, rawData : String)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
ManageStreamsDialog
- successMsg : String = ""
+ ManageStreamsDialog(parentShell : Shell)
- checkExists() : boolean
- checkInput() : boolean
createButtonsForButtonBar(parent : Composite) : void
createContents(parent : Composite) : Control
createDialogArea(parent : Composite) : Control
+ lock() : void
- refreshSpotList() : void
+ unlock(errCode : int) : void
RecordedStreamsDialog
- successMsg : String = ""
- DATE_FORMAT : String = "dd.MM.yyyy HH:mm:ss"
+ RecordedStreamsDialog(parentShell : Shell)
- checkInput() : boolean
createButtonsForButtonBar(parent : Composite) : void
createContents(parent : Composite) : Control
createDialogArea(parent : Composite) : Control
+ lock() : void
- refreshStreamList() : void
+ unlock(errCode : int) : void
- eItem
- recordedStreamsList
model
TopologyConnectionAttributes
TreeObject
Statement
AlternativeStatementList
PageElement
EventItem
RecordedStreamInfo
ILockable
- callbackDialog
- attributes
- selectedObject
- selectedStatement
- altStatementList
- pageElement
ConfigureTopologyConnection
+ ConfigureTopologyConnection(parentShell : Shell, attr : TopologyConnectionAttributes)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
Create3dHelpDialog
+ Create3dHelpDialog(parentShell : Shell)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
CreateNewTreeElementDialog
+ CreateNewTreeElementDialog(parentShell : Shell, object : TreeObject)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
FindAlternativeStatementDialog
+ FindAlternativeStatementDialog(parentShell : Shell, alternativeStatementList : AlternativeStatementList, pageElement : PageElement)
# createButtonsForButtonBar(parent : Composite) : void
# createContents(parent : Composite) : Control
# createDialogArea(parent : Composite) : Control
- createTablesAndEntries(tableSpan : Composite) : Table
- updateAttributeMapping(statement : Statement) : void
ManageStatementsDialog
- successMsg : String = ""
+ ManageStatementsDialog(parentShell : Shell)
- checkInput() : boolean
createButtonsForButtonBar(parent : Composite) : void
createContents(parent : Composite) : Control
createDialogArea(parent : Composite) : Control
+ lock() : void
- refreshStatementList() : void
+ unlock(errCode : int) : void
controller
ActionController

AdminAccessDialog Der AdminAccessDialog fordert die Eingabe des Administratorpassworts und ändert das User-Level.

ConfigureChartColorSettings Dialog um die Farben von Balken- und Kuchendiagrammen zu ändern.

ConfigurePlotSettings Dialog um Attribute eines X,Y - Diagramms zu ändern.

ConfigureServerSettingsDialog In dem ConfigureServerSettingsDialog kann die Odysseus URL geändert werden, so dass eine neue Odysseusinstanz ausgewählt werden kann und zu dieser verbunden wird.

ConfigureTopologyConnection Dialog zur Anpassung der Schwellwertfunktion von einer Topologie-Verbindung

CreateNewGraphDialog Dialog zur Erstellung eines neuen Graphens

CreateNewTableDialog Dialog zur Erstellung einer neuen Tabelle

CreateNewTreeElementDialog Dialog zur Erstellung einer neuen Perspektive bzw. eines neuen Ordners

CriticalValuesDialog Der CriticalValuesDialog listet die lokalen Einstellungen der kritischen Werte auf. Grenzen für die kritischen Werte können hinzugefügt, entfernt und bearbeitet werden.

ExceptionDialog Der ExceptionDialog kann zum Anzeigen einer Exception verwendet werden. Dieser informiert den Benutzer über das Problem und kann optional den Stacktrace anzeigen.

FindAlternativeStatementDialog Dialog, um einen StatementViewPart mit einer alternativen Anfrage zu versorgen.

ILockable Dialoge, die indirekt mit Odysseus kommunizieren, sollten das Interface ILockable implementieren. Für solche Dialoge kann der ActionController dann eine callback Funktionalität bereitstellen, wenn der Webserviceaufruf beendet wurde.

InstructionsDialog Der InstructionsDialog dient zur Anzeige von lokal gespeicherten Handlungsanweisungen für Attribute, dessen Grenzen überschritten wurden. Administratoren können diese Anweisungen über den CriticalValues-Dialog editieren.

ManageStatementsDialog Der ManageStatementsDialog listet alle aktuell in Odysseus installierten Anfragen auf. Es können neue Anfragen geschrieben und installiert werden.

ManageStreamsDialog Der ManageStatementsDialog listet alle aktuell in Odysseus eingerichteten Quellen auf. Neue Quellen können mit der dazugehörigen IP und dem Port, sowie dem Schema, hinzugefügt werden. Es können Quellen aufgezeichnet werden, um diese später wiederzugeben. Dazu kann die Aufnahme für jeden Stream gestartet und gestoppt werden.

RecordedStreamsDialog Der RecordedStreamsDialog listet alle Streams auf, für die aufgezeichnete Daten vorhanden sind. Diese Daten können in einem neuen Wiedergabe-Stream, der in diesem Dialog erstellt werden kann, erneut gesendet werden. Die Aufzeichnung für die hier aufgelisteten Streams kann gestoppt werden.

TupleRawDataDialog Der TupleRawDataDialog zeigt alle Informationen an, die zu einem Tupel gehören. Das sind hauptsächlich die Attribute und Überschreitungen von Grezen. Wenn der Dialog über das Ereignislog aufgerufen wird, können alle zu diesem Tupel gehörigen Ereignisse bestätigt werden.

Create3dModuleDialog Dialog zum Erstellen von Kraftwerken. Bei der Erstellung ist es möglich direkt Statements auf die Kraftwerke zu legen und somit Tupel über dem Kraftwerk anzuzeigen.

Create3dStaticalModuleDialog Auch die statischen Modelle können über einen Dialog zur Welt hinzugefügt werden. Dabei wird über ein Dropdown-Menü einfach nur das Modell ausgesucht und los geht es.

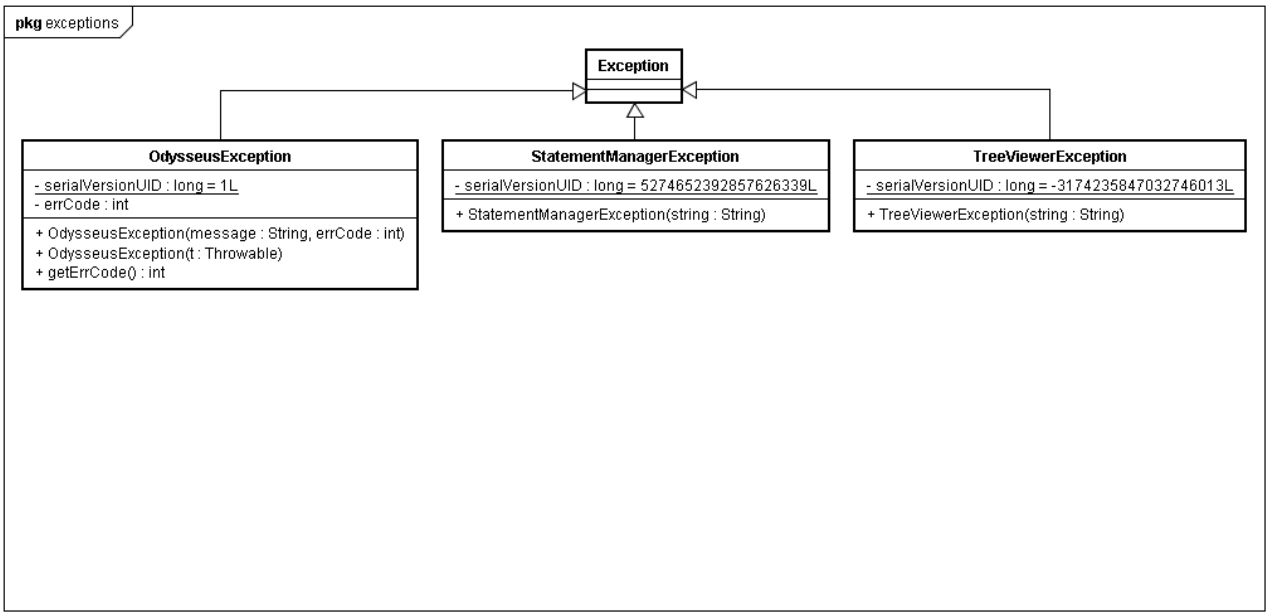
SaveWorldDialog Das Speichern der 3D-Welt funktioniert über einen Dialog. In diesem kann für das Szenario ein Name gewählt werden oder bereits gespeicherte Szenarien überschrieben werden.

LoadWorldDialog Mit Hilfe dieses Dialogs lassen sich gespeicherte Szenarien wieder laden.

Create3dHelpDialog Dieser Dialog gibt Hilfestellung bei der Bewegung und Bedienung in der 3D-Welt.

ChangeStatementDialog Dieser Dialog ermöglicht es die Statements die auf einem Kraftwerk liegen im Nachhinein zu ändern.

exceptions

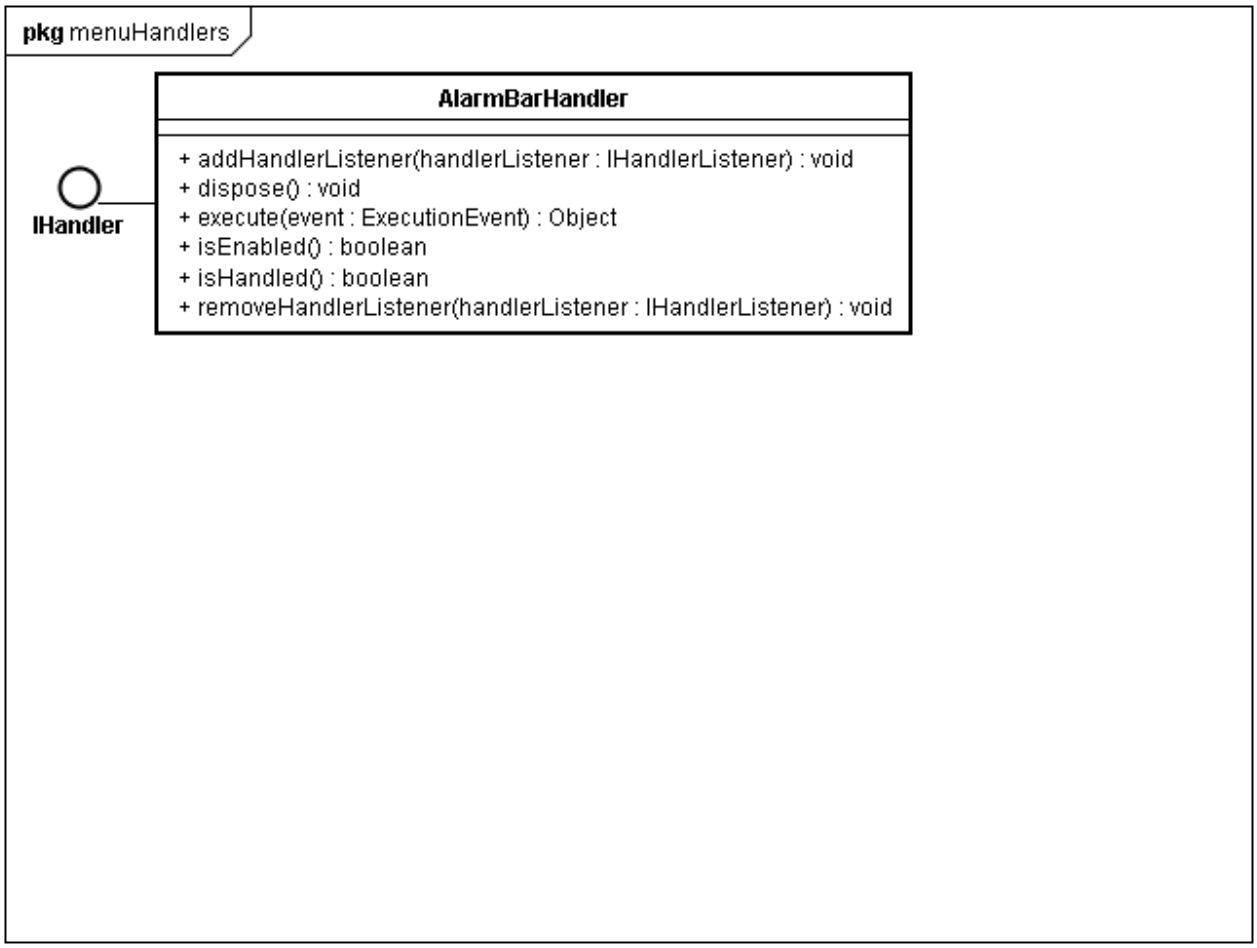


OdysseusException Eine OdysseusException ist eine interne Exception, die einen Fehler bei der Kommunikation mit Odysseus über den WebService repräsentiert. Ein Errorcode kann benutzt werden, um den Fehler zu identifizieren. WEBSERVICE_FAILURE bedeutet, dass es ein Problem mit der Konnektivität zu dem WebService gibt.

StatementManagerException Interne Exception, die geworfen wird, falls der StatementManager mit falschen Informationen versorgt wird, z.B. eine nicht existente Statement-ID.

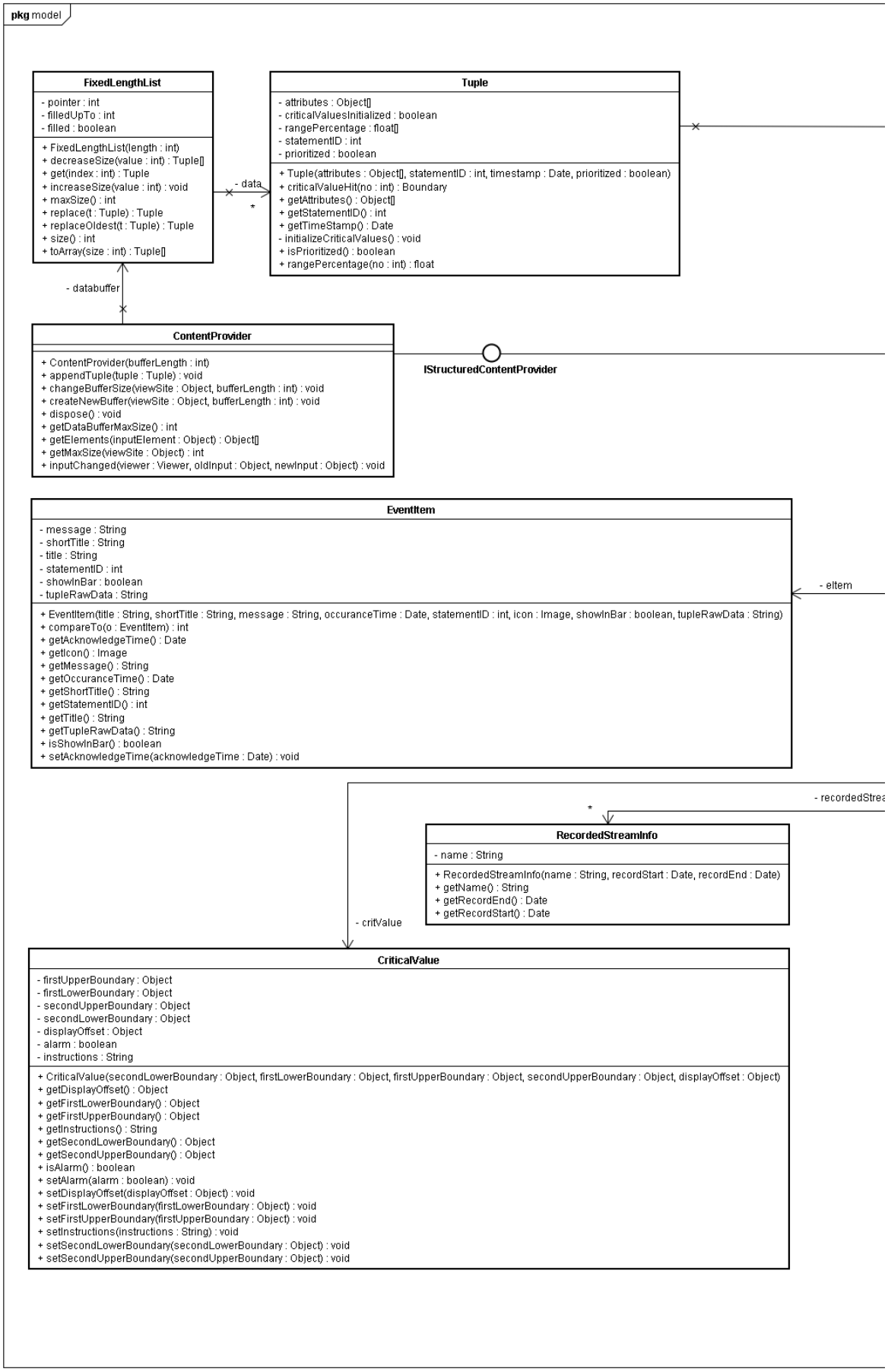
TreeViewerException Interne Exception, die geworfen wird falls irreguläre Aktionen im TreeViewer durchgeführt werden.

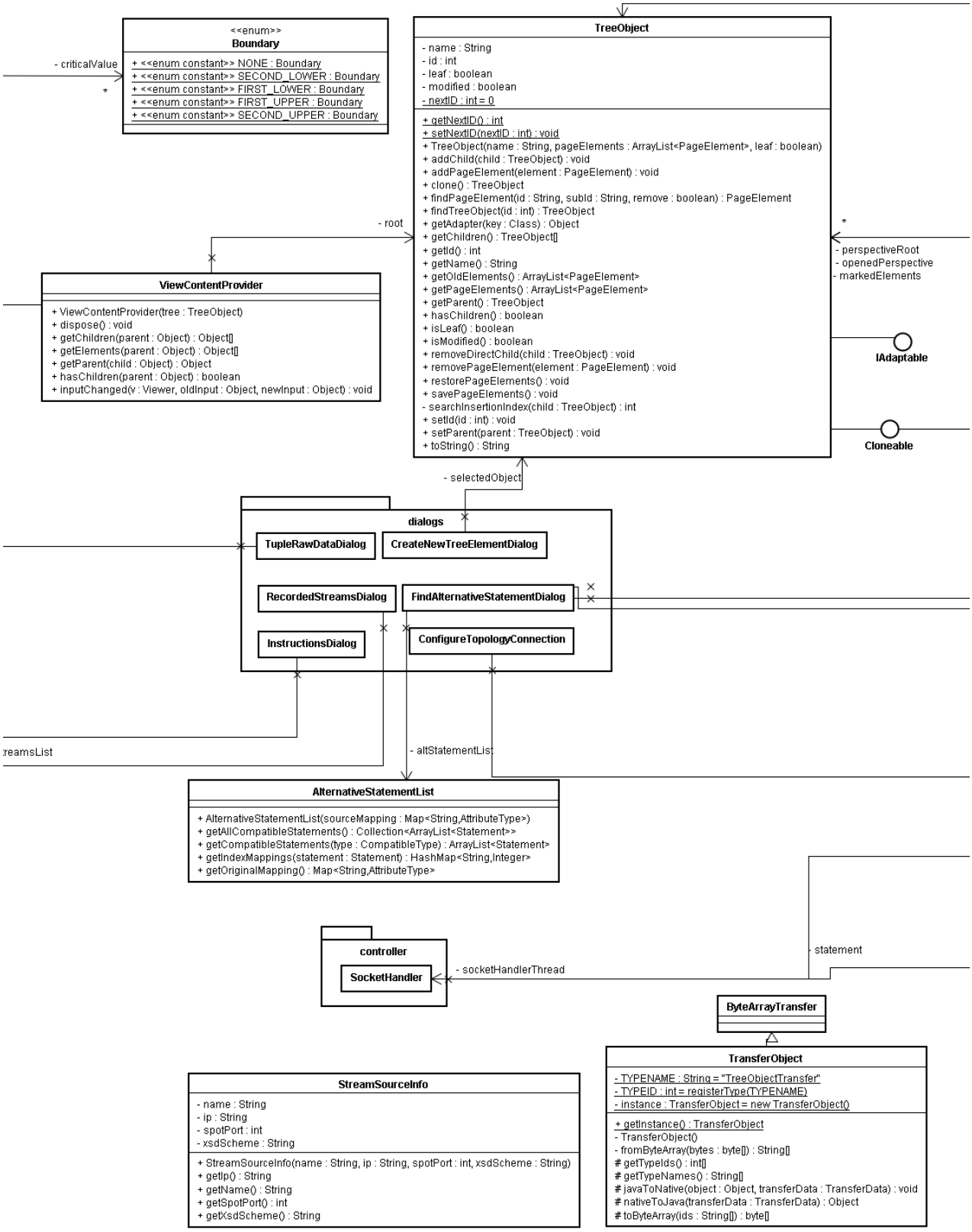
menuHandlers

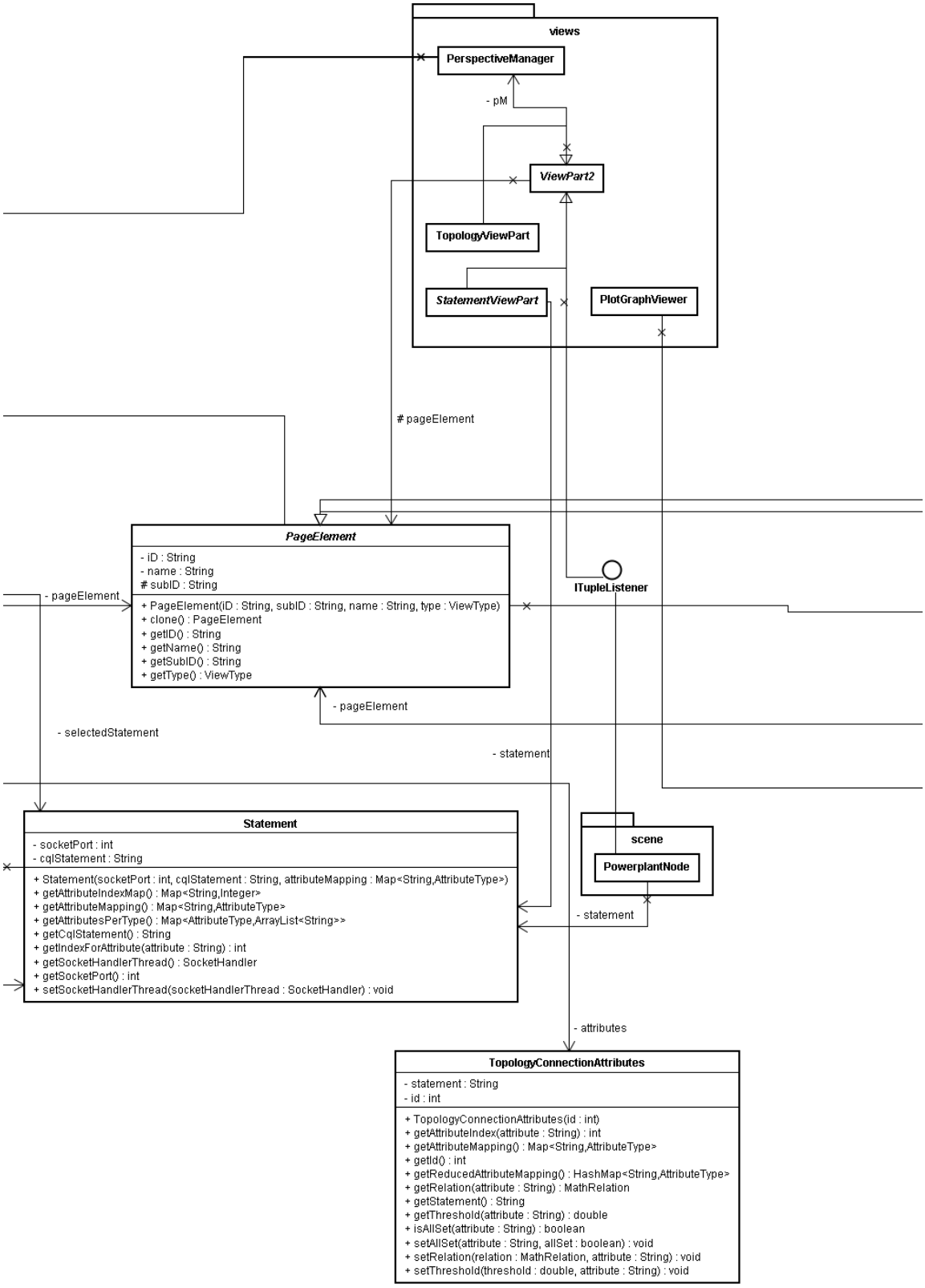


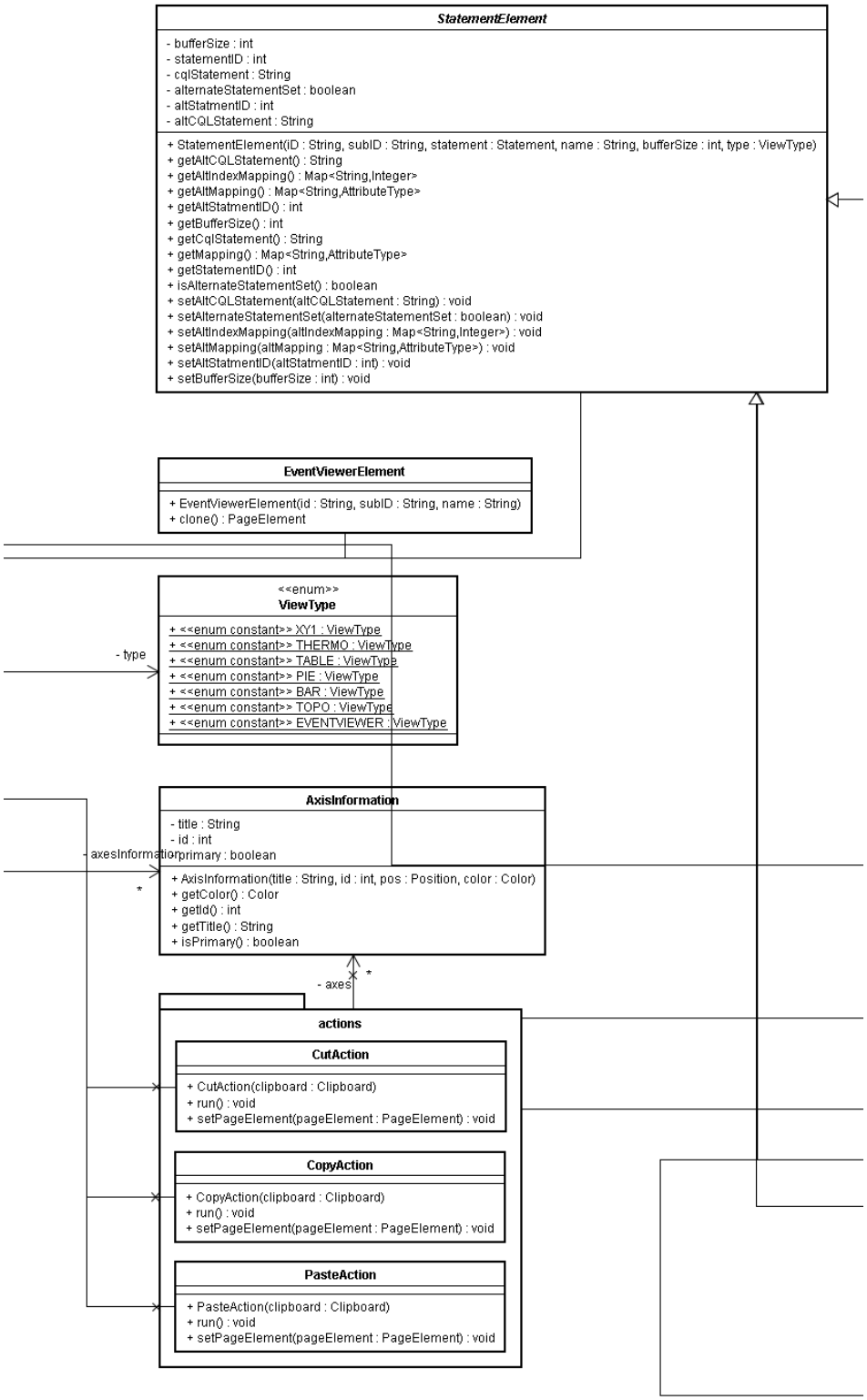
AlarmBarHandler MenüHandler, zeigt die AlarmBar, wenn der Eintrag im Menü ausgewählt wurde

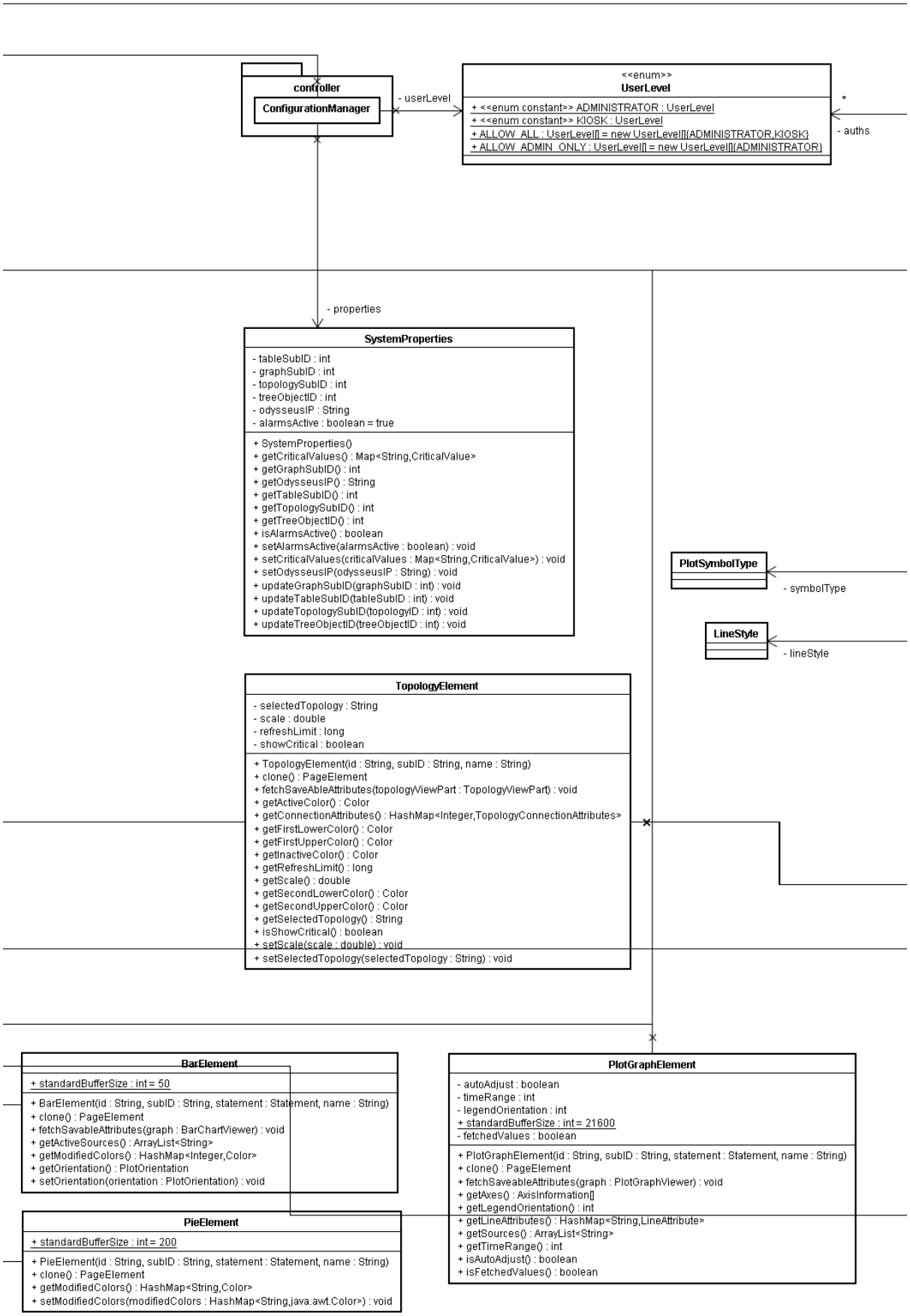
model

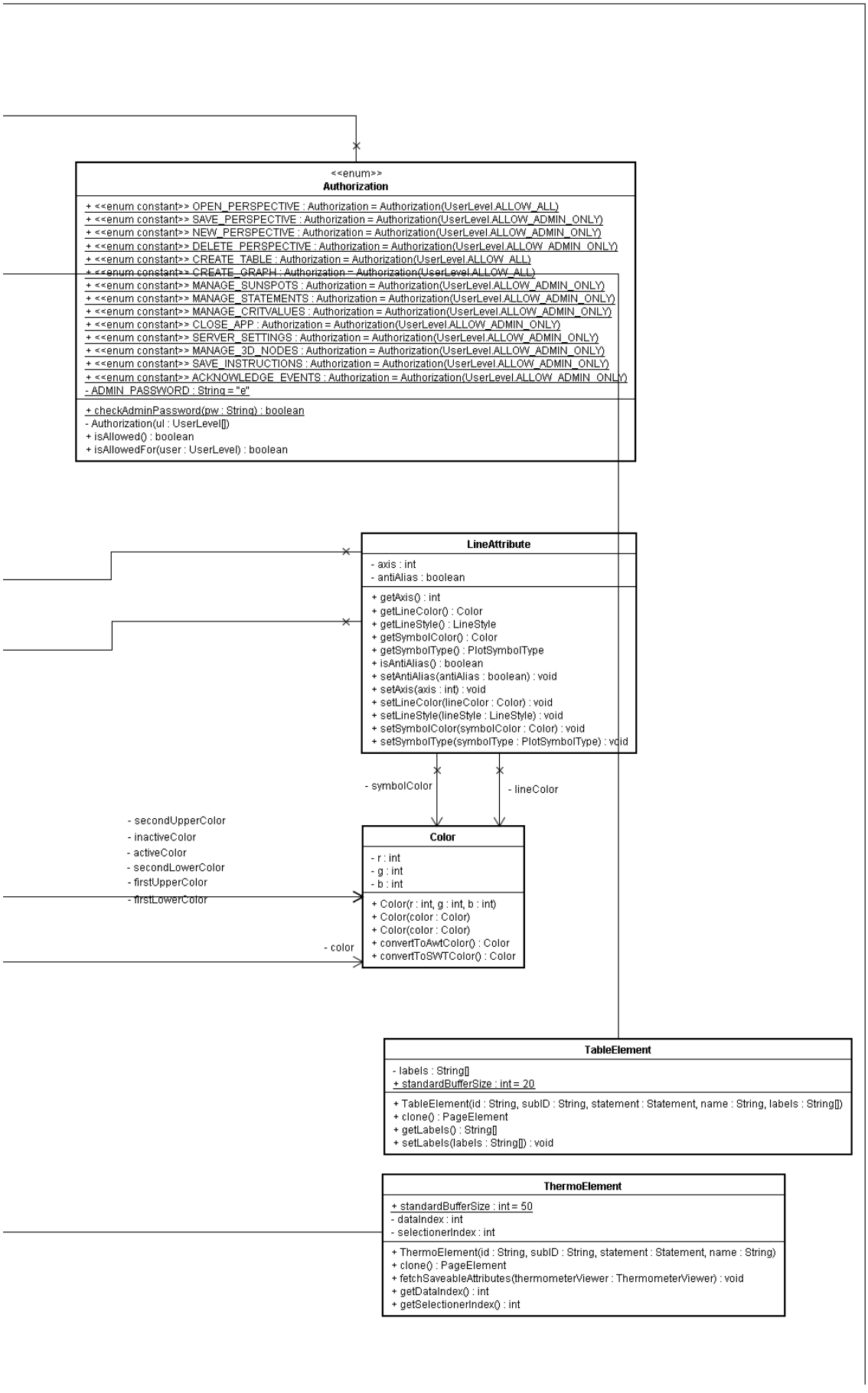

pkg model
FixedLengthList
- pointer : int
- filledUpTo : int
- filled : boolean
+ FixedLengthList(length : int)
+ decreaseSize(value : int) : Tuple[]
+ get(index : int) : Tuple
+ increaseSize(value : int) : void
+ maxSize() : int
+ replace(t : Tuple) : Tuple
+ replaceOldest(t : Tuple) : Tuple
+ size() : int
+ toArray(size : int) : Tuple[]
- data
*
Tuple
- attributes : Object[]
- criticalValuesInitialized : boolean
- rangePercentage : float[]
- statementID : int
- prioritized : boolean
+ Tuple(attributes : Object[], statementID : int, timestamp : Date, prioritized : boolean)
+ criticalValueHit(no : int) : Boundary
+ getAttributes() : Object[]
+ getStatementID() : int
+ getTimeStamp() : Date
- initializeCriticalValues() : void
+ isPrioritized() : boolean
+ rangePercentage(no : int) : float
- databuffer
ContentProvider
+ ContentProvider(bufferLength : int)
+ appendTuple(tuple : Tuple) : void
+ changeBufferSize(viewSite : Object, bufferLength : int) : void
+ createNewBuffer(viewSite : Object, bufferLength : int) : void
+ dispose() : void
+ getDataBufferMaxSize() : int
+ getElements(inputElement : Object) : Object[]
+ getMaxSize(viewSite : Object) : int
+ inputChanged(viewer : Viewer, oldInput : Object, newInput : Object) : void
IStructuredContentProvider
EventItem
- message : String
- shortTitle : String
- title : String
- statementID : int
- showInBar : boolean
- tupleRawData : String
+ EventItem(title : String, shortTitle : String, message : String, occuranceTime : Date, statementID : int, icon : Image, showInBar : boolean, tupleRawData : String)
+ compareTo(o : EventItem) : int
+ getAcknowledgeTime() : Date
+ getIcon() : Image
+ getMessage() : String
+ getOccuranceTime() : Date
+ getShortTitle() : String
+ getStatementID() : int
+ getTitle() : String
+ getTupleRawData() : String
+ isShowInBar() : boolean
+ setAcknowledgeTime(acknowledgeTime : Date) : void
- eItem
- recordedStrea
*
RecordedStreamInfo
- name : String
+ RecordedStreamInfo(name : String, recordStart : Date, recordEnd : Date)
+ getName() : String
+ getRecordEnd() : Date
+ getRecordStart() : Date
- critValue
CriticalValue
- firstUpperBoundary : Object
- firstLowerBoundary : Object
- secondUpperBoundary : Object
- secondLowerBoundary : Object
- displayOffset : Object
- alarm : boolean
- instructions : String
+ CriticalValue(secondLowerBoundary : Object, firstLowerBoundary : Object, firstUpperBoundary : Object, secondUpperBoundary : Object, displayOffset : Object)
+ getDisplayOffset() : Object
+ getFirstLowerBoundary() : Object
+ getFirstUpperBoundary() : Object
+ getInstructions() : String
+ getSecondLowerBoundary() : Object
+ getSecondUpperBoundary() : Object
+ isAlarm() : boolean
+ setAlarm(alarm : boolean) : void
+ setDisplayOffset(displayOffset : Object) : void
+ setFirstLowerBoundary(firstLowerBoundary : Object) : void
+ setFirstUpperBoundary(firstUpperBoundary : Object) : void
+ setInstructions(instructions : String) : void
+ setSecondLowerBoundary(secondLowerBoundary : Object) : void
+ setSecondUpperBoundary(secondUpperBoundary : Object) : void


«enum»
Boundary
+ «enum constant»» NONE : Boundary
+ «enum constant»» SECOND_LOWER : Boundary
+ «enum constant»» FIRST_LOWER : Boundary
+ «enum constant»» FIRST_UPPER : Boundary
+ «enum constant»» SECOND_UPPER : Boundary
- criticalValue
*
TreeObject
- name : String
- id : int
- leaf : boolean
- modified : boolean
- nextID : int = 0
+ getNextID() : int
+ setNextID(nextID : int) : void
+ TreeObject(name : String, pageElements : ArrayList<PageElement>, leaf : boolean)
+ addChild(child : TreeObject) : void
+ addPageElement(element : PageElement) : void
+ clone() : TreeObject
+ findPageElement(id : String, subId : String, remove : boolean) : PageElement
+ findTreeObject(id : int) : TreeObject
+ getAdapter(key : Class) : Object
+ getChildren() : TreeObject[]
+ getId() : int
+ getName() : String
+ getOldElements() : ArrayList<PageElement>
+ getPageElements() : ArrayList<PageElement>
+ getParent() : TreeObject
+ hasChildren() : boolean
+ isLeaf() : boolean
+ isModified() : boolean
+ removeDirectChild(child : TreeObject) : void
+ removePageElement(element : PageElement) : void
+ restorePageElements() : void
+ savePageElements() : void
- searchInsertionIndex(child : TreeObject) : int
+ setId(id : int) : void
+ setParent(parent : TreeObject) : void
+ toString() : String
- root
*
- perspectiveRoot
- openedPerspective
- markedElements
IAdaptable
Cloneable
ViewContentProvider
+ ViewContentProvider(tree : TreeObject)
+ dispose() : void
+ getChildren(parent : Object) : Object[]
+ getElements(parent : Object) : Object[]
+ getParent(child : Object) : Object
+ hasChildren(parent : Object) : boolean
+ inputChanged(v : Viewer, oldInput : Object, newInput : Object) : void
- selectedObject
dialogs
TupleRawDataDialog
CreateNewTreeElementDialog
RecordedStreamsDialog
FindAlternativeStatementDialog
InstructionsDialog
ConfigureTopologyConnection
reamsList
- altStatementList
AlternativeStatementList
+ AlternativeStatementList(sourceMapping : Map<String,AttributeType>)
+ getAllCompatibleStatements() : Collection<ArrayList<Statement>>
+ getCompatibleStatements(type : CompatibleType) : ArrayList<Statement>
+ getIndexMappings(statement : Statement) : HashMap<String,Integer>
+ getOriginalMapping() : Map<String,AttributeType>
controller
SocketHandler
- socketHandlerThread
- statement
ByteArrayTransfer
TransferObject
- TYPENAME : String = "TreeObjectTransfer"
- TYPEID : int = registerType(TYPENAME)
- instance : TransferObject = new TransferObject()
+ getInstance() : TransferObject
- TransferObject()
- fromByteArray(bytes : byte[]) : String[]
getTypeIds() : int[]
getTypeNames() : String[]
javaToNative(object : Object, transferData : TransferData) : void
nativeToJava(transferData : TransferData) : Object
toByteArray(ids : String[]) : byte[]
StreamSourceInfo
- name : String
- ip : String
- spotPort : int
- xsdScheme : String
+ StreamSourceInfo(name : String, ip : String, spotPort : int, xsdScheme : String)
+ getIp() : String
+ getName() : String
+ getSpotPort() : int
+ getXsdScheme() : String


views
PerspectiveManager
- pM
ViewPart2
TopologyViewPart
StatementViewPart
PlotGraphViewer
pageElement
PageElement
- iD : String
- name : String
subID : String
+ PageElement(iD : String, subID : String, name : String, type : ViewType)
+ clone() : PageElement
+ getID() : String
+ getName() : String
+ getSubID() : String
+ getType() : ViewType
ITupleListener
- pageElement
- pageElement
- selectedStatement
- statement
Statement
- socketPort : int
- cqlStatement : String
+ Statement(socketPort : int, cqlStatement : String, attributeMapping : Map<String,AttributeType>)
+ getAttributeIndexMap() : Map<String,Integer>
+ getAttributeMapping() : Map<String,AttributeType>
+ getAttributesPerType() : Map<AttributeType,ArrayList<String>>
+ getCqlStatement() : String
+ getIndexForAttribute(attribute : String) : int
+ getSocketHandlerThread() : SocketHandler
+ getSocketPort() : int
+ setSocketHandlerThread(socketHandlerThread : SocketHandler) : void
scene
PowerplantNode
- statement
- attributes
TopologyConnectionAttributes
- statement : String
- id : int
+ TopologyConnectionAttributes(id : int)
+ getAttributeIndex(attribute : String) : int
+ getAttributeMapping() : Map<String,AttributeType>
+ getId() : int
+ getReducedAttributeMapping() : HashMap<String,AttributeType>
+ getRelation(attribute : String) : MathRelation
+ getStatement() : String
+ getThreshold(attribute : String) : double
+ isAllSet(attribute : String) : boolean
+ setAllSet(attribute : String, allSet : boolean) : void
+ setRelation(relation : MathRelation, attribute : String) : void
+ setThreshold(threshold : double, attribute : String) : void


StatementElement
- bufferSize : int
- statementID : int
- cqlStatement : String
- alternateStatementSet : boolean
- altStatmentID : int
- altCQLStatement : String
+ StatementElement(iD : String, subID : String, statement : Statement, name : String, bufferSize : int, type : ViewType)
+ getAltCQLStatement() : String
+ getAltIndexMapping() : Map<String,Integer>
+ getAltMapping() : Map<String,AttributeType>
+ getAltStatmentID() : int
+ getBufferSize() : int
+ getCqlStatement() : String
+ getMapping() : Map<String,AttributeType>
+ getStatementID() : int
+ isAlternateStatementSet() : boolean
+ setAltCQLStatement(altCQLStatement : String) : void
+ setAlternateStatementSet(alternateStatementSet : boolean) : void
+ setAltIndexMapping(altIndexMapping : Map<String,Integer>) : void
+ setAltMapping(altMapping : Map<String,AttributeType>) : void
+ setAltStatmentID(altStatmentID : int) : void
+ setBufferSize(bufferSize : int) : void
EventViewerElement
+ EventViewerElement(id : String, subID : String, name : String)
+ clone() : PageElement
<<enum>>
ViewType
+ <<enum constant>> XY1 : ViewType
+ <<enum constant>> THERMO : ViewType
+ <<enum constant>> TABLE : ViewType
+ <<enum constant>> PIE : ViewType
+ <<enum constant>> BAR : ViewType
+ <<enum constant>> TOPO : ViewType
+ <<enum constant>> EVENTVIEWER : ViewType
- type
AxisInformation
- title : String
- id : int
- primary : boolean
+ AxisInformation(title : String, id : int, pos : Position, color : Color)
+ getColor() : Color
+ getId() : int
+ getTitle() : String
+ isPrimary() : boolean
- axesInformation
*
- axes
*
actions
CutAction
+ CutAction(clipboard : Clipboard)
+ run() : void
+ setPageElement(pageElement : PageElement) : void
CopyAction
+ CopyAction(clipboard : Clipboard)
+ run() : void
+ setPageElement(pageElement : PageElement) : void
PasteAction
+ PasteAction(clipboard : Clipboard)
+ run() : void
+ setPageElement(pageElement : PageElement) : void


controller
ConfigurationManager
- userLevel
<<enum>>
UserLevel
+ <<enum constant>> ADMINISTRATOR : UserLevel
+ <<enum constant>> KIOSK : UserLevel
+ ALLOW_ALL : UserLevel[] = new UserLevel[]{ADMINISTRATOR,KIOSK}
+ ALLOW_ADMIN_ONLY : UserLevel[] = new UserLevel[]{ADMINISTRATOR}
*
- auths
- properties
SystemProperties
- tableSubID : int
- graphSubID : int
- topologySubID : int
- treeObjectID : int
- odysseusIP : String
- alarmsActive : boolean = true
+ SystemProperties()
+ getCriticalValues() : Map<String,CriticalValue>
+ getGraphSubID() : int
+ getOdysseusIP() : String
+ getTableSubID() : int
+ getTopologySubID() : int
+ getTreeObjectID() : int
+ isAlarmsActive() : boolean
+ setAlarmsActive(alarmsActive : boolean) : void
+ setCriticalValues(criticalValues : Map<String,CriticalValue>) : void
+ setOdysseusIP(odysseusIP : String) : void
+ updateGraphSubID(graphSubID : int) : void
+ updateTableSubID(tableSubID : int) : void
+ updateTopologySubID(topologyID : int) : void
+ updateTreeObjectID(treeObjectID : int) : void
PlotSymbolType
- symbolType
LineStyle
- lineStyle
TopologyElement
- selectedTopology : String
- scale : double
- refreshLimit : long
- showCritical : boolean
+ TopologyElement(id : String, subID : String, name : String)
+ clone() : PageElement
+ fetchSaveAbleAttributes(topologyViewPart : TopologyViewPart) : void
+ getActiveColor() : Color
+ getConnectionAttributes() : HashMap<Integer,TopologyConnectionAttributes>
+ getFirstLowerColor() : Color
+ getFirstUpperColor() : Color
+ getInactiveColor() : Color
+ getRefreshLimit() : long
+ getScale() : double
+ getSecondLowerColor() : Color
+ getSecondUpperColor() : Color
+ getSelectedTopology() : String
+ isShowCritical() : boolean
+ setScale(scale : double) : void
+ setSelectedTopology(selectedTopology : String) : void
BarElement
+ standardBufferSize : int = 50
+ BarElement(id : String, subID : String, statement : Statement, name : String)
+ clone() : PageElement
+ fetchSavableAttributes(graph : BarChartViewer) : void
+ getActiveSources() : ArrayList<String>
+ getModifiedColors() : HashMap<Integer,Color>
+ getOrientation() : PlotOrientation
+ setOrientation(orientation : PlotOrientation) : void
PieElement
+ standardBufferSize : int = 200
+ PieElement(id : String, subID : String, statement : Statement, name : String)
+ clone() : PageElement
+ getModifiedColors() : HashMap<String,Color>
+ setModifiedColors(modifiedColors : HashMap<String,java.awt.Color>) : void
PlotGraphElement
- autoAdjust : boolean
- timeRange : int
- legendOrientation : int
+ standardBufferSize : int = 21600
- fetchedValues : boolean
+ PlotGraphElement(id : String, subID : String, statement : Statement, name : String)
+ clone() : PageElement
+ fetchSaveableAttributes(graph : PlotGraphViewer) : void
+ getAxes() : AxisInformation[]
+ getLegendOrientation() : int
+ getLineAttributes() : HashMap<String,LineAttribute>
+ getSources() : ArrayList<String>
+ getTimeRange() : int
+ isAutoAdjust() : boolean
+ isFetchedValues() : boolean


«enum»
Authorization
+ «enum constant» OPEN_PERSPECTIVE : Authorization = Authorization(UserLevel.ALLOW_ALL)
+ «enum constant» SAVE_PERSPECTIVE : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» NEW_PERSPECTIVE : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» DELETE_PERSPECTIVE : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» CREATE_TABLE : Authorization = Authorization(UserLevel.ALLOW_ALL)
+ «enum constant» CREATE_GRAPH : Authorization = Authorization(UserLevel.ALLOW_ALL)
+ «enum constant» MANAGE_SUNSPOTS : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» MANAGE_STATEMENTS : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» MANAGE_CRITVALUES : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» CLOSE_APP : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» SERVER_SETTINGS : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» MANAGE_3D_NODES : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» SAVE_INSTRUCTIONS : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
+ «enum constant» ACKNOWLEDGE_EVENTS : Authorization = Authorization(UserLevel.ALLOW_ADMIN_ONLY)
- ADMIN_PASSWORD : String = "e"
+ checkAdminPassword(pw : String) : boolean
- Authorization(ul : UserLevel[])
+ isAllowed() : boolean
+ isAllowedFor(user : UserLevel) : boolean
LineAttribute
- axis : int
- antiAlias : boolean
+ getAxis() : int
+ getLineColor() : Color
+ getLineStyle() : LineStyle
+ getSymbolColor() : Color
+ getSymbolType() : PlotSymbolType
+ isAntiAlias() : boolean
+ setAntiAlias(antiAlias : boolean) : void
+ setAxis(axis : int) : void
+ setLineColor(lineColor : Color) : void
+ setLineStyle(lineStyle : LineStyle) : void
+ setSymbolColor(symbolColor : Color) : void
+ setSymbolType(symbolType : PlotSymbolType) : void
- symbolColor
- lineColor
- secondUpperColor
- inactiveColor
- activeColor
- secondLowerColor
- firstUpperColor
- firstLowerColor
Color
- r : int
- g : int
- b : int
+ Color(r : int, g : int, b : int)
+ Color(color : Color)
+ Color(color : Color)
+ convertToAwtColor() : Color
+ convertToSWTColor() : Color
- color
TableElement
- labels : String[]
+ standardBufferSize : int = 20
+ TableElement(id : String, subID : String, statement : Statement, name : String, labels : String[])
+ clone() : PageElement
+ getLabels() : String[]
+ setLabels(labels : String[]) : void
ThermoElement
+ standardBufferSize : int = 50
- dataIndex : int
- selectionerIndex : int
+ ThermoElement(id : String, subID : String, statement : Statement, name : String)
+ clone() : PageElement
+ fetchSaveableAttributes(thermometerViewer : ThermometerViewer) : void
+ getDataIndex() : int
+ getSelectionerIndex() : int

AlternativeStatementList Erzeugt eine Liste kompatibler Anfragen. Dazu wird das Mapping der originalen Anfragen benötigt.

Authorization Das Authorization enum definiert, welche Aktionen auf welchem UserLevel erlaubt sind. CONSTANT.isAllowed() kann dazu benutzt werden, um zu prüfen, ob eine Aktion mit dem aktuellen UserLevel erlaubt ist.

AxisInformation Leichtgewichtiges Modell, das eine Achse eines X,Y - Diagramms repräsentiert. Wird für Serialisierung genutzt.

BarElement Modell für ein Balkendiagramm. Hält alle Attribute. Wird für Serialisierung genutzt.

fetchSavableAttributes(BarChartViewer graph)
Holt alle relevanten Attribute aus dem Graphen und speichert sie.

Color Leichtgewichtiges Modell, welches eine Farbe repräsentiert. Wird für Serialisierung genutzt.

convertToAwtColor()
Erzeugt eine AWT-Farbe basierend auf den RGB-Werten des Color Objekts.

convertToSWTColor()
Erzeugt eine SWT-Farbe basierend auf den RGB-Werten des Color Objekts.

ContentProvider Der ContentProvider beliefert eine Menge von StatementViewparts mit Daten, die angezeigt werden sollen. Dabei werden x Tupel gepuffert.

appendTuple(Tuple tuple)
Fügt ein Tupel in den Ringpuffer des ContentProviders ein.

changeBufferSize(Object viewSite, int bufferLength)
Ordnet der angegebenen viewSite eine Buffergröße zu, sodass diese nur noch eine begrenzte Anzahl an Tupeln erhält.

createNewBuffer(Object viewSite, int bufferLength)
Erzeugt eine neuen Puffer-Bereich mit angegebener Größe zur viewSite.

CriticalValue Information über die Grenzen bei einem Attribut.

EventItem Ein EventItem repräsentiert das Auftreten eines kritischen Wertes und hält nähere Informationen zu diesem Ereignis. Diese Informationen müssen einmalig generiert werden und können dann im Ereignislog und in der Alarmleiste benutzt werden.

compareTo(EventItem o)
Zur Sortierung nach Datum des Auftretens.

EventViewerElement Das EventViewerElement repräsentiert ein Ereignislog. Es erlaubt die Wiederherstellung beim Perspektivenwechsel.

FixedLengthList Die FixedLengthList bietet eine spezielle Liste für den ContentProvider an. Die Liste hat eine feste Länge und funktioniert wie ein Ringpuffer. Elemente können nur über die replace() Methode hinzugefügt werden, wodurch gleichzeitig das älteste Element entfernt wird. Auf die Elemente kann über die get(index) Methode zugegriffen werden. Die Benutzung einer FixedLengthList anstatt einer ArrayList in dem ContentProvider bringt einen deutlichen Performaceschub.

decreaseSize(int value)
Verringert die Größe der Liste und liefert die entfernten Elemente zurück.

get(int index)
Liefert das Element bei dem gegebenen Index.

increaseSize(int value)
Vergrößert die Liste.

replaceOldest(Tuple t)
Ersetzt das nach Zeitstempel älteste Element und fügt das Neue chronologisch in die Liste ein. Liefert das entfernte Element zurück.

toArray(int size)
Liefert die Liste als Array zurück mit maximal size Elementen.

ITupleListener Ein ITupleListener ist entweder ein ViewPart2 oder eine 3D-Welt Klasse. ViewPart2s werden bei neuen Tupeln aktualisiert, andere Klassen bekommen nur das Tupel über appendTuple().

LineAttribute Modell, das die Attribute einer Linie eines X,Y - Diagramms wiederspiegelt. Wird für die Serialisierung genutzt.

PageElement Das PageElement repräsentiert einen ViewPart mit allen Eigenschaften. Es erlaubt die Wiederherstellung von geschlossenen Viewparts bzw. die Wiederherstellung nach einem Neustart der Anwendung.
clone()
Erstellt eine Kopie des PageElement. Entsprechende IDs werden verändert.

PieElement Spezielles StatementElement, das ein Kuchendiagramm mit allen Eigenschaften repräsentiert.

PlotGraphElement Spezielles StatementElement, das ein X,Y - Diagramm mit allen Eigenschaften repräsentiert.
fetchSaveableAttributes(PlotGraphViewer graph)
Holt alle zu speichernden Attribute aus dem Graphen.

RecordedStreamInfo Ein RecordedStreamInfo hält Metainformationen über einen Stream, für den Daten aufgezeichnet wurden. Es enthält den Namen des Streams sowie Zeitstempel von den ersten und letzten aufgezeichneten Daten.

Statement Ein Statement hält Informationen und eine installierte Anfrage.

StatementElement Spezielles PageElement, das ein StatementViewpart repräsentiert. Hält alle Eigenschaften des Viewparts. Zudem werden original und alternativ Anfrage inklusive Mapping gespeichert.

StreamSourceInfo Ein StreamSourceInfo hält Informationen und eine eingerichtete Quelle in Odysseus.

SystemProperties Hält einige globale Einstellung, die serialisiert werden müssen.

TableElement Spezielles StatementElement, das eine Tabelle mit allen Eigenschaften repräsentiert.

ThermoElement Spezielles StatementElement, das ein Thermometer-Diagramm mit allen Eigenschaften repräsentiert.
fetchSaveableAttributes(ThermometerViewer thermometerViewer)
Holt alle zu speichernden Attribute aus dem Graphen.

TopologyConnectionAttributes Beschreibt die Schwellwert-Funktionen einer Kante einer Topologie

TopologyElement Spezielles PageElement, das eine Topologie mit allen Eigenschaften repräsentiert.

TransferObject Das TransferObject erlaubt es Objekte über „Drag and Drop“ zu verschieben und zu kopieren.

TreeObject Das TreeObject ist ein Objekt in dem Perspektiven-Baum des PerspectiveManagers. Es hält zudem eine Menge an PageElementen, die bei Öffnen des TreeObjects angezeigt werden sollen.
addChild(TreeObject child)
Fügt ein TreeObject als Kindknoten hinzu.
addPageElement(PageElement element)
Fügt ein PageElement dem TreeObject hinzu.

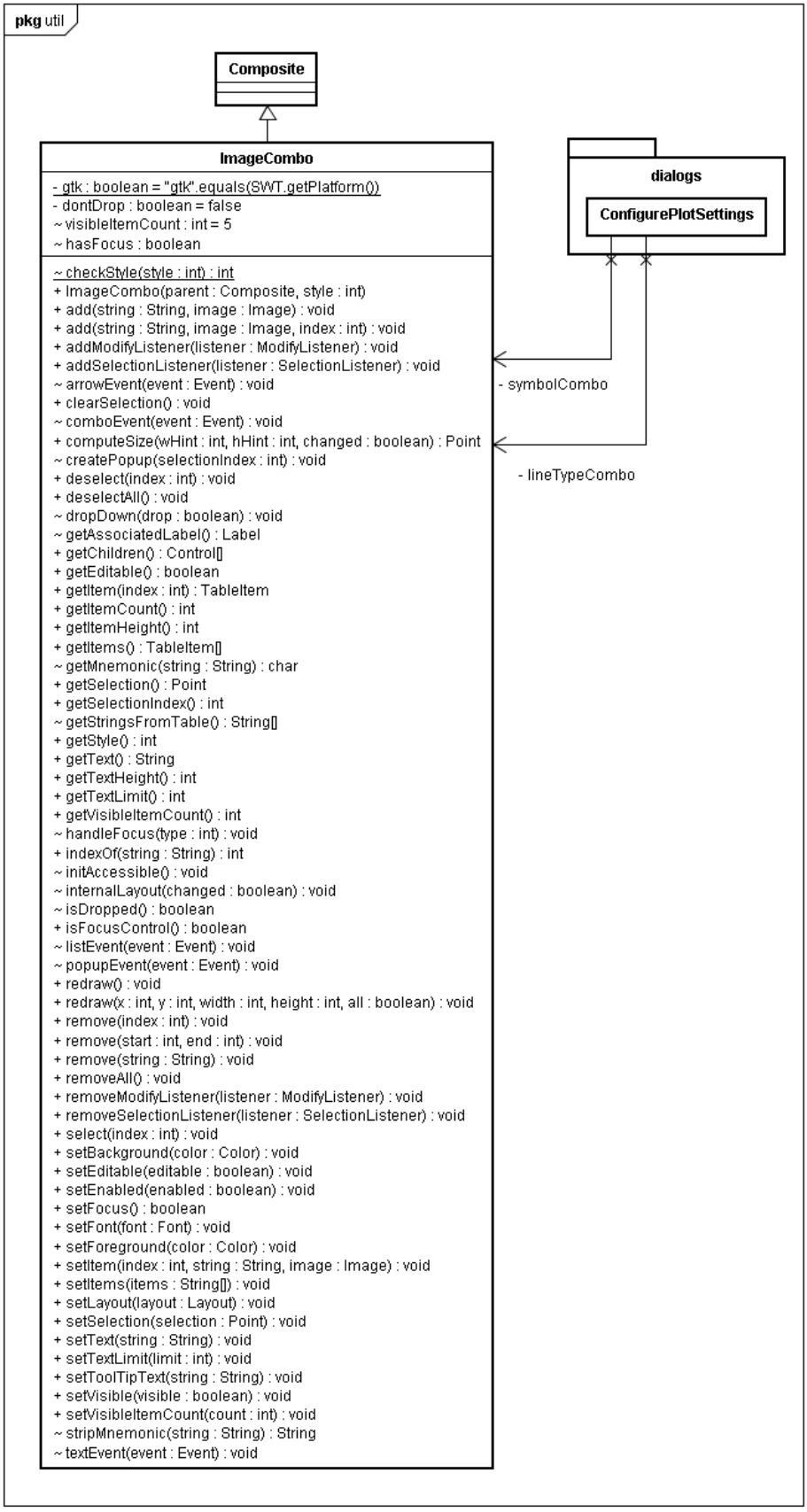
Tuple Ein Tuple repräsentiert ein Datenstromtupel.

rangePercentage(int no)

Liefert eine Prozentangabe, wo sich der Tupelwert in Bezug auf die Grenzen befindet. Wird für die 3D-Welt benötigt.

UserLevel Das UserLevel enum bietet verschiedene Berechtigungsstufen für Benutzer an, die in der Authorization Klasse verwendet werden können.

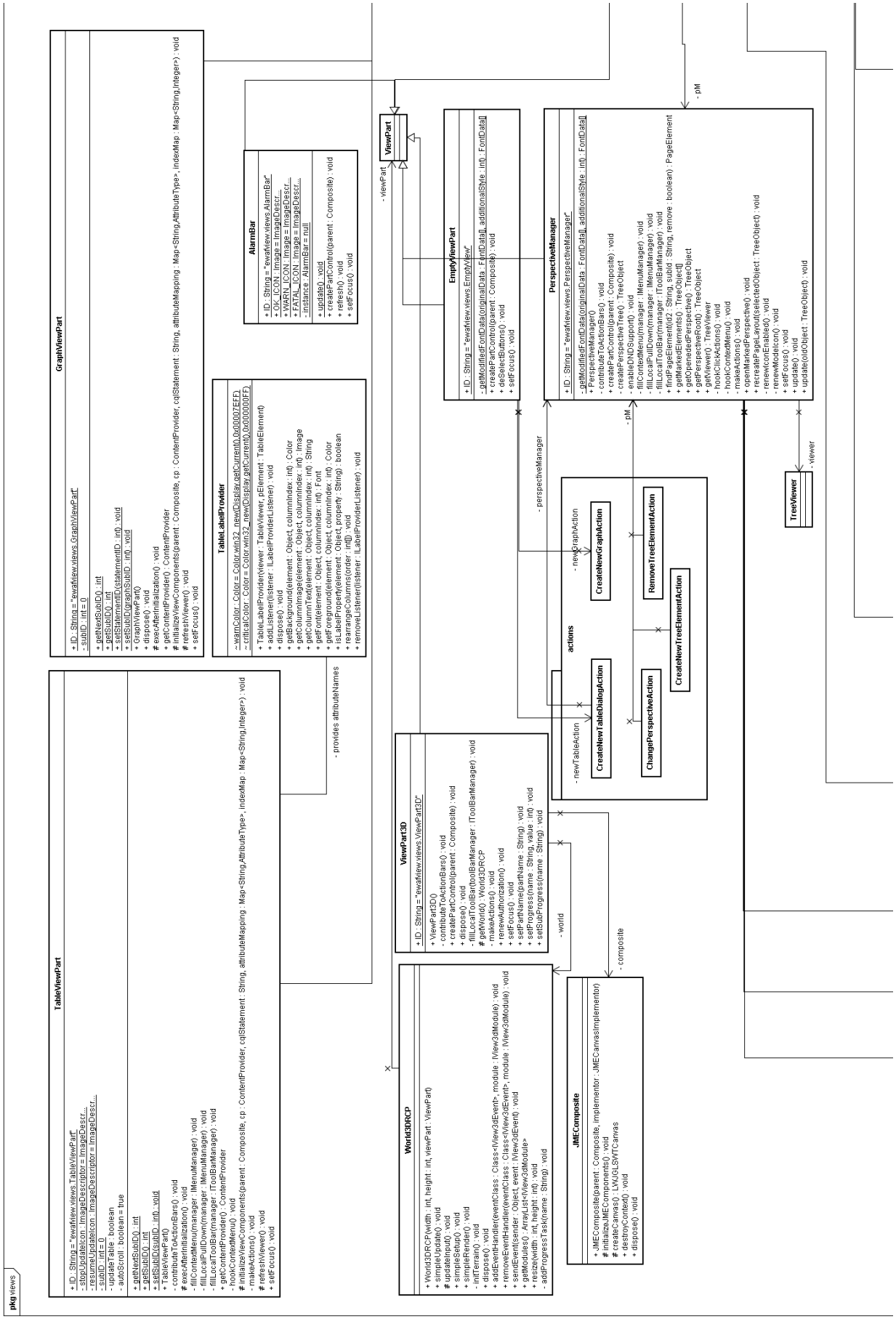
util

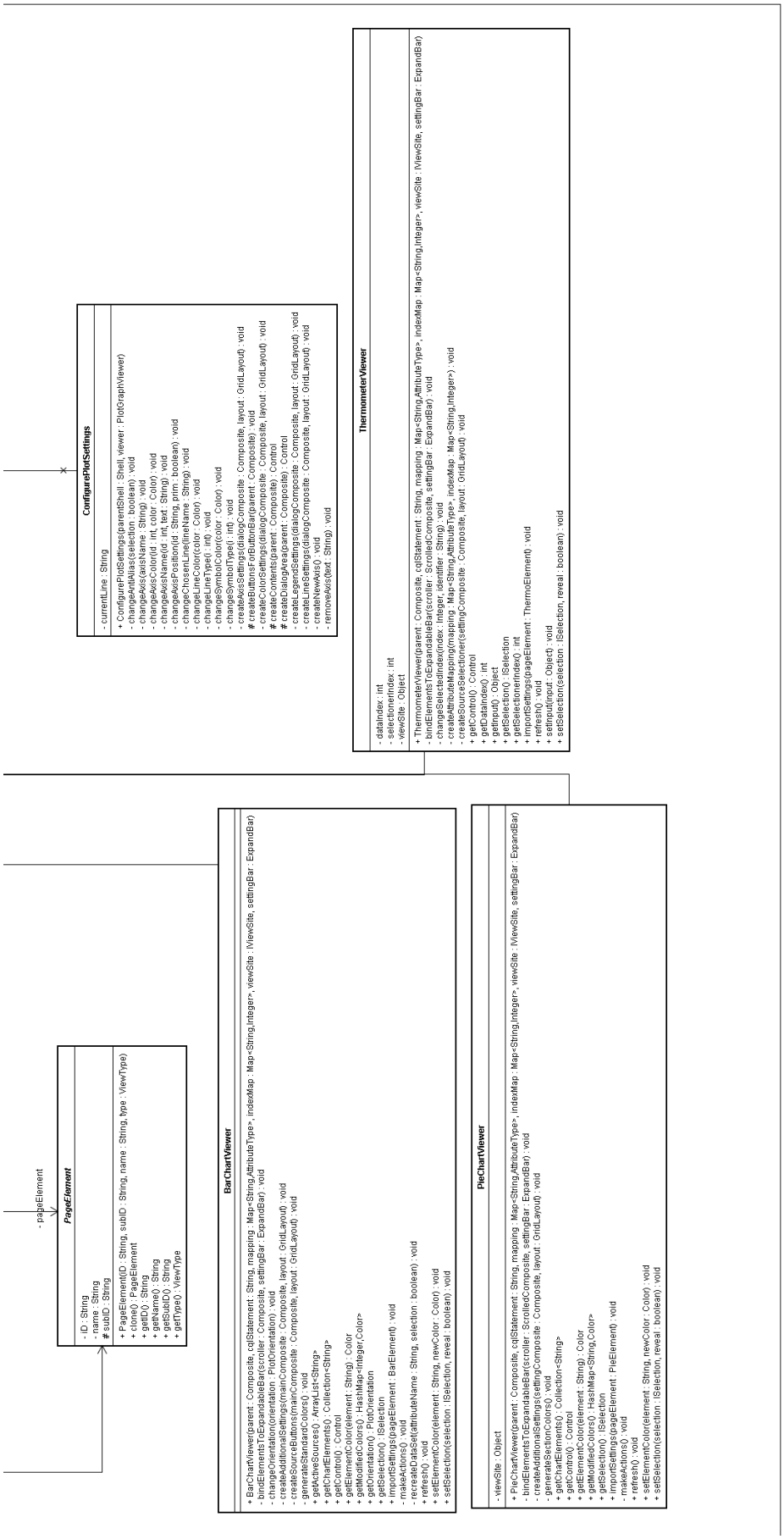

pkg util
Composite
ImageCombo
- gtk : boolean = "gtk".equals(SWT.getPlatform())
- dontDrop : boolean = false
~ visibleItemCount : int = 5
~ hasFocus : boolean
~ checkStyle(style : int) : int
+ ImageCombo(parent : Composite, style : int)
+ add(string : String, image : Image) : void
+ add(string : String, image : Image, index : int) : void
+ addModifyListener(listener : ModifyListener) : void
+ addSelectionListener(listener : SelectionListener) : void
~ arrowEvent(event : Event) : void
+ clearSelection() : void
~ comboEvent(event : Event) : void
+ computeSize(wHint : int, hHint : int, changed : boolean) : Point
~ createPopup(selectionIndex : int) : void
+ deselect(index : int) : void
+ deselectAll() : void
~ dropDown(drop : boolean) : void
~ getAssociatedLabel() : Label
+ getChildren() : Control[]
+ getEditable() : boolean
+ getItem(index : int) : TableItem
+ getItemCount() : int
+ getItemHeight() : int
+ getItems() : TableItem[]
~ getMnemonic(string : String) : char
+ getSelection() : Point
+ getSelectionIndex() : int
~ getStringsFromTable() : String[]
+ getStyle() : int
+ getText() : String
+ getTextHeight() : int
+ getTextLimit() : int
+ getVisibleItemCount() : int
~ handleFocus(type : int) : void
+ indexOf(string : String) : int
~ initAccessible() : void
~ internalLayout(changed : boolean) : void
~ isDropped() : boolean
+ isFocusControl() : boolean
~ listEvent(event : Event) : void
~ popupEvent(event : Event) : void
+ redraw() : void
+ redraw(x : int, y : int, width : int, height : int, all : boolean) : void
+ remove(index : int) : void
+ remove(start : int, end : int) : void
+ remove(string : String) : void
+ removeAll() : void
+ removeModifyListener(listener : ModifyListener) : void
+ removeSelectionListener(listener : SelectionListener) : void
+ select(index : int) : void
+ setBackground(color : Color) : void
+ setEditable(editable : boolean) : void
+ setEnabled(enabled : boolean) : void
+ setFocus() : boolean
+ setFont(font : Font) : void
+ setForeground(color : Color) : void
+ setItem(index : int, string : String, image : Image) : void
+ setItems(items : String[]) : void
+ setLayout(layout : Layout) : void
+ setSelection(selection : Point) : void
+ setText(string : String) : void
+ setTextLimit(limit : int) : void
+ setToolTipText(string : String) : void
+ setVisible(visible : boolean) : void
+ setVisibleItemCount(count : int) : void
~ stripMnemonic(string : String) : String
~ textEvent(event : Event) : void
dialogs
ConfigurePlotSettings
- symbolCombo
- lineTypeCombo

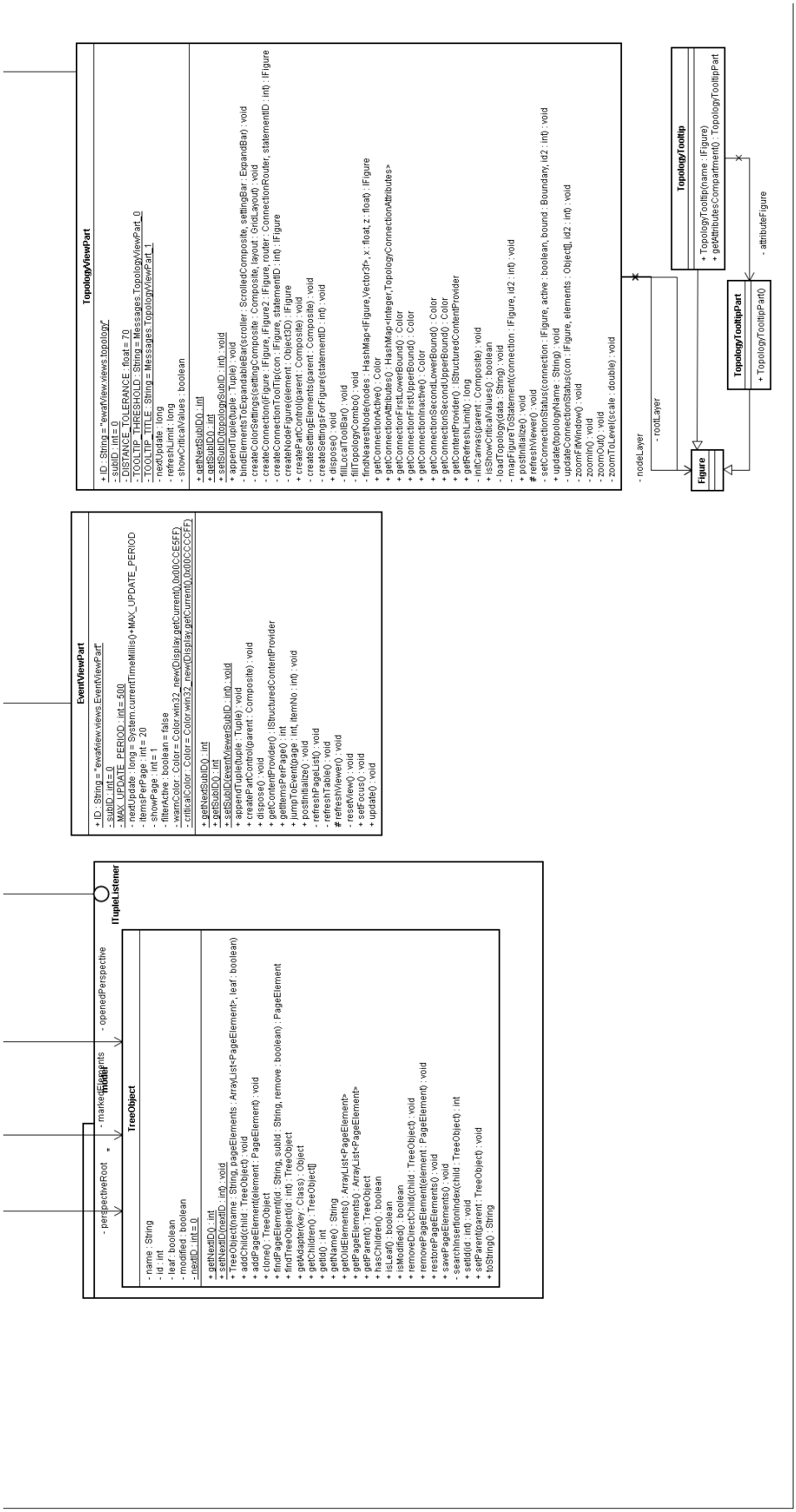
ImageCombo Erweiterte Auswahl-Combo. Erlaubt das gleichzeitige Anzeigen von einem Bild und Text in der Combo. Entwickelt von IBM, Tom Seidel.

views

AxisInformation
- title : String
- id : int
- primary : boolean
+ AxisInformation(title : String, id : int, pos : Position, color : Color)
+ getColor() : Color
+ getId() : int
+ getTitle() : String
+ isPrimary() : boolean
PlotGraphViewer
- timeRange : int = 60000
- autoAdjust : boolean
- autoScaleY : boolean
- autoScaleX : boolean
- viewSite : Object
- legendOrientation : int
- axesInformation
+ PlotGraphViewer(parent: Composite, cqlStatement : String, mapping : Map<String,AttributeType>, indexMap : Map<String,Integer>, viewSite : IViewSite, settingBar: ExpandBar)
- addPopUpMenu(plotArea : Composite) : void
- addYAxis() : int
- adjustRange() : void
- adjustTimePeriod(startTime : double) : void
- bindElementsToExpandableBar(scroller : ScrolledComposite, settingBar : ExpandBar) : void
+ changeLegendOrientation(orientation : int) : void
- changeTimeRange() : int) : void
- createAdditionalSettings(settingComposite : Composite) : void
- createSourceButtons(mainComposite : Composite, layout : GridLayout) : void
- createTimeperiod(mainComposite : Composite, layout : GridLayout) : void
- createYAxisSettings(mainComposite : Composite, layout : GridLayout) : void
- diffPopUpMenu(plotArea : IViewManager) : void
- generateDefaultLineAttributes() : void
- generateStandardColors() : void
+ getActiveSources() : ArrayList<String>
+ getAxesInformation() : AxisInformation[]
+ getControl() : Control
+ getLegendOrientation() : int
+ getLineAttributes() : HashMap<String,LineAttribute>
+ getLineColor(lineName : String) : Color
+ getLineStyle(lineName : String) : LineStyle
+ getSchemeInformation() : LinkedHashMap<String,Integer>
+ getSelection() : ISelection
+ getSymbolColor(lineName : String) : Color
+ getSymbolType(lineName : String) : PlotSymbolType
+ getTimeRange() : int
+ getYAxes() : IAxis[]
+ getYAxis(lineName : String) : IAxis
+ importSettings(pageElement : PlotGraphElement) : void
+ isAntiAlias(lineName : String) : boolean
+ isAutoAdjust() : boolean
- makeActions() : void
- recreateDataSet(lineName : String, selected : boolean) : void
+ refresh() : void
+ removeYAxis(id : String) : String
- restoreAxis(pageElement : PlotGraphElement) : void
+ setAntiAlias(lineName : String, aa : boolean) : void
+ setAutoAdjust(function : boolean) : void
+ setLineColor(lineName : String, color : Color) : void
+ setLineStyle(lineName : String, style : LineStyle) : void
+ setSelection(selection : ISelection, reveal : boolean) : void
+ setSymbolColor(lineName : String, color : Color) : void
+ setSymbolType(lineName : String, type : PlotSymbolType) : void
+ setYAxis(lineName : String, yAxisID : int) : void
+ setYAxisColor(id : int, color : Color) : void
+ setYAxisDirection(id : int, prim : boolean) : void
+ setYAxisTitle(id : int, text : String) : void
+ syncAxesInformation() : void
- viewer
Statement
- socketPort : int
- cqlStatement : String
+ Statement(socketPort : int, cqlStatement : String, attributeMapping : Map<String,AttributeType>)
+ getAttributeIndexMap() : Map<String,Integer>
+ getAttributeMapping() : Map<String,AttributeType>
+ getAttributesPerType() : Map<AttributeType,ArrayList<String>>
+ getCqlStatement() : String
+ getIndexForAttribute(attribute : String) : int
+ getSocketHandlerThread() : SocketHandler
+ getSocketPort() : int
+ setSocketHandlerThread(socketHandlerThread : SocketHandler) : void
statement
<<enum>>
FailStatus
+ <<enum.constant>> NO_STATEMENT : FailStatus
+ <<enum.constant>> ALL_STATEMENT : FailStatus
+ <<enum.constant>> OK : FailStatus
# failState
StatementViewPart
statementID : int
- nextUpdate : long
refreshLimitTime : long
+ setStatementID(id : int) : void
+ StatementViewPart()
+ createPartControl(Tuple) : void
+ setRefreshRate(rate : int) : void
+ createPartControl(parent : Composite) : void
+ createPartControl(parent : Composite, cp : ContentProvider, cqlStatement : String, attributeMapping : Map<String,AttributeType>, indexMap : Map<String,Integer>, statementID : int) : void
# executeAfterInitialization() : void
# initializeViewComponents(parent : Composite, cp : ContentProvider, cqlStatement : String, attributeMapping : Map<String,AttributeType>, indexMap : Map<String,Integer>) : void
- initializeViewComponents(cqlStatement : String, cp : ContentProvider, attrMap : Map<String,AttributeType>, indexMap : Map<String,Integer>, statementID : int) : void
+ postInitialize() : void
+ reinitializeViewPart() : boolean
+ setFocus() : void
+ update() : void
+ updateBufferText() : void
- view
ErrorCaseViewer
+ ErrorCaseViewer(parent : Composite, title : String, view : StatementViewPart, pageElement : PageElement)
+ getControl() : Control
+ getSelection() : ISelection
- makeActions() : void
+ refresh() : void
+ setDescriptionText(description : String) : void
+ setSelection(selection : ISelection, reveal : boolean) : void
# viewer
ContentViewer
ViewPart2
- deletePageElement : boolean
- clipboard : Clipboard = new Clipboard(Display.getCurrent())
type : ViewType
+ setViewType(viewType : ViewType) : void
+ ViewPart2()
- addCopyPasteToActionBars() : void
+ createPartControl(parent : Composite) : void
+ dispose() : void
+ getContentProvider() : IStructuredContentProvider
+ getPageElement() : PageElement
+ isDeletePageElement() : boolean
+ isViewPart2() : boolean
- makeCopyPasteActions() : void
- postInitialize() : void
setDeletePageElement(deletePageElement : boolean) : void
+ setFocus() : void
+ setPageElement(pageElement : PageElement) : void
+ setPartName(partName : String) : void
+ setPasteActionEnabled(enable : boolean) : void
+ update() : void


pkg views
GraphViewPart
- ID : String = "ewaf4view.views.GraphViewPart"
- subID : int = 0
+ getNextSubID() : int
+ getSubID() : int
+ setStatementID(statementID : int) : void
+ setSubIDGraphSubID : int) : void
+ GraphViewPart()
+ dispose() : void
# doExtAfterInitialization() : void
+ getContentProvider() : ContentProvider
# initializeViewComponents(parent : Composite, cp : ContentProvider, cqlStatement : String, attributeMapping : Map<String,AttributeType>, indexMap : Map<String,Integer>) : void
# refreshViewer() : void
+ setFocus() : void
TableViewPart
- ID : String = "ewaf4view.views.TableViewPart"
- stopIconDescriptor : ImageDescriptor = ImageDescr...
- startIconDescriptor : ImageDescriptor = ImageDescr...
- subID : int = 0
- updateTable : boolean
- autoScroll : boolean = true
+ getNextSubID() : int
+ getSubID() : int
+ setSubIDSubID : int) : void
+ TableViewPart()
- contributeToActionBars() : void
doExtAfterInitialization() : void
- fillLocalPullDown(manager : IMenuManager) : void
- fillLocalToolBar(manager : IToolBarManager) : void
+ getContentProvider() : ContentProvider
- hookContextMenu() : void
initializeViewComponents(parent : Composite, cp : ContentProvider, cqlStatement : String, attributeMapping : Map<String,AttributeType>, indexMap : Map<String,Integer>) : void
- makeActions() : void
refreshViewer() : void
+ setFocus() : void
TableLabelProvider
- warnColor : Color = Color.win32_newDisplay(getCurrent(),0x00000FF)
- criticalColor : Color = Color.win32_newDisplay(getCurrent(),0x000000FF)
+ TableLabelProvider(viewer : TableViewer, pElement : TableElement)
+ addListener(listener : ILabelProviderListener) : void
+ dispose() : void
+ getBackground(element : Object, columnIndex : int) : Color
+ getColumnImage(element : Object, columnIndex : int) : Image
+ getColumnText(element : Object, columnIndex : int) : String
+ getFont(element : Object, columnIndex : int) : Font
+ getForeground(element : Object, columnIndex : int) : Color
+ isLabelProperty(element : Object, property : String) : boolean
+ rearrangeColumns(order : int[]) : void
+ removeListener(listener : ILabelProviderListener) : void
AlarmBar
+ ID : String = "ewaf4view.views.AlarmBar"
- OK_ICON : Image = ImageDescr...
- WARN_ICON : Image = ImageDescr...
- FATAL_ICON : Image = ImageDescr...
- instance : AlarmBar = null
+ update() : void
+ createPartControl(parent : Composite) : void
+ refresh() : void
+ setFocus() : void
ViewPart
- viewPart
EmptyViewPart
+ ID : String = "ewaf4view.views.EmptyView"
- getModifiedFontData(originalData : FontData[], additionalStyle : int) : FontData[]
+ createPartControl(parent : Composite) : void
+ deSelectButtons() : void
+ setFocus() : void
PerspectiveManager
+ ID : String = "ewaf4view.views.PerspectiveManager"
- getModifiedFontData(originalData : FontData[], additionalStyle : int) : FontData[]
+ PerspectiveManager()
+ contributeToActionBars() : void
+ createPartControl(parent : Composite) : void
- createPerspectiveTree() : TreeObject
- enableDnDSupport() : void
- fillContextMenu(manager : IMenuManager) : void
- fillLocalPullDown(manager : IMenuManager) : void
- fillLocalToolBar(manager : IToolBarManager) : void
+ findPageElementId(id2 : String, subId : String, remove : boolean) : PageElement
+ getMarkedElements() : TreeObject[]
+ getOpenededPerspective() : TreeObject
+ getPerspectiveRoot() : TreeObject
+ getViewer() : TreeViewer
- hookClickActions() : void
- hookContextMenu() : void
- makeActions() : void
+ openMarkedPerspective() : void
+ recreatePageLayout(selectedObject : TreeObject) : void
- renewContentEnabled() : void
- renewModelIcon() : void
+ setFocus() : void
+ update() : void
+ update(oldObject : TreeObject) : void
TreeViewer
- viewer
- pM
- perspectiveManager
actions
- newTableAction
CreateNewTableDialogAction
- newGraphAction
CreateNewGraphAction
ChangePerspectiveAction
RemoveTreeElementAction
CreateNewTreeElementAction
- provides attributeNames
World3DRCP
+ World3DRCP(width : int, height : int, viewPart : ViewPart)
+ simpleUpdate() : void
# updateInput() : void
+ simpleSetup() : void
+ simpleRender() : void
- initTerrain() : void
+ dispose() : void
+ addEventHandler(eventClass : Class<IView3dEvent>, module : IView3dModule) : void
+ removeEventHandler(eventClass : Class<IView3dEvent>, module : IView3dModule) : void
+ sendEvent(sender : Object, event : IView3dEvent) : void
+ getModules() : ArrayList<IView3dModule>
+ resize(width : int, height : int) : void
- addProgressTask(name : String) : void
- world
ViewPart3D
- ID : String = "ewaf4view.views.ViewPart3D"
+ ViewPart3D()
- contributeToActionBars() : void
+ createPartControl(parent : Composite) : void
+ dispose() : void
- fillLocalToolBar(toolBarManager : IToolBarManager) : void
getWorld() : World3DRCP
- makeActions() : void
+ renewAuthorization() : void
+ setFocus() : void
+ setPartName(partName : String) : void
+ setProgress(name : String, value : int) : void
+ setSubProgress(name : String) : void
- composite
JMEComposite
+ JMEComposite(parent : Composite, implementor : JMECanvasImplementor)
# initialize,IMEComponents() : void
# createCanvas() : LWJGLSWTCanvas
+ destroyContext() : void
+ dispose() : void


- pageElement
PageElement
- ID : String
- name : String
subID : String
+ PageElement(ID : String, subID : String, name : String, type : ViewType)
+ clone() : PageElement
+ getID() : String
+ getName() : String
+ getSubID() : String
+ getType() : ViewType
BarChartViewer
+ BarChartViewer(parent : Composite, cqlStatement : String, mapping : Map<String,AttributeType>, indexMap : Map<String,Integer>, viewSite : IViewSite, settingBar : ExpandBar)
- bindElementsToExpandableBar(scroller : Composite, settingBar : ExpandBar) : void
- changeOrientation(orientation : PlotOrientation) : void
- createAdditionalSettings(mainComposite : Composite, layout : GridLayout) : void
- createSourceButtons(mainComposite : Composite, layout : GridLayout) : void
- generateStandardColors() : void
+ getAvailableSources() : Array<String*
+ getChartElements() : Collection<String*
+ getControl() : Control
+ getElementColor(element : String) : Color
+ getModifiedColors() : HashMap<Integer,Color>
+ getOrientation() : PlotOrientation
+ getSelection() : ISelection
+ importSettings(pageElement : BarElement) : void
- makeActions() : void
- recreateDataSet(attributeName : String, selection : boolean) : void
+ refresh() :void
+ setElementColor(element : String, newColor : Color) :void
+ setSelection(selection : ISelection, reveal : boolean) :void
PieChartViewer
- viewSite : Object
+ PieChartViewer(parent : Composite, cqlStatement : String, mapping : Map<String,AttributeType>, indexMap : Map<String,Integer>, viewSite : IViewSite, settingBar : ExpandBar)
- bindElementsToExpandableBar(scroller : ScrolledComposite, settingBar : ExpandBar) : void
- createAdditionalSettings(settingComposite : Composite, layout : GridLayout) :void
- generateSetIconColors() : void
+ getChartElements() : Collection<String*
+ getControl() : Control
+ getElementColor(element : String) : Color
+ getModifiedColors() : HashMap<String,Color*
+ getSelection() : ISelection
+ importSettings(pageElement : PieElement) : void
+ makeActions() :void
+ refresh() :void
+ setElementColor(element : String, newColor : Color) :void
+ setSelection(selection : ISelection, reveal : boolean) :void
ConfigurePlotSettings
- currentLine : String
+ ConfigurePlotSettings(parentShell : Shell, viewer : PlotGraphViewer)
- changeAntiAlias(selection : boolean) :void
- changeAxisName(axisName : String) :void
- changeAxisColor(id : int, color : Color) :void
- changeAxisName(id : int, text : String) :void
- changeAxisPosition(id : String, prim : boolean) :void
- changeChooseLineStyle(lineName : String) :void
- changeLineColor(color : Color) :void
- changeLineType() : int) :void
- changeSymbolColor(color : Color) :void
- changeSymbolType() : int) :void
- createAxisSettings(dialogComposite : Composite, layout : GridLayout) :void
createButtonsForButtonBar(parent : Composite) :void
- createColorSettings(dialogComposite : Composite, layout : GridLayout) :void
createContents(parent : Composite) : Control
createDialogArea(parent : Composite) : Control
- createLegendSettings(dialogComposite : Composite, layout : GridLayout) :void
- createLineSettings(dialogComposite : Composite, layout : GridLayout) :void
- createNewAxis() :void
- removeAxis(text : String) :void
ThermometerViewer
- dataIndex : int
- selectionIndex : int
- viewSite : Object
+ ThermometerViewer(parent : Composite, cqlStatement : String, mapping : Map<String,AttributeType>, indexMap : Map<String,Integer>, viewSite : IViewSite, settingBar : ExpandBar)
- bindElementsToExpandableBar(scroller : ScrolledComposite, settingBar : ExpandBar) :void
- changeSelectedIndex(index : Integer, identifier : String) :void
- createAttributeMapping(mapping : Map<String,AttributeType>, indexMap : Map<String,Integer>) :void
- createSourceSelectioner(settingComposite : Composite, layout : GridLayout) :void
+ getControl() : Control
+ getDataIndex() : int
+ getInput() : Object
+ getSelection() : ISelection
+ getSelectionIndex() : int
+ importSettings(pageElement : ThermoElement) :void
+ refresh() :void
+ setInput(input : Object) :void
+ setSelection(selection : ISelection, reveal : boolean) :void


TopologyViewPart
+ ID : String = "ewafview.views.topology"
- subID : int = 0
- MAX_UPDATE_PERIOD : int = 500
- DISTANCE_TOLERANCE : float = 70
- TOOLTIP_THRESHOLD : String = Messages.TopologyViewPart_0
- TOOLTIP_TITLE : String = Messages.TopologyViewPart_1
- nextUpdate : long
- needsUpdate : boolean
- showPage : int = 1
- itemsPerPage : int = 20
- showCriticalValues : boolean
+ getNextSubID() : int
+ getSubID() : int
+ setSubID(topologySubID : int) : void
+ appendTuple(tuple : Tuple) : void
- bindElementsToExpandableBar(scroller : ScrolledComposite, settingBar : ExpandBar) : void
- createColorSettings(settingComposite : Composite, layout : GridLayout) : void
- createConnection(fFigure : IFigure, fFigure2 : IFigure, router : ConnectionRouter, statementID : int) : IFigure
- createConnectionToolTip(con : IFigure, statementID : int) : IFigure
- createNodeFigure(element : Object3D) : IFigure
- createPartControl(parent : Composite) : void
- createSettingsElements(parent : Composite) : void
- createSettingsForFigure(statementID : int) : void
+ dispose() : void
- fillLocalToolBar() : void
- fillTopologyCombo() : void
- findNearestNode(nodes : HashMap<IFigure,Vector3f>, x : float, z : float) : IFigure
+ getConnectionActive() : Color
+ getConnectionAttributes() : HashMap<Integer,TopologyConnectionAttributes>
+ getConnectionFirstLowerBound() : Color
+ getConnectionFirstUpperBound() : Color
+ getConnectionInactive() : Color
+ getConnectionSecondLowerBound() : Color
+ getConnectionSecondUpperBound() : Color
+ getContentProvider() : IStructuredContentProvider
+ getRefreshLimit() : long
- initCanvas(parent : Composite) : void
+ isShowCriticalValues() : boolean
- loadTopologyData : String() : void
- mapFigureToStatement(connection : IFigure, id2 : int) : void
+ postInitialize() : void
# refreshViewer() : void
- setConnectionStatus(connection : IFigure, active : boolean, bound : Boundary, id2 : int) : void
+ update(topologyName : String) : void
- updateConnectionStatus(con : IFigure, elements : Object[], id2 : int) : void
- zoomFitView() : void
- zoomIn() : void
- zoomOut() : void
- zoomToLevel(scale : double) : void
- nodeLayer
- rootLayer
Figure
TopologyTooltip
+ TopologyTooltip(name : IFigure)
+ getAttributesCompartment() : TopologyTooltipPart
TopologyTooltipPart
+ TopologyTooltipPart()
- attributeFigure
EventViewPart
+ ID : String = "ewafview.views.EventViewPart"
- subID : int = 0
- MAX_UPDATE_PERIOD : int = 500
- nextUpdate : long = System.currentTimeMillis()+MAX_UPDATE_PERIOD
- itemsPerPage : int = 20
- showPage : int = 1
- filterActive : boolean = false
- warnColor : Color = Color.win32_new(Display.getCurrent(),0x00CCE5FF)
- criticalColor : Color = Color.win32_new(Display.getCurrent(),0x00CCCCFF)
+ getNextSubID() : int
+ getSubID() : int
+ setNextID(nextID : int) : void
+ SetSubID(eventViewerSubID : int) : void
+ appendTuple(tuple : Tuple) : void
+ createPartControl(parent : Composite) : void
+ dispose() : void
+ getContentProvider() : IStructuredContentProvider
+ getItemsPerPage() : int
+ jumpToEvent(page : int, itemNo : int) : void
+ postInitialize() : void
- refreshPageList() : void
- refreshTable() : void
refreshViewer() : void
- resetView() : void
+ setFocus() : void
+ update() : void
ITupleListener
- openedPerspective
- markedElements
- perspectiveRoot
*
TreeObject
- name : String
- id : int
- leaf : boolean
- modified : boolean
- nextID : int = 0
+ getNextID() : int
+ setNextID(nextID : int) : void
+ TreeObject(name : String, pageElements : ArrayList<PageElement>, leaf : boolean)
+ addChild(child : TreeObject) : void
+ addPageElement(element : PageElement) : void
+ clone() : TreeObject
+ findPageElement(id : String, subId : String, remove : boolean) : PageElement
+ findTreeObject(id : int) : TreeObject
+ getAdapter(key : Class) : Object
+ getChildren() : TreeObject[]
+ getId() : int
+ getName() : String
+ getOldElements() : ArrayList<PageElement>
+ getPageElements() : ArrayList<PageElement>
+ getParent() : TreeObject
+ hasChildren() : boolean
+ isLea() : boolean
+ isModified() : boolean
+ removeDirectChild(child : TreeObject) : void
+ removePageElement(element : PageElement) : void
+ restorePageElements() : void
+ savePageElements() : void
- searchInsertionIndex(child : TreeObject) : int
+ setId(id : int) : void
+ setParent(parent : TreeObject) : void
+ toString() : String

AlarmBar Die AlarmBar hebt kritische Ereignisse als Schaltflächen hervor. Es werden nur Ereignisse für Attribute angezeigt, die in dem CriticalValuesDialog markiert wurden. Die Schaltflächen öffnen ein Ereignislog und springen zu dem neusten Ereignis des jeweiligen Attributs.

update()

Aktualisiert die Alarmleiste.

BarChartViewer Der BarChartViewer ist ein ContentViewer, welcher in einen ViewPart integriert werden kann. Dieser stellt ein Balkendiagramm dar.

importSettings(BarElement pageElement)

Importiert die im PageElement gespeicherten Einstellungen und passt dementsprechend den Viewer an.

refresh()

Der BarChartViewer holt sich die aktuellen Elemente vom ContentProvider und aktualisiert die Anzeige.

EmptyViewPart ViewPart, dem keine weiteren ViewParts zugeordnet sind. Enthält Knöpfe zum erzeugen von Graphen, Tabellen und Topologien.

ErrorCaseViewer ContentViewer, der Fehler anzeigt. Wird angezeigt, falls die Anfragen, die einem StatementViewPart zugeordnet ist, nicht verfügbar ist.

EventViewPart Das EventViewPart ist ein Ereignislog, wo alle Ereignisse in einer Tabelle angezeigt werden können. Alte Ereignisse können seitenweise angezeigt werden oder über eine Filterfunktion gefunden werden. Ebenfalls können hier Ereignisse bestätigt werden.

update()

Aktualisiert das Ereignislog.

GraphViewPart StatementViewPart, der alle möglichen Graphtypen anzeigt. Ist immer einer Perspektive zugeordnet.

IMultiColorChart Interface, das Methoden zur Veränderung der Farben von Graphelementen bereitstellt.

PerspectiveManager ViewPart, der einen Baum aller vorhanden Perspektiven anzeigt. Elemente des Baums sind TreeObjects und werden per Doppelklick geöffnet. Enthält zudem eine Button-Toolbar und ein Kontextmenü zur Verwaltung der Perspektiven. Kann nicht geschlossen werden.

recreatePageLayout(TreeObject selectedObject)

Schließt die aktuell geöffnete Perspektive mit all ihren Viewparts und öffnet die übergebene (selectedObject).

update()

Aktualisiert den Perspektiven-Baum.

update(TreeObject oldObject)

Aktualisiert den Perspektiven-Baum und expandiert den Pfad zur gegebenen Perspektive.

PieChartViewer Der PieChartViewer ist ein ContentViewer, welcher in einen ViewPart integriert werden kann. Dieser stellt ein Kuchendiagramm dar.

importSettings(PieElement pageElement)

Importiert die im PageElement gespeicherten Einstellungen und passt dementsprechend den Viewer an.

refresh()

Der PieCHartViewer holt sich die aktuellen Elemente vom ContentProvider und aktualisiert die Anzeige.

PlotGraphViewer Der PlotGraphViewer ist ein ContentViewer, welcher in einen ViewPart integriert werden kann. Dieser stellt ein X,Y - Diagramm dar.

importSettings(PlotElement pageElement)

Importiert die im PageElement gespeicherten Einstellungen und passt dementsprechend den Viewer an.

refresh()

Der PlotGraphViewer holt sich die aktuellen Elemente vom ContentProvider und aktualisiert die Anzeige.

StatementViewPart Abstrakte Klasse. Basis für Tabellen und Graphen. Bietet Methoden zur Aktualisierung und verbindet den ViewPart mit einem PageElement.

postInitialize()

Initialisiert View-Komponenten, die während des createPartControl(Composite parent) Prozesses nicht initialisiert werden konnten.

reinitializeViewPart()

Versucht den ViewPart mit einem alternativen Statement zu initialisieren

update()

Aktualisiert den assoziierten ContentViewer.

TableLabelProvider Definiert für Tabellen die Formatierung der Zelleninhalte.

TableViewPart ViewPart für die Darstellung einer Tabelle. In den Spalten werden die Attribute und in den Zeilen die Tupelwerte abgetragen.

refreshViewer()

Aktualisiert die Tabelle und scrollt zu dem neusten Element, falls es nicht sichtbar ist.

ThermometerViewer Der ThermometerViewer ist ein ContentViewer, welcher in einen ViewPart integriert werden kann. Dieser stellt ein Thermometer dar.

importSettings(ThermoElement pageElement)

Importiert die im PageElement gespeicherten Einstellungen und passt dementsprechend den Viewer an.

refresh()

Der ThermometerViewer holt sich die aktuellen Elemente vom ContentProvider und aktualisiert die Anzeige.

TopologyTooltip Tooltip figure, welche einen Titel als IFigure und eine unbegrenzte Anzahl an IFigures darunter anzeigt.

TopologyTooltipPart Teil eines TopologyTooltip. Hält zusätzliche Informationen über den Knoten eines TopologyViewPart.

TopologyViewPart Zeigt eine zwei-dimensionale geographische Darstellung des Stromnetzes an. Das Stromnetz wird in der 3D-Welt definiert und gespeichert.

postInitialize()

Initialisiert View-Komponenten, die während des createPartControl(Composite parent) Prozesses nicht initialisiert werden konnten.

update()

Aktualisiert die Farben der Kanten, anhand von kritischen Werten bzw. Schwellwert-Funktionen.

ViewPart2 Abstrakte Klasse, die alle ViewParts zusammenfasst, die einer Perspektive zugeordnet werden können.

public void dispose()

Deregistriert den ViewPart vom Controller, sodass dieser nicht mehr versucht ihn zu aktualisieren. Wird beim Schließen des ViewParts automatisch aufgerufen.

postInitialize()

Initialisiert View-Komponenten, die während des createPartControl(Composite parent) Prozesses nicht initialisiert werden konnten.

update()

Aktualisiert den zugeordneten ContentViewer.

ViewPart3D ViewPart3D fasst 3D-Elemente zusammen und stellt diese in SWT dar.

makeActions()

Mit Hilfe dieser Methode werden alle Actions erzeugt und den Buttons zugeordnet.

CreatePartControl

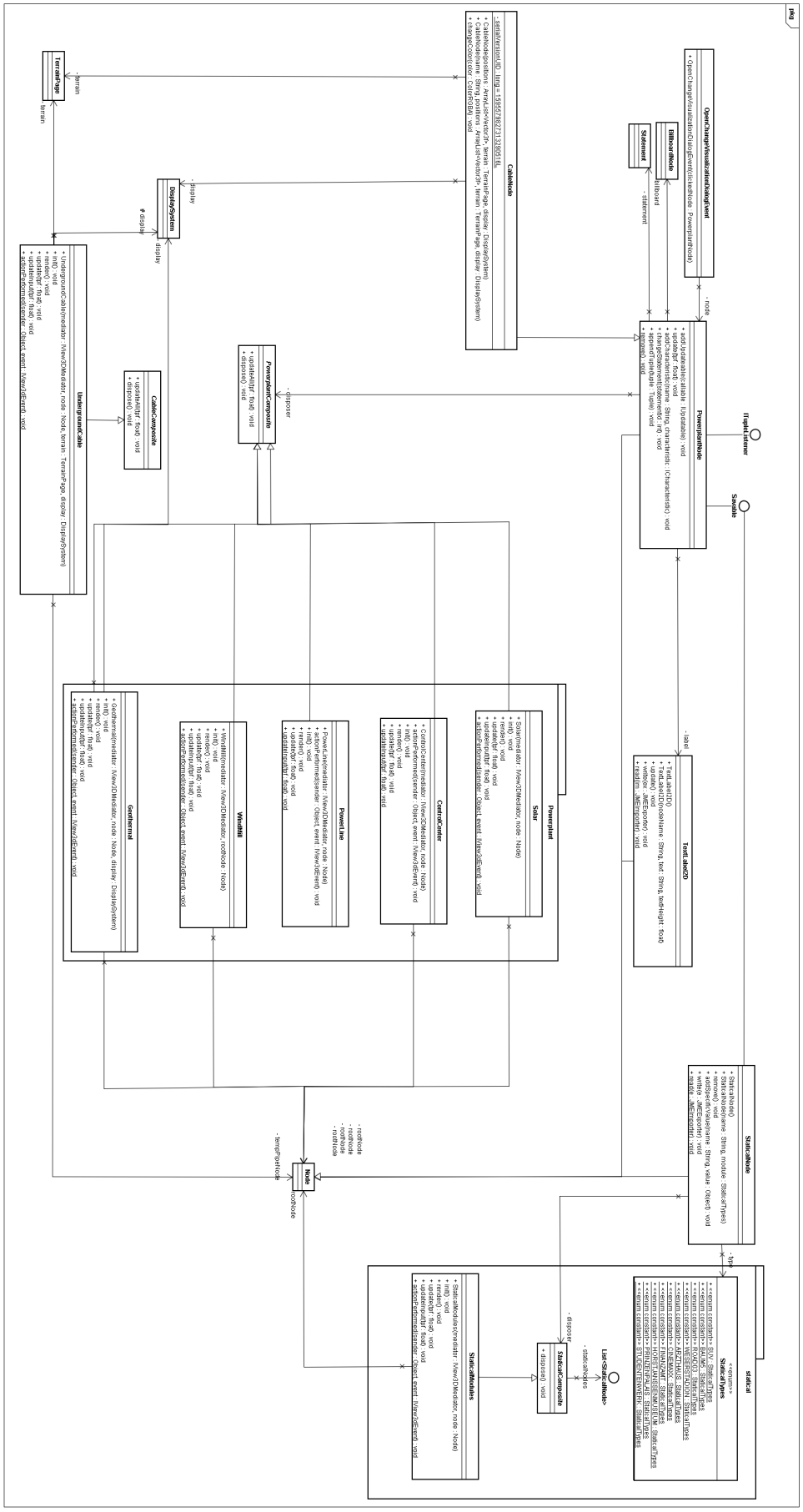
Innerhalb dieser Methode wird die 3D-Welt erstellt und direkt in Eclipse geöffnet.

fillLocalToolBar()

Legt die Buttons in die Toolbar.

4.3.6 3D-View

Scene



PowerplantNode Diese Klasse stellt Knoten, die Tupel anzeigen und diese an Merkmale binden können, dar. Alle Kraftwerke sind ein PowerplantNode.

changeStatement()
ändert das zugeordnete Statement.

appendTupel()
Überliefert alle neuen Tupel an das zu modifizierende Modell

PowerplantComposite Dieses Composite enthält und verwaltet alle PowerplantNodes. Es werden speziell alle Update- und Disposeanweisungen.

TextLabel2d Wandelt ankommende Tupel in ein Objekt um, welches in der 3D-Welt angezeigt werden kann.

StaticalNode Wird dazu verwendet um statische Modelle in die 3D-Welt einzubinden.

StaticalComposite Stellt den Kontainer für alle statischen Modelle dar.

StaticalTypes Bietet eine Reihe von Enums an um statische Modelle zu erzeugen.

StaticalModels Erzeugt statische Modelle für die 3D-Welt

actionPerformed
Methode zum Erstellen der Modelle.

dispose()
Entfernt alle Knoten vom Elternknoten und entfernt die Registrierungen von allen Listeners

OpenChangeVisualizationEvent Das Event wird erzeugt sobald man in der 3D-Welt die Statements ändern will, die an ein Kraftwerk gebunden sind.

Powerplant

Solar Stellt ein Solarkraftwerk in der 3D-Welt dar. Das Erstellen von mehreren Solarkraftwerken funktioniert mit Hilfe von einem Cloner

actionPerformed
Methode zum erstellen des Kraftwerks

ControlCenter Repräsetiert ein Leitstand an dem die Energieleitungen einen Knotenpunkt finden

actionPerformed
Methode zum erstellen des Kraftwerks

PowerLine Stellt die Stromleitungen dar. Diese lassen sich in der 3D-Welt frei bewegen und beliebig anordnen.

actionPerformed
Methode zum erstellen des Kraftwerks

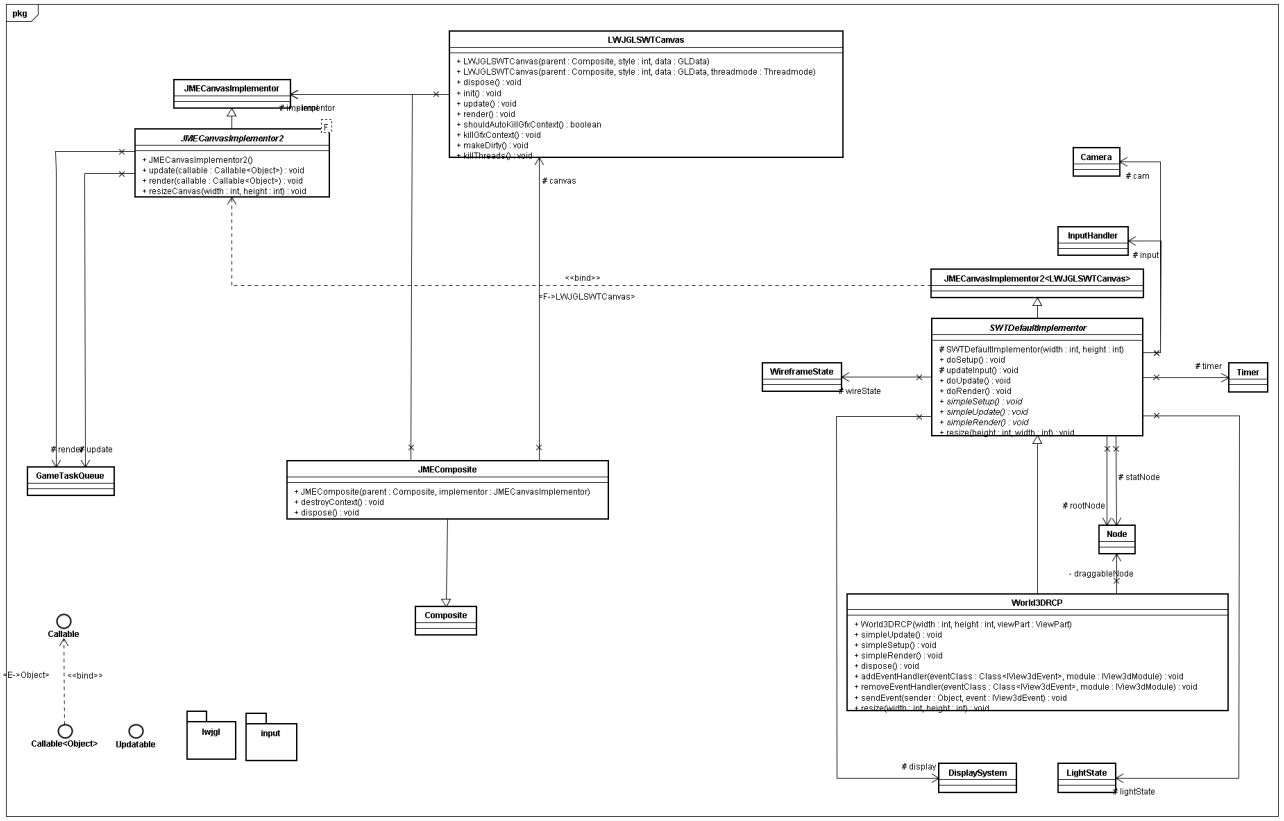
WindMill Stellt eine Windmühle in der 3D-Welt dar.

actionPerformed
Methode zum erstellen des Kraftwerks.

Geothermal Stellt ein Geothermalkraftwerk dar. Dieses lässt sich in der 3D-Welt frei bewegen und beliebig anordnen.

actionPerformed
Methode zum erstellen des Kraftwerks.

swt



World3dRCP Liefert den Haupteinstiegspunkt in die 3D-View.

runWorld()

Startet die 3D-View, die in einem extra Fenster bereitgestellt wird.

addEventHandler

Fügt ein Modul (zB. Kraftwerk) zu dem Mediator hinzu. Dieses Modul wird dann mit Hilfe der sendEvent()-Methode informiert, dass ein bestimmtes Event aufgetreten ist.

sendEvent()

Informiert alle registrierten Module, das ein bestimmtes Event aufgetreten ist. Der Mediator ruft die Funktion actionPerformed() bei allen registrierten Modulen auf.

SWTDefaultImplementor Ein auf SWT basierender Implementor, welcher der grundlegende Gamehandler von JME ist. Abstrakte Klasse die viele Methoden bereitstellt. Grundlegende Methoden wie, simpleUpdate(), simpleRender().

JMECanvasImplementor2 Generischer Implementor, der das Canvas kennt an dem es gebunden ist.

update()

Reiht ein Callable in die Warteschlange für Updates ein.

render()

Reiht ein Callable in die Warteschlange zum Rendern ein.

LWJGLSWTCanvas Implementiert einen OpenGL SWT-Canvas als einen JMECanvas.

update()

Wird durch den Update-Thread aufgerufen.

render()

Wird durch den Render-Thread aufgerufen.

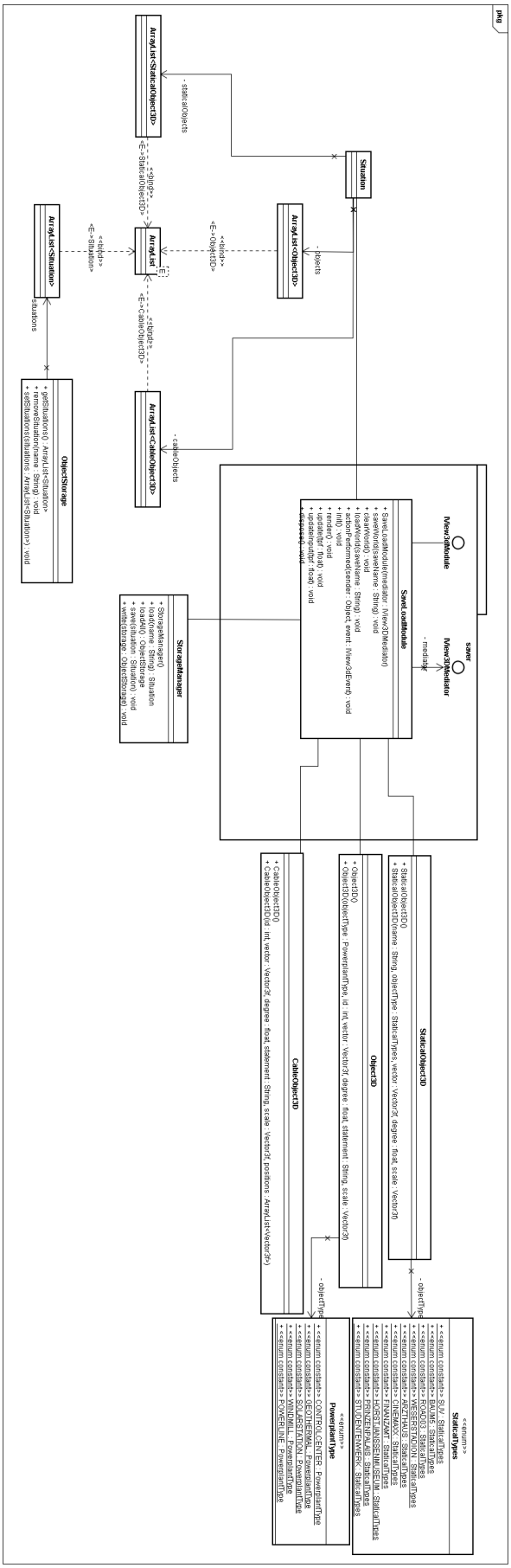
killThreads()

Beendet Threads.

JMEComposite Basic Composite welches ein Canvas und einen Implementor enthält.
destroyContext()
Entfernt den Context.

LWJGLSWTCanvasConstructor Implementiert einen CanvasConstruktor für den LWJGLSWTCanvas.
makeCanvas()
Erstellt das Canvas

Storage


pkg
Situation
ArrayList<StaticalObject3D>
ArrayList<Object3D>
ArrayList<Situation>
ArrayList<CableObject3D>
ArrayList
ObjectStorage
SaveLoadModule
StorageManager
StaticalObject3D
Object3D
CableObject3D
StaticalTypes
PowerplantType

Situation Das Situationobjekt beinhaltet alle dynamischen, statischen und Kabelobjekte.

ObjectStorage Speicher für alle 3D-Objekte.

StorageManager Der StorageManager speichert die Situations als XML mit Hilfe eines Webservices. Aus debug-Gründen ist es möglich die Daten auf der Festplatte zu speichern, ohne den Webservice dazu zu nutzen.
load(String name)
Läd die gespeicherte Situation zum passenden Namen.
save(Situation situation) Speichert die übergebene Situation.

SaveLoadModule Stellt das Modul zum Laden und Speichern der Welt dar. Das SaveLoadModule ist als Listener für das Load- und SaveEvent eingetragen.
saveWorld()
Speichert die Welt.
loadWorld()
Laden der Welt.

IView3dModule Ein Modell innerhalb der 3D-Welt. Ein Modell wird durch einen IView3dMediator verwaltet.

IView3dMediator Verwaltet die Modelle in der 3D-Welt.

StaticalObject3D Hält die Werte für Position, Rotation und Scale eines statischen Modells vor.

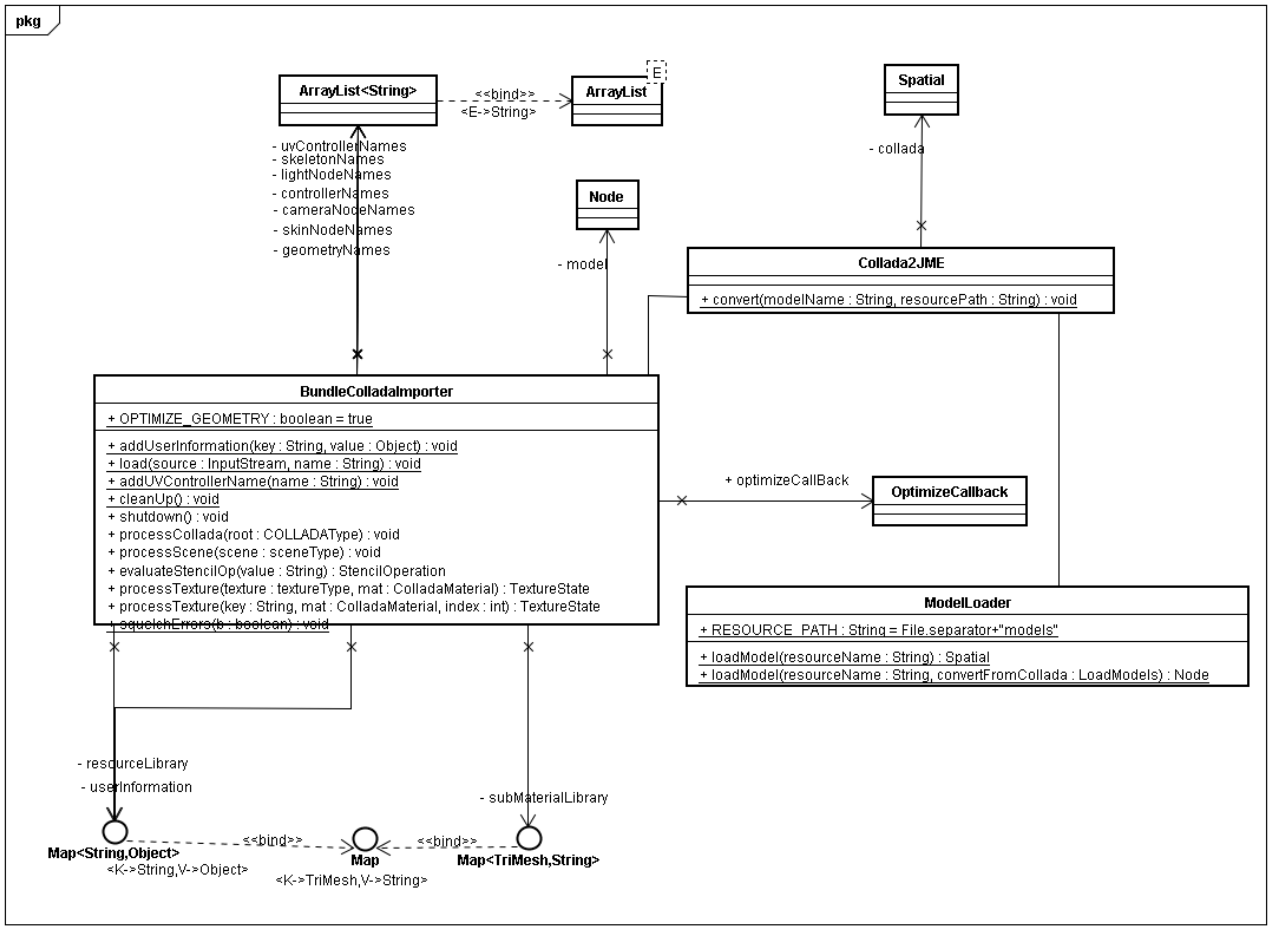
Object3D Hält die Werte für Position, Rotation und Scale eines Modells vor.

CabelObject3D Hält die Werte für Position, Rotation und Scale eines Kabelobjekts vor.

StaticalTypes Enums für statische Modelle.

PowerplantTypes Enums für Kraftwerkmodelle.

Modelloader



BundleColladaImporter Ein Collada-Importer der mit RCP-Bundles arbeitet.

processCollada(ColladaType root)

Nutzt einen ColladaType der alle Informationen aus der Collada-XML enthält. Dieses root-Element wird genutzt um Modelle in der 3D-Welt zu rendern.

processScene(Scene scene)

Schließt den Modellknoten ab um ihn als Collada zu returnen.

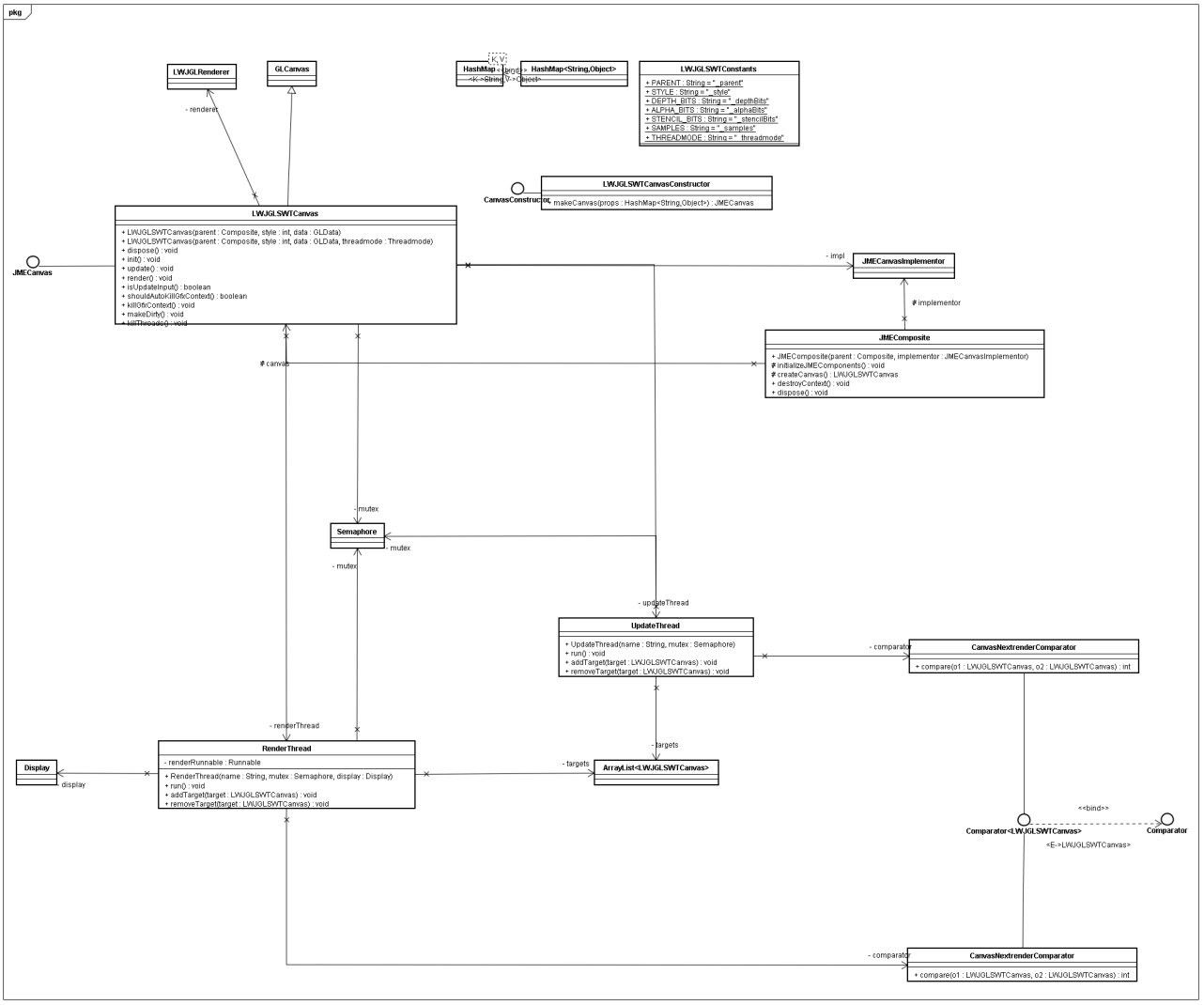
processTexture(TextureType texture)

Generiert einen TextureState der die Bilder und die Texturkoordinaten beinhaltet.

Modelloader Läd sämtliche Modelle von der Festplatte.

Collada2JME Konvertiert Colladedatein zu JMEdatein.

lwjgl

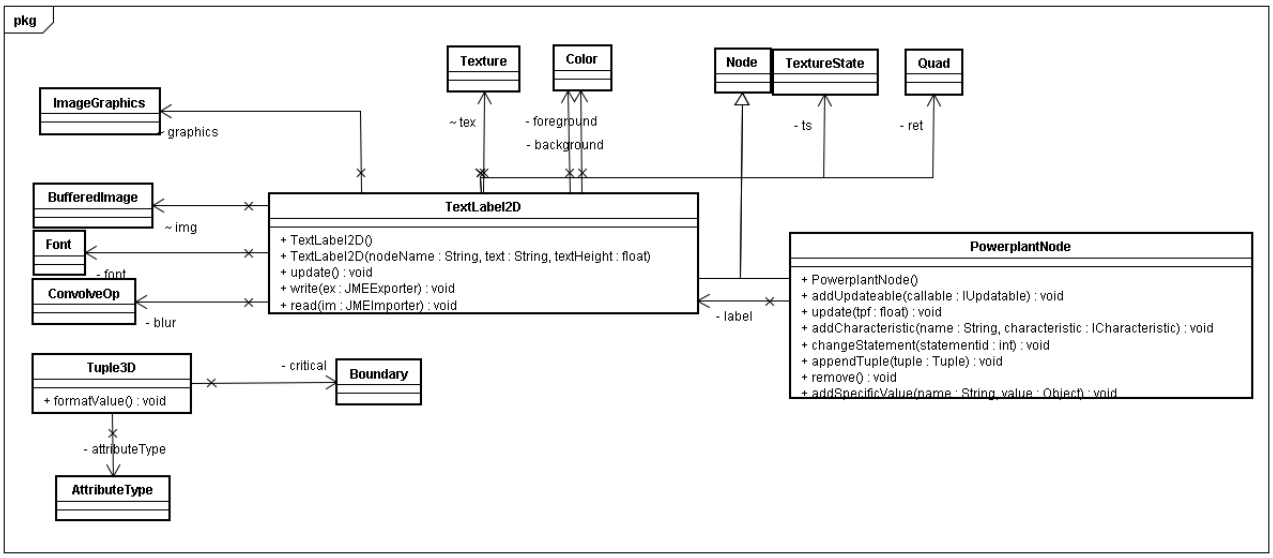


LWJGLSWTConstants Stellt Konstanten für das Canvas bereit.

RenderThread Ein extra Thread um GLCanvas in SWT zu rendern.

UpdateThread Ein extra Thread um GLCanvas in SWT zu updaten.

label

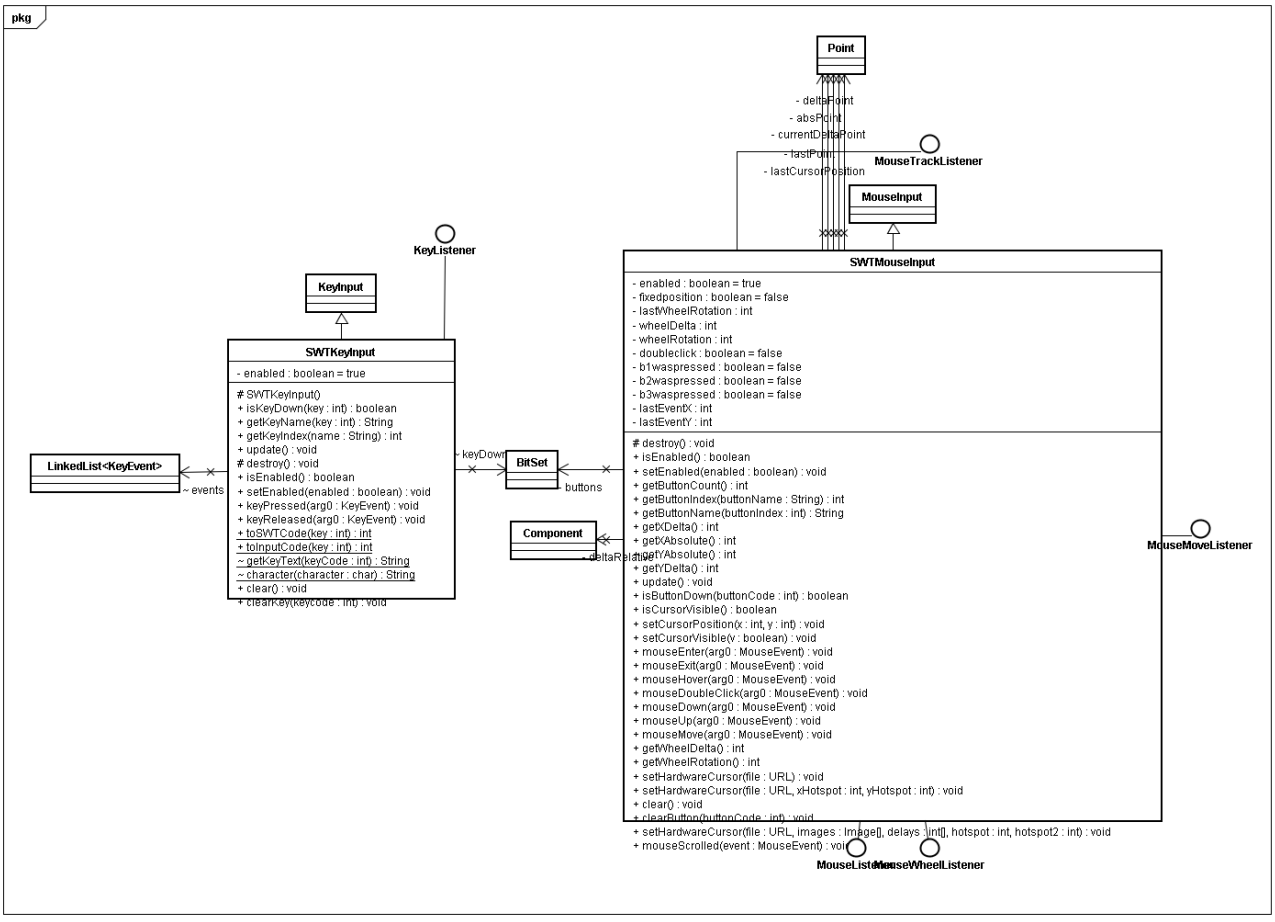


Tupel3D Repräsentiert ein einzelnes Tupel aus Odysseus.

formatValue()

Ändert die Größe in eine lesbare Größe.

Input

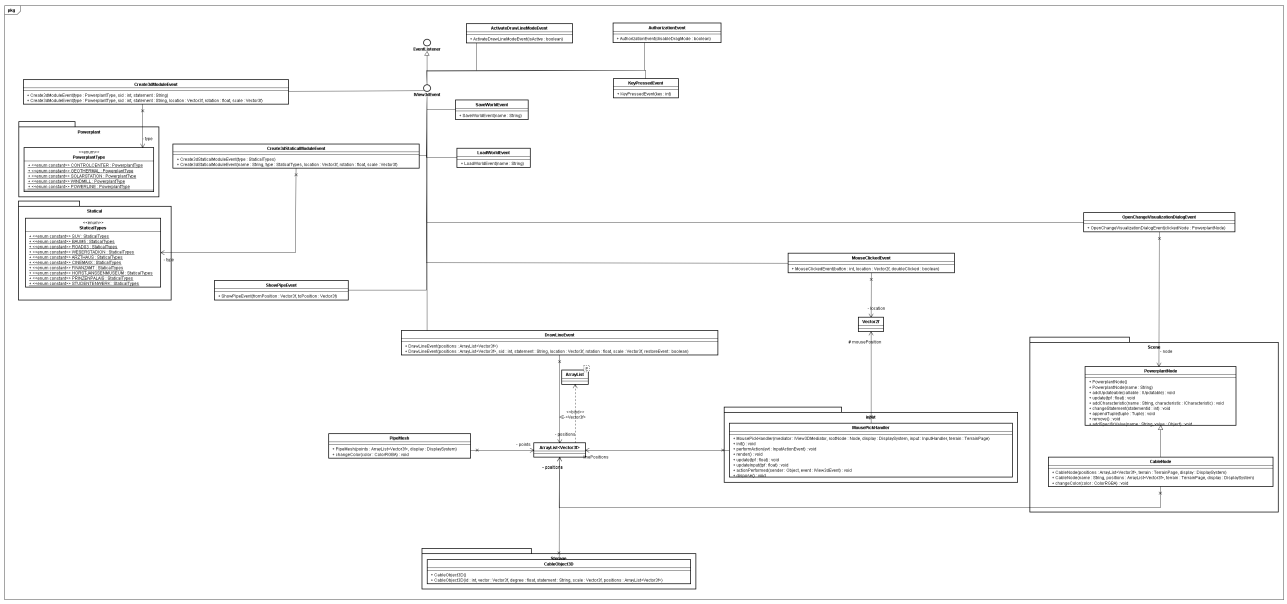


SWTKeyInput Verbindet die JMEKeyInputs mit den SWTKeylistener.

SWTMouseInput Verbindet die JMEKeyInputs mit den SWTKeyInputs.

SWTMouseInputListener Definiert einen JMEMouseInputListener mit einem zusätzlichen Doppelklick.

Events



Create3dModuleEvent Event, dass das erschaffen von Modellen ermöglicht.

Create3dStaticalModuleEvent Event, dass das erschaffen von statischen Modellen ermöglicht.

ShowPipeEvent Event, welches das Anzeigen der Pipes ermöglicht.

ActivateDrawLineModeEvent Event zum Aktivieren des Zeichnenmodus.

AuthorizationEvent Eine Authorisierung zum Verlassen des Kioskmodus wird dadurch gestartet.

KeyPressedEvent Event das geschickt wird wenn eine Taste gedrückt wird.

SaveWorldEvent Event zum Speichern der Welt.

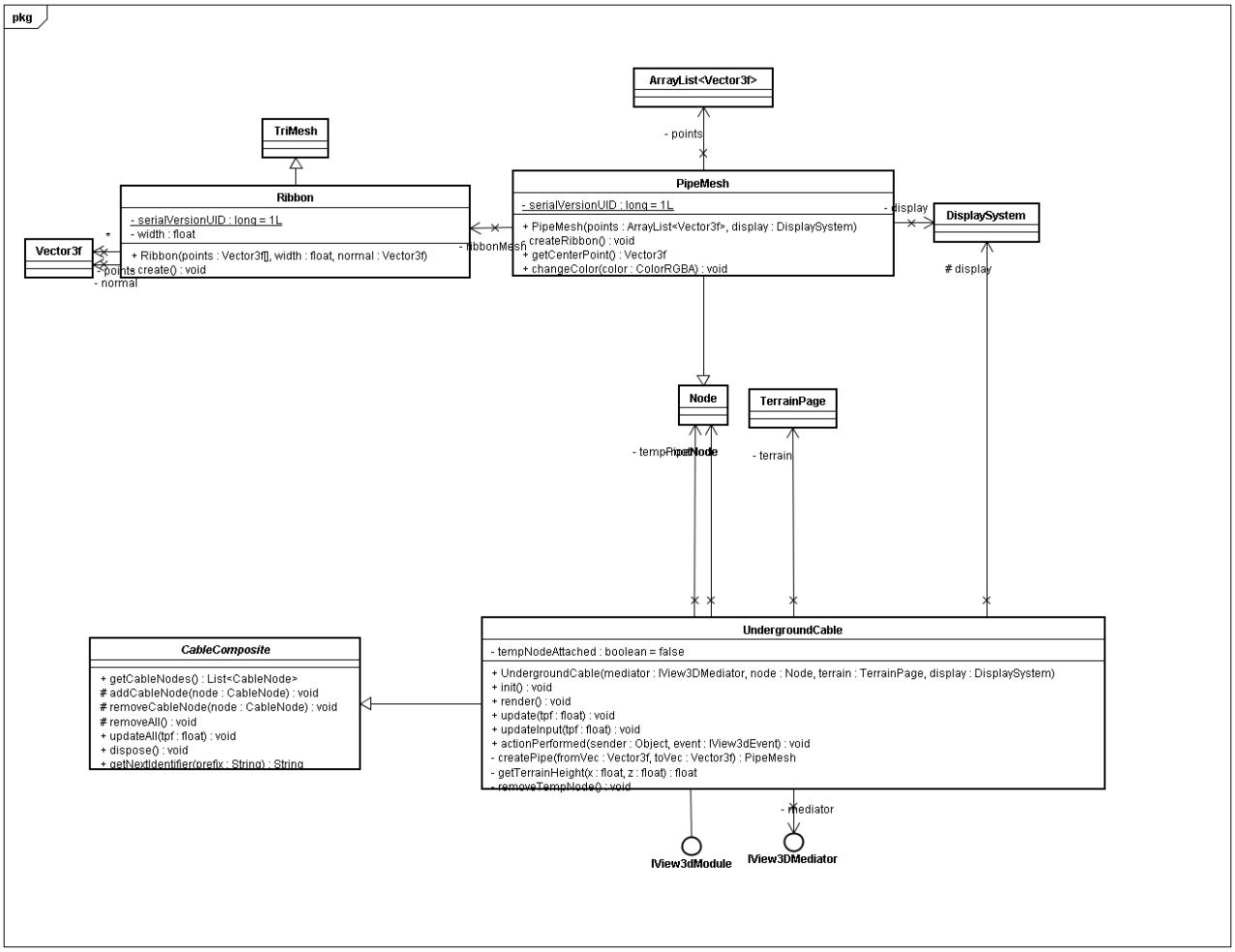
LoadWorldEvent Event zum Laden der Welt.

DrawLineEvent Event, dass das Zeichnen von Linien ermöglicht.

MouseClickedEvent Event, dass das Klicken der Maus symbolisiert.

MousePickHandler Wird genutzt um herauszufinden welches Modell angeklickt wurde. Enthält Funktionen die drag 'n' drop rotating oder removing.

Cable



PipeMesh Repräsentiert eine Pipe und animiert diese.

Ribbon Unterstützt bei der Gestalltung der Pipes

UndergroundCabel Hält verschiedene UndergroundCabelTeile vor und verwaltet diese.
actionPerformed()
Nimmt die Events für diese Klasse an und führt eine eventspezifische Aktion aus.

CableComposite Stellt den Kontainer für alle Kabelelemente dar und führt die Update/Disposeanweisungen aus.

CableNode Bindet Kabelelemente in die 3D-Welt ein.
changeColor(ColorRGBA color) ändert die Farbe in die, die übergeben wird.

4.3.7 Management von persistenten Daten

Dem Benutzer soll ermöglicht werden, einzelne oder alle Streams aufzuzeichnen, um sie später wieder abspielen zu können. Dazu ist eine persistente Datenhaltung nötig.

Um Datenstrom-Daten persistent zu speichern, setzt Odysseus MySQL ein. Falls in dieser MySQL-Instanz noch keine Datenbank angelegt ist, so wird diese von Odysseus angelegt. Der Name dieser Datenbank wird im ArchiveManager festgelegt. In der View kann für angelegten Stream einzeln festgelegt werden, ob dieser aufgezeichnet werden soll. Wenn ein Stream aufgezeichnet werden soll, dann wird eine Tabelle mit den Namen des Streams angelegt. In

dieser Tabelle werden die einzelnen Attribute des Tupels gespeichert und außerdem wird gespeichert, wann dieser Tupel aufgezeichnet wurde. Die Tabelle hat dann dabei folgende Struktur:

timestamp	**Attribut 1**	**Attribut 2**	...	**Attribut n**
1251753910482	Datum 1	Datum 2	...	Datum n
1251753910615	Datum 1	Datum 2	...	Datum n
1251753910894	Datum 1	Datum 2	...	Datum n

Dabei wird allerdings nicht gespeichert, welches SunSpot-Profil die jeweilige Tabelle benutzt. Damit Odysseus trotzdem bei dem Replaying das korrekte SunSpot-Profil auswählt, wird diese Information extra in der Tabelle „sunSpotProfiles“ ab. Diese Tabelle hat dabei folgende Struktur:

name	**sunSpotProfile**
wind1:test	WindPowerPlant
solar1:test	SolarPowerPlant
geo1:test	GeoThermalPlant

Auf diese Art und Weise kann Odysseus herausfinden, welche Tabelle in der Datenbank welches SunSpot-Profil benutzt.

4.3.8 Globaler Kontrollfluss

Der Kontrollfluss stellt auf einer hohen Abstraktionsebene da, wie sich das Programm gegenüber dem Benutzer verhält.

Sofern keine Perspektive vorhanden ist, muss erst eine erstellt werden. In dieser Perspektive kann der Benutzer dann Tabellen oder verschiedene Graphen erstellen. Hierzu müssen Streams und Statements installiert werden.

Zu jedem Wert eines Tupels können kritische Werte eingestellt werden. Werden diese Werte überschritten, erscheint ein Ereignis in der Alarm Bar. Von dort aus können die Warnungen im Event Viewer angezeigt und Quittiert werden.

Die Topologie gibt die Positionsdaten von Kraftwerken aus der 3D Welt wieder, hierzu muss erst die gewünschte Welt ausgewählt werden.

In der 3D-Welt lassen sich Kraftwerke erstellen, denen ein Statement zugewiesen werden muss, außerdem können die Attribute eingestellt werden, die über dem Kraftwerk angezeigt werden sollen. Außer Kraftwerken lassen sich noch statische Gebäude, Überlandleitungen und Untergrundkabel bauen. Die 3D-Welt kann gespeichert und später wieder geladen werden.

Zur Veranschaulichung dienen die Grafiken 4.2 und 4.3

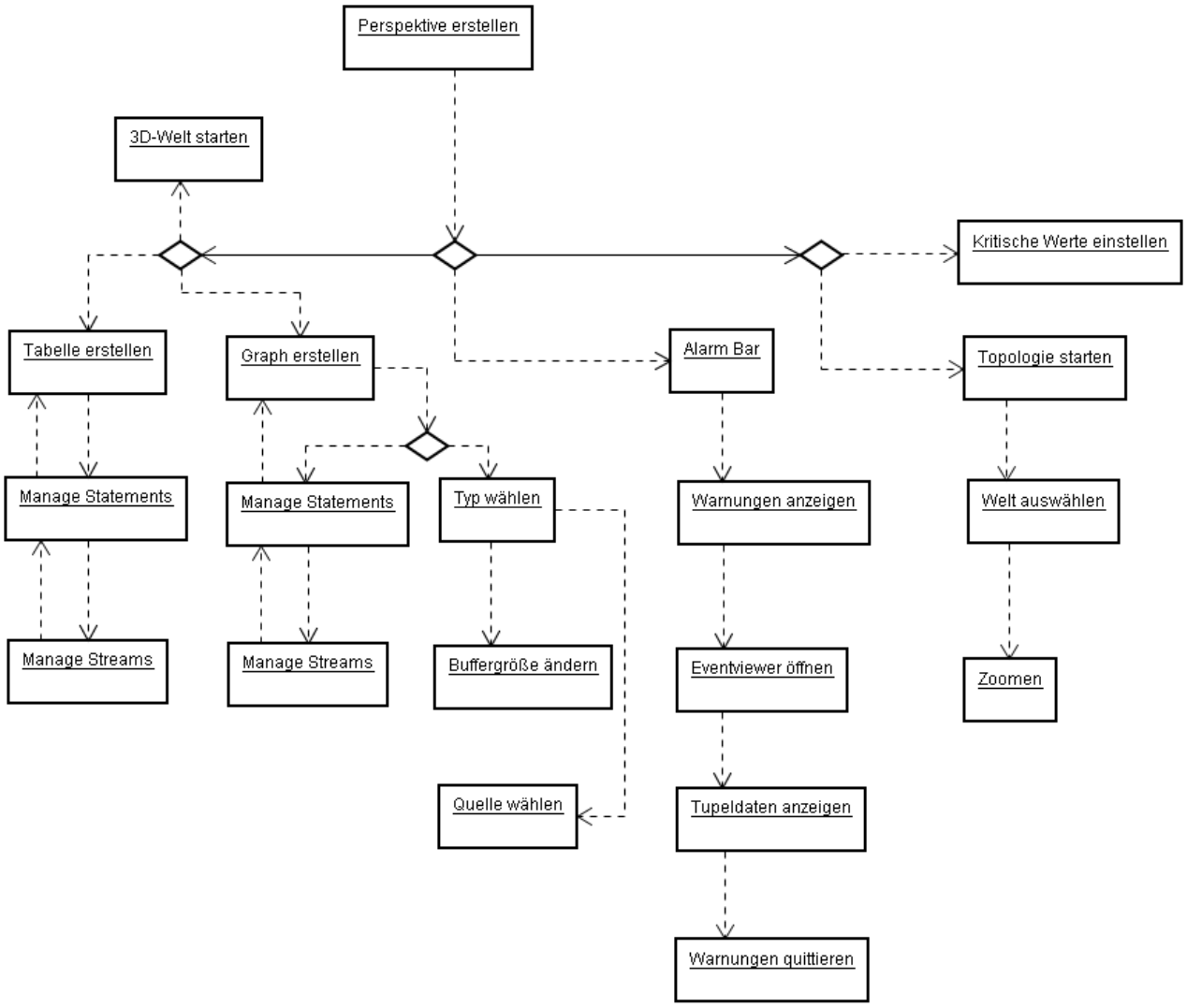


Abbildung 4.2: Kontrollflussdiagramm View

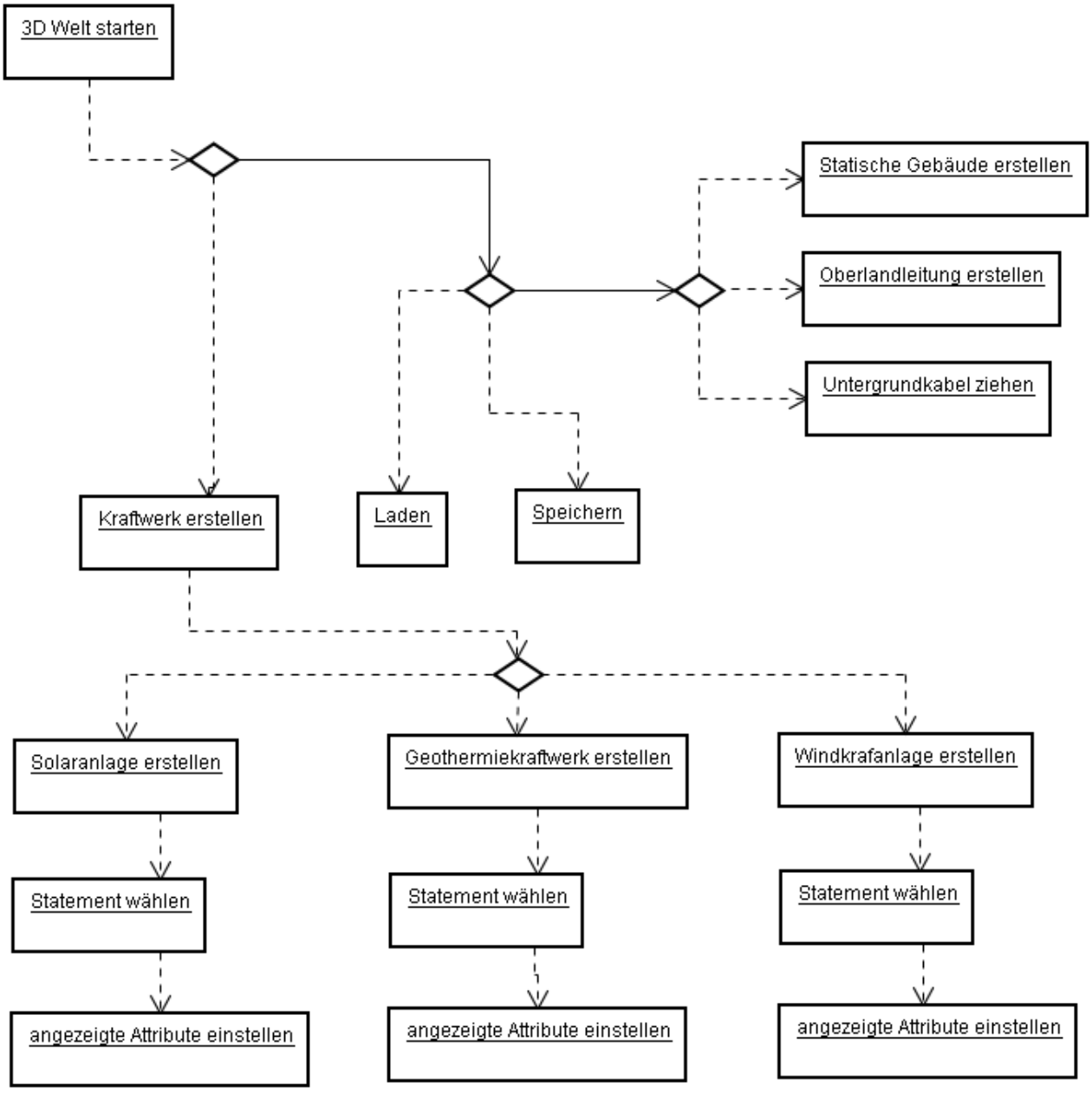


Abbildung 4.3: Kontrollflussdiagramm 3D View

Kapitel 5

Umsetzung

5.1 Kraftwerkssimulation

5.1.1 Verwendete Technologien

Zur Umsetzung der Kraftwerkssimulation werden SunSPOT Sensoren verwendet, die in Kapitel 2.2.1 näher vorgestellt wurden. Auf diesen steht die Squawk Java Virtual Machine [SCC+06] zur Verfügung, welche mit der Java Micro Edition [SM09] programmiert werden kann. Diese ist im Vergleich zur Java Standard Edition in dem Umfang der Klassenbibliothek sehr eingeschränkt und enthält auch nicht alle programmiersprachlichen Ausdrücke. Aufgrund dieser Einschränkung ist die Erweiterung mit üblichen frei verfügbaren Klassen nur eingeschränkt möglich.

Die Kommunikation zwischen den Sensoren erfolgt nach IEEE 802.15.4 [Soc06] für Personal Area Networks. Für die virtuellen Sensoren kommt die Java Standard Edition in der Version 6 zum Einsatz.

5.1.2 Architektur

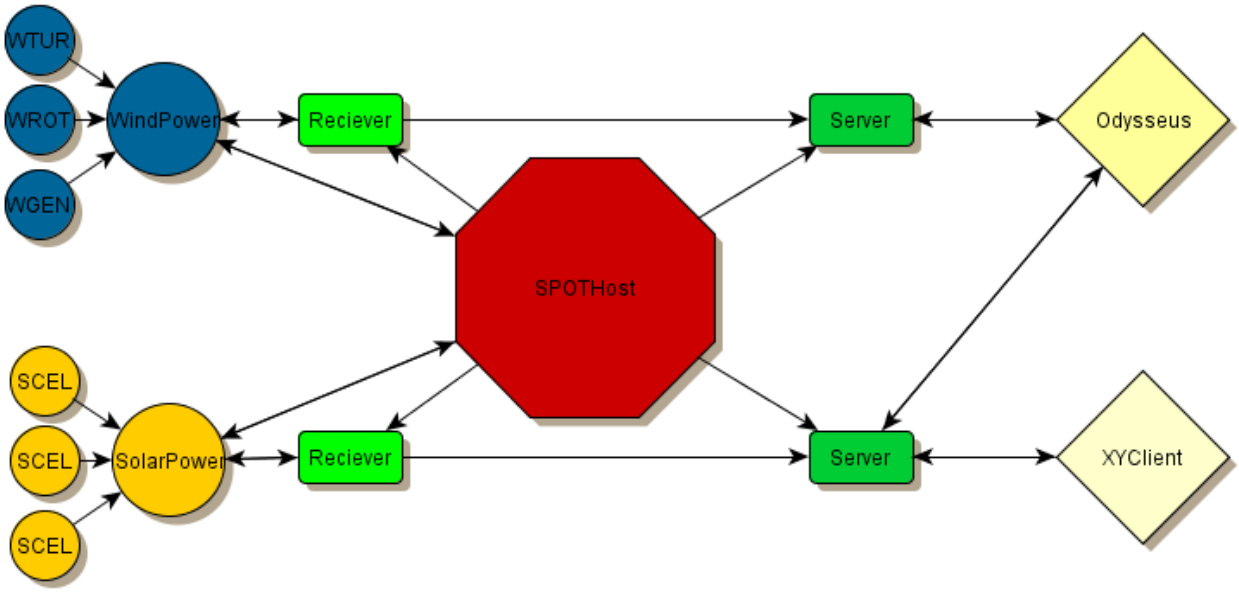


Abbildung 5.1: Architecture

Um größtmögliche Flexibilität hinsichtlich der gleichzeitigen Verarbeitung von einer hohen Anzahl LDSPOTS und Clients (wie z.B. Odysseus) zur ermöglichen werden Threads eingesetzt. Auf Abbildung 5.1 ist dies dargestellt. Dabei senden die einzelnen LNSPOTs (z.B. *WTUR*) an ihren jeweilgen LDSPOT (in diesem Fall *WindPower*). Will sich ein LDSPOT mit dem SpotHost verbinden so sendet er einen Rundspruch und bekommt eine Antwort

vom *SPOTReceptionist*. Dieser öffnet für jeden LD einen neuen *Reciever*-Thread, der die Daten des LDSPOTs in Empfang nimmt. Damit potenziell eine beliebige Anzahl von Clients die Daten abrufen kann erstellt jeder Reciever einen *Server*-Thread, der auf Verbindungen von Clients lauscht.

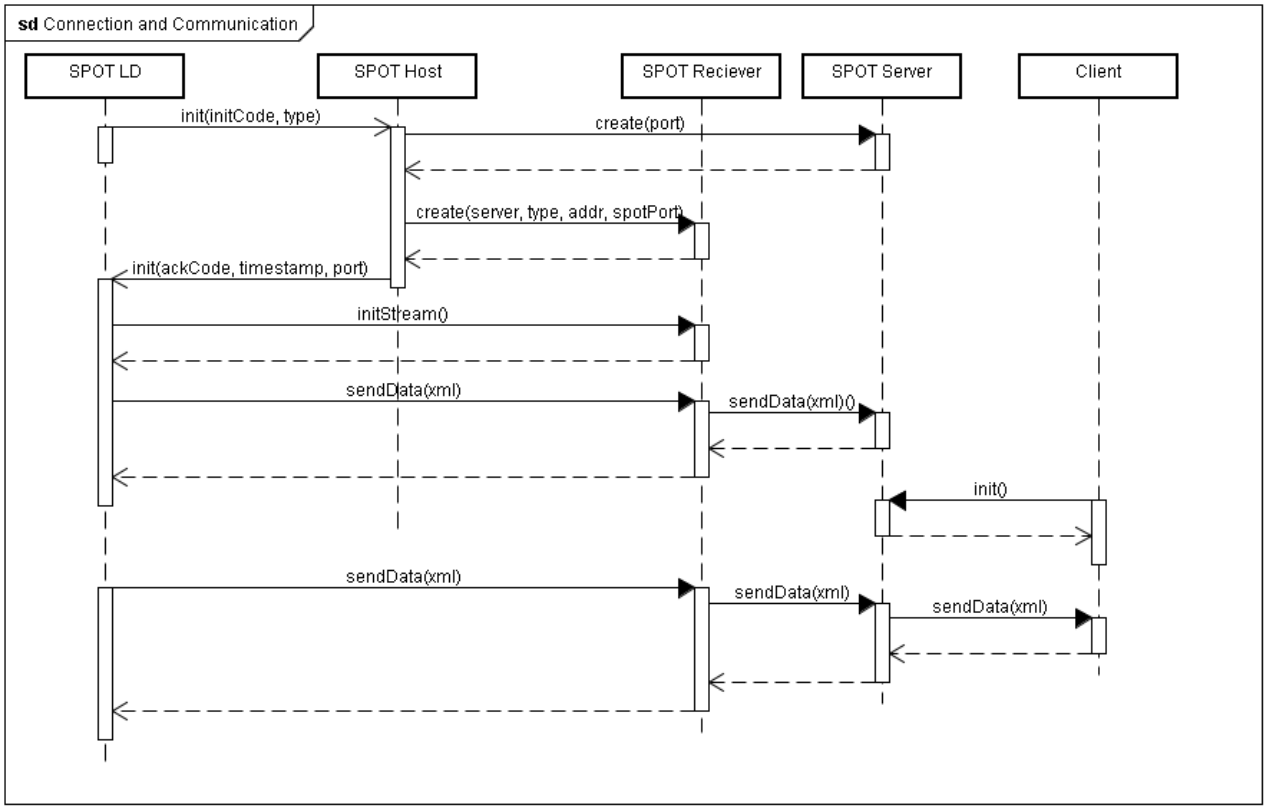
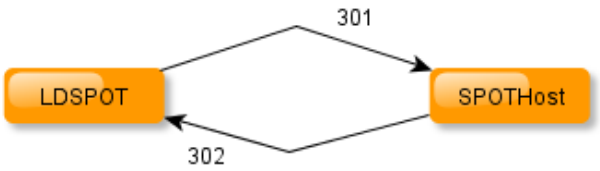


Abbildung 5.2: Connection and Communication

Die genaue Sequenz dieser Anfragen wird nochmal in Abb. 5.2 gezeigt.

5.1.3 Challenge-Response Authentifikation



Für den Aufbau einer Kommunikation zwischen LDSPOT und SPOTHost wird ein einfaches Challenge-Response-Verfahren [Zel04] eingesetzt. Der LDSPOT sendet als Challenge *301* als Rundspruch des Netzwerk. Diese kann im hier vorliegenden Fall nur der SPOTHost beantworten und zwar mit der Response *302* direkt an den LDSPOT, wie auf Abbildung 5.1.3 gezeigt. Es handelt sich also im Prinzip um eine ganz konventionelle Passwortabfrage. Beachtet werden sollte hierbei, dass bei der Wahl des Aufbaus dieses Protokolls Sicherheitsbedenken keine Rolle spielten.

5.1.4 Zeitsynchronisation

Eine Synchronisation der Zeit aller Sensorkomponenten ist im Zusammenhang mit einem Datenstrommanagement von sehr großer Bedeutung. Für die SunSPOTs wird dazu der Algorithmus von Christian [TS07] verwendet. Dies erfolgt hierarchisch von SPOTHost zu LDSPOT und von dort weiter zu allen zugehörigen LNSPOTs. Als erstes sendet dazu

ein SPOT zum Zeitpunkt T_0 eine Anfrage an seinen jeweiligen Server. Dieser antwortet mit seiner aktuellen Zeit und der Client erhält die Antwort zum Zeitpunkt T_1. Er errechnet nun die aktuelle Zeit T, indem zur erhaltenen Zeit $(T_1 - T_0)/2$ addiert wird. Dabei wir idealisiert angenommen, dass die Zeit zum Bearbeitung einer solchen Anfrage konstant ist und die Zeit für den Nachrichtentransport von Client zu Server sowie Server zu Client gleich ist. Da ein WPAN (wireless personal area network) eingesetzt wird ist davon auszugehen, dass die Übermittlungszeiten insgesamt niedrig sind und der Fehler, der durch diese Annahmen auftreten könnte, nicht ins Gewicht fällt. Die Systemzeit des SunSPOTs wird abschliessend auf die ermittelte Zeit gesetzt.

5.2 View

Die View ist eine Eclipse Rich Client Platform Anwendung zur Darstellung von Datenströmen. Die Daten können sowohl in Form von Tabellen oder Graphen dargestellt werden. Im Folgenden werden die Funktionen der Applikation vorgestellt.

5.2.1 Verwendete Technologien

Die Anforderung an die View war, dass sie als Eclipse Plugin verwendet werden kann. Eclipse Plugins sind Bestandteil des Eclipse Equinox Projekts, welches eine Implementierung der OSGi R4 core framework Spezifikation ist. Zudem stützen sich Plugins auf das Eclipse Rich Client Platform Framework (RCP-Framework). Das RCP-Framework nutzt zur Implementierung der grafischen Benutzeroberfläche das Standard Widget Toolkit (SWT), eine Bibliothek zur Erstellung grafischer Oberflächen in Java.

Bestimmte Konfigurationen und Einstellungen der Anwendungen sollen gespeichert und wiederhergestellt werden. Dazu bietet sich XStream, eine Bibliothek zur Objekt-Serialisierung in XML, an.

Die View stellt die Datenströme von Odysseus in verschiedenen Formen dar. Eine Möglichkeit dazu sind Graphen. Als Hauptframework wurde dazu JFreeChart verwendet. Dieses Framework bietet eine übersichtliche API, viele verschiedene Graphentypen an und unterstützt direkt SWT. Das X,Y-Diagramm von JFreeChart ist jedoch sehr unperformant, daher wurde für diesen speziellen Graphtypen SWTChart als weitere Bibliothek eingesetzt.

Eine zusätzliche Anforderung war die Darstellung einer Topologie, also die geographische Darstellung des Stromnetzes und ihrer Komponenten. Dazu bietet sich die Draw2D Komponente des Graphical Editing Frameworks (GEF) an. Diese kann direkt Knoten und Kanten zeichnen, sowie die Kanten routen.

Zur Verwaltungskommunikation mit Odysseus wurden Webservices eingesetzt, da entfernte Methoden nativ, direkt aufgerufen werden können. Zur Generierung wurde die Java API for XML Web Services (JAX-WS) Referenz Implementierung des Metro Projekts genutzt. Die Übertragung der Tupel-Daten hingegen wird über Socket-Kommunikation gelöst, da dabei minimaler Overhead entsteht.

5.2.2 Angewendete Konzepte

Durch die Verwendung des RCP-Frameworks unterliegt die Architektur der View gewissen Beschränkungen, da die Schnittstellen zum Framework strikt vorgeschrieben sind. Die gesamte Anwendung folgt den Entwurfsmuster „Plugin“. So besitzt die View verschiedene Erweiterungspunkte, u.a. Perspektiven, Views und Menüerweiterungen. Die Anwendung ist außerdem nach dem „Model-View-Controller“ -Architekturmuster gegliedert.

Die Initialisierung des Plugins erfolgt durch den *Activator*, der das Plugin mit dem entsprechenden *BundleContext* versorgt. Draufhin wird die *Application* gestartet und zu einer Workbench hinzugefügt. Während der Initialisierung des „Plugin-Lifecycles“ wird zudem der *Controller* initialisiert. Dieser folgt dem Entwurfsmuster „Singleton“. Dieser hält einen *StatementManager*, welcher für die Verwaltungskommunikation mit Odysseus zuständig ist. Zudem erstellt er einen *SocketHandler*, der für jede Anfrage einen Socket öffnet und auf diesem in einem eigenen Thread lauscht, um Tupel von Odysseus zu empfangen. Der *Controller* nimmt diese Daten entgegen und leitet sie an die entsprechenden ContentProvider weiter. Diese versorgen die Views mit den anzuzeigenden Daten und halten diese in einem Ringpuffer, der durch die FixedLengthList realisiert ist. Dabei fungiert der *Controller* als „Beobachter“ für die Views, indem er sie bei neuen Daten benachrichtigt.

Der *ConfigurationManager* folgt auch dem Muster „Singleton“ und ist für die Serialisierung der Einstellungen und deren Wiederherstellung zuständig.

Beim erstmaligen Start der View wird das Initiallayout, welches in der *Perspective* definiert ist, angezeigt. Dieses enthält den *PerspectiveManager*, der Aktionen zum Erzeugen von Views bereitstellt. Jede View ist einer Perspektive zugeordnet, die in einem *TreeObject* gespeichert ist.

Die Kommunikation von Dialogen zum Modell findet über den *ActionController* statt. Dieser ist ebenfalls nach dem Muster „Singleton“ entworfen und bietet den Dialogen gekapselte Schnittstellen zum Modell bzw. Odysseus an.

5.3 3D-View

Als Bestandteil der View stellt die 3D-View eine virtuelle Welt bereit, in der Kraftwerke und Verbraucher dargestellt werden. Einzelne Modelle können Datenströme visualisieren und passen ihre Darstellung den Werten eines Datenstroms an. Im Folgenden werden die verwendeten Technologien und angewendete Konzepte beschrieben.

5.3.1 Verwendete Technologien

Die 3D-View ist Bestandteil der View und ist ebenfalls in Java 1.6 realisiert. Um die 3D-Computergraphik zu erstellen wird die jMonkey Engine [Pow09] verwendet. jMonkey ist ein objektorientierter Szenengraph. Ein Szenengraph ist eine Datenstruktur die wie ein Baum aufgebaut ist. Ausgehend von einer Wurzel können Objekte an diese angehängt werden. Verschiebt man nun einen Knoten in der 3D-Welt werden automatisch alle Kinder ebenfalls verschoben. Dies vereinfacht die Komplexität der Programmierung erheblich. Weitere Funktionen des Frameworks sind unter anderem Funktionen zur Kollisionserkennung oder der Verarbeitung von Eingaben.

Die 3D-Darstellung wird mit OpenGL realisiert. OpenGL bietet eine API um 3D-Darstellung zu ermöglichen. Die Grafikkarte muss dazu das OpenGL-API unterstützen. Dies wird meist durch die Treiber realisiert. Um aus Java auf das OpenGL-API zugreifen zu können, bedient sich jMonkey der Lightweight Java Game Library (LWJGL). Abbildung 5.3 zeigt wie die einzelnen Komponenten aufeinander aufbauen.

Abbildung 5.3: 3D Software Komponenten

Innerhalb der 3D-View werden verschiedene 3D-Modelle wie z.B. ein Windrad oder Häuser verwendet. Die 3D-View erlaubt es Modelle im COLLADA Format (Version 1.4.1) [Mar08] zu laden. COLLADA steht für COLLAborative Design Activity und beschreibt ein 3D-Model in XML. Mit diesem verbreiteten Austauschformat ist es möglich 3D-Modelle aus verschiedenen Quellen zu laden. So können unter anderem weitere Modelle aus der Google Sketchup Gallery geladen werden.

5.3.2 Verwendete Konzepte

Die verwendeten Konzepte teilen sich zum einen in die besondere Art der modularen Gesamtarchitektur und zum anderen in einzelne Schwerpunkte der Implementierung auf.

Architektur

Die Grundarchitektur der 3D-View baut auf dem Vermittler-Entwurfsmuster auf. Wie Abbildung 5.4 zeigt, nimmt IView3dMediator dabei die Vermittlerrolle und *IView3dModule* eine Kollegenrolle ein. Als konkreter Vermittler ist die *World3DRCP* auch der Einstiegspunkt für die 3D-View. *World3DRCP* stellt ein SWT-Canvas für die RCP bereit, so

dass die 3D-Welt als Eclipse-ViewPart lauffähig ist. Es verknüpft somit SWT mit der jMonkey Engine bzw. OpenGL. Dem konkreten Vermittler sind ebenfalls mehrere Module als konkrete Kollegen zugeordnet. Diese Module stellen alle Funktionalitäten für die 3D-Welt bereit.

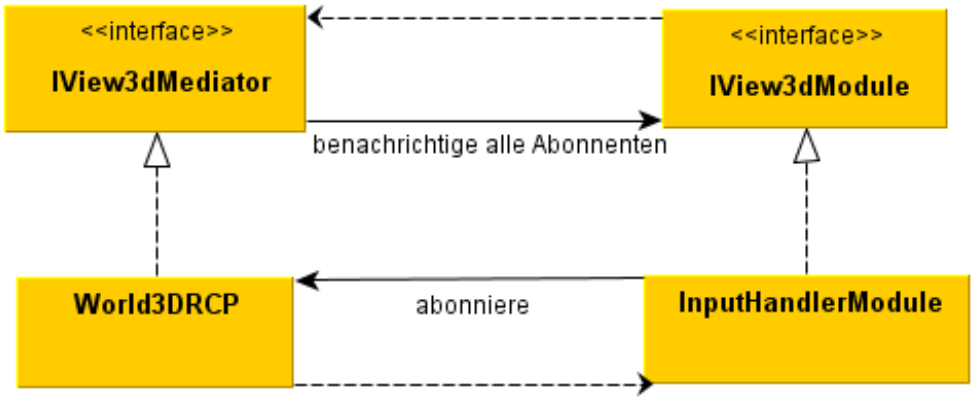


Abbildung 5.4: 3D Grundarchitektur

Des Weiteren basiert die Architektur auf dem Observer-Entwurfsmuster und bietet somit ein modularen Eventmechanismus. Dabei stellt jedes Modul einen Beobachter dar und der *IView3dMediator* stellt den Mediator bzw. das Subjekt dar. Die einzelnen Module können verschiedene Events vom Typ *IView3dEvent* abonnieren und werden beim auftreten eines abonnierten Events informiert. Diese Architektur erlaubt die Kommunikation zwischen einzelnen Modulen, ohne dass diese von einander in Kenntnis sein müssen. Ebenso ist diese lose Koppelung robust gegen das Entfernen oder Hinzufügen von Modulen. So ist es möglich einzelne Module für eine bessere Performanz auszustellen.

Implementierungsschwerpunkte

Die Integration der 3D-Umgebung in das RCP-Framework von Eclipse war eine besondere Herausforderung. Die Standardimplementierung der jMonkey Engine stellt für die Ausgabe ein fertiges AWT-Fenster zur Verfügung. Weiter konnte das von der Engine genutzte AWT-Canvas nicht als Zeichenfläche für die 3D-Welt verwendet werden, da auf Grund des RCP-Framework nur die Verwendung eines Canvas aus dem Standard Widget Toolkit (SWT) möglich ist. Entsprechend musste eine eigene Lösung implementiert werden, die ein OpenGL-Canvas verwendet. Dies erlaubte die Verknüpfung der jMonkey-Engine mit dem SWT. Damit das Zeichnen der 3D-Welt kein flackern erzeugt und noch parallel zur eigentlich View (dem RCP-Plugin) lauffähig ist, musste hier sowohl das Neuzeichnen der Objekte (Rendern) als auch die Neuberechnung der Zustände in zwei zusätzliche Threads verlagert werden. Da alle Threads auf die selben Objekte zugreifen war hier zusätzlich noch eine Synchronisation durch Semaphore nötig.

Ein weiteres Design der 3D-View ist die Verwaltung der Modelle. Jede Art von Modell ist in einem Kompositum zusammengefasst. Ein solches Kompositum wird wiederum von einem Modul verwaltet. So sind alle Windkraftwerke in *PowerplantComposite* als Kompositum zusammengefasst und dieses Kompositum wird von dem *WindMill*-Modul verwaltet. Dadurch werden gemeinsame Ausprägungen und grundlegende Verhaltensweisen zentral durch das Modul für alle Modelle des Kompositums verwaltet. Jedoch können einzelne Modelle auch eigene Ausprägungen und Verhalten haben. Beispielsweise hat ein Windkraftwerk eine Flügelrotation als grundlegende Verhaltensweise. Diese Verhaltensweise wird zentral verwaltet. Die aktuelle Geschwindigkeit dieser Rotation wird hingegen von jedem Modell selbst verwaltet.

Des Weiteren haben Modelle Charakteristiken. Für die Darstellung der Datenströme können Modelle ihr Erscheinen entsprechend eines Attributwertes ändern. So passt sich z.B. die Rotationsgeschwindigkeit der Flügel eines Windkraftwerks dem Wert an. Die Grundimplementierungen für diese Funktionalität wird durch ein *PowerplantNode* gegeben, die, wie vorher beschrieben, in einem Kompositum *PowerplantComposite* zusammengefasst sind. Ein *PowerplantNode* ist dabei ein Beobachter des Controllers der View (siehe 5.2.2). Für jedes Tuple, welches vom Controller empfangen wird, wird anhand eines Filters geprüft, ob es für die Charakteristik ein relevantes Attribut enthält. Daraufhin werden die entsprechenden Charakteristiken ausgeführt. Das Verhalten einer *PowerplantNode* ist an das Decorator-Entwurfsmuster angepasst. So kann ein *PowerplantNode* verschiedene Charakteristiken enthalten. Welche Charakteristik genau ein *PowerplantNode* haben kann, wird durch das Modul bestimmt, welches das *PowerplantNode* über das Kompositum verwaltet. Entsprechend bestimmt z.B. das *WindMill*-Modul, dass jedes *PowerplantNode*, das

sie verwaltet, mit einer Charakteristik für die Flügelrotation dekoriert wird. Abbildung 5.5 zeigt einen beispielhaften

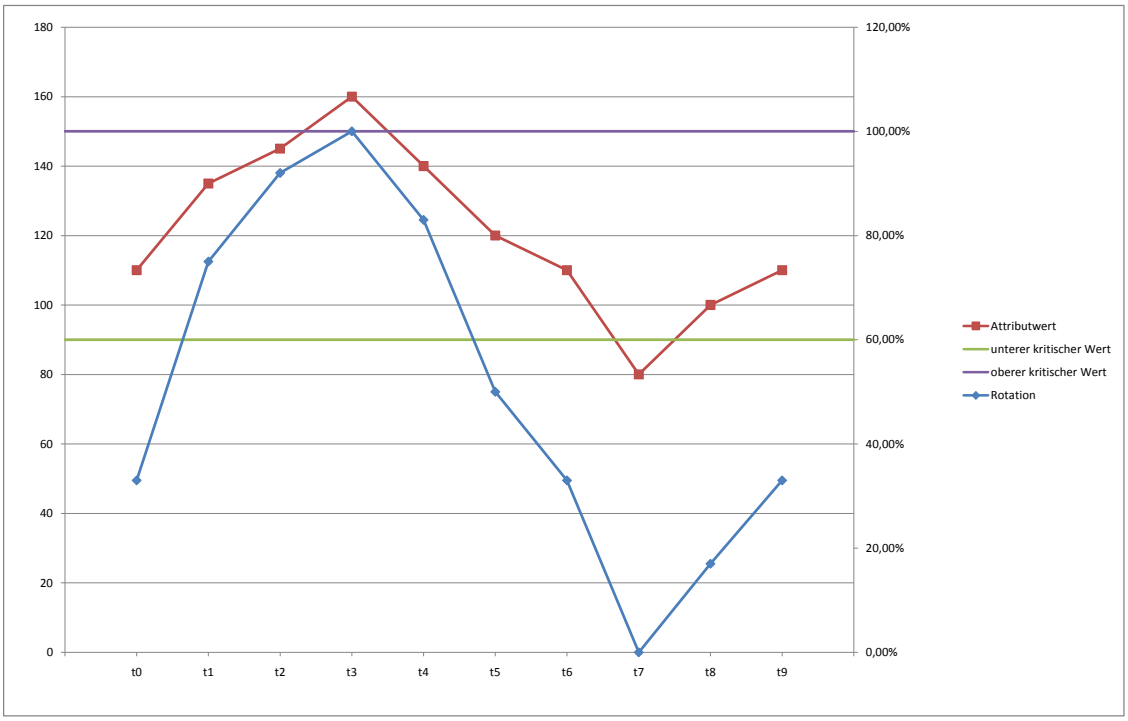


Abbildung 5.5: Normierung der Attributwerte für eine Ausprägung in der 3D-Welt

Verlauf. Dort wurde ein Attribut an die Rotation des Windkraftwerks geknüpft. Das Attribut hat über eine Zeitspanne von 10 Zeiteinheiten (t_0 bis t_9) die Werte $\{110, 135, 145, 160, 140, 120, 110, 80, 100, 110\}$. Für das Attribut muss vorher ein oberer und ein unterer kritischer Wert in der View festgelegt worden sein. Hier ist der untere kritische Wert 90 und der obere kritische Wert ist 150. Die ankommenden Werte des Attributes werden nun anhand der Skala zwischen dem oberen und dem unteren kritischen Wert normiert, hier also zwischen 90 und 150. Die Skala wird hier mit 100% vorgegeben, so dass ein Attributwert von 90 entsprechend 0% und ein Attributwert von 150 analog 100% ist. Tritt ein Attributwert unter 90 bzw. über 150 auf, so entspricht dies 0% bzw. 100%. Angenommen es wird zu einem Attributwert n die entsprechende Normierung $f(n)$ in Prozent gesucht, so ergibt dies:

$$f(n) = \begin{cases} 0, & \text{wenn } n < \text{ untere Grenze} \\ \frac{(n - \text{obere Grenze}) \cdot 100}{(\text{obere Grenze} - \text{untere Grnze})}, & \text{wenn untere Grenze } \leq n \leq \text{obere Grenze} \\ 100, & \text{wenn } n > \text{ untere Grenze} \end{cases}$$

Für die oben genannten Werte in diesem Beispiel ergibt dies folgende Normierungen:

Zeit		t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Attributwert		110	135	145	160	140	120	110	80	100	110
Normiert		33%	75%	92%	100%	83%	50%	33%	0%	17%	33%

Anhand der normierten Werte wird das entsprechende Modell animiert. In diesem Beispiel bedeutet dies, dass die Flügel des Windkraftwerks sich mit 33%, 75%, 92% usw. der maximalen Umdrehungsgeschwindigkeit rotieren. Abbildung 5.5 zeigt einen Verlauf von Attributwerten, sowie den normierten Prozentwerten, die als Grundlage für die Rotationsgeschwindigkeit dienen. Diese Geschwindigkeit bleibt konstant bei dem zuletzt empfangen Attributwert und ändert sich entsprechend bei Eintreffen eines neuen Attributwertes.

Der *ModelLoader* dient zum Laden der Modelle. Es werden, wie oben beschrieben, Modelle im COLLADA-Format unterstützt. Eine Beschreibung, wie zusätzliche Modelle installiert werden können, befindet sich im Installationshandbuch. Der *ModelLoader* lädt das entsprechende COLLADA-Modell und konvertiert es in ein JME-Format. Das JME-Format ist ein bytebasiertes Format der jMonkey Engine und erlaubt ein schnelleres Laden. Alle JME-Dateien werden nach dem konvertieren in einem Verzeichnis gespeichert. Wird ein weiteres Modell geladen und existiert es bereits als JME-Modell, so wird das Modell nicht neu konvertiert, sondern das schnellere JME-Modell wird geladen. Diese Vorgehensweise kann als eine Art Cache betrachtet werden. Wird das COLLADA-Modell geändert, so muss die JME-Datei aus dem Cache gelöscht werden, da es nun invalide ist und daher neu in den Cache geladen werden muss. Ein weiteres Caching wird beim Erzeugen der Modelle vorgenommen. Alle möglichen Modelle werden beim Starten der 3D-View in den Speicher geladen. Jede eigentliche Modellinstanz wird daraufhin nur aus dem Speicher kopiert und instantiiert. Durch dieses Prefetching muss nicht jeder Modelltyp erneut von der Festplatte geladen werden.

5.4 Odysseus

Die mittlere Schicht des EWaF-Softwarestacks wird hier mit Odysseus bezeichnet. Es handelt sich dabei um einen Serverprozess, der das Odysseus-Framework und dessen Datenstrommanagementfunktionalität nutzt, um eine Verbindung zwischen der SunSpot-Schicht und der View-Schicht herzustellen. Im Folgenden werden die verwendeten Technologien und angewendeten Konzepte dieses Serverprozesses beschrieben. Die Technologien des Odysseus-Frameworks selbst werden nicht weiter ausgeführt.

5.4.1 Verwendete Technologien

Der Serverprozess verwendet die Java Virtual Machine 1.6. Um mehrere Verbindungen zur View sowie zu den SunSpots zu verwalten, werden Threads eingesetzt. Der Datenaustausch erfolgt mittels Sockets.

Der Serverprozess bietet seine Funktionalität gegenüber der View mit Hilfe eines Web Services an. Dieser Web Service nutzt die Java 1.6 Web Services Technologie.

Die Daten der SunSpots werden als XML-Elemente übertragen. Sie können dauerhaft gespeichert werden, wofür eine MySQL-Datenbank per JDBC eingebunden wird.

5.4.2 Angewendete Konzepte

Der Serverprozess versucht möglichst robust gegenüber Netzwerkfehlern zu sein. Daher wird sämtliche Netzwerkkommunikation in separaten Threads durchgeführt. Dadurch wird die vorübergehende Störung von Verbindungen transparent gehandhabt. Dies hat zur Folge, dass eine oder mehrere Views beliebig viele Verbindungen jederzeit öffnen und schließen können, ohne dass laufende Abfragen beeinträchtigt werden. Zudem kann mit ungültigen Daten der SunSpots umgegangen werden und bei Verbindungsfehlern findet ein automatisches Neuverbinden statt.

Für die Bereitstellung der API, die von der View verwendet wird, wurde in der zweiten Iteration die Umsetzung mit Hilfe eines Web Services gewählt. Vorherige Experimente mit Java RMI hatten zu einem extrem hohen Aufwand während der Entwicklung sowie beim Einrichten der Software geführt. Mit Hilfe des Web Services konnte die Entwicklung mit wesentlich weniger Aufwand erfolgen und in der Nutzung während des laufenden Betriebs hat sich diese Lösung als robust und problemlos erwiesen.

Der Web Service wird jedoch nur genutzt, um die eigentliche Datenverbindung anzufordern und vorzubereiten. Die Daten selber werden aus Performanzgründen über eine extra Netzwerkverbindung übermittelt, wofür sie unter Verwendung eines Java Object Streams serialisiert werden.

Die MySQL-Datenbank, welche für die Aufzeichnung ausgewählter Datenströme verwendet wird, ist optional. Nur wenn sie vorhanden ist, steht die Aufzeichnungsfunktionalität zur Verfügung. Die Datenbank selbst wird automatisch von dem Serverprozess vorbereitet und passende Tabellen werden angelegt, um die ausgewählten SunSpots aufzuzeichnen. Abgesehen von der Konfiguration von Zugangsdaten ist keine weitere Vorbereitung nötig.

Kapitel 6

Test

6.1 Einführung

Dieser Testplan behandelt die Testpläne für die Software Earth, Wind and Fire, die im Rahmen der Projektgruppe Earth, Wind and Fire an der Carl-von-Ossietzky-Universität Oldenburg erstellt wird.

6.1.1 Zweck

Zweck des Testplans ist es, ein organisiertes und systematisches Testen des Systems Earth, Wind and Fire zu ermöglichen. Es werden die zu testenden Komponenten des Systems benannt und zu diesen adäquate Testfälle entworfen. Auch werden Kriterien gesetzt, wann der Test als erfolgreich durchgeführt gilt und wann er abgebrochen werden muß. Desweiteren werden Anforderungen an Hardware, Software und andere Betriebsmittel, die für die Tests benötigt werden, definiert. Der Testplan richtet sich an den Kunden, an das Entwicklungsteam sowie insbesondere die Tester.

6.1.2 Umfang

Inhalt und Aufbau dieses Testplanes ist Angelegt an den IEEE Standard 829-1998 [**?**] . Schwerpunkt der Testfälle sollen hier Integrations-, Funktions-und Leistungstests sein.

6.1.3 Beziehungen zu anderen Dokumenten

Im Testplan wird dargelegt, wie die Erfüllung der Anforderungen aus dem Lastenheft und Pflichtenheft sichergestellt wird. Dadurch ist ein direkter Bezug zu diesen Dokumenten gegeben. Die Anwendungsfälle aus dem Lastenheft werden direkt als Testfälle für Abnahmetests dienen. Weitere Testfälle werden aus den Klassendiagrammen des Pflichtenheftes erstellt.

6.1.4 Aufbau der Testbezeichner

Die Testbezeichner sind nach folgendem Prinzip aufgebaut:

- Die ersten beiden Buchstaben stehen für die Art des Testes(vgl. Tabelle 6.1)
- Die folgende Zahl steht für die durchlaufende Nummer des Testes nach der angegebenen Testart.
- Der abschließende Buchstabe bezeichnet die Variante des Testes. Dieser Teil ist optional.

Beispiel: IT-3-B steht für Integrationstest Nummer 3, Variante 2.

IT	Integrationstest
FT	Funktionstest
LT	Leistungstest

Tabelle 6.1: Kürzelordnung der Testberichte

6.2 Merkmale

Da nicht jede Codezeile getestet werden muss, wird hier aufgeführt, was zu testen ist und welche Merkmale nicht getestet werden brauchen.

6.2.1 Zu testende Merkmale

Die implementierten Klassen müssen korrekt funktionieren. Das Zusammenspiel zwischen einzelnen Klassen bzw. Komponenten muss korrekt ausgeführt werden. Die Module sind auf Einhaltung ihrer Spezifikation zu testen. Das System muss angemessen auf auftretende Fehler und unerwartete Eingaben reagieren.

6.2.2 Funktionale Anforderungen

Besonders die funktionalen Anforderungen, die im Lastenheft identifiziert worden sind, müssen erfüllt werden. Hierbei liegt das Hauptaugenmerk auf der Trennung von Kioskmodus und Adminmodus.

6.2.3 Nicht zu testende Merkmale

Verwendete Java-Bibliotheken von Drittanbietern müssen nicht getestet werden, da diese von den Anbietern selbst getestet wurden. Getestet wird nur das Zusammenspiel mit diesen Bibliotheken.

Methoden von Java-Klassen, wie z.B. set- und get-Methoden, die lediglich auf die Attribute eines Objektes zugreifen, brauchen nicht getestet zu werden.

6.3 Abnahme- und Testendekriterien

Die „wahre“ Fehlerzahl einer Software ist unbekannt. Daher müssen im Vorfeld Kriterien bestimmt werden, die festlegen, in welchem Umfang das Testen stattfindet und wann die Tests beendet werden können.

Wenn ein Test fehlschlägt, also einen Fehler gefunden hat, wird dieser Fehler nach der Tabelle 6.2 in Fehlerklassen eingeteilt. Fehler der Fehlerklassen 3 und 4 sind in der Abgabe nicht gestattet und müssen unbedingt behoben werden. Fehler der Fehlerklassen 1 und 2 sollten behoben werden, wenn es die Zeit erlaubt. Sie sind ansonsten nicht kritisch.

Fehlerklassen	Beschreibung
1	**unwesentlich**: Fehler, der keine sichtbaren Schäden verursacht, aber trotzdem korrigiert werden muss
2	**gering**: Fehler, der das System in seiner Leistung einschränken oder dessen Verfügbarkeit reduzieren könnte
3	**kritisch**: Fehler, der schwere Schäden am System verursachen kann
4	**katastrophal**: Fehler, der die Unterbrechung des Systems verursachen kann bzw. das System unbrauchbar macht

Tabelle 6.2: Fehlerklassen

6.4 Vorgehensweise

6.4.1 Integrationstest

Die einzelnen Gruppen stellen die richtige Funktion ihrer einzelnen Klassen sicher. Die einzelnen Klassen werden in diesem Testplan nicht mehr beachtet. Nach Abschluss der Implementierung werden die einzelnen Komponenten unabhängig von einander getestet. Für die Simulation von Datenströmen werden Dummys verwendet.

Nach den einzelnen Komponententests folgt der Integrationstest. Hier werden zuerst die SunSpots und Odysseus zusammengefügt. Hier werden Daten von den SunSpots erzeugt, durch Odysseus geleitet und aufbereitet.

Der Integrationstest von Odysseus und der View erfolgt direkt im Anschluss. Hier soll sichergestellt werden, das die Daten von Odysseus richtig an die View weitergereicht werden. Auch müssen hier die CQL-Abfragen getestet werden.

Sind diese beiden Test bestanden, werden die drei Subsysteme gemeinsam getestet. Hier soll der komplette Umfang getestet werden.

6.4.2 Funktionstest

Diese Tests beziehen sich direkt auf die Anwendungsfälle aus dem Lastenheft. Sie testen die Funktionen, die der Nutzer am fertigen Produkt bedienen kann. Mindestanforderung hierfür ist es, jeden Fall einmal auf seine korrekte Funktion zu testen.

6.4.3 Leistungstest

Durch die Leistungstests sollen die Grenzen des Systems getestet werden.

6.5 Aufhebung und Wiederaufnahme

Wenn sich während des Testverlaufs zeigt, dass die Anzahl fehlerhafter Tests einen Schwellwert übersteigt, ist es sinnvoll, die Tests abzubrechen und zur Implementierungsphase zurückzukehren. Dieser Schwellwert ist abhängig von der jeweiligen Testphase, der Testart sowie der Schwere des Fehlers.

Fehler in den unteren Schichten haben eine starke Auswirkung auf die Funktionsweise des Systems, da sie zu fehlerhaften Übergaben von Daten an Methoden einer höheren Schicht führen. Die abhängigen Komponenten können dann nicht mehr sinnvoll getestet werden. Bei Testfällen in tiefer liegenden Schichten muß deshalb ein niedrigerer Schwellwert angesetzt werden als bei Tests in oberen Schichten (z.B. Tests der Controller-Schicht oder Funktionstests).

Bei der Wiederaufnahme der Testphase muß beachtet werden, daß Veränderungen an Komponenten dazu führen, dass alle davon abhängigen Komponenten neu getestet werden müssen.

6.6 Testfälle

In diesem Kapitel werden alle zu testenden Testfälle aufgelistet.

6.6.1 Integrationstests

Die Integrationstests dienen zur Unterstüzung der Zusammenfügung der einzelnen Komponenten. Sie sollten nach jeder größeren Änderung einer Komponente durchgeführt werden um die Kommunikation zwischen den Komponenten zu gewährleisten.

IT-1	Integration Kraftwerkssimulation - Odysseus	
	Testziel	SunSpots senden Daten an Odysseus.
	Vorbedingungen	SunSpot kann Daten senden und Odysseus kann Daten empfangen.
	Eingabe	Daten senden von SunSpots.
	erwartete Ausgabe	Gesendete Daten werden durch Odysseus durchgeschleust.
	Abhängigkeiten	Kommunikation zwischen SunSpot- und Odysseus-Rechner ist möglich.

IT-2	Integration Odysseus - View	
IT-2-A	Testziel	Daten werden von Odysseus an die View gesendet und dort angezeigt.
	Vorbedingungen	Odysseus kann Daten senden und View kann Daten empfangen.
	Eingabe	Daten senden von Odysseus.
	erwartete Ausgabe	Gesendete Daten werden von der View korrekt angezeigt.
	Abhängigkeiten	Kommunikation zwischen Odysseus- und View-Rechner ist möglich.
IT-2-B	Testziel	CQL Anfragen können installiert werden
	Vorbedingungen	Odysseus kann Daten senden und View kann Daten empfangen.
	Eingabe	CQL Anfrage in der View.
	erwartete Ausgabe	Empfangene Daten sind gefiltert.
	Abhängigkeiten	Kommunikation zwischen Odysseus- und View-Rechner ist möglich.
IT-2-C	Testziel	Mehrere Anfragen können zeitgleich erfolgen
	Vorbedingungen	Odysseus kann Daten senden und View kann Daten empfangen.
	Eingabe	Verschiedene Anfragen an Odysseus.
	erwartete Ausgabe	Daten werden gleichmäßig durch die verschiedenen Anfragen angezeigt.
	Abhängigkeiten	Kommunikation zwischen Odysseus- und View-Rechner ist möglich.

IT-1	Integration Kraftwerkssimulation - Odysseus - View	
	Testziel	SunSpots senden Daten an Odysseus, diese werden von Odysseus an die View gesendet und dort angezeigt.
	Vorbedingungen	SunSpot kann Daten senden, Odysseus kann Daten verarbeiten und View kann Daten empfangen.
	Eingabe	Daten senden von SunSpots.
	erwartete Ausgabe	Gesendete Daten werden durch Odysseus durchgeschleust und von der View korrekt angezeigt.
	Abhängigkeiten	Kommunikation zwischen SunSpot-, Odysseus- und View-Rechner ist möglich.

6.6.2 Funktionstests

Die Funktionstests beziehen sich auf die Anwendungsfälle aus dem Lastenheft.
Jeder Testfall wird auf korrekte Funtkionalität und auf ein korrektes Verhalten im Fehlerfall getestet werden.

Kraftwerkssimulation

FT-KWS-1	Simuliere Kraftwerk	
FT-KWS-1-A	Testziel	Kraftwerk werden korrekt simuliert
	Vorbedingungen	SunSpots eingeschaltet und mit SpotHost verbunden
	Eingabe	SunSpot sammelt Daten und schickt diese an den SpotHost
	erwartete Ausgabe	Generierte Daten werden vom SpotHost angezeigt
FT-KWS-1-B	Testziel	Ermittle Temperaturdaten
	Vorbedingungen	SunSpots eingeschaltet.
	Eingabe	SunSpot erwärmen und abkühlen.
	erwartete Ausgabe	Momentane Umgebungstemperatur.
FT-KWS-1-C	Testziel	Ermittle Lichtintensität
	Vorbedingungen	SunSpots eingeschaltet.
	Eingabe	SunSpot beleuchten und abdunkeln.
	erwartete Ausgabe	Momentanen Lichtwerte.
FT-KWS-1-D	Testziel	Ermittle Beschleunigungsdaten
	Vorbedingungen	SunSpots eingeschaltet und mit Windrad verbunden.
	Eingabe	Windrad schneller und langsammer drehen.
	erwartete Ausgabe	Momentanen Drehzahl aus dem erzeugten Strom des Dynamo umgewandelt.

FT-KWS-2	Virtuelle Kraftwerke	
	Testziel	Virtuelle Kraftwerke liefern logische Daten
	Vorbedingungen	Virtuell SpotHost gestartet
	Eingabe	Virtuelles Windkraftwerk starten
	erwartete Ausgabe	XML-Tupel mit standartkonformen Daten und sinnvollen Größen

Odysseus

FT-Ody-1	Odysseus	
FT-Ody-1-A	Testziel	Stream initialisieren
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden
	Eingabe	IP, Port und Art der Datenquelle
	erwartete Ausgabe	Bestätigung über die erfolgreiche Eingabe
	Fehlerfallbehandlung	Fehlermeldung ausgeben
FT-Ody-1-B	Testziel	Statement hinzufügen
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert
	Eingabe	Statement
	erwartete Ausgabe	Bestätigung über die erfolgreicht Eingabe
	Fehlerfallbehandlung	Fehlermeldung ausgeben

View

FT-View-1	Initialisieren

FT-View-1-A	Testziel	Verbindung mit Odysseus herstellen
	Vorbedingungen	IP von Odysseus bekannt
	Eingabe	IP
	erwartete Ausgabe	Bestätigung über die erfolgreicht Eingabe
	Fehlerfallbehandlung	Fehlermeldung ausgeben
FT-View-1-B	Testziel	Stream initialisieren
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden
	Eingabe	IP, Port und Art der Datenquelle
	erwartete Ausgabe	Bestätigung über die erfolgreicht Eingabe
	Fehlerfallbehandlung	Fehlermeldung ausgeben
FT-View-1-C	Testziel	Statement hinzufügen
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert
	Eingabe	Statement
	erwartete Ausgabe	Bestätigung über die erfolgreicht Eingabe
	Fehlerfallbehandlung	Fehlermeldung ausgeben

FT-View-2	Darstellung von Daten	
FT-View-2-A	Testziel	Darstellung als Tabelle
	Testziel	Daten werden als Tabelle dargestellt
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Gewünschtes Statement, dessen Daten als Tabelle dargestellt werden sollen
	erwartete Ausgabe	Tabelle mit den Daten aus den Tupeln die dem Statement entsprechen
FT-View-2-B	Testziel	Darstellung als Liniendiagramm
	Testziel	Daten werden als Liniendiagramm dargestellt
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Gewünschtes Statement, dessen Daten als Liniendiagramm dargestellt werden sollen
	erwartete Ausgabe	Liniendiagramm mit den Daten aus den Tupeln die dem Statement entsprechen
FT-View-2-C	Testziel	Darstellung als Balkendiagramm
	Testziel	Daten werden als Balkendiagramm dargestellt
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Gewünschtes Statement, dessen Daten als Balkendiagramm dargestellt werden sollen
	erwartete Ausgabe	Balkendiagramm mit den Daten aus den Tupeln die dem Statement entsprechen
FT-View-2-D	Testziel	Darstellung als Tortendiagramm
	Testziel	Daten werden als Tortendiagramm dargestellt
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Gewünschtes Statement, dessen Daten als Tortendiagramm dargestellt werden sollen
	erwartete Ausgabe	Tortendiagramm mit den Daten aus den Tupeln die dem Statement entsprechen

FT-View-3	Topologie	
	Testziel	Topologie mit Positionsdaten aus 3D-View

	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert, 3D-Welt mit Kraftwerken gespeichert
	Eingabe	Gespeicherte Welt auswählen
	erwartete Ausgabe	Topologie mit Kraftwerken und Verbindungen

FT-View-4	Kritische Werte	
FT-View-4-A	Testziel	Kritische Werte einstellen
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Wertname und Wertebereiche
	erwartete Ausgabe	Überschreiten von Grenzen wird in Tabelle, Topologie und 3d-Welt kenntlich gemacht
FT-View-4-B	Testziel	Alarm Bar
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert, Kritische Werte eingestellt
	Eingabe	Bestätigung, das bei Überschreitung der Werte eine Anzeige in der Alarm Bar erfolgen soll
	erwartete Ausgabe	Anzeige bei überschrittenen Werten in der Alarm Bar

FT-View-5	Event Viewer	
FT-View-4-A	Testziel	Anzeige von Events
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert, Kritische Werte eingestellt
	Eingabe	Keine
	erwartete Ausgabe	Anzeige bei überschrittenen Werten in dem Event Viewer
FT-View-4-B	Testziel	Bestätigen von Events
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert, Kritische Werte eingestellt
	Eingabe	Bestätigen einzellner oder sämtlicher Events
	erwartete Ausgabe	Kenntlichmachung als „Bestätigt“ mit Zeitangabe der Bestätigung

3D-Welt

FT-3D-1	Kraftwerke	
FT-3D-1-A	Testziel	Geothermalkraftwerk erstellen
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Statement für das Kraftwerk
	erwartete Ausgabe	Kraftwerk wird erstellt und reagiert auf Datenquelle
	Fehlerfallbehandlung	Fehlermeldung ausgeben
FT-3D-1-B	Testziel	Solaranlage erstellen
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Statement für das Kraftwerk
	erwartete Ausgabe	Kraftwerk wird erstellt und reagiert auf Datenquelle
	Fehlerfallbehandlung	Fehlermeldung ausgeben
FT-3D-1-A	Testziel	Windkraftanlage erstellen
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Statement für das Kraftwerk

	erwartete Ausgabe	Kraftwerk wird erstellt und reagiert auf Datenquelle
	Fehlerfallbehandlung	Fehlermeldung ausgeben

FT-3D-1	Stromleitungen	
FT-3D-1-A	Testziel	Optische Verbindung zwischen Kraftwerken
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Start- und Zielpunkt der Leitung
	erwartete Ausgabe	Leitung wird erstellt
FT-3D-1-B	Testziel	Leitung reagiert auf kritische Werte
	Vorbedingungen	Datenquelle vorhanden, View mit Odysseus verbunden, Stream initialisiert, Statement installiert, Kritische Werte eingestellt
	Eingabe	Statement für die Leitung
	erwartete Ausgabe	Leitung verfärbt sich bei Überschreitungen der eingestellten Werte

6.6.3 Leistungstests

Da die Leitsung der Software erheblich von der verwendeten Hardware abhängig ist, wird auf einen Großteil der Leistungstests verzichtet. Es ist zum Beispiel nicht möglich, eine Aussage über die anzeigbare Anzahl von Tabellen, Graphen oder Kraftwerken in der 3D-Welt zu machen, da diese je nach Hardwareausstattung stark variieren kann.

Odysseus

LT-Ody-1	Mehrere Views	
	Testziel	Mehrere Views erhalten Daten von einem Odysseus
	Vorbedingungen	Datenquelle vorhanden, Views mit Odysseus verbunden, Stream initialisiert, Statement installiert
	Eingabe	Tabelle vom gleichen Statement in mehreren Views erzeugen
	erwartete Ausgabe	Tabelle wird zeitnah und ohne Verzögerungen mit Daten versorgt
	Fehlerfall	Starke verzögerte Ankunft der Tupel

6.7 Durchgeführte Tests

Hier werden sämtliche durchgeführten Tests protokolliert.

6.7.1 Integrationstest

Die Integrationstests wurden hauptsächlich in der Endphase der Entwicklung durchgeführt.

29.07.2009

Kraftwerkssimulation: Die SunSpots müssen für einen dauerhaften Einsatz stabiler laufen. Bei den gesammelten Daten sind viele Felder der Norm frei, hier müssen Daten ermittelt und übertragen werden.

Odysseus: Booleans werden nicht unterstützt. Die Zeitstempel werden standartkonform in Sekunden, nicht Millisekunden übertragen, so das diese umgerechnet werden müssen. Mehrere Views werden nicht unterstützt. Wird eine angeschlossene View beendet, stürzt Odysseus ab.

View: Bei den Liniendiagrammen fehlt in der Liste der Quellen eine Scrollleiste.

3D-Welt: bei der Auswahl der anzuzeigenden Daten fehlt eine Scrollleiste.

12.08.2009

Die festgestellten Mängel der vorangegangenen Tests wurden komplett abgestellt.

Odysseus: das Archiving erzeugt einen Nullpointer, JOIN als CQL nicht möglich.

09.09.2009

JOINs sind möglich, wenn man die Werte mit AS umbenennt und einer RANGE angibt.

Kraftwerkssimulation: Probleme mit der Übertragung zu Odysseus, hier werden mehr Daten angekündigt als tatsächlich gesendet werden, dadurch kommt es bei Odysseus zu einer Exception mit Nullpointer.

6.7.2 Funktionstest

Der abschließende Funktionstest wurde vom Qualitätsmanager am 11.09.2009 durchgeführt. Es wurden keine Beanstandungen festgestellt und der Test als bestanden gewertet.

6.7.3 Leistungstest

Der abschließende Leistungstest wurde vom Qualitätsmanager am 11.09.2009 durchgeführt. Hierzu wurden vier Views mit einer Installation von Odysseus verbunden und konnten ohne Probleme Daten zu Statements abfragen.

Kapitel 7

Ablauf

7.1 KickOff-Phase

Zu Beginn der Projektgruppe wurde eine Seminarphase abgehalten. Hier hat sich jeder Projektgruppenteilnehmer in einen Teilbereicht des zu erwartenden Umfeldes intensiv eingearbeitet und eine Präsentation über seine Ergebnisse gegeben sowie eine Ausarbeitung angefertigt.

In diesen Zeitraum viel auch das erste Gespräch mit der Kundin. Hier wurden der groben Umfang der zu erstellenden Software festgelegt und Einzelheiten zu gewünschten Features geklärt.

7.2 Erste Iteration

7.2.1 Meilensteinbetrachtung

Meilensteine	Termin	Status
Fertigstellung des Angebots	16.01.2009	abgeschlossen
Fertigstellung des Lastenhefts	16.01.2009	abgeschlossen
Fertigstellung des Pflichtenhefts	06.02.2009	abgeschlossen
Lauffähige View	06.02.2009	abgeschlossen
Lauffähiges Odysseus	06.02.2009	abgeschlossen
Lauffähige SunSpots	06.02.2009	abgeschlossen
Abschluss erste Iteration	11.02.2009	abgeschlossen

Tabelle 7.1: Tabelarische Übersicht der Meilensteine

In der obigen Tabelle ist eine Zusammenstellung der Meilensteine inklusive ihrer Termine und dem entsprechenden Erledigungsgrad.

Anhand der Meilensteine ist eine grobe Gliederung der Projektgruppe zu erkennen. Wir gliedern uns in vier Bereiche. Der Erste ist der Bereich für die Planung sowie die Erstellung des Angebots. Da das Lastenheft auf dem Angebot aufsetzt zählt das Lastenheft sowie das Pflichtenheft auch dazu. Der Zweite Bereich ist die Visualisierung der Datenströme. Der Dritte Bereich ist die Verarbeitung der Daten aus den Sensoren und die Weiterleitung mittels Odysseus. Die SunSpots bilden den vierten Vereich.

Die Projektgruppe befindet sich zur Zeit sehr gut im Zeitplan. Es wurde von allen Teilbereichen geschafft alle Meilensteine einzuhalten. Die oben dargelegte Planung bezieht sich auf die 1. Iteration. Die Planung wurde dementsprechend auf den SIT ausgelegt, sodass zum SIT ein bereits vorzeigbares Ergebnis entwickelt werden konnte.

Alle Ziele der einzelnen Teams zur ersten Iteration konnten eingehalten werden. Sodass der Integrationstest auch erfolgreich abgeschlossen werden konnte. Zum Ende der ersten Phase kamen neue Ideen bezüglich der Schnittstelle zwischen View und Odysseus auf. Allerdings wäre das Risiko die erste Iteration durch die Neuplanung und Implementierung zu verlieren zu groß gewesen. Aus diesem Grund wurde der derzeitige Stand für die erste Iteration eingefroren, da der Integrationstest zu diesem Zeitpunkt bereits erfolgreich war. Um die Ideen und bereits geänderte Konzepte nicht zu verlieren wurde im SVN ein Tag angelegt, welcher zur Weiterentwicklung genutzt werden kann.

Für die nächste Iteration muss jetzt zu Beginn eine weitere Planungsphase eingeleitet werden. Es müssen die Arbeitspakete für die zweite Iteration ermittelt und in einen zeitlichen Verlauf gebracht werden und deren Abhängigkeiten berücksichtigt werden. Dazu dienlich ist ebenfalls ein weiterer Netzplan. Es wurden bereits Arbeitspakete aus der 1. Iteration in die zweite Iteration verschoben, da diese aus zeitlichen Gründen nicht mehr zu schaffen waren. Eine Verschiebung und damit eine Reduzierung der Leistung war nicht mehr zu vermeiden, wenn die Meilensteine gehalten werden wollten. Zu diesen Arbeitspaketen gehören die Datenhaltung in Odysseus, sowie die dynamische Anfrageverwaltung mit Hilfe von CQL.

7.2.2 Fortschritte weiterer Arbeitspakete

Verwaltung der Tabellen (View)

Es ist jetzt möglich sich die ankommenden Daten in einer Tabelle anzeigen zu lassen. Diese Tabellen werden auch in gewissen Zeitabständen aktualisiert (immer wenn neue Daten ankommen) um einen aktuellen Stand immer gewährleisten zu können. Dieses Arbeitspaket wurde mit einem Fertigstellungsgrad von 100% abgeschlossen.

Erstellung von Graphen (View)

Zusätzlich zu der Visualisierung der Daten durch die Tabellen wird die Möglichkeit gegeben Graphen zu erstellen. Diese Graphen wurden mit dem Framework JFreeChart realisiert. Es ist derzeit möglich Graphen unterschiedlichen Typs zu instanziieren um die Graphen somit auf die Datentypen anzupassen. Als Graphtypen stehen derweil zwei Typen zur Auswahl. Diese können bei der Erstellung des Graphens ausgewählt werden.

Dieses Arbeitspaket wurde mit 100% abgeschlossen.

Ansichtenverwaltung (View)

Es können verschiedene Views(Ansichten) verwaltet und gespeichert werden. Die unterschiedlichen Views können erstellt werden und als XML-Datei gespeichert werden. Zur Verwaltung der Views dient eine Baumstruktur durch die der User navigieren und sich die entsprechenden Views anzeigen lassen kann. Dieses Arbeitspaket ist zu 100% Fertig.

Anfrageverwaltung (View)

Das hinzufügen von Tabellen zu einer View funktioniert über ein Dialogfenster. In diesem ist es möglich eine CQL-Anfrage zu stellen um diese in Odysseus zu installieren oder bereits vorgefertigte CQL-Statements auszuwählen. Diese Statements werden dann per RMI direkt in Odysseus installiert bzw. deinstalliert. Seitens der View ist alles implementiert und das Arbeitspaket somit zu 100% abgeschlossen.

Anfragenverwaltung (Odysseus)

Es wurden einige Schemainformationen integriert, sodass Anfragen vom Typ 'Select * from test:solar where identifier $>$ 50' funktionieren. Weitere Schemata werden nach und nach integriert. Eine Verarbeitung von dynamischen Anfragen (unteranderem die Erstellung neuer Datenströme) wurde in die zweite Iteration verschoben. Die "Create Stream.."-Statements sind derzeit fest im Code integriert. Es wurde allerdings mit den geänderten Zielen für dieses Arbeitspaket ein Fertigstellungsgrad von 100% erreicht.

Schnittstellen (Oysseus)

Des Weiteren ist der Wrapper zwischen SunSpot und Odysseus implementiert (für Windenergie), der die Daten der SunSpots annimmt und in RelationalTuple umwandelt und diese dann an Odysseus übergibt. In der zweiten Iteragtion werden die Daten als Rohdaten per java.nio Kommunikation gesendet um eine Effizienzsteigerung zu erhalten. Die Schnittstellen für die erste Iteration wurden zu 100% abgeschlossen.

Datenhaltung (Odysseus)

Um Anfragen stellen zu können, die Daten über einen ganzen Zeitraum aggregieren, wird mit Hilfe einer Datenbank eine Datenhaltung eingerichtet. Diese wird allerdings aus Zeitmangel und um die erste Iteration nicht zu riskieren erst in der zweiten Iteration umgesetzt.

KraftwerkServer (SunSpot)

Für jeden Kraftwerktyp (Windkraft, Geothermal und Solar) wurde ein Server implementiert, der die Daten von den SunSpots annimmt und per XML über eine TCP-Verbindung weiter an Odysseus schickt. Die Daten werden zuvor von den Servern in eine standardkonforme Struktur gebracht. Dieses Arbeitspaket ist zu 100%abgeschlossen.

KraftwerkClient (SunSpot)

Verschiedene Daten werden mit Hilfe der SunSpots bzw. deren Sensoren ermittelt. Die SunSpots bereiten diese Daten geringfügig vor und leiten diese weiter an die SunSpotServer (Host). Alle Kraftwerktypen sind dabei getrennt zu betrachten. Sie sind alle zu 100% abegeschlossen. Da der PG kein Standard für Solar und Geothermal vorliegt, lehnen sich die Implementierung an den Windkraftwerkstandard an.

Schnittstellen (SunSpot)

Die Daten werden wie bereits erwähnt vom Host in XML gewandelt und per TCP an Odysseus geschickt. Der implementierte Wrapper liegt schon auf der Odysseusseite. Die Schnitstelle steht also zu 100%.

Tests

Um die Qualität unseres Produkts sicher zu stellen, führen wir verschiedene Test durch. Die erste Testphase wurden Anfang Februar abgeschlossen. Testpläne sind erstellt worden um ein organisiertes Testen durchführen zu können. Abschluss der Testphase war ein erfolgreich durchgeführter Integrationstest.

Schülerinformationstag

Ein für den SIT entwickeltes Windrad wurde gebaut, um produzierte Energie simulieren zu können. Dazu wurde Rotationsbewegung mit Hilfe eines Dynamos in Elektrizität umgewandelt. Die SunSpots können diese produzierte Energie messen und an Odysseus weiterleiten. Der SIT diente als gute Erprobungsphase für unseren EWaF-Client. Zur Demonstration wurden weitere Sensoren aufgebaut, die Lichtintensität und Temperatur gemessen haben. Weiterhin gehörte zum SIT eine Betreuung der Besucher. Diese konnten Fragen stellen oder erhielten von uns eine Einweisung in die Software.

7.2.3 Übergabe des Projekts an Christian

Zu Beginn des Projekts wurde sich darauf geeinigt, dass ich die erste Iteration als Projektleiter betreue und Christian die zweite Iteration übernimmt. Die offizielle Übergabe wird am Donnerstag den 26.2.09 erfolgen. Ich werde weiterhin zur Verfügung stehen und die Rolle des stellvertretenden Projektleiters übernehmen.

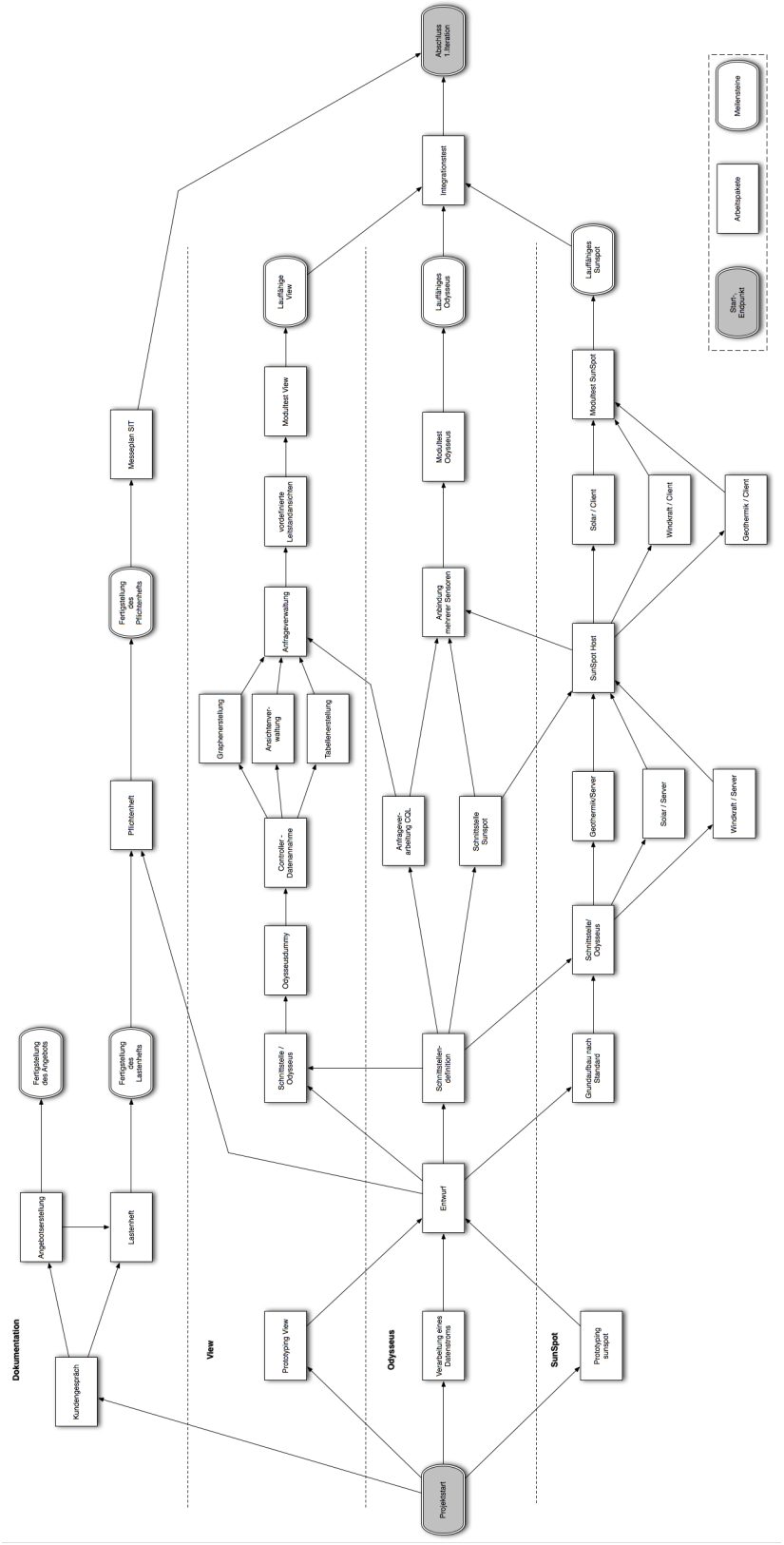


Abbildung 7.1: Netzplan der 1. Iteration

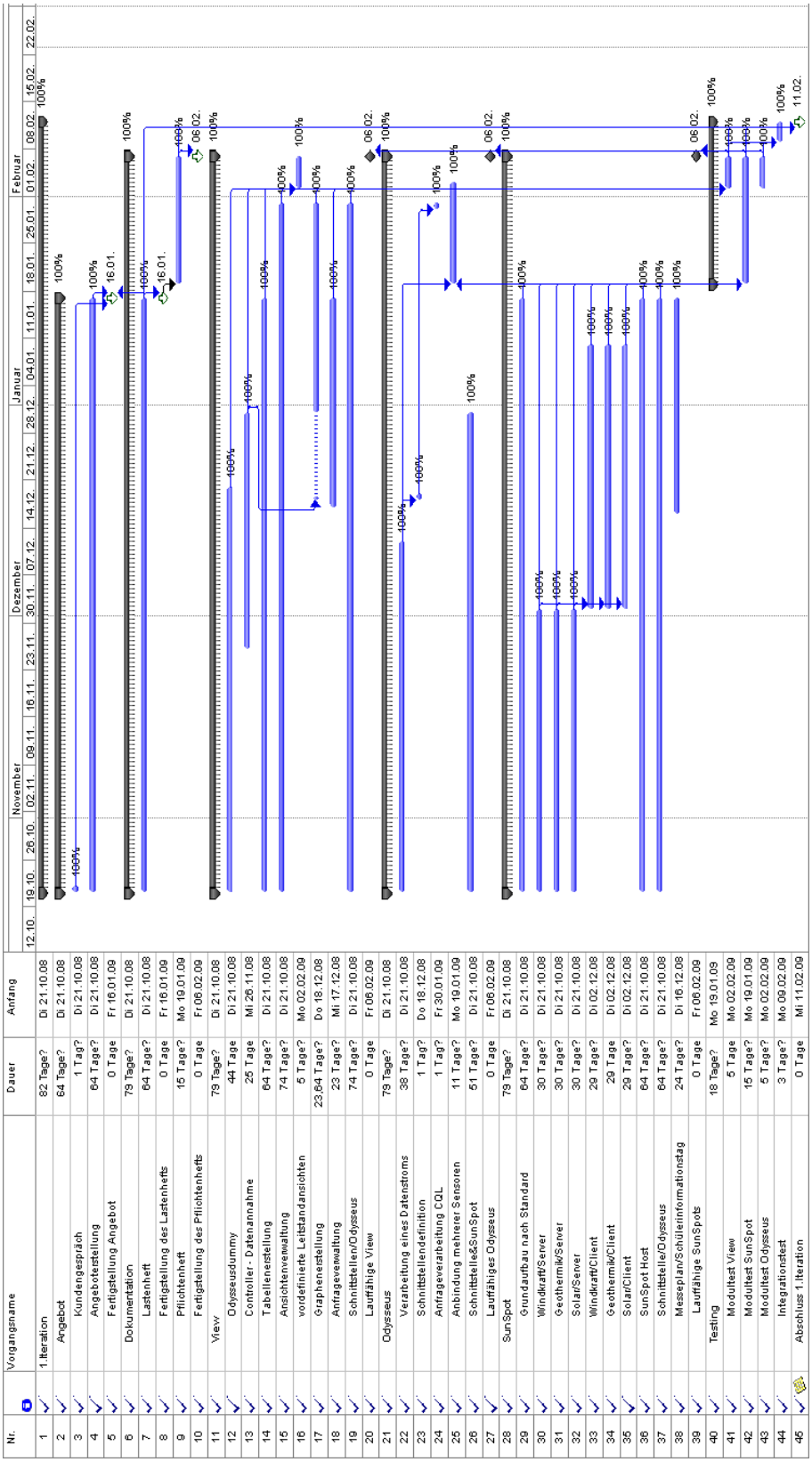


Abbildung 7.2: Projektplan der 1. Iteration

7.3 Zweite Iteration

7.3.1 Neue Gruppeneinteilung

Durch den in der ersten Iteration nicht betrachteten Part der 3D-Welt und die Übergabe der Projektleitung wurden die einzelnen Gruppen neu aufgeteilt um die Arbeitskraft optimal zu verteilen. Die neue Aufteilung sieht wie folgt aus:

Projektleiter und **Qualitätsmanager**: Christian Hülsen
Stellvertretender Projektleiter: Marcel Sommerlatt
Team View:

- Simon Flandergan (Teamleiter)
- Tobias Witt

Team 3D-View:

- Dennis Geesen (Teamleiter)
- Gunnar Schulte-Loh
- Marcel Sommerlatt

Team Odysseus:

- Wolfgang König
- Kai-Andre Pancratz (Teamleiter)
- Jan Vornberger

Team SunSpot:

- Jan-Ole Brode
- Ulf Truderung (Teamleiter)

7.3.2 Meilensteinbetrachtung

Meilensteine	Termin	Status
3D-View	26.08.2009	Abgeschlossen
View 2. Iteration	26.08.2009	abgeschlossen
Odysseus 2. Iteration	26.08.2009	abgeschlossen
SunSpots 2. Iteration	26.08.2009	abgeschlossen
Abschluss zweite Iteration	26.08.2009	abgeschlossen

Tabelle 7.2: Tabelarische Übersicht der Meilensteine

In der obigen Tabelle ist eine Zusammenstellung der Meilensteine inklusive ihrer Termine und dem entsprechenden Erledigungsgrad.

Anhand der Meilensteine ist eine grobe Gliederung der Projektgruppe zu erkennen. Wir gliedern uns in vier Bereiche die jeweils von einer Gruppe bearbeitet werden.

Die zweite Iteration endete am 21.08.2009 mit einem Featurefreeze, alle bis dahin noch nicht begonnenen Features wurden nicht mehr weiter bearbeitet. Zu diesem Zeitpunkt waren bereits alle geforderten Funktionen fertiggestellt, so das nur zusätzliche Features nicht bearbeitet worden sind.

Die Zeit bis zum Projektabschluss wird jetzt für das Testen des Programms sowie die daraus resultierenden Bugfixes und das Erstellen des Abschlussberichtes benötigt.

7.3.3 Fortschritte weiterer Arbeitspakete

Kritische Werte (View)

Als kritische Werte werden Grenzen bezeichnet, deren Über- oder Unterschreitung eine Meldung hervorrufen soll. Diese Werte lassen sich für jedes Element einstellen und werden in einer Alarm Bar und einem Event Viewer dargestellt. Damit ist dieses Packet zu 100 % abgeschlossen.

Erstellung von Graphen (View)

Zusätzlich zu den bereits bestehenden Graphentypen wurden Balken- und Tortendiagramme eingefügt. Des weiteren ist es jetzt möglich, die Quellen der Graphen zu bestimmen und nur einzellne Werte anzuzeigen. Dieses Arbeitspaket wurde mit 100% abgeschlossen.

Erstellung von Topologien (View)

Eine Topologie zeigt in einer 2D-Darstellung die Position von Kraftwerken und die Verbindungen unter diesen an. Die Topologie lässt sich aus bestehenden Welten der 3D-Welt automatisch erstellen. Durch die kurze Zeit, die für dieses Paket blieb, werden die Kanten mit der von Draw2D gegebenen Technologie geroutet, was zur Folge haben kann, das sich Kanten und Knoten überschneiden. Dieses haben wir in Kauf genommen um.

Interface zwischen Odysseus und View

Das Interface wurde komplett neu gestaltet. Die Erstellung von Datenströmen erfolgt nun über einen Webservice per Dialog in der View. Die Ausgabe der Daten geschieht über eine Socketverbindung. Diese Methode erwies sich als schnell und stabil und war somit die offensichtlich richtige Entscheidung. Damit ist dieses Packet zu 100 % abgeschlossen.

Anlegen von Kraftwerken (3D-View)

In der 3D-Welt müssen Kraftwerke angelegt werden können, diese sollen dann mit einem Statement verbunden werden und auf die Daten dieses Statements reagieren. Hierzu wurden Modelle der gewünschten Kraftwerke erstellt. Diese lassen sich über einen Dialog in die 3D-Welt einfügen und mit einem Statement verbinden. Dieses Arbeitspaket ist zu 100% Fertig.

Anlegen von Kabelleitungen (3D-View)

Zur Visualisierung des Stromflusses wurden Kabel in die 3D-Welt eingefügt, die zwischen den Kraftwerken und statischen Modellen verlegt werden können. Auch diese Leitungen lassen sich mit einem Statement verbinden und reagieren mit Farbwechsel auf kritische Werte. Damit ist dieses Packet zu 100 % abgeschlossen.

Anlegen von statischen Modellen (3D-View)

Zur optischen Aufwertung der 3D-Welt wurden einige statische Objekte eingefügt, die sich nicht direkt steuern lassen, aber zum Beispiel als Verbraucher fungieren können. Auch dieses Arbeitspaket ist zu 100% fertig.

Speichern der Welt (3D-View)

Damit eine Welt nicht ständig neu erstellt werden muss, soll die Anordnung der Objekte auf der Welt und die zugehörigen Statements, soweit vorhanden, gespeichert und wieder geladen werden können. Damit ist dieses Packet zu 100 % abgeschlossen.

Archiving (Odysseus)

Die Tupel der Datenströme sollen gespeichert werden um später wieder abgerufen zu werden. Dieses erfolgt durch einen Dialog in der View. Die Datenströme werden dann direkt beim Eingang in Odysseus abgegriffen und in einer Datenbank gespeichert. Dieses Arbeitspaket ist zu 100% abgeschlossen.

Weitere CQL-Statements (Oysseus)

Die Unterstützung weiterer CQL-Statements wie JOINs oder die Angabe von Zeitintervallen wurde zu 100 % fertiggestellt.

Leistungsoptimierung (Odysseus)

Durch Optimierungen am Code wurde die Leistung von Odysseus optimiert und damit die Hardwareanforderungen minimiert.

Virtuelle Kraftwerke (SunSpot)

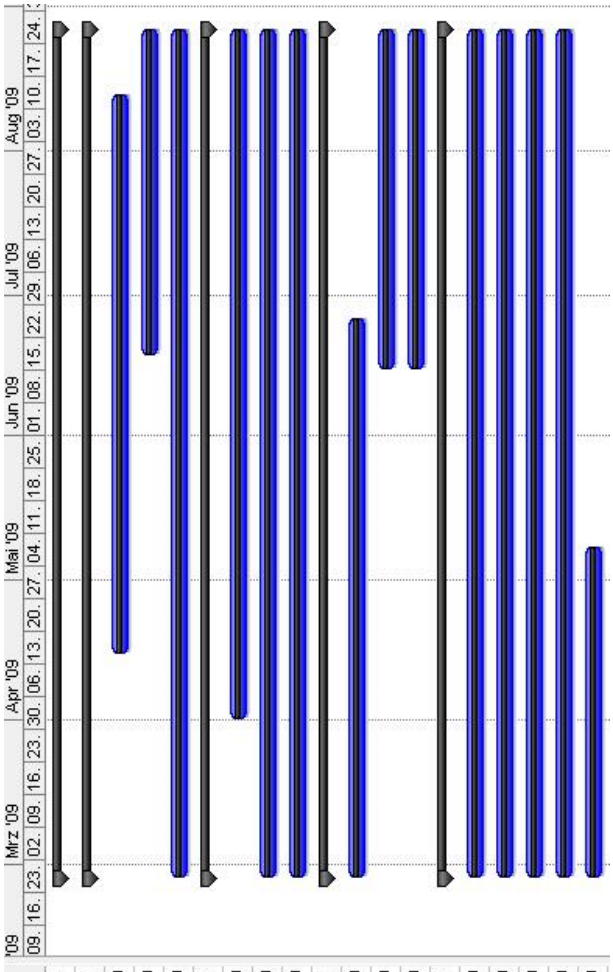
Virtuelle Kraftwerke reagieren für Odysseus und View genau so wie mit SunSpots simulierte Kraftwerke. Aus der Tatsache, das wir nicht jedes angezeigte Kraftwerk mit SunSpots simulieren können, ergibt sich die Notwendigkeit dieser Virtualisierung. Dieses Arbeitspaket ist zu 100%abgeschlossen.

Senden nach Standard (SunSpot)

Auch für die Geothermal- und Solaranlagen wurden bestehende Standarts umgesetzt, so das die Daten der SunSpots jetzt alle Standartkonform gesendet werden.

Präsentator (SunSpot)

Um dem Zuschauer auf Messen einen Eindruck von der Arbeitsweise der SunSpots zu geben und eine optische ansprechende Darstellung zu bieten, wurde für jedes simulierte Kraftwerk eine Form der Präsentation geschaffen. Hier werden die Sensoren der SunSpots gezielt mit Daten versorgt. Der Präsentator ist zu 100 % fertiggestellt.



Vorgangsname	Dauer	Anfang	Ende
2.Iteration	**130 Tage?**	**Do 26.02.09**	**Mi 26.08.09**
SunSpot	**130 Tage?**	**Do 26.02.09**	**Mi 26.08.09**
Virtuelle Kraftwerke	86 Tage?	Mi 15.04.09	Mi 12.08.09
Präsentator	50 Tage?	Do 18.06.09	Mi 26.08.09
Senden nach Standart	130 Tage?	Do 26.02.09	Mi 26.08.09
Odysseus	**130 Tage?**	**Do 26.02.09**	**Mi 26.08.09**
Archiving	106 Tage?	Mi 01.04.09	Mi 26.08.09
weitere CGL-Statements	130 Tage?	Do 26.02.09	Mi 26.08.09
Leistungsoptimierung	130 Tage?	Do 26.02.09	Mi 26.08.09
View	**130 Tage?**	**Do 26.02.09**	**Mi 26.08.09**
weitere Graphen	86 Tage?	Do 26.02.09	Do 25.06.09
Kritische Werte	53 Tage?	Mo 15.06.09	Mi 26.08.09
Topologie	53 Tage?	Mo 15.06.09	Mi 26.08.09
3D-View	**130 Tage?**	**Do 26.02.09**	**Mi 26.08.09**
Kraftwerke	130 Tage?	Do 26.02.09	Mi 26.08.09
Kabelleitungen	130 Tage?	Do 26.02.09	Mi 26.08.09
Statische Modelle	130 Tage?	Do 26.02.09	Mi 26.08.09
SpeichernLaden	130 Tage?	Do 26.02.09	Mi 26.08.09
Interface	51 Tage?	Do 26.02.09	Do 07.05.09

Abbildung 7.3: Projektplan der 2. Iteration

7.4 Test- und Dokumentenphase

7.4.1 Meilensteinbetrachtung

Meilensteine	Termin	Status
Testphase	24.09.2009	Abgeschlossen
Dokumentenphase	30.09.2009	Offen

Tabelle 7.3: Tabelarische Übersicht der Meilensteine

In dieser Phase konzentriert sich alles auf das Testen und Dokumentieren der erstellten Software. Neue Features werden nicht mehr programmiert.

Dadurch, das dieses Dokument selber in der Dokumentenphase erstellt wird, ist der Meilenstein „Dokumentenphase“ natürlich noch nicht abgeschlossen.

7.4.2 Fortschritte weiterer Arbeitspakete

Integrationstests (Test)

Die Integrationstests liefen ab dem Abschluss der Implementierung wöchentlich. Hier wurden einzellne kleinere Schwachstellen festgestellt, die jedoch schnell behoben werden konnten. Insgesammt deckten die Tests keine größeren Probleme auf.

Funktionstest (Test)

Der Funktionstest wurde vom Qualitätsmanager am 11.09.2009 durchgeführt und ergab keine schwerwiegenden Fehler. Ein weiterer Test wird kurz vor der Abgabe stattfinden.

Leistungstest (Test)

Der Leistungstest wurde vom Qualitätsmanager am 11.09.2009 zufriedenstellent durchgeführt.

Endbericht (Dokumententation)

Die Erstellung des Endberichtes ist zu diesem Zeitpunkt weit vortgeschritten, es fehlen nur noch einzelne Punkte.

Installationshandbuch (Dokumententation)

Das Installationshandbuch enthält detailierte Informationen, wie die einzelnen Programmteile zu installieren sind.

Benutzerhandbuch (Dokumententation)

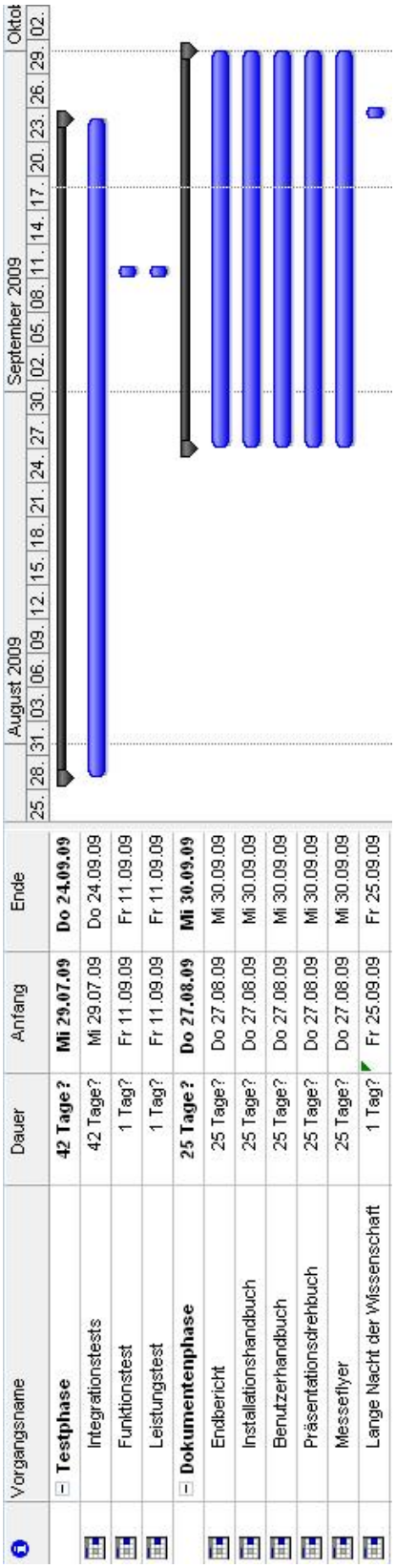
Das Benutzerhandbuch erklärt dem zukünftigen Benutzer anhand einer bebilderten Anleitung die einzelnen Funktionen des Programms.

Präsentationsdrehbuch (Dokumententation)

Für die Präsentation des Programms auf Messen oder sonstigen Vorführungen wurde ein Präsentationsdrehbuch angefertigt, welches dem Präsentierenden einen genauen Ablauf aufzeigt, mit dem der gesamten Funktionsumfang des Programms vorgeführt werden kann.

Messeflyer (Dokumententation)

Um den Besuchern auf Messen einen kurzen Überblick über die Software zu geben, wurde ein Messeflyer erstellt, der das Programm in einer kurzen Übersicht erklärt. Für weitere Fragen stehen dann die Betreuer vor Ort zur Verfügung.



Vorgangsname	Dauer	Anfang	Ende
Testphase	**42 Tage?**	**Mi 29.07.09**	**Do 24.09.09**
Integrationstests	42 Tage?	Mi 29.07.09	Do 24.09.09
Funktionstest	1 Tag?	Fr 11.09.09	Fr 11.09.09
Leistungstest	1 Tag?	Fr 11.09.09	Fr 11.09.09
Dokumentenphase	**25 Tage?**	**Do 27.08.09**	**Mi 30.09.09**
Endbericht	25 Tage?	Do 27.08.09	Mi 30.09.09
Installationshandbuch	25 Tage?	Do 27.08.09	Mi 30.09.09
Benutzerhandbuch	25 Tage?	Do 27.08.09	Mi 30.09.09
Präsentationsdrehbuch	25 Tage?	Do 27.08.09	Mi 30.09.09
Messeflyer	25 Tage?	Do 27.08.09	Mi 30.09.09
Lange Nacht der Wissenschaft	1 Tag?	Fr 25.09.09	Fr 25.09.09

Abbildung 7.4: Projektplan der Test- und Dokumentenphase

Kapitel 8

Fazit

Da es für ein solch großes Projekt schwierig ist, ein zusammenfassendes Fazit zu geben, wurde dieser Teil in verschiedene Kapitel aufgeteilt.

8.1 Thematik Datenstrommanagement und regenerative Energien

Datenstromtechnologie gewinnt immer mehr an Relevanz, da es in immer mehr Bereichen zu großen Datenmengen kommt, die möglichst ohne Verzögerung erfasst und abgefragt werden müssen. Hier steht mit Odysseus ein Framework bereit, das diese Abfragen auf Datenströme mittels einer deklarativen Anfrage ermöglicht.

Im Bereich der Energieerzeugung wird vermehrt auf regenerative Energien zurückgegriffen. Dieses geschieht zum Großteil aus Umweltschutzgründen. Aber auch die Entwicklung von kostengünstigen Verfahren zur Energiegewinnung spielen eine Rolle. Ein Problem hierbei ist die Verteilung vieler kleiner Anlagen auf eine große Fläche. Es besteht kein einzelnes Kraftwerk zur Überwachung sondern viele kleine Anlagen, die jedes einzeln und im Verband überwacht werden müssen. Auch hiermit setzt sich Earth Wind and Fire auseinander, indem es Anfragen für einzelne Kraftwerke oder eine Aggregation von Datenströmen mehrerer Kraftwerke ermöglicht.

8.2 Ablauf der Projektgruppe

Durch den frühen ersten Präsentationstermin auf dem Schüler Informations Tag der Informatik am 12.02.2009 war die Gruppe gezwungen die Entwurfsphase zu kürzen und zügig mit der Programmierung anzufangen. Dieses hatte zur Folge, dass sich einige Designentscheidungen nach der ersten Iteration als schlecht herausstellten. So konnten beispielsweise die nötigen Datenmengen nicht durch RMI übertragen werden.

Andererseits kamen diese Erkenntnisse erst im Verlauf der Programmierung hervor und wären bei einem typischen Softwareentwurf nicht aufgefallen. Die Änderungen hätten also in jedem Fall erfolgen müssen, nur stand uns jetzt mehr Zeit dafür zur Verfügung.

Nach dem Besuch bei BTC und der Vorführung der Prins Software kamen einige neue Anregungen zusammen, die wir in die Software übernommen haben. Leider war der Termin sehr spät in der laufenden Projektgruppe, so das für einige Funktionen nicht mehr ausreichend Zeit für eine zufriedenstellende Implementierung zur Verfügung stand.

8.3 Ausblick

Da die Software als Demonstrator entwickelt wurde, ist ein Einsatz im professionellen Bereich, etwa zur Überwachung realer Energieerzeugungsanlagen, im Moment nicht vorstellbar. Dazu fehlen unter anderem eine Möglichkeit zur Steuerung dieser Anlagen und eine exaktere Benutzerverwaltung.

Zudem haben wir die Sensordaten in ein IEC-konformes XML-Dokument verpackt, jedoch hält sich kaum ein Hersteller von Energieanlagen an diesen Standard in der Reinform. In diesen Standards gibt es viele offene Felder, die von den Herstellern frei mit Daten befüllt werden können. Das hat zur Folge, dass bei einer Nutzung von Earth Wind and Fire für jedes Model einer Energieerzeugungsanlage ein eigener Wrapper geschrieben werden muss, um die Daten untereinander kompatibel zu gestalten.

Als Demonstrator kann die Software auch auf andere Szenarien angepasst werden. Dazu müssen die Datenquellen auf dieses neue Szenario angepasst und das XSD-Schema entsprechend geändert werden. Die möglichen Anfragen an

Odysseus bleiben dieselben, womit auch die Tabellen und Graphen der RCP-View unangetastet bleiben können. Für die 3D-Welt müssen entsprechend des neuen Szenario neue Modelle erstellt und eingepflegt werden.

Kapitel 9

Installationshandbuch

9.1 Einführung

Dieses Dokument erläutert die Installation sämtlicher Teilelemente, die für die Ausführung der Präsentationssoftware Earth, Wind and Fire nötig sind.

Hierbei wird davon ausgegangen, das sowohl eine Java Laufzeitumgebung der Version 6.0 oder höher installiert und Eclipse in der Version 3.5 oder höher vorhanden ist.

9.2 Kraftwerkssimulation

Die Simulation von Kraftwerken kann virtuell oder durch die Erfassung von Echtdaten mithilfe von SunSPOTs erfolgen.

9.2.1 Installation der SunSPOTs

Um die SunSpots mit den entsprechenden Anwendungen auszustatten, wird das SunSPOT SDK in Version „blue“ benötigt. Dieses lässt sich durch den SunSPOT Manager installieren und verwalten(`http://www.sunspotworld.com/SPOTManager/index.html`. Desweiteren wird eine Installation von Apache Ant vorrausgesetzt. Das Projekt der Anwendung für die LN-simulierenden SunSPOTS kann im SVN unter svn/apps/SpotClient/LN/trunk ausgecheckt werden, das Projekt für die LD-simulierenden SunSPOTS unter svn/apps/SpotClient/LD/trunk. Im jeweiligen Stammverzeichnis kann der Build-Prozess durch „ant jar-app“ gestarted werden und durch „ant jar-deploy“ auf einem per USB verbundenen SunSPOT installiert werden.

Zum Betrieb wird weiter die SpotHost Anwendung benötigt, welche im SVN unter svn/apps/SpotHost/trunk zu finden ist. Bei dieser handelt es sich ebenfalls um ein SunSPOT-Projekt, da Sie eine SunSPOT-Basisstation verwendet, um den kabellosen SunSPOTs eine Schnittstelle zur Anwendung anzubieten. Die Anwenung wird mit ant host-run gestartet.

9.2.2 Installation ohne SunSPOTs

Stehen keine SunSpots und/oder Messdaten zur Verfügung, so kann auf den virtuellen SpotHost zurückgegriffen werden. Er kann unter „svn/apps/VirtualSpotHost/trunk“ ausgecheckt und als reguläre Java-Anwendung gestartet werden.

9.3 Odysseus

Die Schnittstelle zwischen den installierten SunSpots und der View ist Odysseus. Odysseus besteht hierbei aus dem eigentlichen Odysseus-Framework und einer sogenannten OdysseusApp, welche für das Starten der eigentlichen Mittelschicht verantwortlich. Des Weiteren kümmert sich diese Mittelschicht, um das Aufzeichnen von Datenströmen. Die permamenente Datenspeicherung erfordert eine MySQL 5.1 Installation.

9.3.1 Installation von Odysseus

Um Odysseus zu installieren, müssen folgende Ordner aus dem SVN ausgecheckt werden:

- Interfaces
- OdysseusApp
- Unser Odysseus Branch als „ODYSSEUS_OURBRANCH“

Wichtig dabei ist, dass unser Branch vom Odysseus-Framework als „ODYSSEUS_OURBRANCH“ ausgecheckt wird. Damit ist Odysseus lauffähig. Gestartet werden kann Odysseus mit der Ausführung von „OdysseusServer.java“ im OdysseusApp Verzeichnis.

9.3.2 Archiving

Das Archiving setzt eine MySQL 5.1 Installation voraus. Der MySQL Server und Odysseus müssen nicht auf dem gleichen Server installiert sein. Allerdings müssen die Zugangsdaten zur der MySQL Instanz in der Datei „Archive-Manager.java“ eingestellt werden. Es müssen keine speziellen Datenbanken oder Tabellen angelegt werden, diese legt Odysseus in der jeweiligen MySQL Instanz selber an.

Prinzipiell ist es möglich auch eine andere Datenbank als MySQL - z.B. PostgreSQL - zu verwenden. Hierfür muß dann allerdings der korrekte Treiber JDBC Treiber in der getInstance() Methode des ArchiveManagers eingestellt werden. Die Verwendung alternativer Datenbanken wurde allerdings nicht getestet. b

9.4 View

Die View wurde als Eclipse Rich Client Platform Anwendung realisiert und ist dementsprechend abhängig von der Eclipse Umgebung. Es gibt nun zwei Möglichkeiten die Anwendung auszuführen. Zum einen kann die Anwendung als Plugin in eine bestehende Eclipse 3.5 Umgebung integriert werden und zum anderen ist es möglich die Anwendung als Eclipse Product zu exportieren.

9.4.1 Installation als Plugin

Die Installation der View als Eclipse Plugin kann in zwei einfachen Schritten durchgeführt werden. Im ersten Schritt werden die Abhängigkeiten des Plugins aufgelöst. Dazu müssen das Draw2d-Plugin, mindestens Version 3.5.1, aus dem Graphical Editing Framework (GEF), sowie das LWJGL-Feature, mindestens Version 2.1.0, installiert werden. Die Plugins können beide über den „Install New Software ..“ Wizard der Eclipse Umgebung installiert werden. Die entsprechenden Seiten sind: http://lwjgl.org/update/ und http://download.eclipse.org/tools/gef/updates/milestones/. Nähere Details zur Installation von Plugins über den Eclipse Wizard können der Eclipse Dokumentation entnommen werden.

Sind die beiden Plugins erfolgreich installiert, so können im zweiten Schritt die beiden Plugins „ewafView“ und „EwafJars“ installiert werden. Dazu müssen die beiden Archive nur in das „dropins“ Verzeichnis der Eclipse Installation kopiert werden.

Nach einem Neustart der Eclipse Umgebung kann nun die View gestartet werden. Dazu muss die „Earth Wind and Fire“-Perspektive geöffnet werden. Diese ist im Menü „Window“ unter dem Eintrag „Open Perspective“, „Other...“ zu finden.

9.4.2 Installation als Product

Ein Eclipse Product ist abhängig vom verwendeten Betriebssystem. Dementsprechend muss es auf der Zielplattform generiert werden.

Zum Export wird der Quellcode der Anwendung, das Projekt, benötigt. Zunächst muss das Projekt in einer Eclipse Umgebung mit installiertem „Plug-in Development“ geöffnet werden. Im „Package-Explorer“ nun die „plugin.deploy.product“ Konfigurationsdatei öffnen. Zum Tab „Dependencies“ wechseln und die bestehenden Plugins entfernen (siehe Abbildung: 9.1; 1). Dann das Workset „Ewaf“ selektieren (siehe Abbildung: 9.1; 2,3) und die benötigten Plugins bestimmen lassen (siehe Abbildung: 9.1; 4). Damit ist die Konfiguration für die Zielplattform abgeschlossen und das Product kann exportiert werden (siehe Abbildung: 9.1; 5). Im folgenden Dialog nur noch den Zielordner angeben (siehe Abbildung: 9.2) und den Export abschliessen.

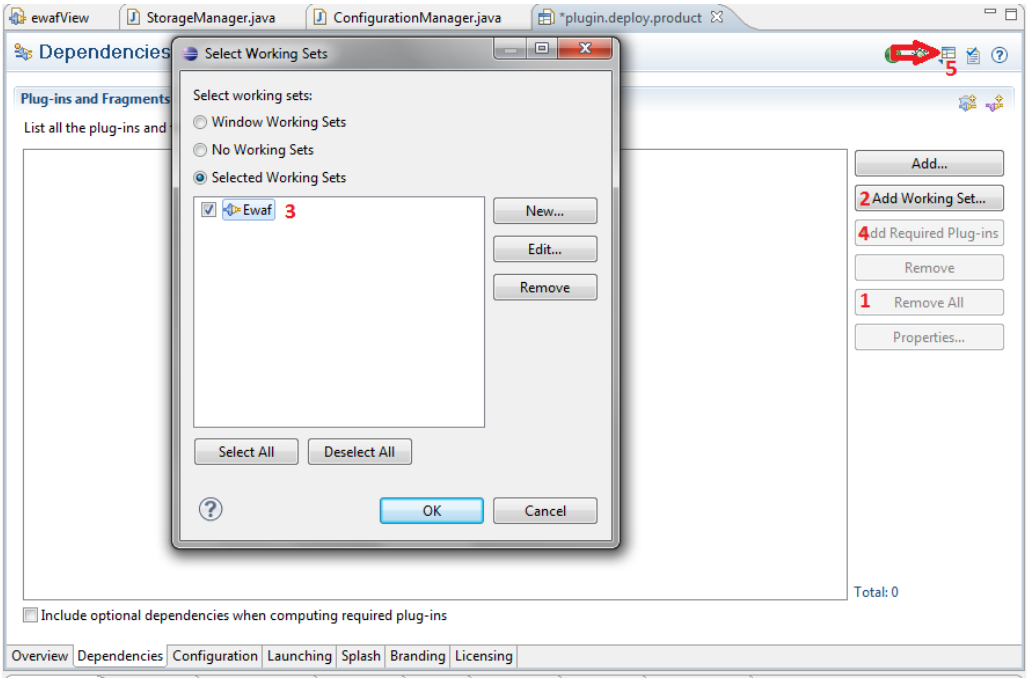


Abbildung 9.1: Product Konfiguration, Abhängigkeiten

Im Zielordner befindet sich nun eine Eclipse Konfiguration mit einer ausführbaren Eclipse Datei, die nur noch gestartet werden muss.

9.5 3D View

Die View Komponente des Projekts (siehe Kapitel 9.4) ermöglicht die Visualisierung der Datenströme. Eine anschauliche 3D-Welt ist ebenfalls enthalten. Diese 3D-View benötigt keine weitere Installation, da sie als Teil der View-Komponente mitgeliefert wird. Anforderungen für die 3D-Darstellung an das Computer-System kann dem Benutzerhandbuch entnommen werden. Besondere zusätzliche Konfigurationen und die Erweiterung der 3D-View sind im Folgenden beschrieben.

9.5.1 Hinzufügen neuer Modelle

Die 3D-View erlaubt es weitere Modelle hinzuzufügen. Dies erfordert ein erneutes Kompilieren des Quelltextes und steht somit nicht in einem Eclipse-Plugin oder -Produkt zur Verfügung.

Alle Modelle müssen im COLLADA-Format sein. Soll beispielsweise ein neues Modell *Neues Haus* hinzugefügt werden, so muss dazu ein Ordner *neues_haus* innerhalb des Ordners *models/collada/* erstellt werden. Das eigentliche Modell *neues_haus.dae* muss dabei wiederum in einem Ordner *models* liegen. Das Modell hat demnach den relativen Pfad *models/collada/neues_haus/models/neues_haus.dae*. Abbildung 9.3 zeigt die Ordnerstruktur.

Um das entsprechende Modell in der 3D-View zu aktivieren muss es der Enumeration *StaticalTypes* hinzugefügt werden. Dieses befindet sich im Paket de.uniol.inf.ewaf.ewafview.view3d.modules.statical. Dort muss der Name des Modells in Großbuchstaben hinzugefügt werden, hier entsprechend *NEUES_HAUS*, wie Abbildung 9.4 zeigt.

Das Modell steht dann, nach dem das Projekt neu kompiliert wurde, als statisches Modell zur Verfügung und kann dort entsprechend ausgewählt werden.

9.5.2 Meta-Daten

In dem Heimatverzeichnis des Benutzers, der die View startet, befindet sich ein Ordner *.ewaf/jme/*. In diesem befinden sich die Dateien des Caches, die während der ersten Ausführung angelegt werden. Falls die COLLADA-Dateien der Modelle geändert wurden, so muss die entsprechende JME-Datei aus diesem Cache-Verzeichnis gelöscht werden.

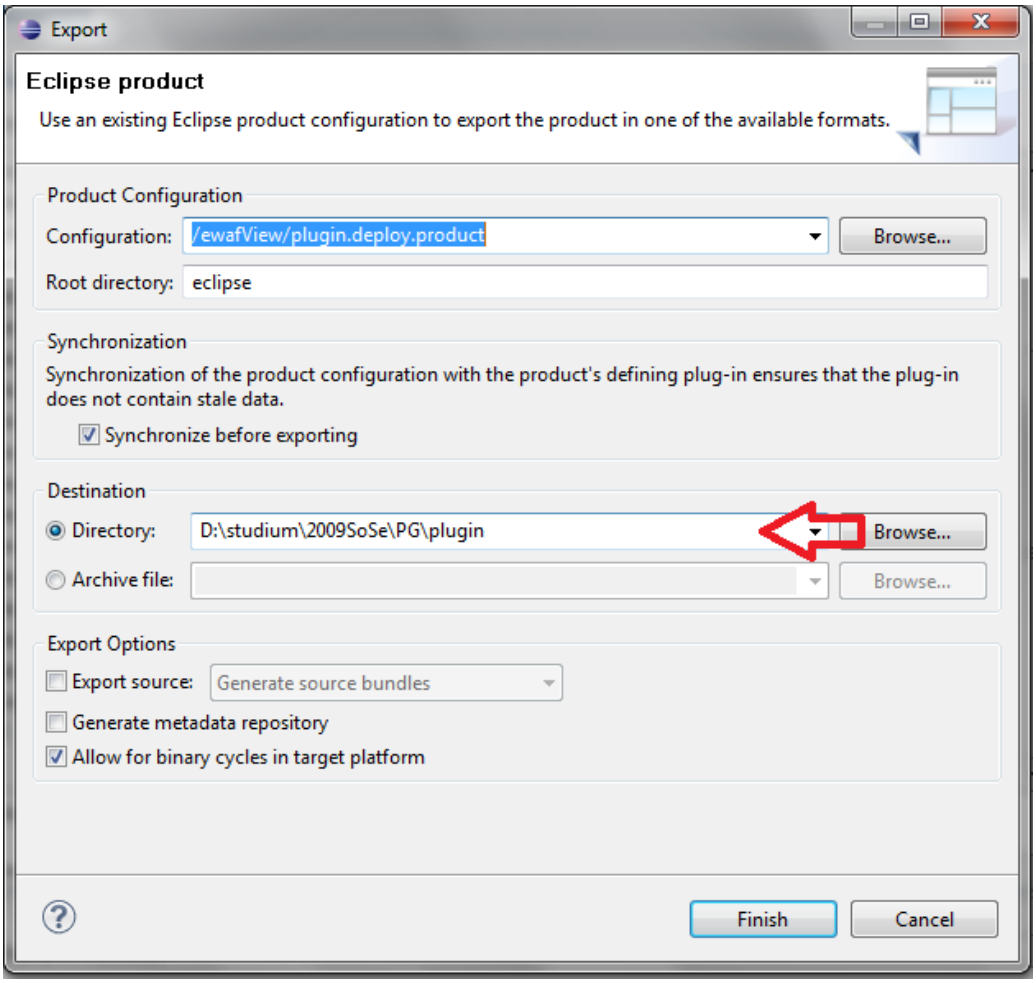


Abbildung 9.2: Product export

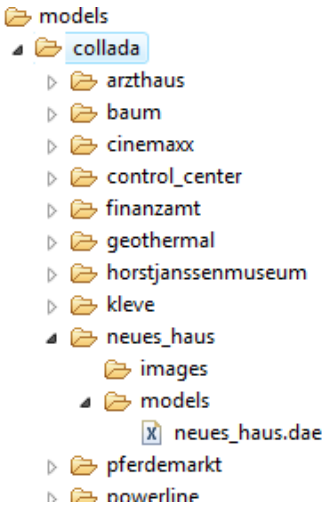


Abbildung 9.3: Ordnerstruktur der Modelle

```
     */
    PFERDEMARKT,

    /**
     * Road called Stauline in Oldenburg
     */
    STAULINIE,

    /**
     * A road of houses
     */
    STREET_OF_HOUSES,


    NEUES_HAUS
}
```

Abbildung 9.4: Erweitern des Enum StaticalTyp

Kapitel 10

Benutzerhandbuch

10.1 Einführung

Das Benutzerhandbuch erklärt dem Endnutzer die Handhabung und Bedienung der Software. Es dient als Nachschlagewerk für die Erklärung sämtlicher Funktionen.

10.2 Kraftwerkssimulation

Grafische Darstellungen sind notwendig, um Messebesuchern das Verständnis für die erstellte Software erleichtern zu können. Dazu ist es erforderlich, dass ein hoher Zusammenhang zwischen Objekten in der modellierter Welt und der realen Welt besteht. Um dies zu erreichen verwenden wir Sensoren, um Daten in der realen Welt zu erfassen, welche die Grundlage für die Vorgänge in der Modellwelt darstellen. Bei den eingesetzen Sensoren handelt es sich um SunSPOTs (Small Programmable Object Technology) von Sun Microsystems, Inc.

10.2.1 SunSPOTs

Die SunSpots von Sun bieten Embedded Entwicklungsumgebungen auf Java-Basis. Die SunSPOTs lassen sich grundsätzlich unterteilen in Basisstationen und Sensor-Knoten. Während die Basisstationen keine interne Stromversorgung haben, sind die Sensorknoten mit einem 750 mAh Lithium-Ionen-Akkumulator ausgestattet. Auch ist auf den Sensorknoten ein zusätzliches Erweiterungsboard angebunden, welches die Sensoren trägt, beziehnungsweise die Anschlüsse für externe Sensoren bereitstellt.

Zum Betrieb sind mindestens zwei Sensorknoten und eine Basistation notwending.

10.2.2 SPOTHost

Zum Starten des SPOTHosts wird das Java Runtime Environment 6.0[1] oder höher benötigt.

Ist der SPOTHost gestartet wartet er auf die Verbindung von SunSPOTs, welche eine LD (Logical Device) Funktion übernommen haben. Dies verläuft völlig autonom und bedarf keiner manuellen Einflussnahme. Im gleichen Verzeichnis wie der SPOTHost selbst wird die portlist.xml als Logdatei abgelegt. Hier werden laufende Server mit zugehöriger Portnummer gespeichert.

10.2.3 Virtual SPOTHost

Zum Starten des Virtual SPOTHost wird das Java Runtime Environment 6.0 oder höher benötigt.

Nach dem Start wird eine Übersicht der virtuellen Server angezeigt, siehe Abb. 10.1. Diese Übersicht besteht aus drei Spalten, welche die drei unterschiedlichen unterstützten Kraftwerkstypen Wind, Solar und Geothermal darstellen. Durch Klicken des Buttons „Start Plant“ unterhalb des jeweiligen Kraftwerktyps wird ein neuer virtueller Server diesen Typs gestartet. Er erscheint dann in der jeweiligen Liste gefolgt von einer Portnummer. Unter dieser Portnummer stellt dieser Serverdienst ab sofort simulierte Daten zur Verfügung. Zu Testzwecken können diese z.B. mit dem Netzwerkprotokoll *telnet* abgefragt werden. Falls Port bereits belegt sind wird eine andere Portnummer verwendet.

Die zur Verfügung gestellten Daten sind nach den IEC Standards 61400-25 und 61850-7 geformt.

[1] http://java.sun.com/javase/downloads/index.jsp

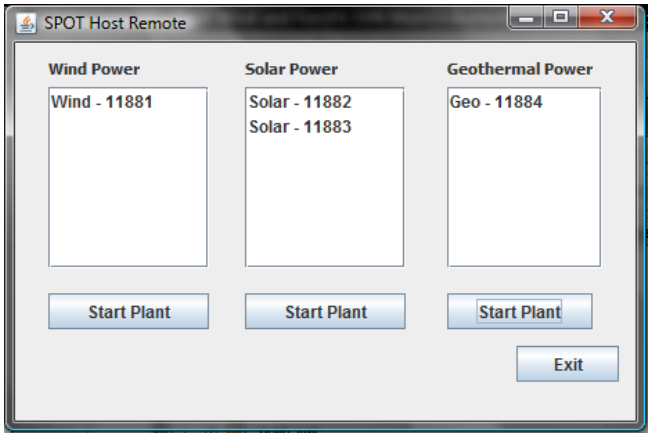


Abbildung 10.1: Virtual SPOTHost Screenshot

Farbe	Kraftwerkstyp
Blau	Windkraft
Grün	Geothermie
Gelb	Photovoltaik

Tabelle 10.1: Kodierung der Kraftwerkstypen

Der LD SunSpot

Auf dem SunSPOT muss der die SunSPOT LD Anwendung installiert sein. Nach dem Starten muss der Kraftwerkstyp eingestellt werden, welcher simuliert werden muss. Sobald der SunSpot eingabebereit ist leuchtet die LED 8 blau auf. Mit Hilfe des Tasters 1 kann nun zehn Sekunden lang durch die Typen durchgeschaltet werden. Zur Ermittlung des gewünschten Typs ist Tabelle 10.1 zu konsultieren. Der SunSpot beginnt nun periodisch per Broadcast auf Port 70 ein Radiogramm mit dem Integer 301 zu senden bis ein SpotHost auf Port 71 mit dem Integer 302 antwortet. An diesen Host sendet der LD-Spot ab diesem Zeitpunkt periodisch seine Daten.

Der LN SunSPOT

Auf dem SunSPOT muss der die SunSPOT LD Anwendung installiert sein. Nach dem Starten muss der Kraftwerkstyp eingestellt werden, welcher simuliert werden muss. Sobald der SunSpot eingabebereit ist leuchtet die LED 8 blau auf. Mit Hilfe des Tasters 1 kann nun der LN-Typ durchgeschaltet werden. Zur Ermittlung des gewünschten Types ist Tabelle 10.2.3 zu konsultieren. Der SunSpot beginnt nun periodisch per Broadcast auf Port 66 einen „Request for parent“ zu senden bis ein LD SunSPOT auf Port 65 antwortet. An die antwortende Adresse sendet der SunSPOT im Anschluss periodisch die von Ihm ermittelten Messwerte.

10.3 Odysseus

Die Main-Methode des Odysseus-Serverprozess befindet sich in der Klasse de.uniol.inf.ewaf.OdysseusServer. Diese kann ausgeführt werden, um den Serverprozess zu starten. Der Web Service steht dann nach kurzer Initialisierung auf

Farbe	LED-Zahl	LN-Typ
Blau	1	WTUR
Blau	2	WROT
Blau	3	WGEN
Grün	1	GTUR
Geld	1	SCEL
Rot	1	MMXN
Rot	2	XCBR

Tabelle 10.2: Kodierung der LN Typen

Fehlercode	Bezeichnung	Erläuterung
200	OK	Kein Fehler
400	PARSE_EXCEPTION	Eine Anfrage konnte nicht erfolgreicht geparst werden.
410	QUERYPARSE_EXCEPTION	wie oben
420	OPENFAILED_EXCEPTION	Das Ausführen einer Anfrage schlug fehl.
430	DATABASE_EXCEPTION	Es trat ein Fehler beim Zugriff auf die Datenbank auf.
440	IO_EXCEPTION	Beim Ausführen einer Anfrage trat ein Netzwerkfehler auf.
450	STREAM_NOT_FOUND_EXCEPTION	Es existiert keine Aufzeichnung für diesen Stream.

Tabelle 10.3: Fehlercodes des Odysseus-Serverprozesses

Port 8000 zur Verfügung.

Logausgaben werden in die Datei logs/odysseus_app.log geschrieben. Das Logging-Level kann von 'INFO' auf 'DEBUG' angehoben werden, indem die Datei bin/log4j.properties editiert wird.

Die Fehlercodes, die im Fehlerfall vom Web Service zurückgeliefert werden, sind in Tabelle 10.3 aufgeführt.

10.4 View

Die View ist eine Eclipse Rich Client Platform Anwendung zur Darstellung von Datenströmen. Die Daten können sowohl in Form von Tabellen oder Graphen dargestellt werden. Im Folgenden werden die Funktionen der Applikation vorgestellt.

10.4.1 Anforderungen der RCP-View

Hardware

- Pentium 4 1.0 GHz oder höher
- 512 MB RAM
- 60 MB freier Festplattenspeicher (Eclipse Product)
- 230 MB freier Festplattenspeicher (Eclipse Plugin)
- VGA-kompatible Grafikkarte
- Breitbandverbindung (oder lokales Odysseus)

Software

Eclipse Product:

- Java SE Runtime Environment 6

Hinweis: Für den Export der Anwendung als Eclipse Product, müssen zusätzliche Anforderungen erfüllt werden. Näheres dazu kann im Installationshandbuch eingesehen werden.

Eclipse Plugin:

- Java SE Runtime Environment 6
- Eclipse 3.5 Galileo
- Draw2D 3.5.1 oder höher
- LWJGL 2.1.0 oder höher

Unterstützte Betriebssysteme

Die folgende Liste unterstützter Betriebssysteme, wurde auf Grund von Tests mit zur Verfügung stehenden Rechnern erstellt. Es ist möglich, dass andere Betriebssysteme auch kompatibel sind. Dies gilt insbesondere für den Betrieb der Anwendung als Plugin.

Eclipse Product:

- Microsoft Windows XP oder höher
- Debian Squeeze, Kernel 2.6 oder ähnliche Distribution

Eclipse Plugin:

- Microsoft Windows XP oder höher
- Debian Squeeze, Kernel 2.6 oder ähnliche Distribution
- Mac OS X 10.6.1 oder höher

10.4.2 Starten der RCP-View

Zur Installation und erstmaligem Einrichten siehe Installationshandbuch.

10.4.3 Verbindung zu Odysseus herstellen

Nach dem Start der Anwendung wird automatisch versucht, eine Verbindung mit Odysseus mit der zuletzt benutzten IP aufzubauen. Wurde die Verbindung erfolgreich hergestellt, wird in der Statusleiste *Odysseus* und dessen IP angezeigt. Kann keine Verbindung hergestellt werden, gibt es eine *WebServiceException* Fehlermeldung (siehe Abbildung 10.2), die mit OK bestätigt werden kann.

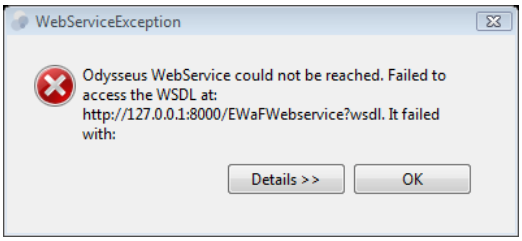


Abbildung 10.2: Odysseus kann nicht erreicht werden

Daraufhin öffnet sich ein Servereinstellungen-Dialog (siehe Abbildung 10.3), bei dem nun die neue IP einer Odysseusinstanz angegeben werden muss. Durch *Apply* wird dann die Verbindung aufgebaut.

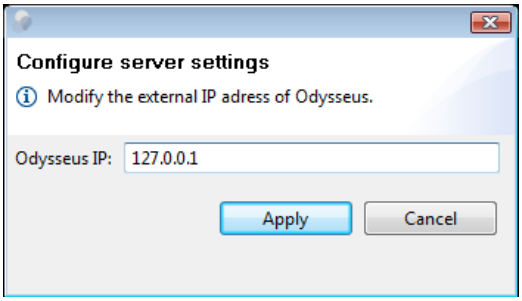


Abbildung 10.3: Ändern der Odysseus Verbindung

Tipp: Auch im laufenden Betrieb kann die Odysseusinstanz über den Servereinstellungen-Dialog gewechselt werden. Dieser kann über das Symbol *Configure server settings* geöffnet werden.

10.4.4 Anfrageverwaltung

Die Anwendung stellt verschiedene Dialoge bereit, um Quellen, im folgenden Streams genannt, hinzuzufügen, CQL-Anfragen zu installieren und aufgezeichnete Daten zu benutzen.

Anfragen verwalten

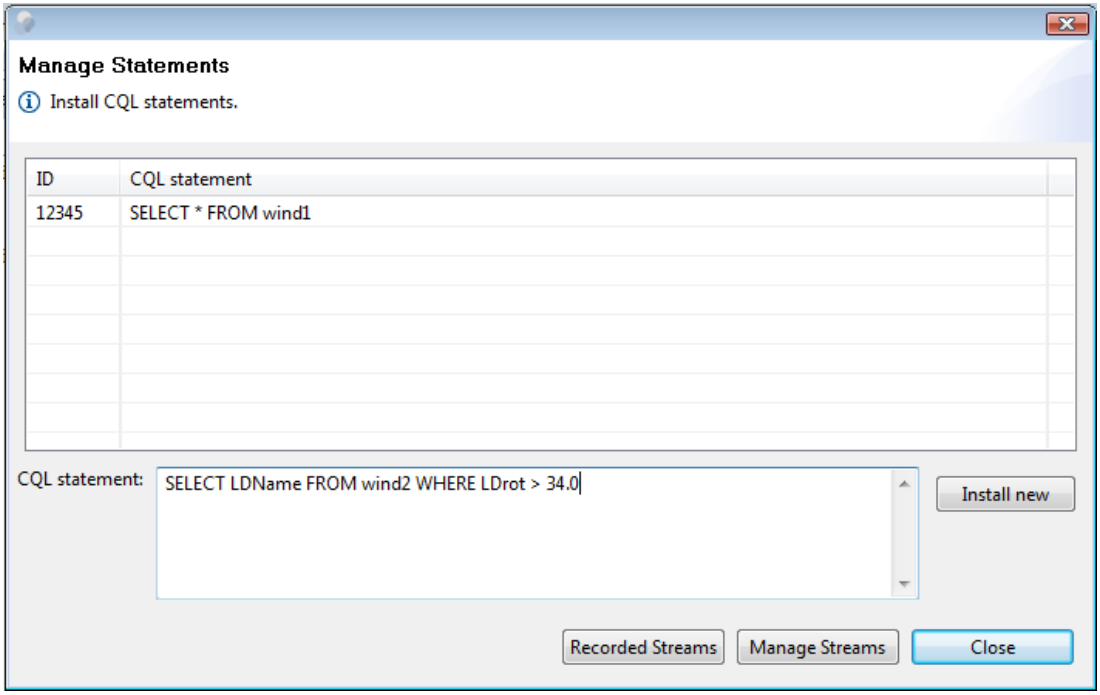


Abbildung 10.4: CQL-Anfragen verwalten

Über das Symbol *Add or remove Statements* wird der *Manage Statements* Dialog aufgerufen, mit dem CQL-Anfragen installiert werden können (siehe Abbildung 10.4). Es wird eine Liste mit bereits in Odysseus installieren Anfragen angezeigt. Eine neue Anfrage kann installiert werden, indem unter *CQL Anfrage* eine CQL-Anfrage eingegeben und mit *Install new* bestätigt wird. Es erfolgt eine Rückmeldung, dass die Anfrage installiert wurde. Tipp: Durch das Auswählen einer bereits installierten Anfrage wird die alte Anfrage in das Textfeld übernommen und kann so als Vorlage verwendet werden. Über *Manage Streams* kann der Dialog zum Verwalten von Streams geöffnet werden und durch *Recorded Streams* gelangt man zum Dialog für aufgezeichnete Streams.

Streams verwalten

Quellen bzw. Streams können über den *Manage Streams* Dialog hinzugefügt werden (siehe Abbildung 10.5). Es wird eine Liste der bereits in Odysseus eingerichteten Streams angezeigt. Um einen neuen Stream hinzuzufügen, muss ein beliebiger *Stream name* gewählt werden, die IP und der Port der Quelle angegeben werden und das XSD-Schema, welches das Format der Quelle definiert, ausgewählt werden. Durch *Add as new* wird dieser Stream dann hinzugefügt. Tipp: Durch das Auswählen eines bereits installieren Streams werden dessen Werte in die Eingabefelder übernommen und können so als Vorlage verwendet werden.

Das Aufzeichnen der Streamdaten kann für einen Stream gestartet und gestoppt werden, indem ein Stream aus der Liste ausgewählt wird und dann *Start recording* bzw. *Stop recording* gewählt wird. Aufgezeichnete Streams können dann über den *Recorded Streams* Dialog benutzt werden, der im nächsten Abschnitt erläutert wird.

Aufgezeichnete Streams benutzen

Wenn Daten eines Streams in Odysseus aufgezeichnet wurden, ist es über den *Recorded Streams* Dialog möglich, einen neuen Stream zu installieren, der die aufgezeichneten Daten wiedergibt (siehe Abbildung 10.6). Es wird eine Liste angezeigt, für welchen Stream aufgezeichnete Daten vorliegen und wann die erste und letzte Aufzeichnung von

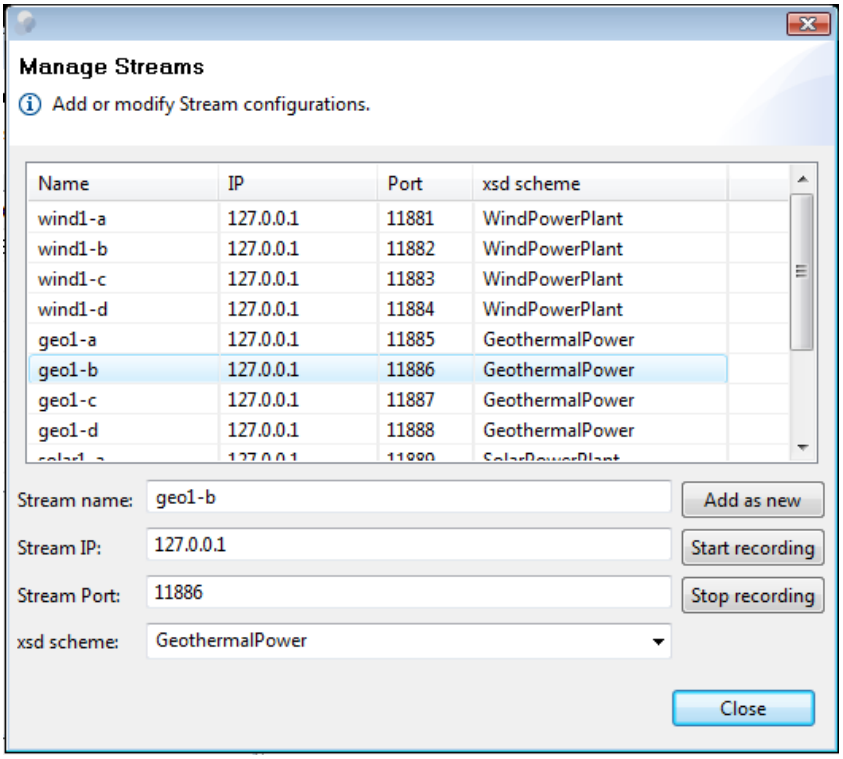


Abbildung 10.5: Streams verwalten

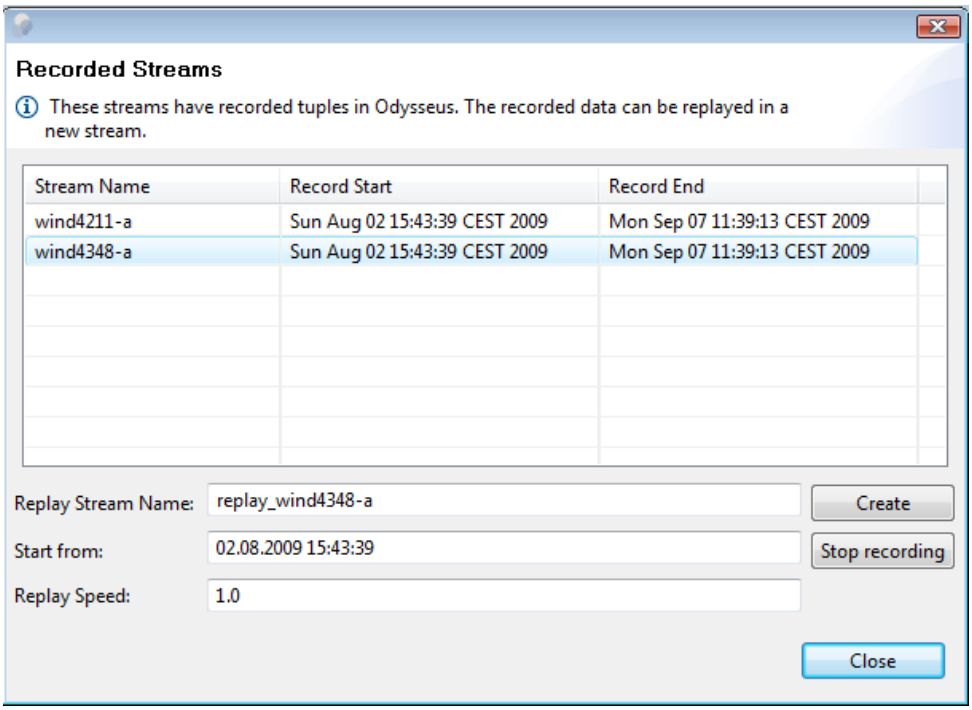


Abbildung 10.6: Aufgezeichnete Streams verwenden

Daten stattgefunden hat. Um diese Daten nun in einem neuen Stream wiederzugeben, muss ein Eintrag aus der Liste gewählt werden. Für den neuen Stream wird ein Name vorgeschlagen, der aber beliebig angepasst werden kann. Als Startzeitpunkt, bei welchen aufgezeichneten Daten begonnen werden soll, wird das Datum der ersten aufgezeichneten Daten vorgeschlagen. Dieses kann frei gewählt werden, dabei sollte allerdings beachtet werden, dass es durchaus

zwischen der ersten und letzten Aufzeichnung größere Zeitspannen geben kann, wo gar keine Daten aufgezeichnet wurden. Zusätzlich kann die Wiedergabegeschwindigkeit als Faktor bei *Replay Speed* erhöht oder verringert werden. Mittels *Create* wird der Wiedergabe-Stream dann erstellt und erscheint auch in der normalen Streamliste im *Manage Streams* Dialog und kann nun in Anfragen verwendet werden.

10.4.5 Perspektiven Management

Die View kann mehrere Perspektiven verwalten und erlaubt so den schnellen Wechsel zwischen verschiedenen Ansicht. So können die Beschränkungen durch die Größe des verwendeten Displays überwunden werden, indem die darzustellenden Daten in verschiedenen Perspektiven untergebracht werden und bei Bedarf gewechselt wird.

Jede Perspektive kann eine beliebige Anzahl an Tabellen, Graphen und Topologien enthalten. Näheres zu den Inhalten der Perspektive wird in den folgenden Kapiteln erläutert.

Perspektiven Manager

Nach dem Start der View wird direkt der Perspektiven Manager angezeigt (siehe Abbildung: 10.7). Dieser zeigt die bestehenden Perspektiven und Ordner an - beim ersten Start ist er selbstverständlich leer.

Neue Perspektiven und Ordner, zur Strukturierung, können über die Toolbar des Perspektiven Managers (siehe Abbildung: 10.7 : 1) oder über das per Rechts-Klick zu öffnende Kontextmenü (siehe Abbildung: 10.7 : 2) erstellt werden. Nach der Auswahl der Funktion wird ein Dialog zur Erstellung der neuen Perspektive/ des neuen Ordners

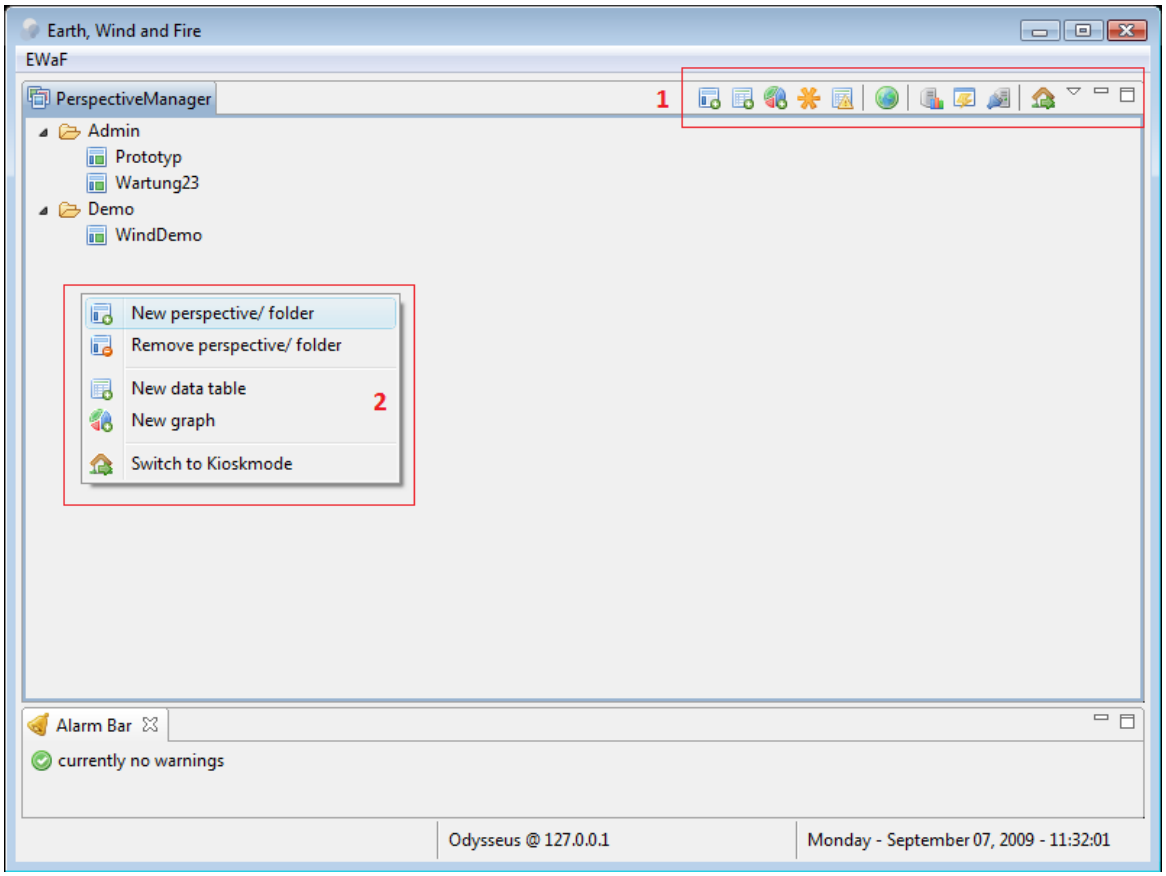


Abbildung 10.7: Perspektivenmanager ohne geöffnete Perspektive

geöffnet. Hier muss nun ein Name, sowie der Typ (Ordner/ Perspektive) angegeben werden (siehe Abbildung: 10.8). Das neue Elemente wird daraufhin unter dem aktuell selektiertem Element eingefügt. Tipp: soll eine Perspektive auf der obersten Ebene eingefügt werden, so muss die aktuelle Auswahl über *strg + Links-Klick* aufgehoben werden.

Die Struktur der Perspektiven kann auch nachträglich verändert werden. Dazu können Elemente, inklusive Unterelemente, verschoben werden. Dies wird über „Drag and Drop“ gelöst. Durch zusätzliches Halten von *strg* werden die Elemente kopiert.

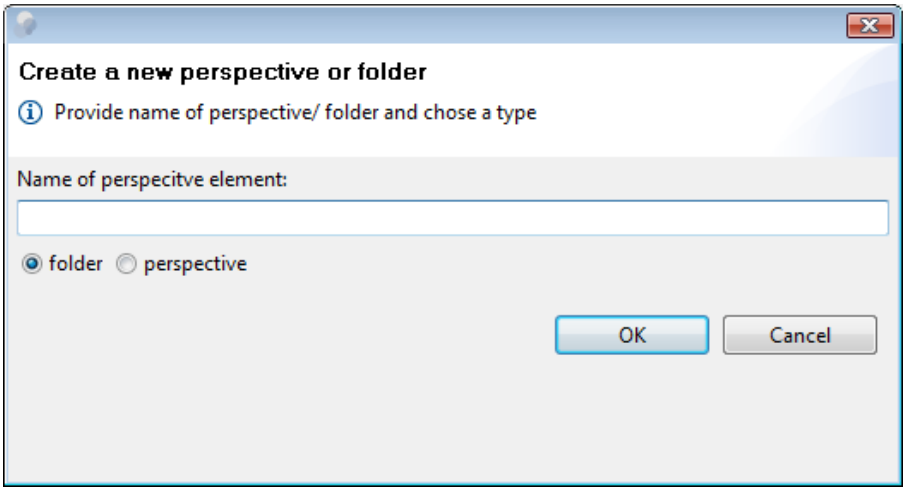


Abbildung 10.8: Neue Perspektive/ Ordner erstellen

Sollte der Bedarf bestehen Perspektiven oder Ordner zu löschen, kann über das Kontextmenü (siehe Abbildung: 10.7 : 2) das ausgewählte Element entfernt werden.

Perspektiven

Eine Perspektive wird über einen Doppelklick geöffnet. Ist dieser noch keine Tabelle, Graph, etc. zugeordnet, so wird eine leere Seite angezeigt (siehe Abbildung: 10.9). Sollen der Perspektive neue Elemente hinzugefügt werden,

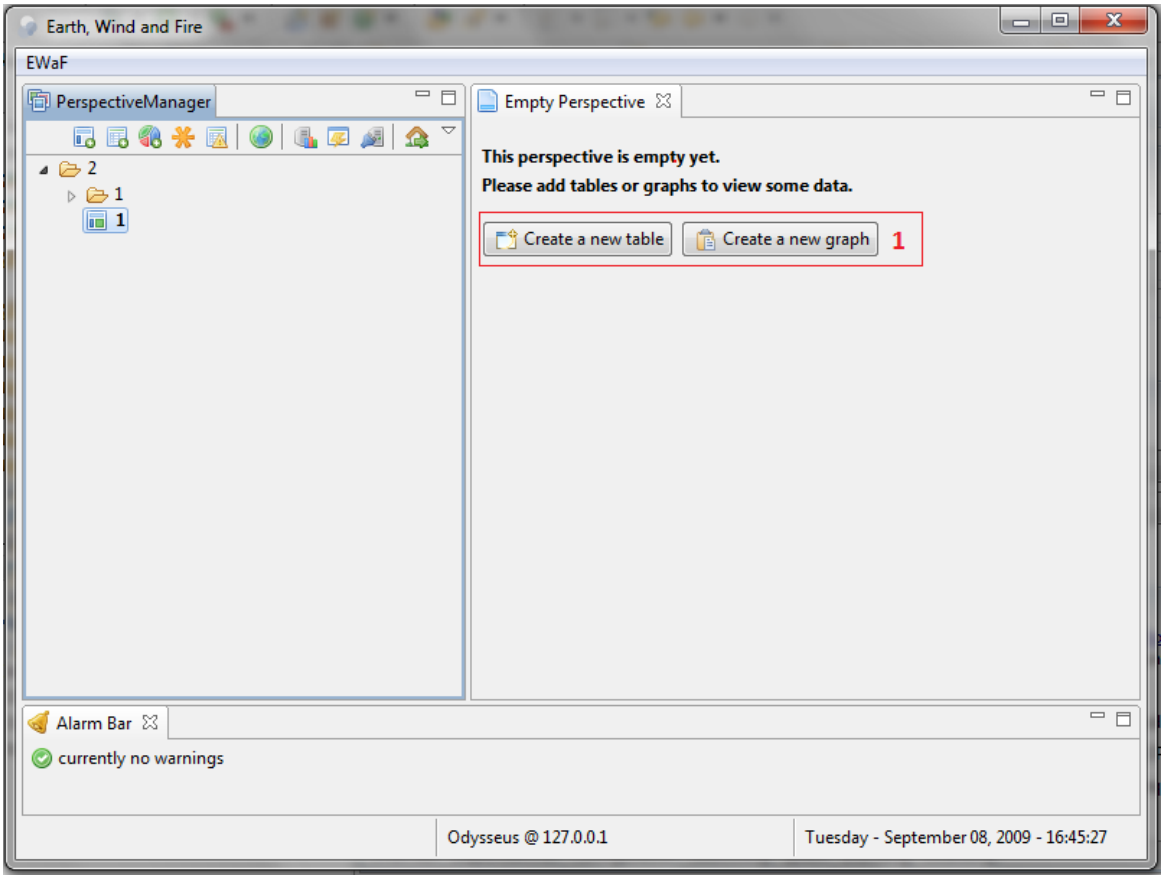


Abbildung 10.9: Leere Perspektive

so können über entsprechende Dialoge Graphen oder Tabellen, sowie Topologien hinzugefügt werden. Die Dialoge können über die Knöpfe der leere Seite (siehe Abbildung: 10.9 : 1), über das Kontextmenü oder die Toolbar (siehe Abbildung: 10.7 : 1,2) erreicht werden.

Jedes Element der Perspektive kann frei verschoben und skaliert werden. Zum Verschieben kann die Titelfläche angeklickt werden und an eine andere Stelle gezogen werden. Die Größe kann verändert werden, indem eine Kante angeklickt und verschoben wird. Zusätzlich können Element vom aktuellen Fenster gelöst werden, dazu muss zunächst das Kontextmenü des Elements per Rechts-Klick auf die Titelleiste geöffnet werden. Ein Klick auf „Detach“ löst das Element vom Hauptfenster und stellt es in einem eigenen Fenster da (siehe Abbildung: 10.10). Dies kann über die gleiche Funktion rückgängig gemacht werden.

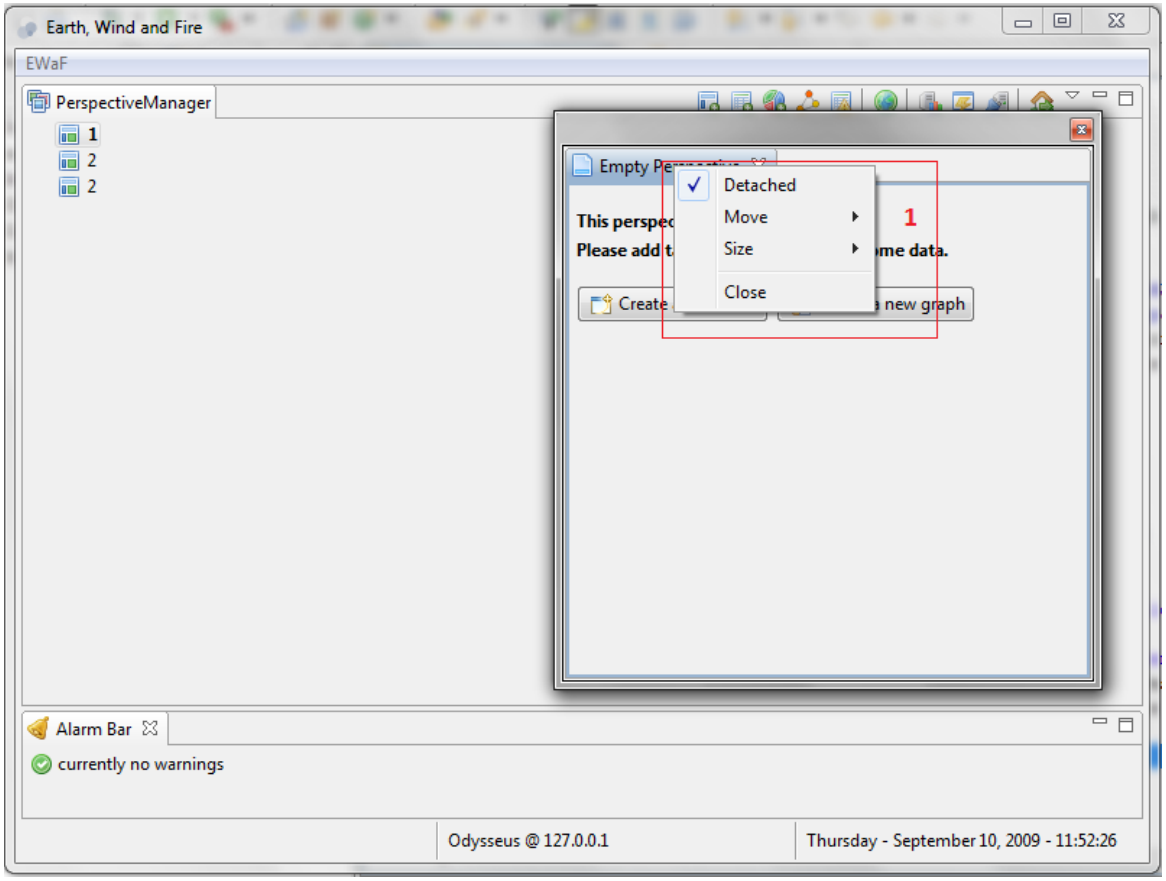


Abbildung 10.10: Element von der Anwendung abdocken

Der Wechsel zu einer neuen Perspektive wird durch Öffnen einer anderen Perspektive gelöst. Daraufhin werden die Elemente der aktuell Geöffneten geschlossen und die neuen geöffnet. Zusätzlich kann entschieden werden, ob Änderungen an der alten Perspektive gespeichert oder verworfen werden sollen.

10.4.6 Tabellen

Eine neue Tabelle kann erstellt werden, indem das Icon *Adds a new data table* benutzt wird. In dem geöffneten Dialog werden alle installierten Anfragen in einer Liste angezeigt (siehe Abbildung 10.11). Es muss ein Tabellenname gewählt werden und eine Anfrage aus der Liste selektiert werden. Über *Create* wird dann die neue Tabelle in der Perspektive erzeugt.

Die Tabelle besitzt für jedes Attribut der Anfrage eine Spalte und die Tupel werden auf eine Zeile abgetragen (siehe Abbildung 10.12). Das neuste Tupel wird unten angefügt und das älteste entfernt. Die Größe des Puffers kann in den *Buffer Settings* verändert werden. Ein sehr großer Puffer kann unter Umständen dazu führen, dass die Anwendung langsamer reagiert. Friert die Anwendung auf Grund eines zu großen Puffers ein, so reguliert das System die Puffergröße automatisch herunter. Die Tabelle scrollt automatisch nach unten zum neusten Element. Sobald die Scrollbar von Hand betätigt wird, wird nicht mehr automatisch nach unten gescrollt. Am Fuß der Tabelle ist eine Legende platziert, die die Hervorhebungen erläutert. Ein orangefarben marktierter Wert stellt eine Überschreitung der

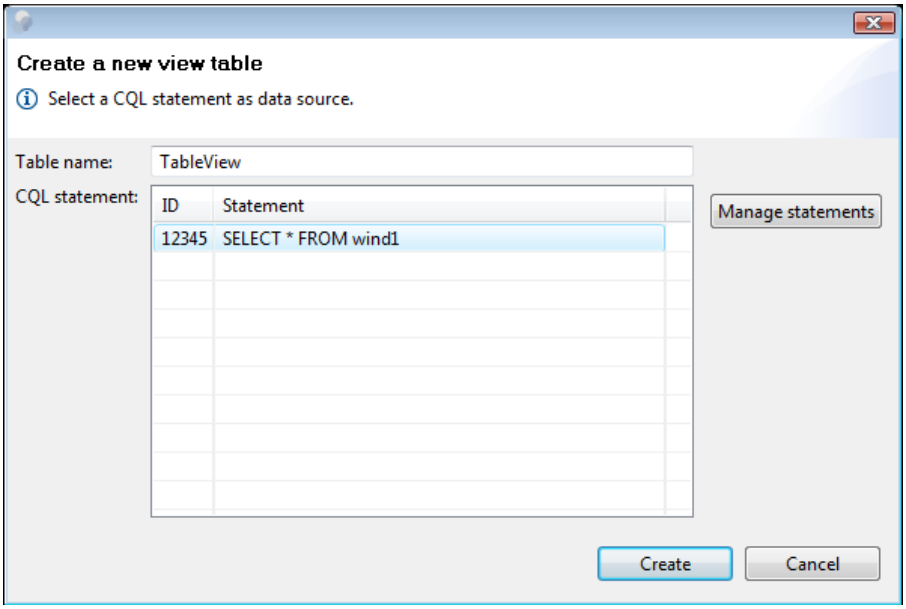


Abbildung 10.11: Eine neue Tabelle erstellen

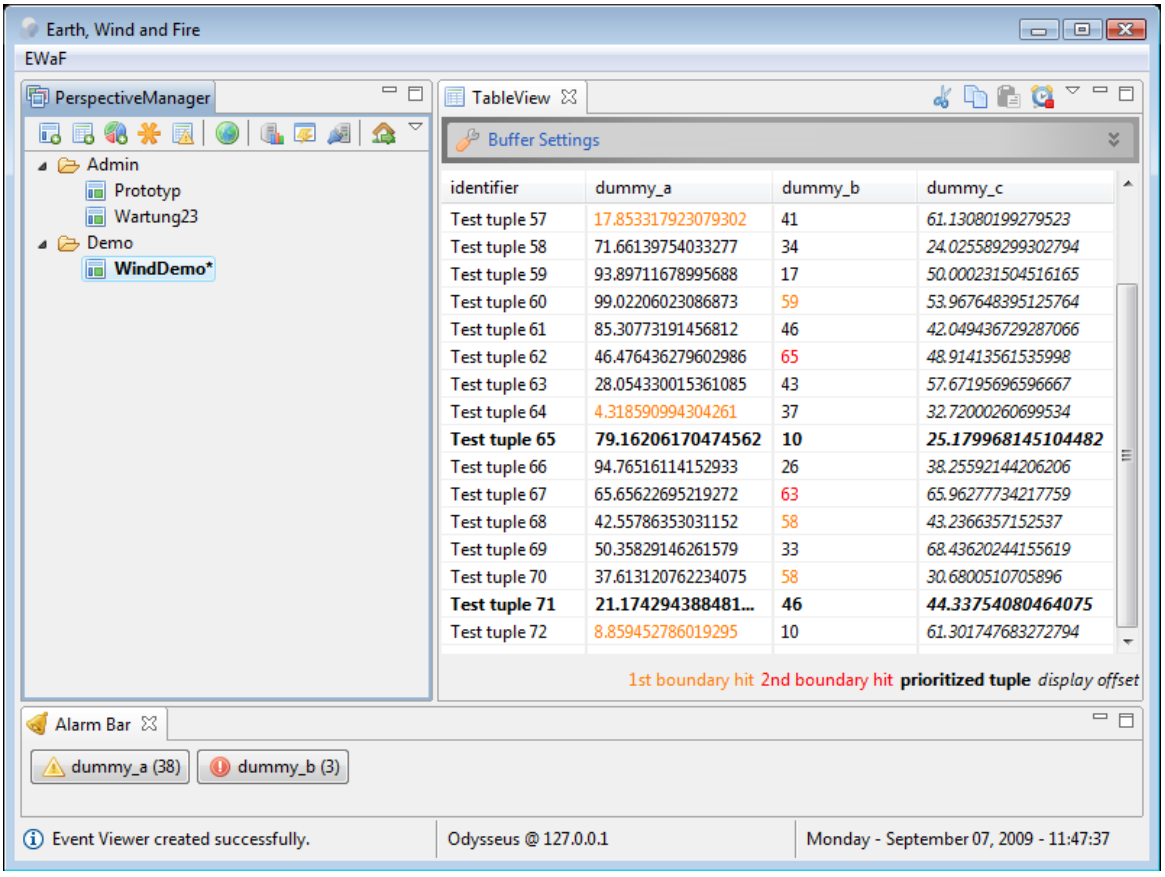


Abbildung 10.12: Tabelle mit verschiedenen Hervorhebungen

ersten Ober- bzw. Untergrenze dar, ein rot marktierter Wert hat die zweite kritische Grenze überschritten. Werte, die mit einem Freiwert verrechnet werden, sind kursiv dargestellt. Bei einer Zeile, dessen Werte fett markiert sind, handelt es sich um ein in Odysseus priorisiertes Tupel. Kritische Werte sind im Abschnitt 10.4.9 näher erläutert. Um die Einträge der Tabelle in Ruhe betrachten zu können, kann das Symbol *Freeze table* gewählt werden, so dass die

aktuelle Ansicht eingefroren wird. Durch einen Doppelklick auf eine Zeile können die Rohdaten des jeweiligen Tupels angezeigt werden.

10.4.7 Graphen

Graphen können wie zuvor erwähnt einer Perspektive hinzugefügt werden. Dazu wird ein Dialog geöffnet (siehe Abbildung: 10.13). Bevor der Graph erstellt wird, müssen nun Name, zugeordnetes Anfrage und Typ des Graphens

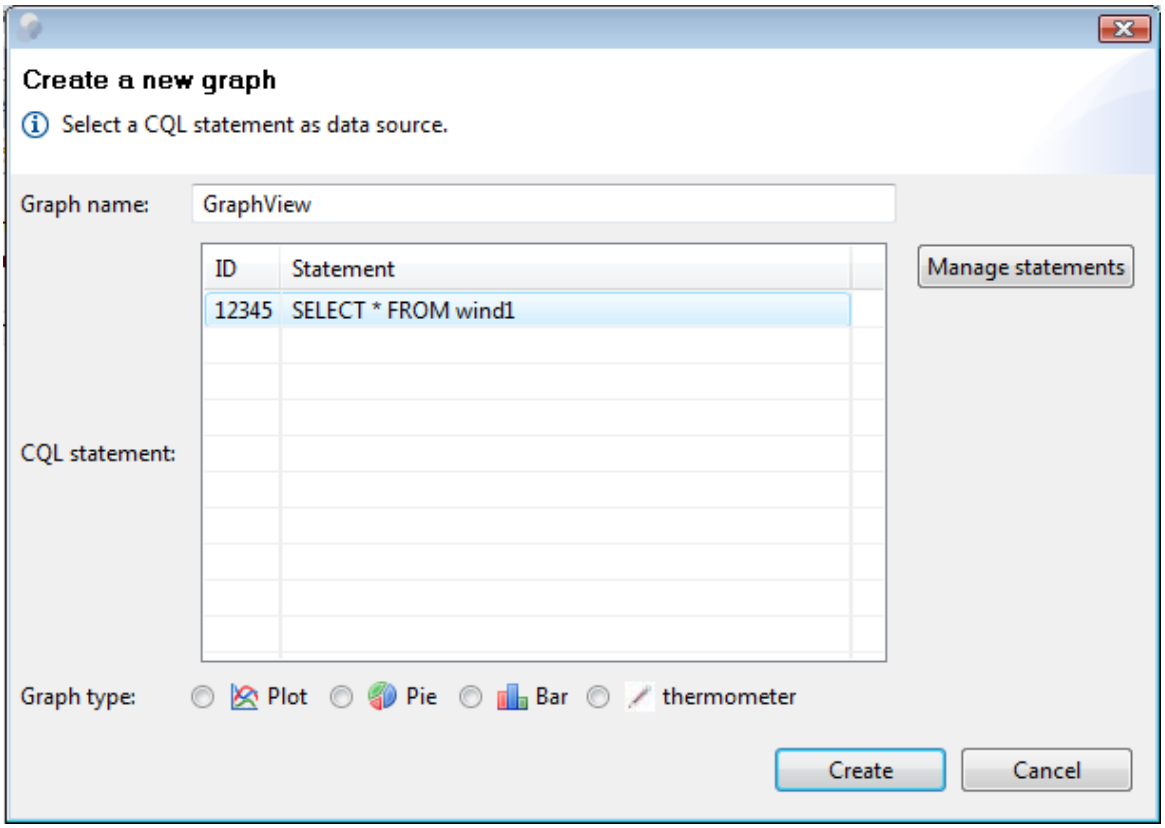


Abbildung 10.13: Dialog zur Erstellung eines Graphens

gewählt werden. Näheres zu den einzelnen Typen im Folgenden.

Die Graphen haben ein Aktualisierungsbeschränkung. Dies bedeutet, dass bei einem hohen Datendurchsatz jedes X,Y - Diagramm maximal einmal in der Sekunde neu gezeichnet wird und alle anderen Graphen alle fünf Sekunden aktualisiert werden.

X,Y - Diagramm

Wurde im Dialog als Typ „Plot“ gewählt, so wird ein X,Y - Diagramm angezeigt. Dieses zeigt auf der X-Achse Zeitstempel und auf der Y-Achse die zu diesen Zeitpunkt gemessenen Werte an. Je nach Einstellung werden die Messwerte mittels einer Linie verbunden.

Es können verschiedene Einstellungen bei einem X,Y - Diagramm vorgenommen werden. Zum einen kann der Puffer für diesen Graphen, wie bei der Tabelle (siehe Abschnitt: 10.4.6), verändert werden, um die Anzahl an Werten, die angezeigt werden können, zu erhöhen oder zu beschränken.

Zusätzlich gibt es noch ein weiteres Register für Einstellungen: *Sources and Graph Option*. Hier kann ausgewählt werden, welche Attribute der gewählten Anfragen angezeigt werden sollen (siehe Abbildung: 10.14 : 1). Achtung: diese Funktion soll nur dazu dienen kurzzeitig einzelne Linien auszublenden, um einige Werte besser betrachten zu können. Es sollten keine Anfragen verwendet werden, die ungenutzte Attribute haben, da dadurch unnötig Daten gepuffert werden und Speicher belegen.

Des Weiteren kann die Zeitspanne, die angezeigt werden soll angepasst werden (siehe Abbildung: 10.14 : 2). Dies kann ein feste Anzahl an Minuten oder eine automatische Anpassung sein. Letztere scrollt die X-Achse abhängig von der Anzahl an Werten.

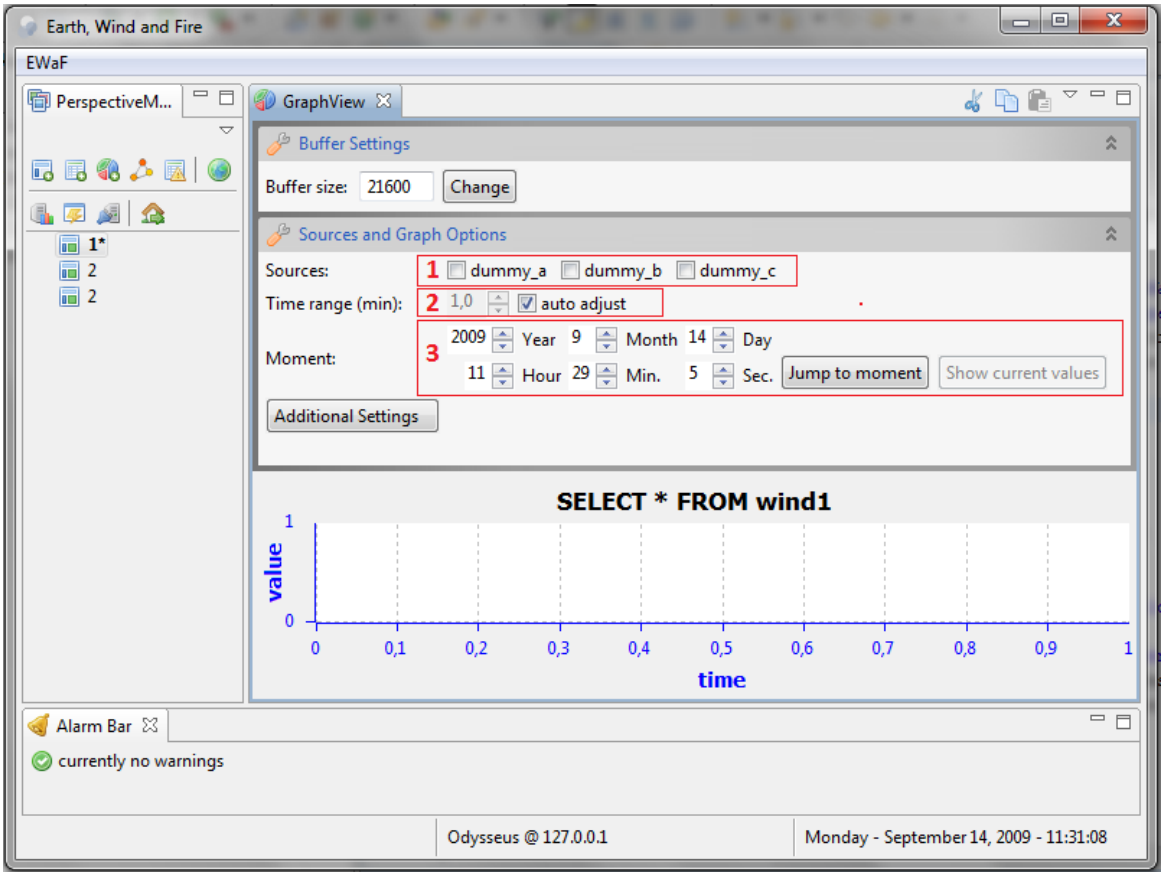


Abbildung 10.14: Einstellungen für X,Y - Diagramm

Zusätzlich zur Zeitspanne kann auch der angezeigte Zeitpunkt angepasst werden. In der Regel ist dies der aktuelle Zeitpunkt, es kann jedoch auch ein Zeitpunkt in der Vergangenheit gewählt werden. Dazu muss der entsprechende Zeitpunkt eingetragen werden (siehe Abbildung: 10.14 : 3) und *Jump to moment* gewählt werden. Danach werden Werte von diesem Zeitpunkt an bis zur zum Ende der Zeitspanne angezeigt. Sollen wieder aktuelle Werte statt des festen Zeitabschnitts angezeigt werden, muss nur *Show current values* gedrückt werden.

Der Graph kann auch optisch angepasst werden. Dazu muss zunächst der entsprechende Dialog geöffnet werden (siehe Abbildung: 10.15). Dieser ist über den Knopf *Additional Settings* erreichbar. Die Einstellungen sind abhängig vom gewählten Attribut. So muss in den allgemeinen Einstellungen zunächst eine Linie gewählt werden (siehe Abbildung: 10.15 : 1). Daraufhin werden die anderen Felder freigeschaltet. Falls die gewählte Linie einer anderen Y-Achse statt der Standard-Achse zugeordnet werden soll. So kann eine neue erzeugt werden und im entsprechenden Feld (siehe Abbildung: 10.15 : 2) der Linie zugeordnet werden. Zusätzliche Achsen können auch wieder entfernt werden. Achtung: dadurch werden Linien, die der zu löschenden Achse zugeordnet sind, wieder mit der Standard-Achse assoziiert.

Bei Bedarf kann Anti-Aliasing eingestellt werden. Dies glättet die Linien, erzeugt jedoch eine erhöhte CPU-Last. Daher sollte diese Funktion mit bedacht eingeschaltet werden.

Im Linienstil-Bereich kann gewählt werden, welche Symbole für die Messwerte der Linie verwendet werden, sowie in welcher Form sie verbunden werden.

Zusätzlich zum Stil können auch die verwendeten Farben eingestellt werden. Dabei können sowohl Symbol-, Linien- und Achsenfarbe verwaltet werden. Die Achse entspricht dabei jener Achse, die der Linie zugeordnet wurde (siehe Abbildung: 10.15 : 2).

Die Achse kann im folgenden Bereich noch weiter angepasst werden. So kann ihr ein Name vergeben werden, der die entsprechenden Daten passend beschreibt, sowie die Platzierung, links oder rechts, verändert werden.

Zuletzt kann noch die Position der Legende, die Zuordnung von Attribut zur Linie, verändert werden.

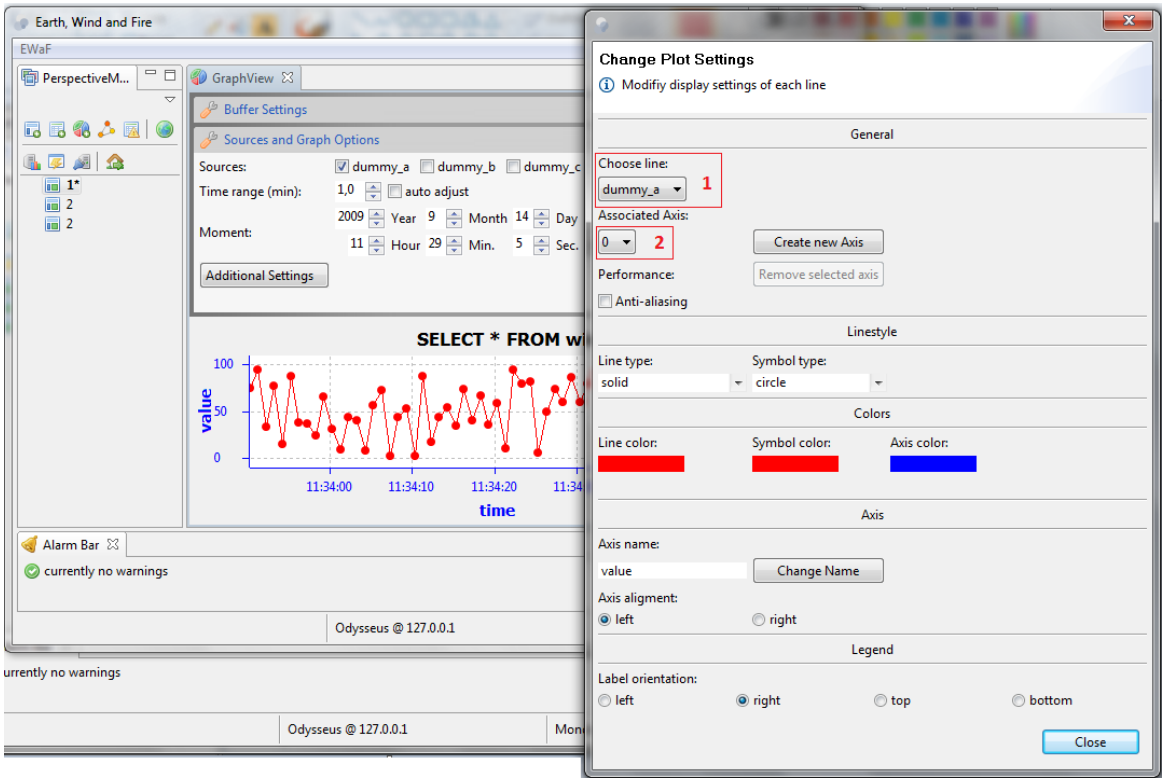


Abbildung 10.15: Zusätzliche Optionen für X,Y - Diagramm

Kuchendiagramm

Das Kuchendiagramm zeigt das Verhältnis zwischen den numerischen Attributen der gewählten Anfrage. Zusätzlich zu dem prozentualen Verhältnis zwischen den Werten kann auch ein Durchschnittswert des Attributs angezeigt werden. Dieser wird angezeigt, sobald der Mauszeiger über den Bereich des Kuchens gehalten wird.

Die Optionen bei diesem Diagramm beschränken sich auf die Puffer-Einstellungen (siehe Abschnitt: 10.4.6), sowie die Möglichkeit die verwendeten Farben zu verändern. Die Farbeinstellungen werden über den Knopf *Color settings* geöffnet. Dort sind alle Attribute mit einem eingefärbtem Feld angeordnet (siehe Abbildung: 10.16). Zur Änderung einer Farbe kann das Feld angeklickt werden, woraufhin sich ein Dialog zur Definition der neuen Farbe öffnet.

Balkendiagramm

Das Balkendiagramm zeigt Durchschnittswerte der Attribute der gewählten Anfrage. Die Anzahl der Werte, die zur Berechnung des Durchschnitts herangezogen werden, entspricht der Anzahl der Tupel im Puffer. Zur Anpassung der Berechnung muss entsprechend die Puffergröße verändert werden.

Über das Register *Sources and Graph Options* (siehe Abbildung: 10.17) können einige Optionen genutzt werden. So können Attribute ausgeblendet oder zwischen horizontalen und vertikalen Balken umgeschaltet werden. Zudem gibt es die Möglichkeit die Farben der Balken zu verändern (siehe Abschnitt: 10.4.7). Achtung: es werden nur die ausgewählten Balken im Farben-Dialog angezeigt.

Thermometerdiagramm

Das Thermometerdiagramm ist ein Graph zur Anzeige eines einzelnen Wertes. Im Gegensatz zu den anderen Graph-Typen ist es bei diesem Diagramm möglich kritische Werte anzuzeigen (siehe Abschnitt: 10.4.9). Das Thermometer wertet dabei die zweite untere Grenze und beide oberen Grenzen aus. Die oberen Grenzen werden durch einen orangen und roten Strich neben dem Thermometer angezeigt (siehe Abbildung: 10.18). Die zweite untere Grenze entspricht dem kleinsten Wert der Skala. Zusätzlich zu den Markierungen wird das Quecksilber des Thermometers entsprechend des Wertes eingefärbt. Werte, die keine der oben genannten Grenzen verletzen werden grün dargestellt. Werte oberhalb der ersten oberen Grenze und kleiner der zweiten oberen Grenzen werden orange gezeichnet. Alle Werte unterhalb der zweiten unteren Grenzen oder oberhalb der zweiten oberen Grenzen werden rot eingefärbt.

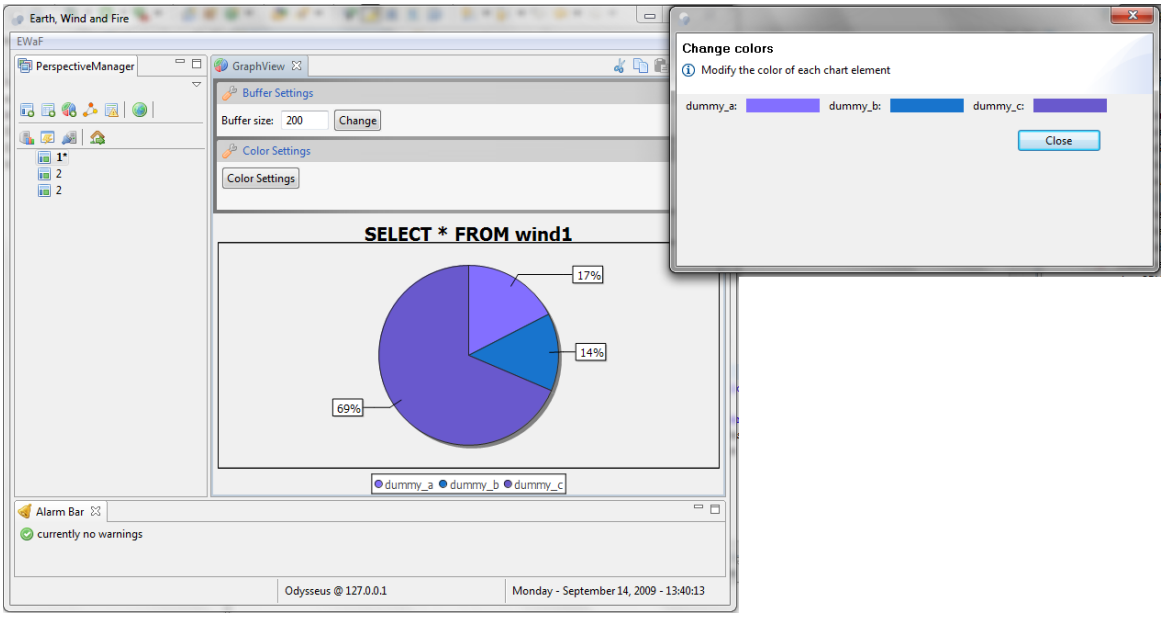


Abbildung 10.16: Kuchendiagramm inklusive Farbeinstellungen

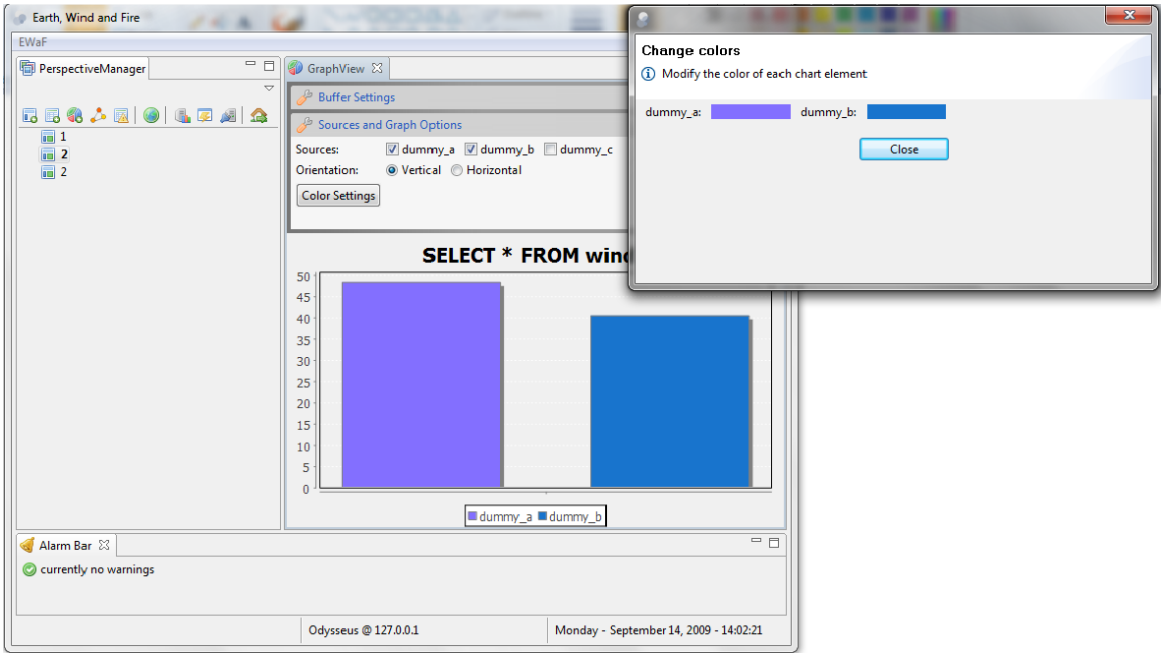


Abbildung 10.17: Balkendiagramm inklusive erweiterter Einstellungen

Die erste untere Grenze wird durch Beschränkungen des JFreeChart Frameworks ignoriert. Achtung: die angezeigten Werte sind Durchschnittswerte. Für eine möglichst akkurate Anzeige der Grenzüberschreitungen, sollte daher der Puffer möglichst klein sein.

Das darzustellende Attribut wird über die *Selected source* Auswahlbox ausgewählt. Die Daten werden bei der nächsten Aktualisierung gezeichnet.

10.4.8 Topologie

Die Topologie ist eine zwei-dimensionale Darstellung der 3D-Welt (siehe Abschnitt: 10.5). Jedes Kraftwerk der 3D-Welt wird durch einen Knoten und die Untergrund-Kabel werden durch Kanten zwischen den Knoten repräsentieren. Die Position der Kraftwerke entsprechen der Position in der 3D-Welt, die Kanten werden geroutet und können

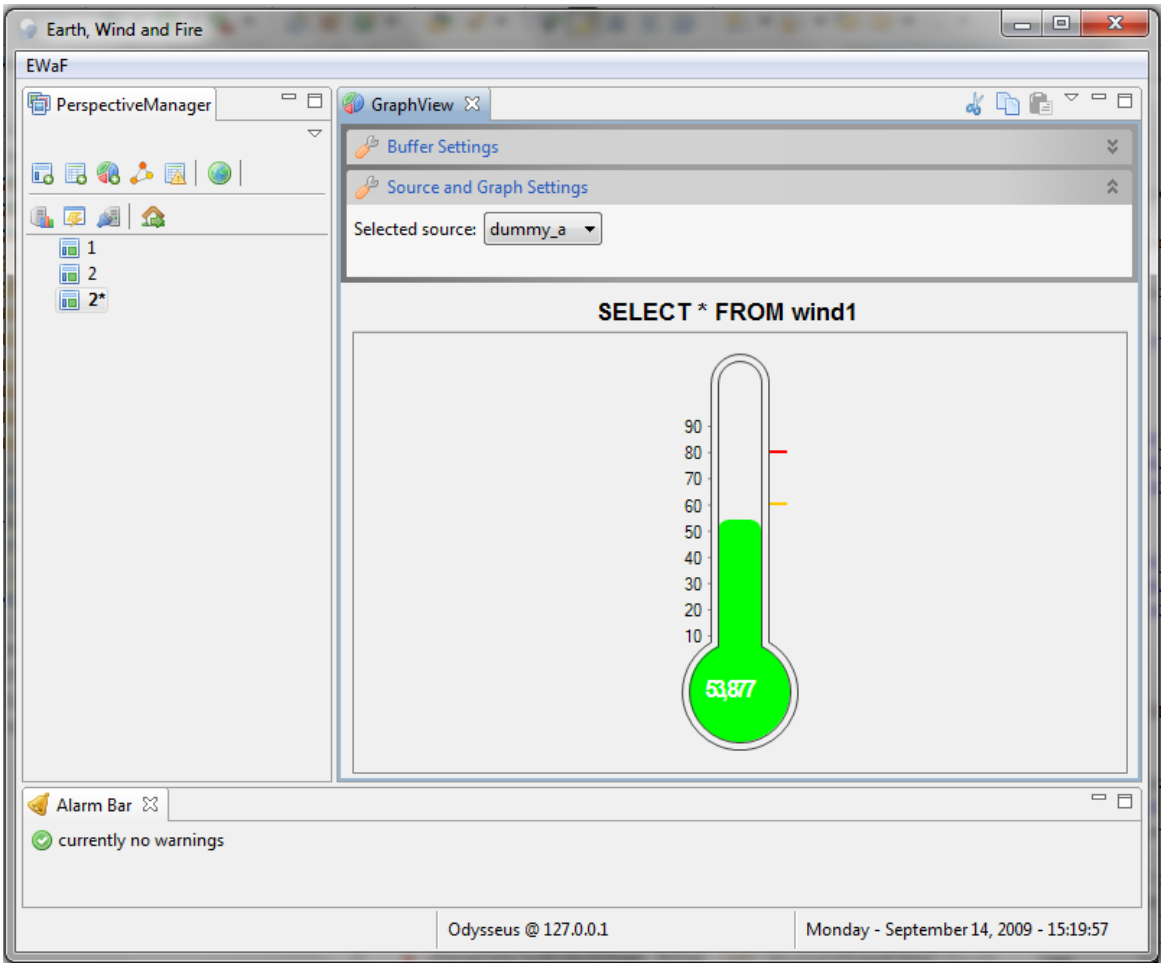


Abbildung 10.18: Thermometerdiagramm

dementsprechend einen anderen Verlauf haben.

Nachdem eine Topologie einer Perspektive hinzugefügt wurde (siehe Abschnitt: 10.4.5). Muss zunächst eine in der 3D-Welt gespeicherte Welt geladen werden (siehe Abbildung: 10.19). Daraufhin kann die angezeigte Topologie angepasst werden. Zum einen können kritische Werte bei Kanten angezeigt und die entsprechenden Farben gesetzt werden (mehr zu kritischen Werten im Abschnitt: 10.4.9).

Zum anderen können Schwellwerte pro Statement gesetzt werden, sodass entsprechende Kanten als aktiv oder inaktiv gezeichnet werden, sobald bestimmte Bedingung erfüllt sind. Jene Bedingungen können über einen Rechts-Klick auf eine Kante festgelegt werden (siehe Abbildung: 10.19 : 2). Dieser öffnet einen Dialog (siehe Abbildung: 10.19 : 3), der es erlaubt für jedes Attribut des Statements eine Bedingung festzulegen. Tipp: die Bedingungen sind an das Statement gebunden. Dementsprechend gelten gesetzte Schwellwerte für alle Kanten, die jenes Statement referenzieren.

Informationen zu den Kanten und Knoten können über die *Hover*-Funktion eingesehen werden (siehe Abbildung: 10.19 : 1).

Zur besseren Übersicht kann die gesamte Topologie gezoomt werden. Dazu stehen herein-, heraus- und auf Fenstergröße-zoomen zur Verfügung (siehe Abbildung: 10.19 : 4).

10.4.9 Kritische Werte

Über das Symbol *Modify critical value information* wird der Dialog zur Verwaltung der Grenzen für kritische Werte aufgerufen (siehe Abbildung 10.20). In dem Dialog werden alle bereits gespeicherten Grenzen angezeigt. Über das + Symbol kann ein neuer Eintrag hinzugefügt werden und ein selektierter Eintrag kann mit dem - Symbol wieder entfernt werden. In *Attribute* muss der Name des Attributes eingetragen werden, für den die Grenzen gelten sollen. Tipp: Ein gültiger Name ist beispielsweise *WTUR_ActivePowerGeneration_Float*. Dieser gilt dann

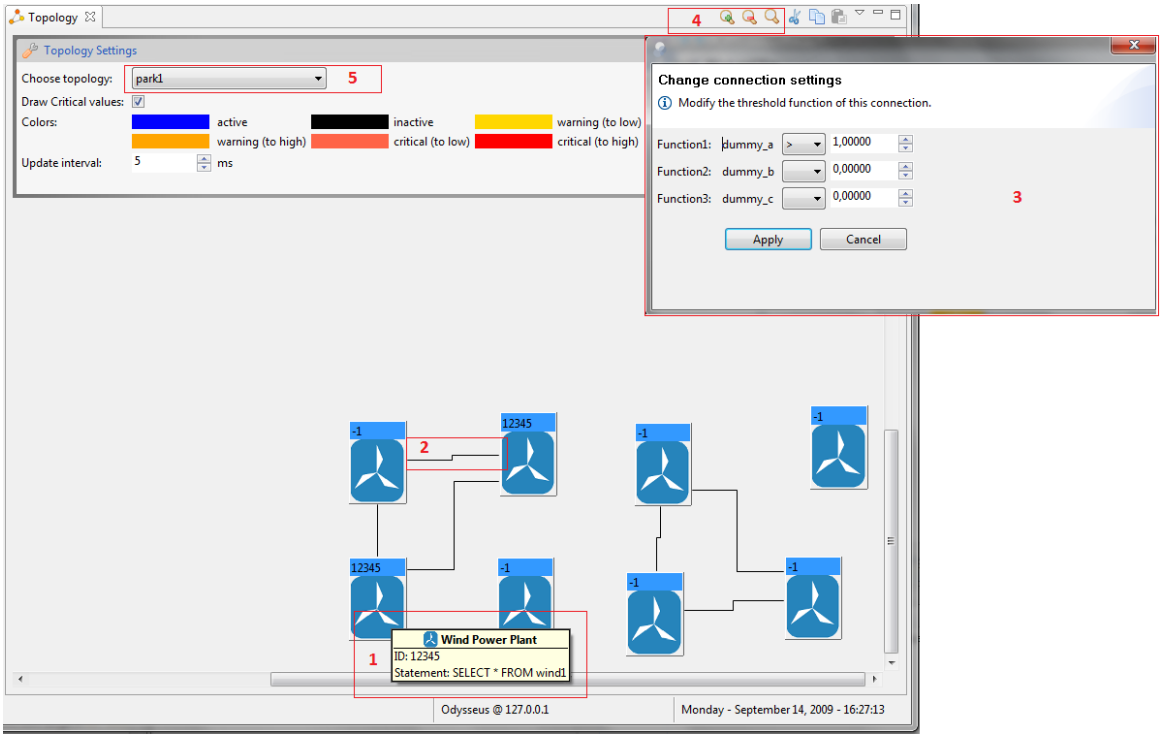


Abbildung 10.19: 2D Topologie

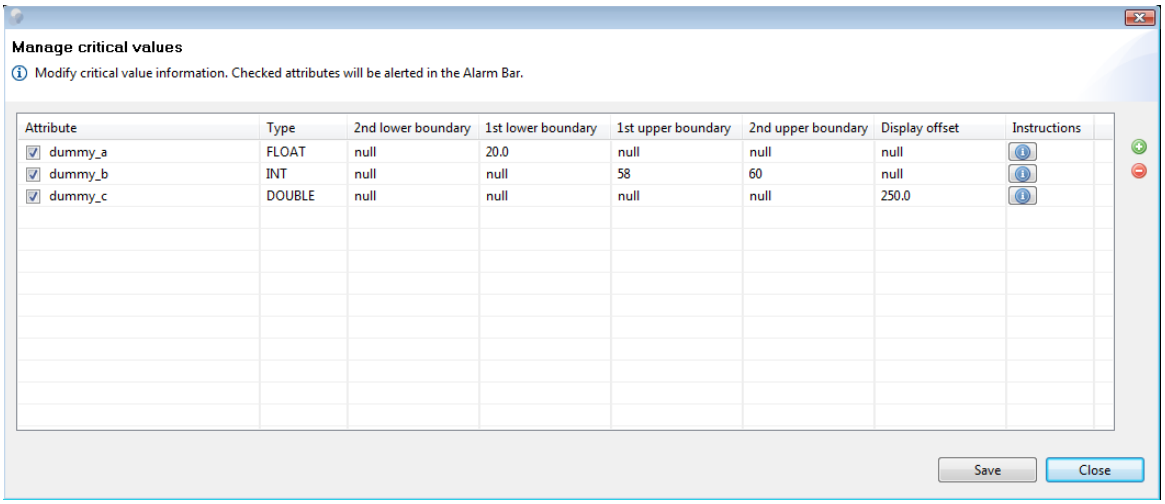


Abbildung 10.20: Kritische Werte definieren

nicht nur für das exakt gleichnamige Attribut, sondern auch für Attribute, die einen Anfragepräfix haben, wie z.B. *wind1.WTUR_ActivePowerGeneration_Float*. Falls gewünscht ist, dass bei zwei Anfragen mit gleichnamigen Attributen unterschiedliche Grenzen benutzt werden sollen, so müssen die betroffenen Attribute in der Anfrage umbenannt werden und in diesem Dialog die umbenannten Attributsnamen verwendet werden. Das Auswählen des Kontrollkästchens bewirkt, dass Werteüberschreitungen zu diesem Attribut in der Alarmleiste angezeigt werden. In der *Type* Spalte muss der Typ des Attributs angegeben werden. Dieser kann *INT, LONG, FLOAT, DOUBLE, STRING* oder *BOOLEAN* sein und muss mit dem Typ in dem Tupel übereinstimmen. Die nächsten vier Spalten bezeichnen die 2. und 1. untere sowie die 1. und 2. obere Grenze. Für die Werte muss gelten $2.\ untere \leq 1.\ untere < 1.\ obere \leq 2.\ obere\ Grenze$. Soll eine Grenze nicht definiert werden, kann der Platzhalter *null* verwendet werden. Eine Grenze gilt als überschritten, wenn der Wert kleiner oder gleich einer Untergrenze oder größer oder gleich einer Obergrenze ist. Bei einer Überschreitung wird immer der schlimmere Fall gemeldet, also das Überschreiten der 2. Grenze.

In der Spalte *Display offset* kann ein Freiwert gewählt werden. In einer Tabelle wird dann nicht der Originalwert des

Attributes angezeigt, sondern die Differenz zu dem Freiwert, mathematisch ausgedrückt $Freiwert - Originalwert$. Diese Einstellung ist sinnvoll, um immer zu sehen, wieviel Raum noch bis zu der Grenze verbleibt, ohne die Grenze an sich kennen zu müssen. Ein negativer Wert stellt somit eine Überschreitung des Freiwertes dar.

Alle Änderungen werden über die Schaltfläche *Save* gespeichert. Einige Einstellungen haben erst Auswirkungen auf View-Elemente, wenn diese neu geöffnet werden. Die Einstellungen werden lokal auf dem Rechner gespeichert. Ungültige Einträge werden beim Speichern entfernt.

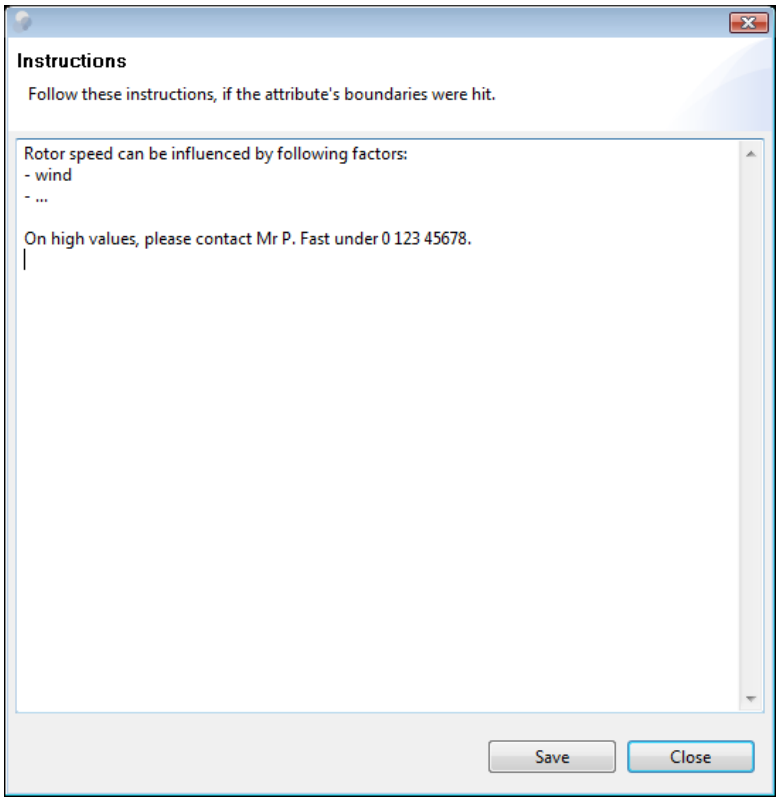


Abbildung 10.21: Handlungsanweisungen vermerken

Über die *i*-Schaltfläche in der letzten Spalte kann der *Instructions* Dialog geöffnet werden (siehe Abbildung 10.21). Dieser Text ist dafür vorgesehen Handlungsanweisungen, Vorschriften oder Hinweise zu den jeweiligen Attributen zu hinterlassen. Der Text kann dann beim Auftreten von kritischen Werten über das Ereignislog angezeigt werden (siehe Abschnitt 10.4.11).

10.4.10 Alarmleiste

In der Abbildung 10.12 ist im unteren Abschnitt die Alarmleiste zu sehen. Hier werden Warnungen für Attribute angezeigt, die im *Critical Values* Dialog ausgewählt wurden. Die Zahl in Klammern hinter dem Attributsnamen zeigt an, wieviele Warnungen für dieses Attribut neu bzw. noch nicht bestätigt sind. Das Symbol ist rot, wenn eines der gruppierten Warnungen eine 2. Grenze überschritten hat, sonst ist es gelb. Durch einen Klick auf die Schaltfläche wird ein Ereignislog geöffnet und zu dem aktuellsten Ereignis für dieses Attribut gesprungen (siehe Abbildung 10.22). Falls die Alarmleiste geschlossen wurde, kann sie wieder über das *EWaF*-Hauptmenü geöffnet werden.

10.4.11 Ereignislog

Über das Symbol *Open Event Viewer* oder eine Schaltfläche in der Alarmleiste kann ein Ereignislog geöffnet werden. Das Ereignislog zeigt alle aufgelaufenen kritischen Werte und weitere Informationen dazu an (siehe Abbildung 10.22). Da sich mit der Zeit sehr viele Ereignisse sammeln können, werden diese seitenweise angezeigt. Im oberen Abschnitt kann ausgewählt werden, wie viele Elemente pro Seite angezeigt werden sollen und direkt zu einer Seite gesprungen werden. Die neusten Ereignisse werden immer am Anfang angefügt. Es kann über das *Filter* Feld nach bestimmten Ereignissen gesucht werden. Dort kann entweder eine Anfragen-ID eingegeben werden oder ein Suchbegriff, der Teile

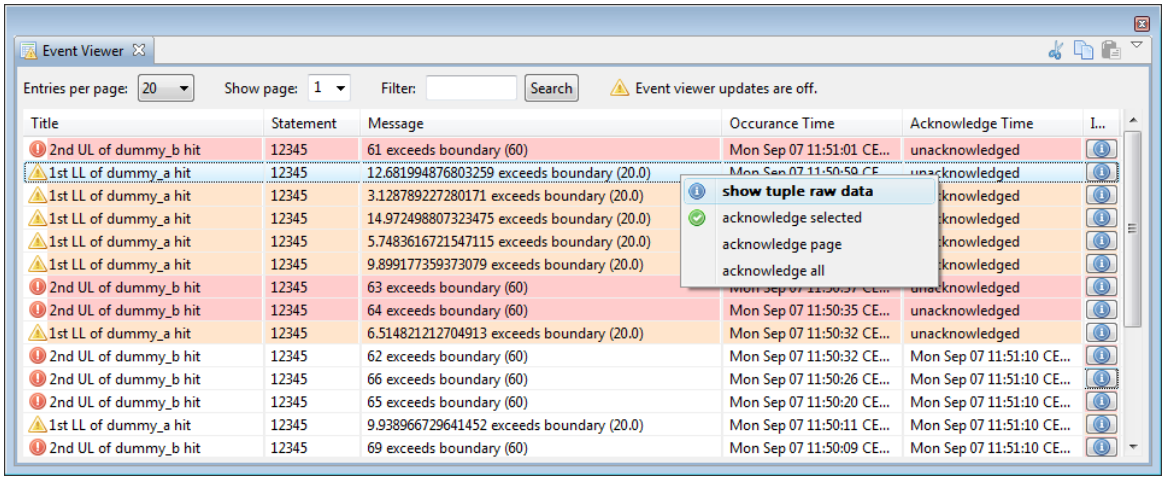


Abbildung 10.22: Ereignislog mit teils bestätigen Ereignissen

eines Attributnamens enthält. Es können auch reguläre Ausdrücke verwendet werden. Damit die Ereignisse in Ruhe betrachtet werden können, wird das automatische Aktualisieren unter folgenden Bedingungen ausgestellt: Ein Eintrag ist ausgewählt, eine alte Seite wird angeschaut, die *Show page* Combobox wird benutzt oder der Filter ist aktiv. Wenn neue Warnungen eintreffen, während das Ereignislog nicht aktualisiert wird, wird ein Hinweis im oberen Abschnitt angezeigt, wie auch in der Abbildung zu sehen ist.

Die Liste der Ereignisse ist in absteigender Reihenfolge nach Auftreten der Ereignisse sortiert. Ein Ereignis gliedert sich in Titel, Anfrage-ID, bei der das Ereignis aufgetreten ist, Nachricht und Zeitpunkt des Auftretens. Neue Ereignisse werden als unbestätigt gesehen und werden farbig markiert. Mit einem Rechtsklick auf Einträge können über das Kontextmenü ein oder mehrere selektierte Ereignisse, die ganze Seite oder alle Ereignisse bestätigt werden. Diese werden dann mit dem aktuellen Zeitpunkt als bestätigt markiert. Um mehr Informationen zu dem Ereignis zu erhalten, können zusätzlich über das Kontextmenü oder einen Doppelklick auf ein Ereignis die Rohdaten des Tupels angezeigt werden, das das Ereignis ausgelöst hat. Näheres zu den Rohdaten ist im folgenden Abschnitt festgehalten. Über die Schaltfläche in der letzten Spalte können Handlungsanweisungen bzw. Hinweise zu dem betroffenen Attribut angezeigt werden, die in dem *Critical Values* Dialog festgelegt wurden.

10.4.12 Rohdaten

Der Rohdaten-Dialog bietet die Möglichkeit, die Rohdaten eines Tupels im Klartextformat anzuzeigen. Rohdaten bezeichnet hier lediglich die Sammlung aller Attribute in unformatierter Form, sowie die Pflichtfelder Zeitstempel, Priorisierung und Anfrage-ID. Der Dialog kann über zwei Wege geöffnet werden: Über einen Doppelklick auf einen Eintrag in einer Tabelle (siehe Abschnitt 10.4.6) oder über einen Doppelklick auf einen Eintrag im Ereignislog (siehe Abschnitt 10.4.11). Für das jeweilige Tupel werden die Pflichtfelder Zeitstempel, Anfrage-ID und Priorisierung, sowie alle Attribute mit ihren Werten aufgelistet (siehe Abbildung 10.23). Tipp: Um komplexere CQL-Anfragen zu formulieren, werden meist Attributnamen benötigt. Diese können über diesen Dialog leicht kopiert und eingefügt werden. Wird der Dialog über das Ereignislog aufgerufen, so besteht die Option über die Schaltfläche *Acknowledge* alle Ereignisse, die dieses Tupel ausgelöst hat, zu bestätigen.

10.4.13 Kioskmodus

Über das Symbol *Switches to Kioskmode with less authorization* kann die Anwendung in den Kioskmodus wechseln. In diesem Modus ist der Benutzer so eingeschränkt, dass er den regulären Ablauf der Anwendung nicht stören kann (siehe Abbildung 10.24). Folgende Features sind im Kioskmodus deaktiviert:

- Anlegen, Löschen oder Speichern einer Perspektive
- Anfragen verwalten
- Streams verwalten
- Kritische Werte definieren

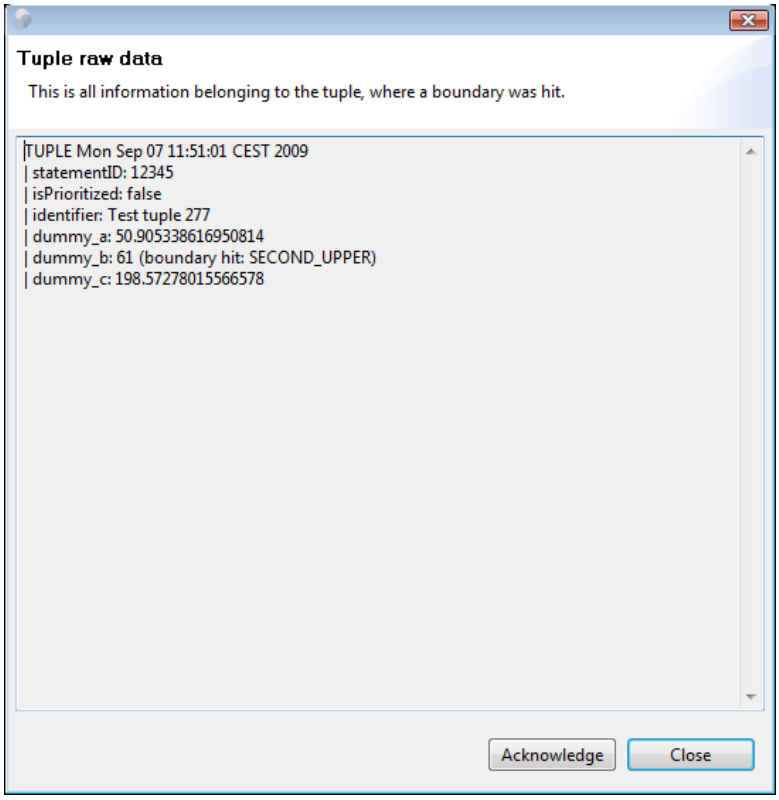


Abbildung 10.23: Rohdaten eines Tupels

- Schließen der Anwendung
- Wechseln zu einer anderen Odysseusinstanz
- Handlungsanweisungen ändern
- Ereignisse als bestätigt markieren
- Modelle in der 3D-Welt ändern

Um wieder die volle Funktionalität freizuschalten, kann über den *Switch to Administratormode* Dialog wieder in den priviligierten Modus gewechselt werden (siehe Abbildung 10.25). Das dazu benötigte Passwort lautet **resetmode**.

10.4.14 Fehlerfall

Unter besonderen Umständen kann es bei der Kommunikation mit Odysseus zu Fehlern kommen. Beispielsweise beim Installieren einer Anfrage kann Odysseus einen Fehlercode zurückliefern, der dann wie in Abbildung 10.26 angezeigt wird. Die Bedeutung dieses Fehlercodes kann im Abschnitt 10.3 nachgeschlagen werden.

Desweiteren kann ein allgemeiner Fehlerdialog auftreten, der eine kurze Beschreibung des Fehlers enthält (siehe Abbildung 10.2), und unter *Details* auch den Stacktrace zur näheren Untersuchung der Ursache ausgibt (siehe Abbildung 10.27).

Möglicherweise treten auch Fehler innerhalb des RCP-Frameworks auf. Diese werden in einer Logdatei gespeichert, die im Fall eines Plugins im Workspace-Ordner unter *.metadata/.log* und bei Ausführung als Eclipse Product im Ordner *workspace/.metadata/.log* zu finden ist.

Im *.metadata*-Ordner werden zudem die Positionen aller GUI-Elemente der View gespeichert. Sollen diese einmal in den ursprünglichen Zustand zurückgesetzt werden, so kann der Ordner einfach gelöscht werden.

Alternativ Anfrage

Sollte der Odysseus-Server neu gestartet oder ausgetauscht werden, so werden die Perspektiven ungültig, da die referenzierten Anfragen nicht mehr vorhanden sind oder unter einer anderen ID registriert sind. Sollte dies der Fall sein,

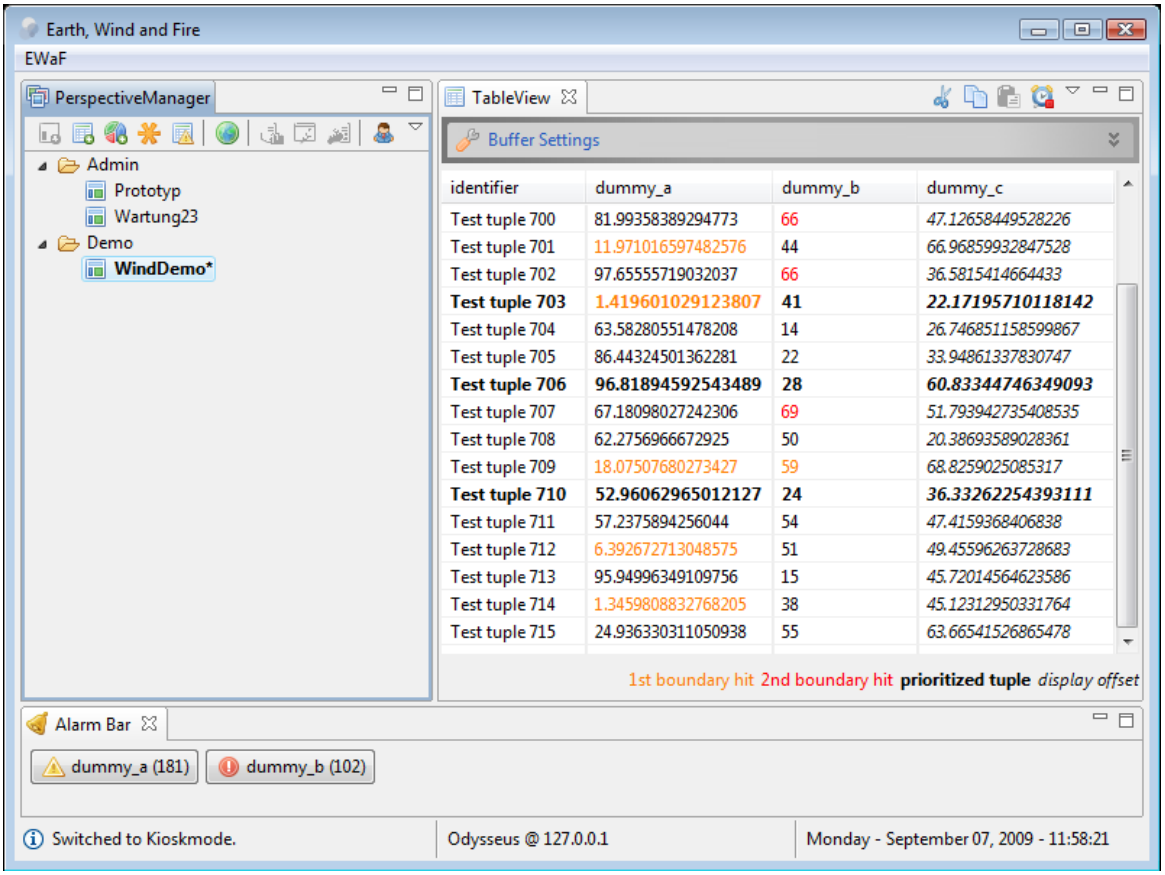


Abbildung 10.24: Anwendung im Kioskmodus

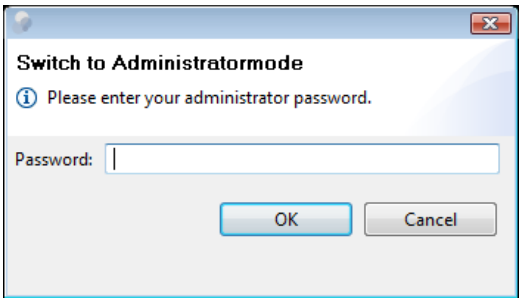


Abbildung 10.25: Passwortabfrage zum Wechsel in den priviligierten Modus

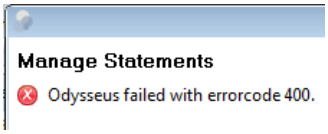


Abbildung 10.26: Odysseus liefert einen Fehlercode beim Installieren einer Anfrage

können die Elemente der Perspektive jedoch wiederhergestellt werden. Dazu stehen zwei Funktionen zur Verfügung (siehe Abbildung: 10.28). Zum einen kann direkt eine neue Anfrage installiert werden: *Install new statement*, oder eine bestehende Anfrage als Alternative ausgewählt werden: *Find alternative statement*. In beiden Fällen muss jedoch im Anschluss die alternative Anfrage angegeben werden. Dazu steht ein Dialog zur Verfügung, der kompatible Anfragen kategorisiert auflistet (siehe Abbildung: 10.29 : 1) Dabei gibt es folgende Kategorien:

- *Statements with same attributes*: Anfragen, die die gleiche Attribute haben (gleicher Name und Datentyp)

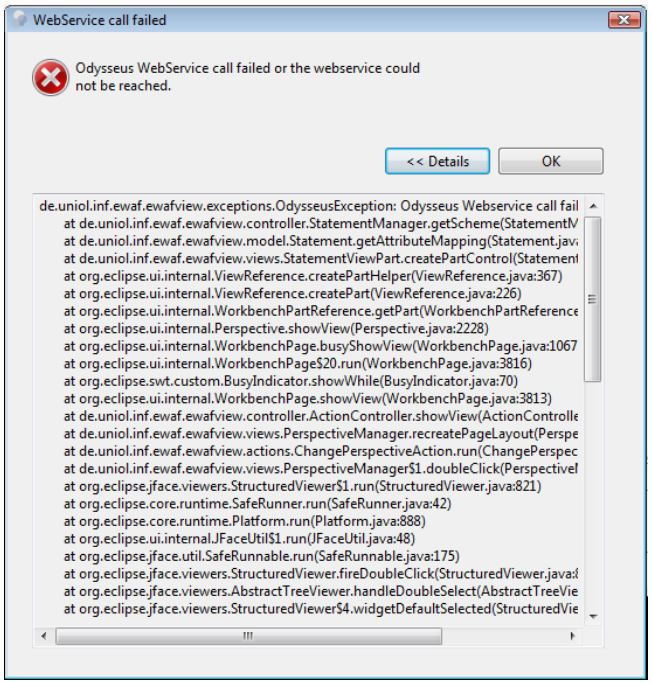


Abbildung 10.27: Fehlerfall bei einem Webservice-Aufruf

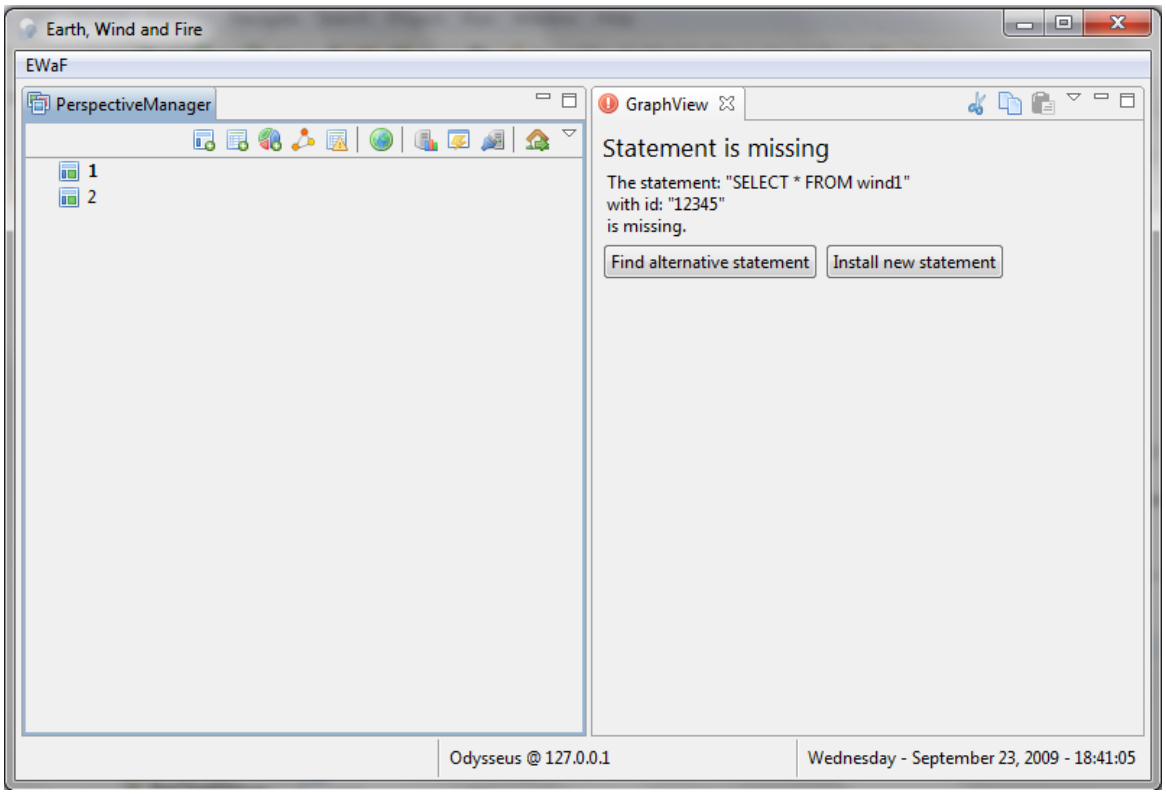


Abbildung 10.28: Graph mit „verlorener“ Anfrage

- *Statements with overlapping attributes*: Anfragen, die die gleichen Attribute haben, sowie zusätzliche Attribute
- *Statements with same types*: Anfragen, die die gleiche Anzahl an Attributen mit den gleichen Datentypen haben, sich jedoch die Namen der Attribute unterscheiden

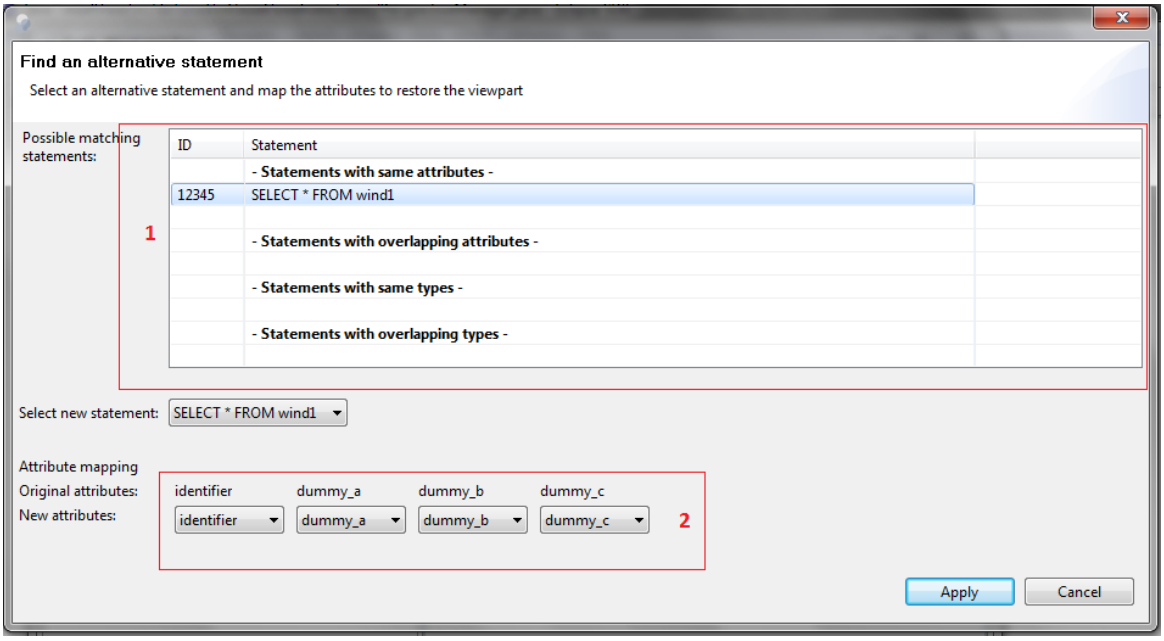


Abbildung 10.29: Dialog zur Auswahl einer alternativen Anfrage

- *Statements with overlapping types*: gleiche Bedingung wie *Statements with same types*, aber es existieren zusätzliche Attribute

Bei allen Kategorien, mit Ausnahme der ersten, erfordern, dass die Attribute zugeordnet werden müssen. Dazu muss jedem ursprünglichen Attribut ein Neues zugeordnet werden (siehe Abbildung: 10.29 : 2). Nachdem die Angaben per *Apply* gespeichert wurden, wird die View wiederhergestellt.

Diese Änderungen sind persistent, d.h. die original Anfrage wird priorisiert geladen, jedoch falls sie nicht vorhanden ist, wird die alternative Anfrage geladen. Falls beide nicht vorhanden sind, wird wieder aufgefordert ein Alternativ-Anfrage auszuwählen (siehe Abbildung: 10.28). Falls die alternative Anfrage genutzt wird, so wird im Titel des Elements „(alt.)“ vermerkt.

10.5 3D-View

Die 3D-View ist eine dynamische, editierbare 3D-Welt in der die Datenströme von Odysseus dargestellt werden. Innerhalb dieser Welt kann man sich mithilfe von Maus und Tastatur frei bewegen. Die verschiedenen Erzeuger von Datenströmen werden anhand von Modellen wie z.B Windrad oder Geothermalkraftwerk visualisiert. Der folgende Abschnitt erklärt die Funktionen und die Bedienung der 3D-View.

10.5.1 Anforderungen der 3D-View

Im Folgenden sind die Anforderungen der 3D-View beschrieben. Da die 3D-View als Teil der View realisiert ist, sind die Anforderungen der View Voraussetzung (siehe 10.4.1). Des Weiteren sind folgende zusätzliche Anforderungen empfehlenswert.

Hardware

- Pentium 4 2.0 GHz oder höher
- 1024 MB RAM
- OpenGL 2.1 kompatible Grafikkarte

Unterstützte Betriebssysteme

- Microsoft Window XP ©
- Microsoft Windosw Vista ©

10.5.2 Starten der 3D-View

Um die 3D-View zu starten wird in der View (siehe 10.7 : 1) der Button „New 3d World“ angeklickt. Dieser befindet sich in der Toolbar des Perspektiven Managers. Beim Starten der 3D-View wird eine Fortschrittsanzeige eingeblendet. Das Laden aller Modelle kann ein wenig Zeit in Anspruch nehmen. Die gestartete 3D-View befindet sich in einem Element innerhalb des Hauptfensters. Um die 3D-View in einem eigenen Fenster anzuzeigen muss mit Rechtsklick auf die Titelleiste geklickt werden. In dem folgenden Kontextmenü kann dann „Detached“ angeklickt werden (siehe auch 10.10). Abbildung 10.30 zeigt die gestartete 3D-View.

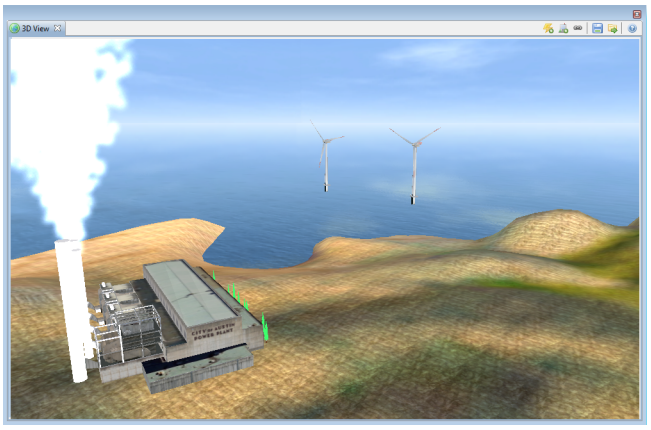


Abbildung 10.30: Gestartete 3D-Welt

10.5.3 Navigation

Die Navigation in der 3D-Welt wird mit der Tastatur vorgenommen. Um sich zu bewegen werden die Tasten w,a,s und d benutzt.

- w - vorwärts bewegen
- s - rückwärts bewegen
- a - nach links bewegen
- d - nach rechts bewegen

Eine Drehung der Ansicht kann mit den Pfeiltasten vorgenommen werden.

- Pfeil links - in die linke Richtung blicken
- Pfeil rechts - in die rechte Richtung blicken
- Pfeil oben - Ansicht nach vorne kippen
- Pfeil unten - Ansicht nach hinten kippen

Mit den Tasten „q“ und „e“, sowie dem Mausrad kann heraus- bzw hereingezoomt werden. Eine Auflösung der Tastaturkürzel befindet sich auch hinter dem Button „Help“ in der Toolbar der 3D-View.

10.5.4 Anpassen der Welt

Die 3D-View besitzt eine eigene Toolbar für die Konfiguration der Darstellung. Der Welt können so neue Modelle hinzugefügt oder bestehende Modelle entfernt werden. Weiter ist es möglich die Darstellung einzelner Modelle zu bearbeiten. Die 3D-View unterscheidet dabei zwischen einem dynamischen Modell, welches Anfragen darstellen kann und einem statischen Modell, welches keine Anfragen darstellen kann.

Dynamisches Modell hinzufügen

Dynamische Modelle werden mit dem Button *Create Module* erstellt. Es erscheint der Dialog *Create 3D-Module* (siehe Abbildung 10.31). Um ein Modell zu erstellen, muss das gewünschte Modell in der Liste (siehe Abbildung 10.31 : 1) ausgewählt werden. Soll an dieses Modell eine Anfrage gelegt werden, so muss die entsprechende Anfrage in der Tabelle ausgewählt werden. Existiert die gewünschte Anfrage noch nicht, so kann dies durch *Manage Statements* (in 10.31 : 2) angelegt werden. Es öffnet sich daraufhin der *Manage Statement* Dialog der View. Dort kann, wie in Abschnitt 10.4.4 beschrieben, eine Anfrage erstellt werden. Eine Anfrage ist nicht Voraussetzung für das Erstellen eines dynamischen Modells. Wurde die gesamte Auswahl getroffen, so kann das Modell durch *Create* erstellt werden. Das Modell erscheint daraufhin in der 3D-Welt. Zum Anpassen der Darstellung siehe Abschnitt 10.5.4.

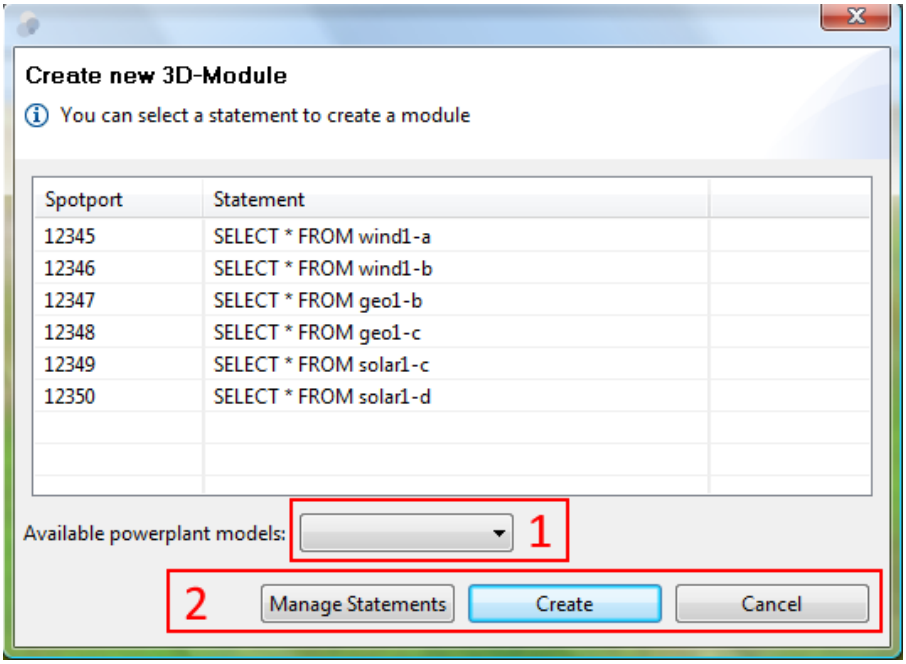


Abbildung 10.31: Dynamisches 3D-Modell hinzufügen

Statisches Modell hinzufügen

Statische Modelle können über den Button *Create statical Module* in der Toolbar erstellt werden. Im darauf erscheinenden Dialog kann das gewünschte Modell ausgewählt und mit *Create* in die Welt platziert werden. Die Anpassung der Darstellung wird in Abschnitt 10.5.4 erklärt.

Erdkabel hinzufügen

Sind mehrere Modelle in der Welt platziert gibt es die Möglichkeit Verbindungen zwischen ihnen zu darzustellen. Zur Visualisierung solcher Erdkabel, die z.B. ein Kraftwerk mit Stromabnehmern verbindet, können halbtransparente Verbindungen gezeichnet werden. Eine solche Verbindung wird als Kantenzug zwischen mehreren Punkten gezeichnet.

Mit einem Klick auf den Button *Draw Underground Cable* in der Toolbar aktiviert man den Modus um eine virtuelle Verbindung zu zeichnen. Hierzu klickt man auf die gewünschte Stelle in der Welt, wo die Verbindung beginnen soll. Jeder weitere Klick lässt einen Punkt des Kantenzuges erstellen. Mit einem zweiten Klick auf den Button *Draw Underground Cable* wird der Modus beendet und der Kantenzug platziert. Auch diese Verbindungen

haben die Möglichkeit Anfragen anzuzeigen. Sie verhalten sich wie Modelle, können also ebenfalls gedreht oder verschoben werden (siehe 10.5.4).

Anpassen von Modellen

Platzierte Modelle können angepasst werden indem man sie mit der linken Maustaste doppelklickt. Jetzt kann das Modell beliebig in der Welt verschoben werden. In diesem Modus ist es auch möglich das Modell zu löschen, zu drehen, die Größe zu verändern sowie eine vertikale Positionierung vorzunehmen. Die Tastenbelegung hierzu:

- + - Modell vertikal nach oben bewegen
- - - Modell vertikal nach unten bewegen
- v - Modell vergrößern
- b - Modell verkleinern
- r - Modell drehen
- Entf - Modell entfernen

10.5.5 Anfragedarstellung

Dynamische Modelle und Erdkabel können Anfragen repräsentieren. Zum einen können die Daten als Text, welcher oberhalb der Modells erscheint, visualisiert werden. Des Weiteren ist es möglich verschiedene Visualisierungen der Modelle durch die empfangenen Daten zu beeinflussen.

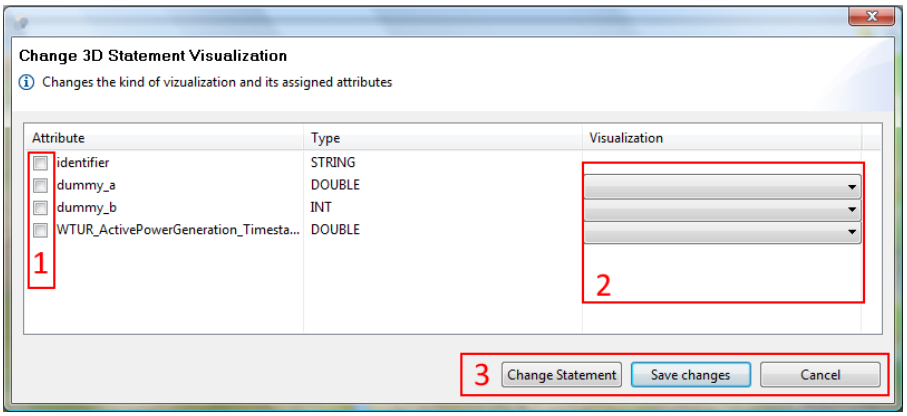


Abbildung 10.32: Anfrage eines 3D-Modells bearbeiten

Ein Rechtsklick auf das gewünschte Modell bzw. Erdkabel öffnet den Dialog um die Anfragedarstellung eines Modells zu ändern. Im folgenden Dialog (siehe Abbildung 10.32) wird die momentan ausgewählte Anfrage angezeigt.

Um einzelne Attribute einer Anfrage als Text oberhalb des Modells anzuzeigen, können die gewünschten Attribute per Checkbox (siehe Abbildung 10.32 : 1) ausgewählt werden. In der Spalte *Visualization* (siehe Abbildung 10.32 : 2) kann für jedes Attribut deren Abbildung auf das Modell gewählt werden. So ist es z.B. möglich, die Flügelrotation eines Windkraftmodells durch die Visualisierung *Blade Rotation* an ein Attribut zu knüpfen. Die Geschwindigkeit der Flügelrotation des Modells passt sich daraufhin der Höhe des Wertes des zugeordneten Attributes an. Für diese Art von Visualisierung ist es notwendig, dass zum entsprechenden Attribut kritische Werte, wie in Abschnitt 10.4.9 beschrieben, definiert sind. Analog existieren weitere Visualisierungen bei den verschiedenen Modellen bzw. dem Erdkabel.

Ist dem Modell noch keine Anfrage zugewiesen oder soll die aktuelle Anfrage geändert oder gelöscht werden, so erlaubt dies die Funktion *Change Statement* (siehe Abbildung 10.32 : 3). Im darauf folgenden Dialog (siehe Abbildung 10.33) kann die zugeordnete Anfrage geändert werden.

Um eine andere Anfrage auszuwählen, muss die gewünschte Anfrage ausgewählt werden und die Änderungen mit *Save Changes* (Abbildung 10.33 : 1) gespeichert werden. Ist die gewünschte Abfrage nicht vorhanden, so kann durch *Manage Statements* (in 10.33 : 1) eine Anfrage angelegt werden. Es öffnet sich daraufhin der *Manage Statement* Dialog der View. Siehe dazu Abschnitt 10.4.4. Des Weiteren kann mit *Remove Statement from Model* eine Anfrage von einem Modell entfernt werden.

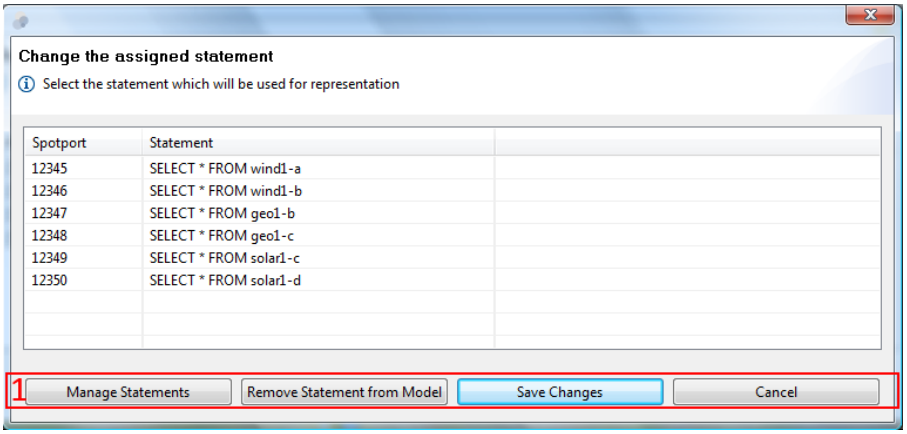


Abbildung 10.33: Anfrage eines 3D-Modells auswählen oder löschen

10.5.6 Laden und Speichern

Erstellte Welten können abgespeichert werden. Die Symbole *Save World* und *Load World* bieten Dialoge (siehe 10.34 zum Speichern und Laden von beliebig vieler Welten. Die Welten werden zentral auf einem Server hinterlegt, so dass sie auch anderen 3D-Views zur Verfügung steht. Über den *Load World* Dialog kann eine vorher ausgewählte Welt unwiderruflich gelöscht werden.

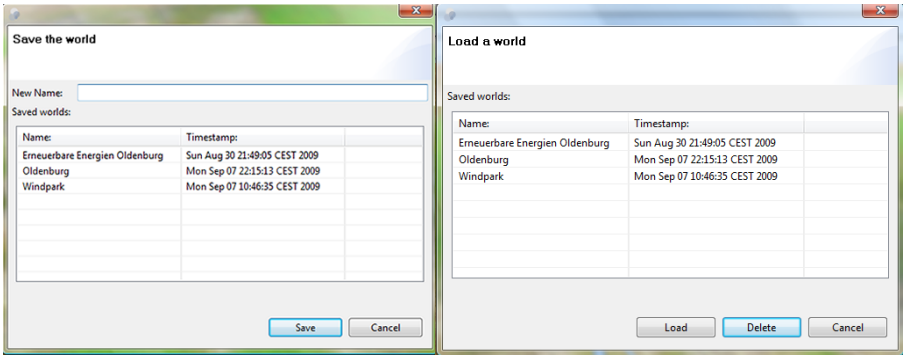


Abbildung 10.34: Speichern- und Ladendialog der 3D-Welt

Kapitel 11

Präsentationsdrehbuch

11.1 Einführung

Dieses Präsentationsdrehbuch dient zur Vorbereitung und Durchführung von Präsentationen. Ziel des Drehbuches ist es, einen gleichbleibende Präsentation zu bieten und zu garantieren, das sämtliche Bereiche der Software vorgestellt werden. Es wird davon ausgegangen, das die Programmteile erfolgreich installiert wurden und der Präsentator mit der Bedienung vertraut ist.

11.2 Vorbereitung

Zur Vorbereitung vor der ersten Präsentation folgen Sie diesen Schritten:

- Starten Sie Odysseus wie in der Bedienungsanleitung beschrieben.
- Starten Sie die SunSpots wie in der Bedienungsanleitung beschrieben.
- Starten Sie die View wie in der Bedienungsanleitung beschrieben.
- Stellen Sie wenn nötig die aktuelle IP-Adresse der Odysseus-Installation ein.

Diese Schritte sind nur zu Beginn nötig und brauchen bei mehreren aufeinanderfolgenden Präsentationen nicht jedes mal wiederholt werden.

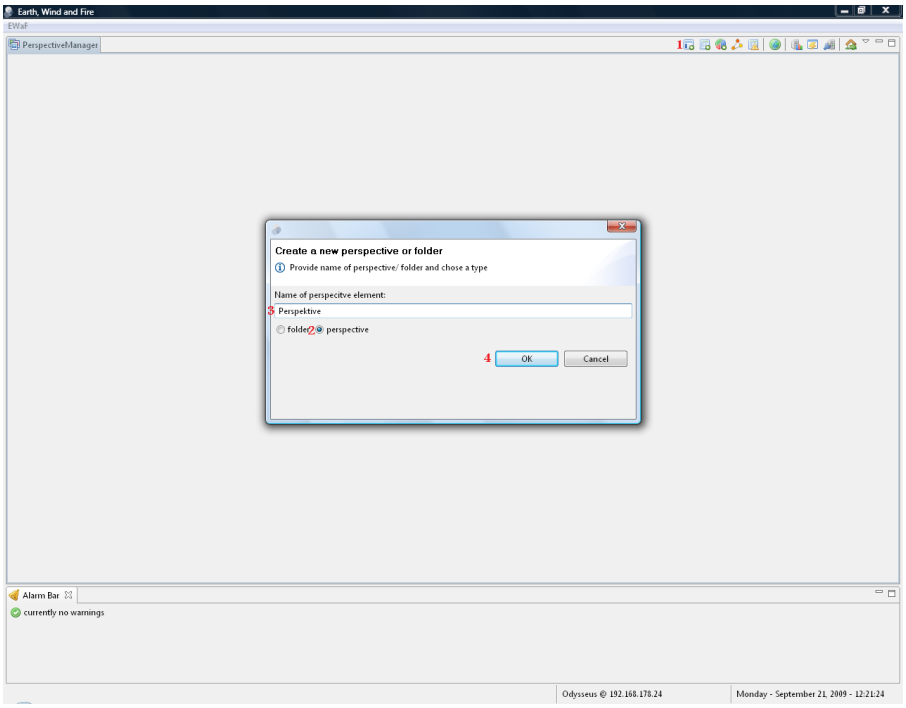
Damit bei einer weiteren Präsentation keine Konflikte mit bereichts installierten Streams und Statements entstehen, wird empfohlen, vor jeder Präsentation Odysseus neu zu starten.

11.3 Präsentation

11.3.1 Datenquelle

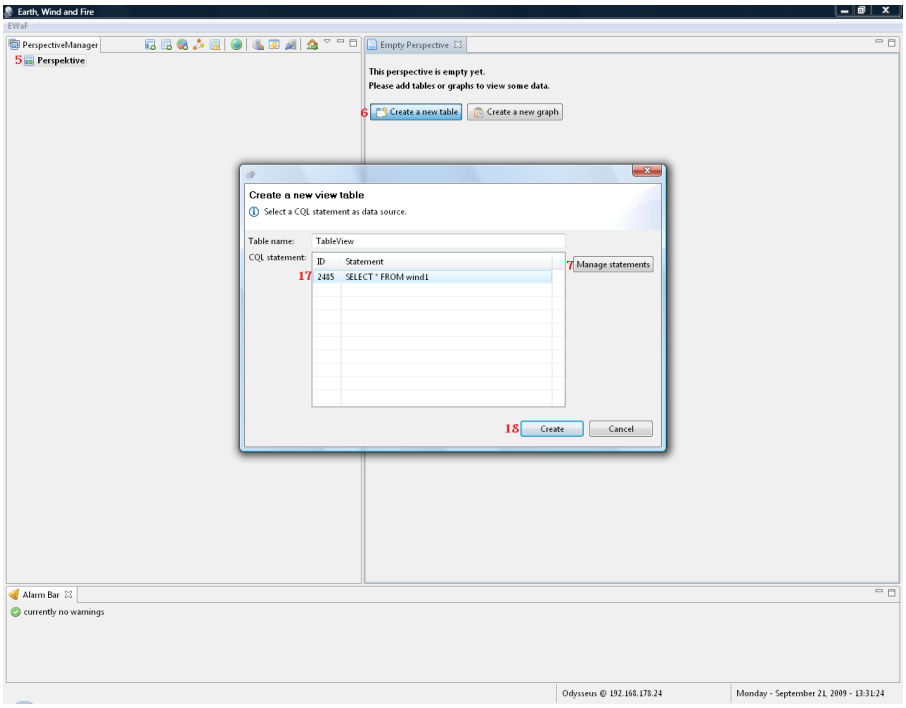
Sollten Sie den kompletten Präsentationsaufbau zur Verfügung haben, können Sie für die Datenquelle Beispielsweise das Windrad mit den SunSpots benutzen. Für dieses Drehbuch wird der VirtualSpotHost mit einem simulierten Windkraftwerk verwendet werden.

11.3.2 Perspektive erstellen

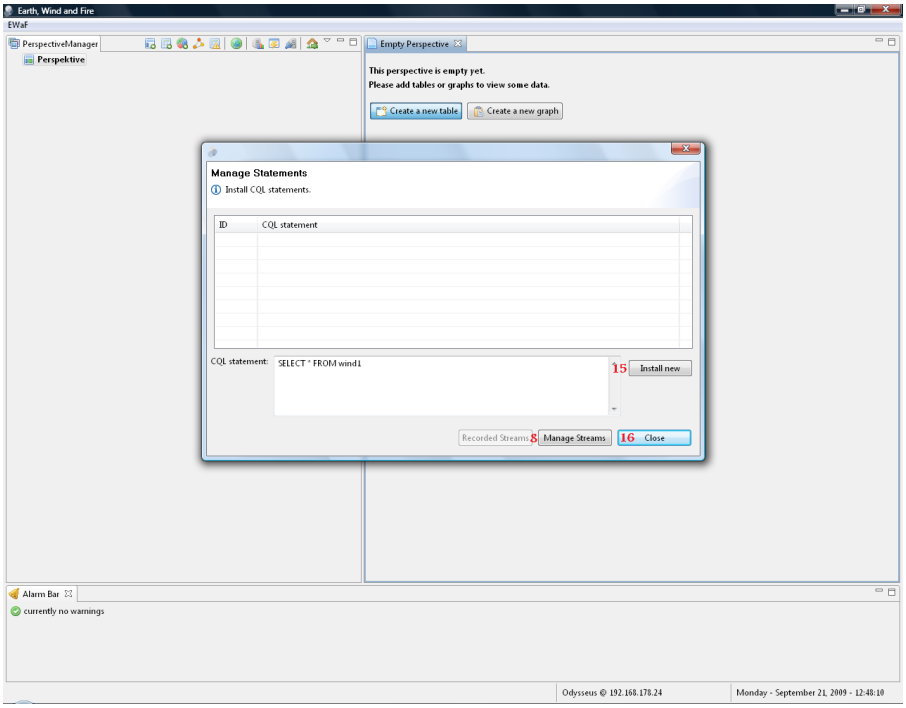


Erstellen Sie zunächste eine neue Perspektive. Dazu klicken Sie auf „adds new perspective or folder“ (1), wählen „perspective“ aus (2), geben einen Namen für die Perspektive ein (3) und bestätigen mit „OK“ (4)

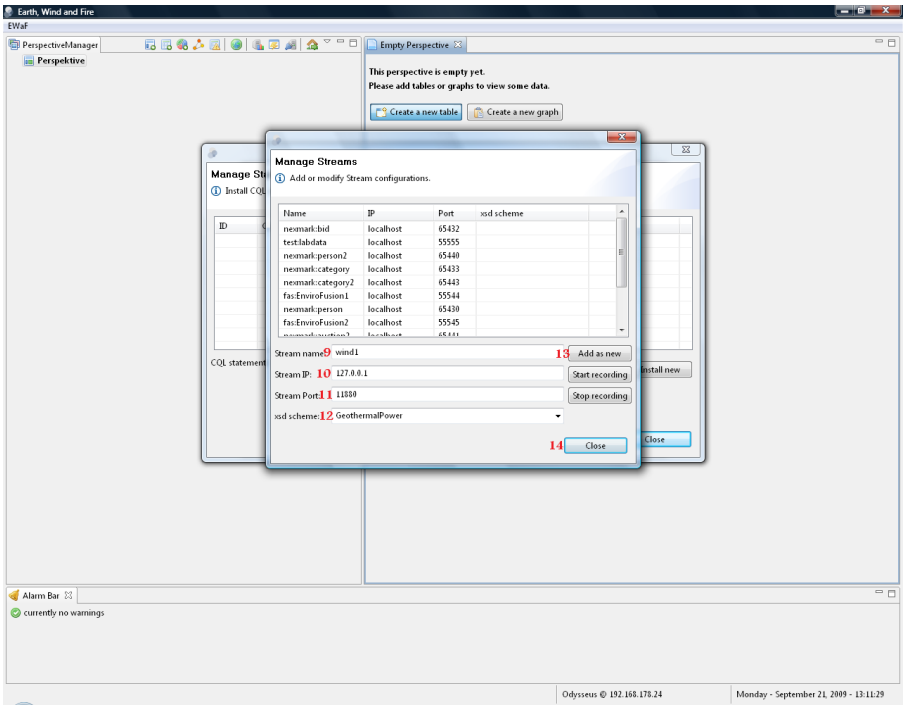
11.3.3 Stream und Statement anlegen, Tabelle erstellen



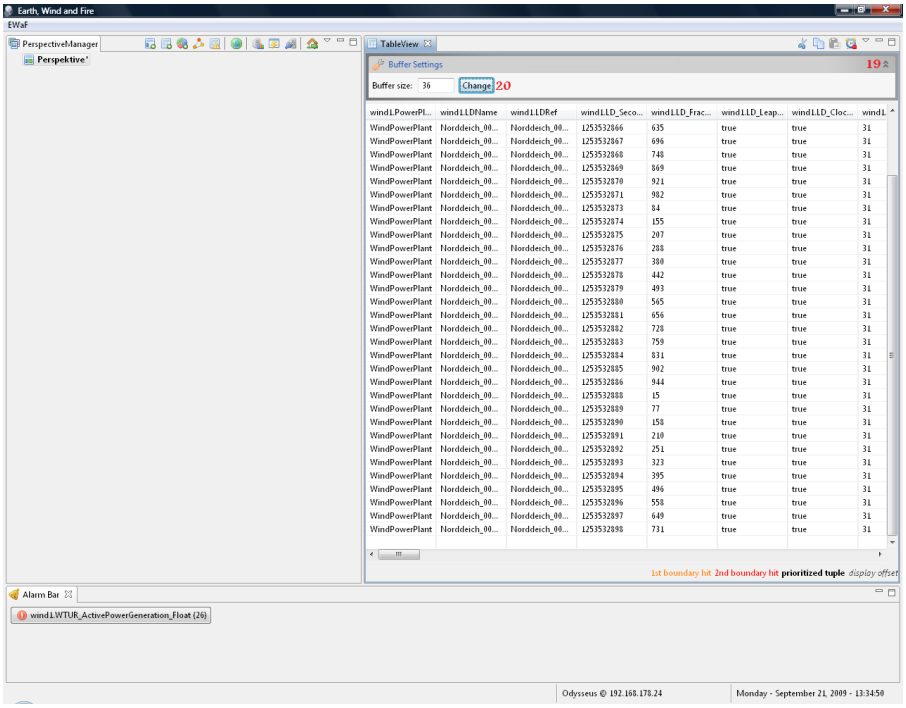
Jetzt soll eine Tabelle angelegt werden. Dazu muss erst ein Stream und ein Statement installiert werden. Doppelklicken Sie hierzu auf Ihre gerade angelegte Perspektive (5). In der rechten Fensterhälfte erscheint daraufhin die Auswahl zwischen Tabelle und Graph, wählen Sie hier „Create a new table“ (6). Im aufgehenden Fenster sind noch keine Statements installiert, darum wählen Sie „Manage statements“ (7).



Da noch kein Stream installiert wurde, wählen Sie im aufgehenden Fenster „Manage Streams“ (8).

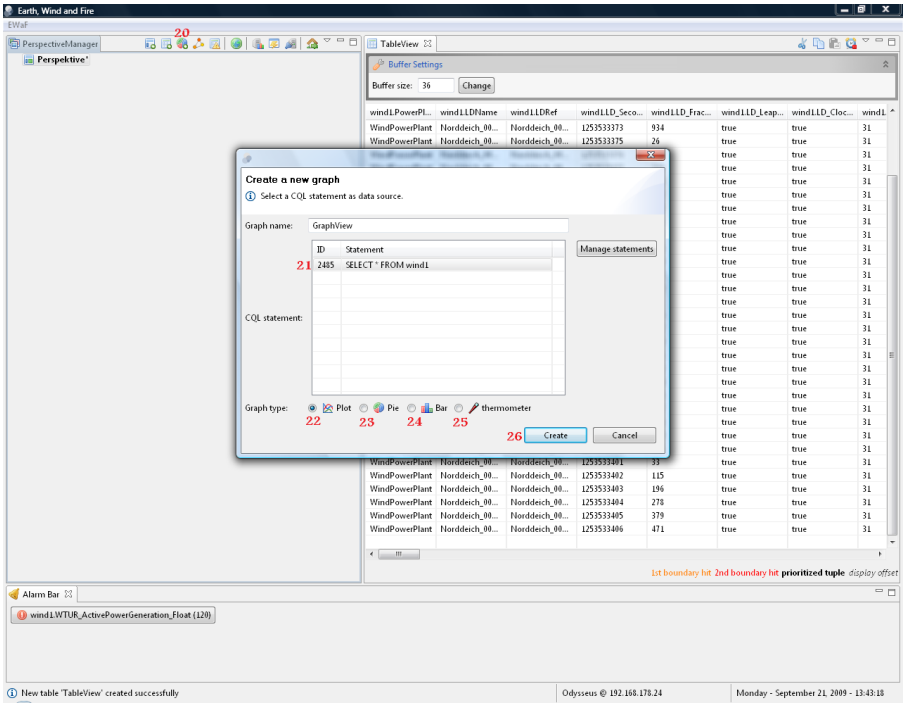


Im nun erscheinenden Fenster geben Sie einen Namen für den Stream ein (9), die IP der SpotHosts (10), den Port des LN SunSpot beziehungsweise den im VirtualSpotHost angegebenen (11) und wählen die Art des xsd-Shemas aus (12). Fügen Sie den Stream mit „Add as new“ (13) hinzu und schließen Sie das Fenster mit „Close“ (14). Sie gelangen wieder zum vorherigen Fenster. Belassen Sie das CQL Statement bei „SELECT * FROM wind1“ und installieren Sie es mit „Install new“ (15). Schließen Sie das Fenster mit „Close“ (16). Jetzt ist in der Liste der „CQL-Statements“ ein Eintrag vorhanden (17). Wählen Sie diesen aus und bestätigen Sie mit „Create“ (18).

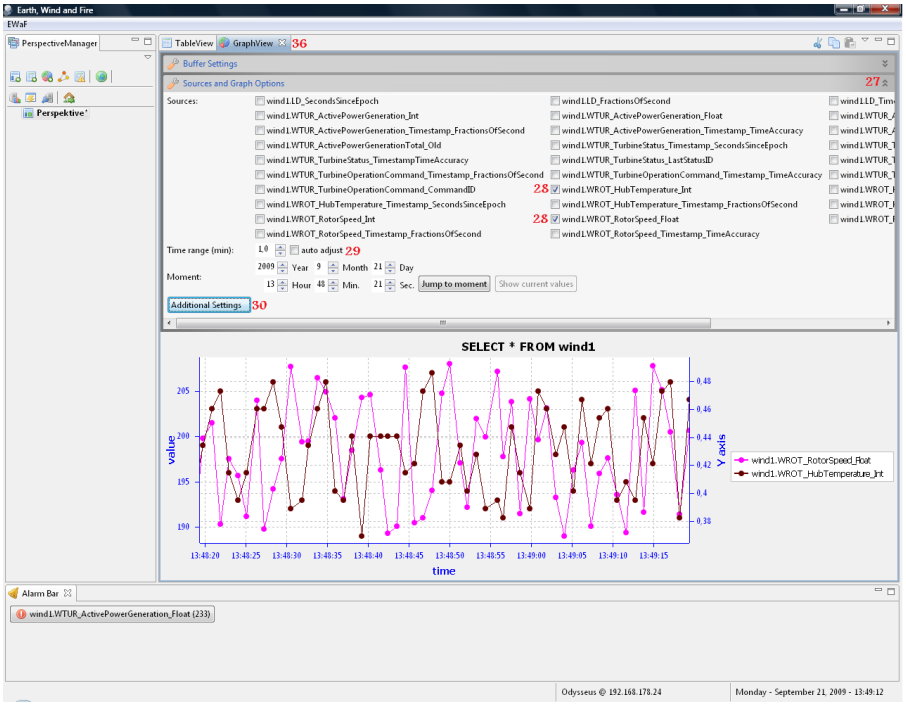


Die Tabelle erscheint jetzt im rechten Fenster. Ändern Sie die Puffergröße indem Sie die „Buffer Settings“öffnen (19). In dem Textfeld geben Sie die neue Größe ein und bestätigen mit „Change“ (20).

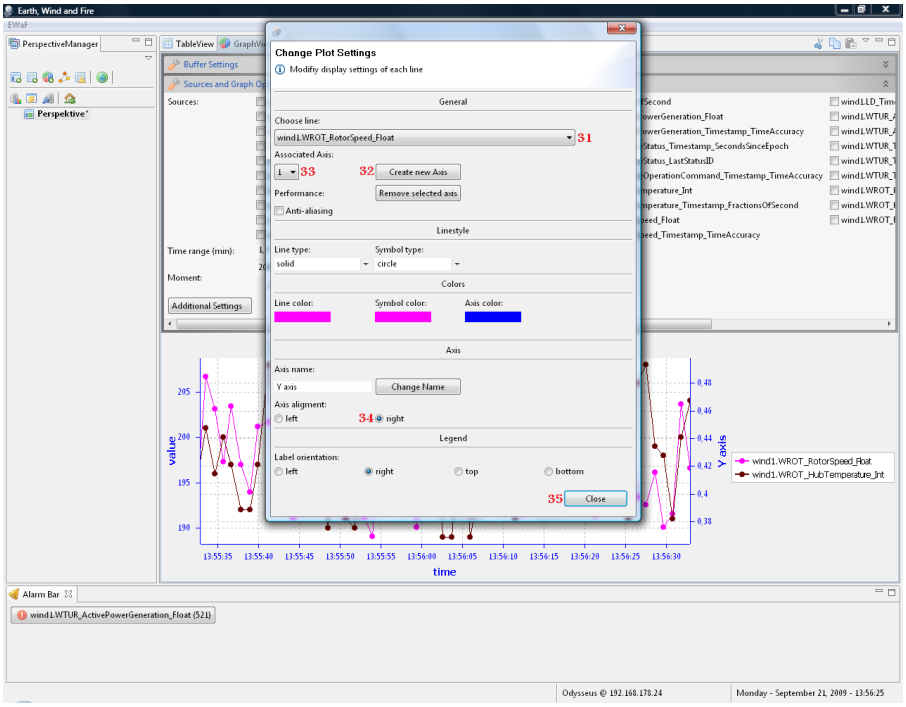
11.3.4 Graph erstellen



Erstellen Sie jetzt einen Graphen. Wählen Sie dazu die Schaltfläche „Add a new Graph“ (21). Im aufgehenden Fenster markieren Sie das Statement (21), wählen „Plot“ (22) und erstellen den Graph mit „Create“ (26)



Ziehen Sie den Graphen neben die Tabelle. Öffnen Sie jetzt die „Sources and Graph Options“(27) und wählen Sie die anzuzeigenden Werte aus. Für Windkraftanlagen eigenen sich „WROT_RotorSpeed_Float“ und „WROT_HubTemperature_Int“ (28). Diese beiden Werte liegen weit auseinander und werden nun am oberen und unteren Rand des Graphen angezeigt. Dieses ändern Sie indem Sie auf „Additional Settings“ (30) klicken.



Wählen Sie im neuen Fenster im Dropdown Menü „WROT_RotorSpeed_Float“ (31). Erstellen Sie mit „Create new Axis“ eine neue Achse (32) und wählen Sie diese unter „Associated Axis“ (33) aus. „WROT_RotorSpeed_Float“ besitzt jetzt eine eigene Achse und die Daten im Graphen werden getrennt voneinander angezeigt. Sie können jetzt noch weitere Einstellungen ändern, zum Beispiel die Position der Achse nach rechts verlegen mit „Axis aligement“ (34). Schließen Sie das Fenster danach mit „Close“ (35)

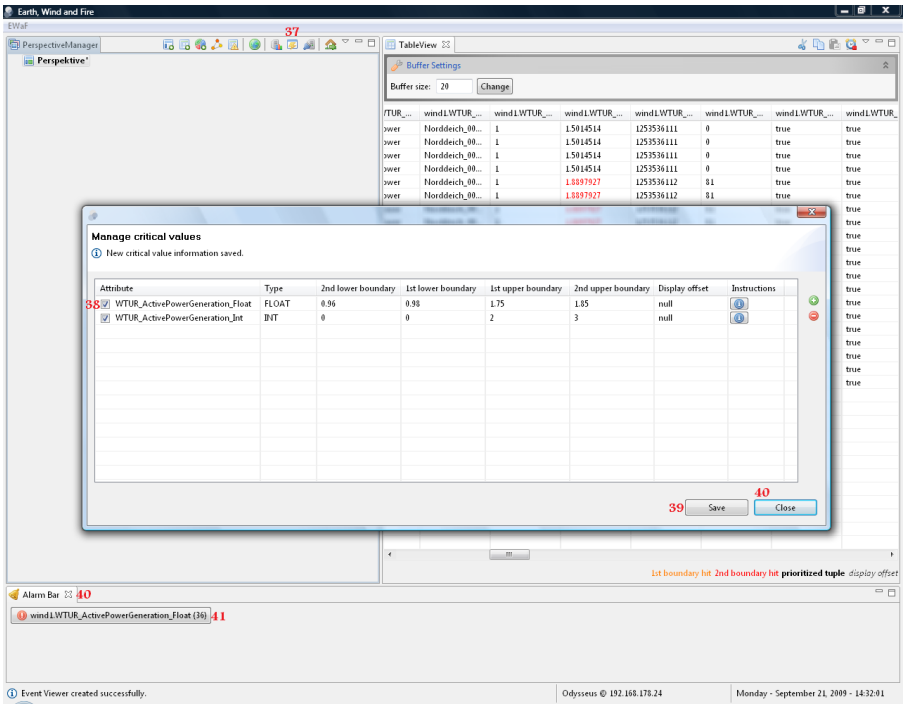
Um den Graphen zu entzerren deaktivieren Sie das „autoadjust“ (29). Schließen Sie den Graphen mit einem Klick auf das Kreuz (36).

Erstellen Sie jetzt einen Graphen mit Tortendiagramm, gehen Sie dafür so vor wie beim ersten Graphen, wählen hierfür jedoch statt dem Liniendiagramm (22) das Tortendiagramm (23) aus. Schließen Sie auch dieses Fenster wie gewohnt.

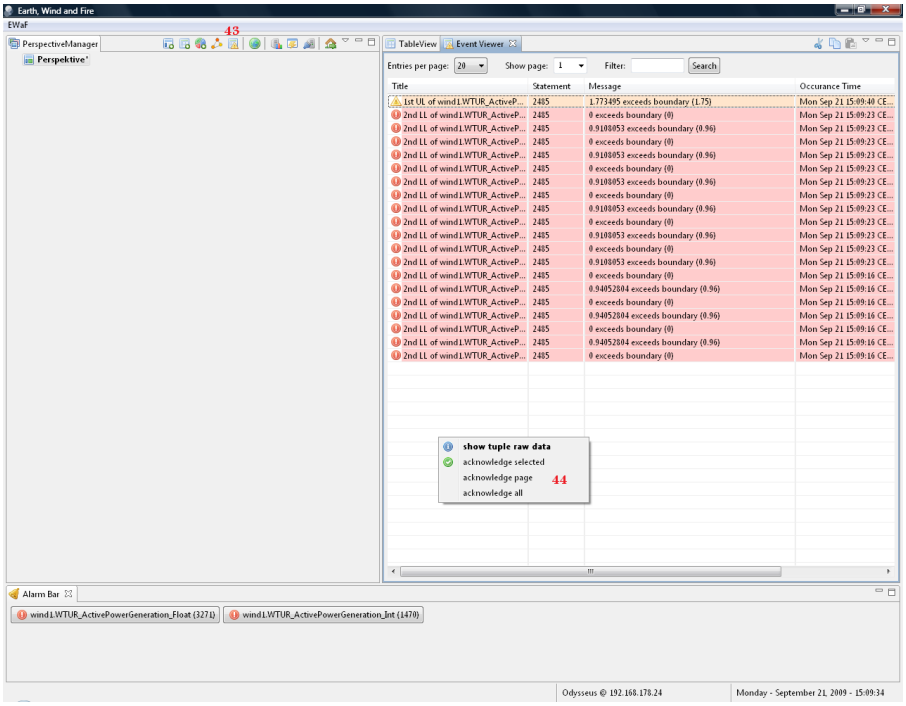
Erstellen Sie jetzt einen Graphen mit Balkendiagramm, gehen Sie dafür so vor wie beim ersten Graphen, wählen hierfür jedoch statt dem Liniendiagramm (22) das Balkendiagramm (24) aus. Wählen Sie auch hier wie beim Liniendiagramm die Quellen aus. Schließen Sie auch dieses Fenster wie gewohnt.

Erstellen Sie jetzt einen Graphen mit Thermometer, gehen Sie dafür so vor wie beim ersten Graphen, wählen hierfür jedoch statt dem Liniendiagramm (22) das Thermometer (25) aus. Wählen Sie auch hier wie beim Liniendiagramm die Quellen aus. Hier eignet sich „WROT_HubTemperature_Int“ als Wert. Schließen Sie auch dieses Fenster wie gewohnt.

11.3.5 Kritische Werte

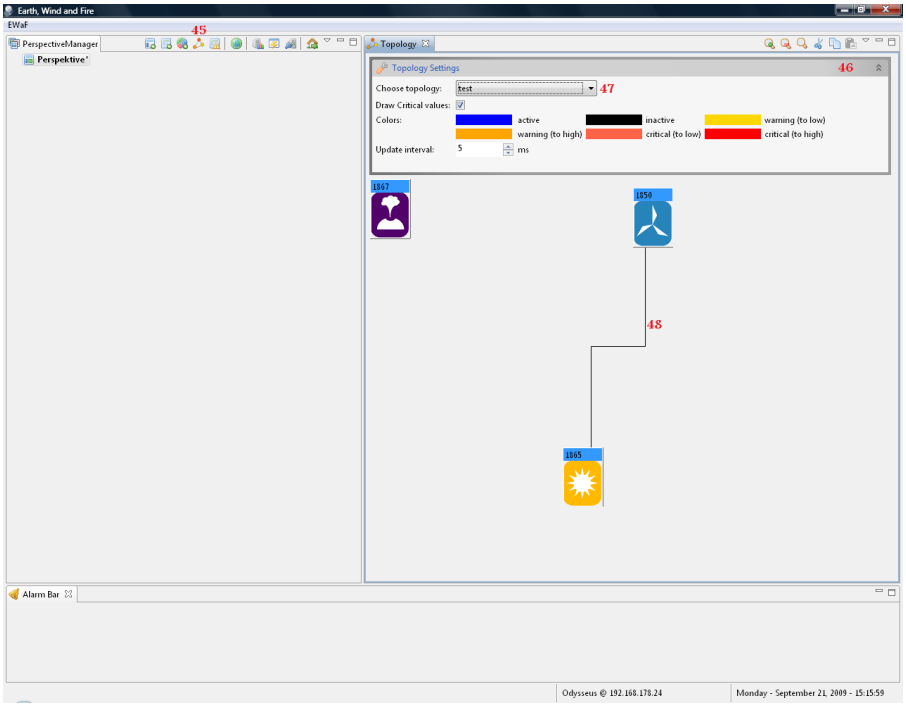


Im jetzt aktuellen Fenster sollte die Tabelle immer noch vorhanden sein. Wenn nicht, erstellen Sie eine neue. Öffnen Sie jetzt den Dialog „Modify critical value information“ (37) und geben hier die Informationen zu den Kritischen Werten wie abgebildet ein, speichern Sie die Einstellungen mit „Save“ (39) und schließen Sie das Fenster mit „Close“ (40).



Bei einer Werteüber- oder -unterschreitung wird jetzt in der Alarm Bar (41) ein Event (42) angezeigt. Bei einem Klick auf ein Event oder die Schaltfläche „Open Event Viewer“ (43) öffnet sich der Event Viewer. Über das Kontextmenü der rechten Maustaste (44) lassen sich einzelne oder sämtliche Events quittieren. Quittieren Sie hier sämtliche Events.

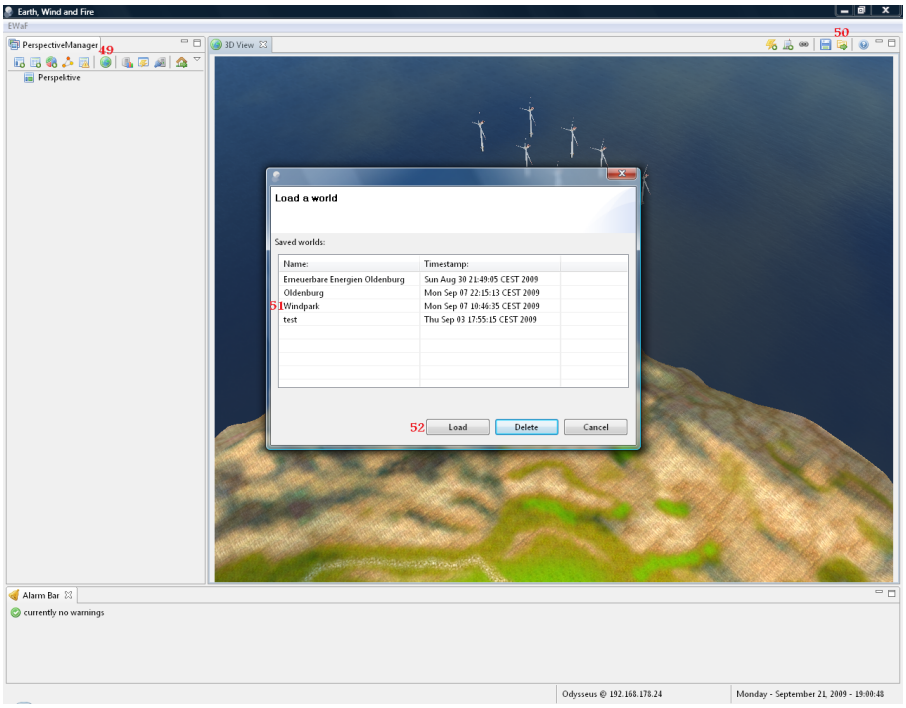
11.3.6 Topologie



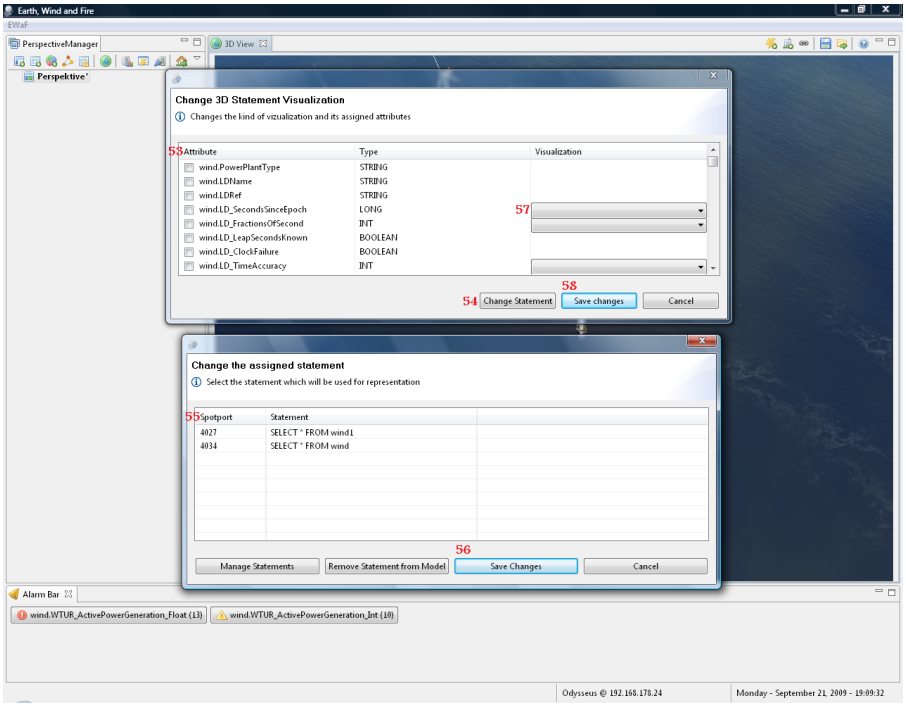
Öffnen Sie die Topologie mit einem Klick auf „Shows topology of connected sensors“ (45). Öffnen Sie die „Topology Settings“ (46) und wählen unter „Chose Topology“ (47) Ihre gespeicherte Welt aus.

11.3.7 3D-Welt

Starten Sie die 3D-Welt über den Button „New 3d world“ (49) und laden eine gespeicherte Welt.



Mit einem Klick auf „load World“ (50) öffnet sich ein neues Fenster, in dem Sie die entsprechende Wlet in der Liste (51) auswählen und mit „load“ (52) laden



Klicken Sie rechts auf ein beliebiges Kraftwerk. Im sich nun öffnenden Fenster sollte die Tabelle (53) noch leer sein. Klicken Sie jetzt auf „Change Statement“ (54). Im neuen Fenster wählen Sie ein passendes Statement aus der Liste (55) aus und bestätigen die Auswahl mit „Save Changes“ (56). Wählen Sie aus der jetzt gefüllten Liste (53) ein Attribut aus und setzen Sie den Haken. Für Windkraftanlagen bietet sich wieder „WROT_HubTemperature_Int“ an. Im Dropdown-Menü (57) wählen Sie für dieses Attribut die Art der Modelmanipulation. Das Erstellen eines neuen Kraftwerks funtkioniert auf die gleiche Weise.

11.4 Eingaben

Diese Liste fasst mögliche Eingaben zusammen und bietet Platz zum Notieren wichtiger Informationen für die Präsentation.

- SpotHost IP - _ _ _ . _ _ _ . _ _ _ . _ _ _
- Plant-Port -
- Plant-Typ -
- Odysseus IP - _ _ _ . _ _ _ . _ _ _ . _ _ _
- 3 - Perspektive
- 9 - wind1
- 10 - SpotHost IP
- 11 - Plant-Port
- 12 - Plant-Typ
- 38 -
  - WTUR_ActivePowerGeneration_Float
  - FLOAT
  - 0.96
  - 0.98
  - 1.75
  - 1.85
  - null
  - WTUR_ActivePowerGeneration_Int
  - INT
  - 0
  - 0
  - 2
  - 3
  - null

Kapitel 12

Messeflyer

12.1 Daten

- Offenes Format: 313mm x 210mm
- geschlossenes Format: DIN Lang (105mm x 210mm)
- 6-seitig, Wickelfalz
- 300 DPI
- Vorderseite: Seite 5 (103mm), Seite 6 (105mm), Seite 1 (105mm)
- Rückseite: Seite 2 (105mm), Seite 3 (105mm), Seite 4 (103mm)
- verwendete Bilder:

  Fotos von www.aboutpixel.de: Bilder sind für kommerzielle Zwecke freigegeben [abo09]

  SunSPOT von www.sunspotworld.com: Freigegeben für Lehrzwecke, in Form von Werbung und Dokumentation [Sun09]

12.2 Flyer

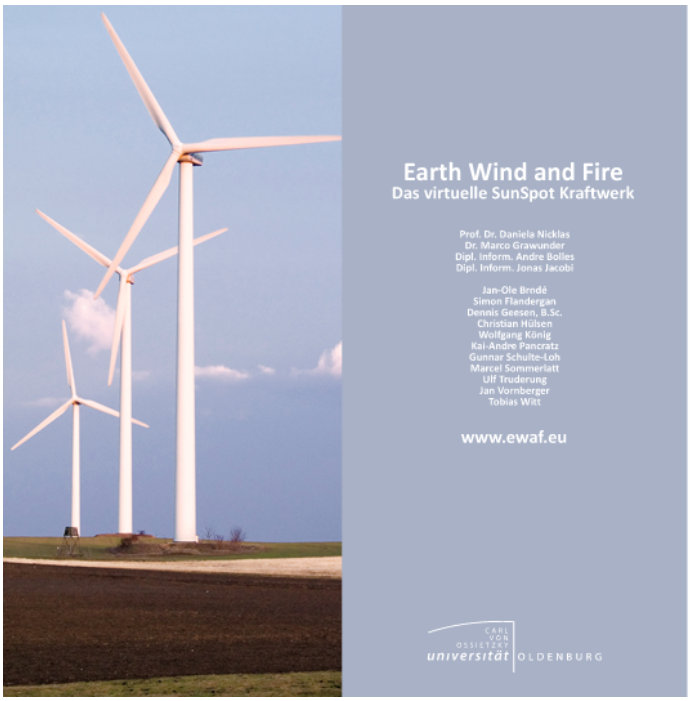


Abbildung 12.1: Vorderseite des Messeflyers

Abbildung 12.2: Rückseite des Messeflyers

Kapitel 13

Glossar

- **Administrationsmodus** - Erweiterte Arbeitsumgebung für Administratoren mit zusätzlichen Einstellungsmöglichkeiten.
- **Administrator** - Benutzer mit umfassenden Zugriffsrechten.
- **Ansichten/Perspektiven** - Ansichten/Perspektiven fassen eine Menge von Graphen und/oder Tabellen in einer Seite zusammen. Diese Perspektive kann gespeichert werden und bei Bedarf wieder aufgerufen werden. So soll der Benutzer zu jeder Zeit in seiner gewohnten Umgebung die Daten betrachten können.
- **Archiving** - Automatische Speicherung von Datenströmen in einer Datenbank mit der Möglichkeit, diese Daten wieder aus der Datenbank auszulesen und neue Datenströme zu erzeugen.
- **Continuous Query Language** - (CQL) Eine deklarative Anfragesprache für Datenströme in Datenstrommanagementsystem.
- **Datenhaltungssystem** - Software zur persistenten Speicherung von Daten.
- **Datenquelle** - Die Datenquelle (Source) ist der Ursprungsort, an dem die Daten generiert wurden
- **Datenstrom** - Kontinuierliche Folge von Datensätzen, die über einen Quelle an ein Datenstrommanagementsystem gesendet werden.
- **Datenstrommanagementsystem** - (DSMS) Software zum Empfangen, Verarbeiten und Weiterleiten von Datenströmen.
- **Demonstrator** - Software nur zur Technologiepräsentation, nicht zur Verwendung durch Endverbraucher gedacht.
- **Extensible Markup Language** - (XML) Die Extensible Markup Language ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten. XML wird u. a. für den Austausch von Daten zwischen Computersystemen eingesetzt.
- **Geothermal** - Geothermal ist die im zugänglichen Teil der Erdkruste gespeicherte Wärme. Sie umfasst die in der Erde gespeicherte Energie, soweit sie entzogen und genutzt werden kann, und zählt zu den regenerativen Energien. Sie kann sowohl direkt genutzt werden, etwa zum Heizen und Kühlen im Wärmemarkt (Wärmepumpenheizung), als auch zur Erzeugung von elektrischem Strom oder in einer Kraft-Wärme-Kopplung.
- **Graphical User Interface** - (GUI) Grafische Benutzeroberfläche, über die der Benutzer mit der Software interagieren kann.
- **Internationale Elektrotechnische Kommission** - (IEC) ein internationales Normierungsgremium mit Sitz in Genf für Normen im Bereich der Elektrotechnik und Elektronik.
- **Kioskmodus** - Gesicherte Arbeitsumgebung für Messevorführungen.
- **Kraftwerktypen** - Es werden drei unterschiedliche Kraftwerke simuliert: Windkrafträder, Geothermiekraftwerke und Solarkraftwerke.

- **LogicalNodeSpot** - (LNSpot) Diese SunSpots stellen als logische Knoten die Sensoren dar. Sie ermitteln einzelne Daten und senden dies an die LDSpots.
- **LogicalDeviseSpot** - (LDSpot) Diese SunSpots aggregieren aus den Daten der LNSpots ein IEC-konformes XML-Dokument und senden dieses über den SpotHost an ODYSSEUS.
- **Messebesucher** - Besucher auf Messen und Vorführungen ohne Vorkenntnisse des Systems.
- **Odysseus** - Unser Datenstrommanagementsystem.
- **Regenerative Energien** - Als regenerative Energien bezeichnet man nachhaltige Energiequellen. Sie bleiben - nach menschlichen Zeiträumen gemessen - kontinuierlich verfügbar und stehen hiermit im Gegensatz zu fossilen Energieträgern und konventionellen Kernbrennstoffen, deren Vorkommen bei kontinuierlicher Entnahme stetig abnimmt.
- **Remote Method Invocation** - (RMI) Der Aufruf einer Methode eines entfernten Java-Objekts.
- **Solar**- Solartechnik ist die Nutzbarmachung der solaren Einstrahlung unserer Sonne mittels technischer Hilfsmittel und Einrichtungen. Aus der Sonnenstrahlung kann Wärme / Hitze / Prozesswärme oder auch elektrische Energie gewonnen werden.
- **SunSpot** - Batteriebetriebene Plattform für drahtlose Sensorennetzwerke mit Licht-, Bewegungs- und Temperatursensoren.
- **View** - Oberfläche zum Anzeigen und Abfragen der Daten aus dem Datenstrommanagementsystem
- **3D-View** - 3-Dimensionale OpenGL Welt zum Anzeigen der Daten aus dem Datenstrommanagementsystem
- **Windkraft** - Bei der Windenergie handelt es sich um die kinetische Energie der bewegten Luftmassen der Atmosphäre. Sie ist eine indirekte Form der Sonnenenergie und zählt deshalb zu den erneuerbaren Energien. Die Windenergie-Nutzung mittels Windkraftanlagen ist eine seit dem Altertum bekannte Möglichkeit, um Energie aus der Umwelt zu schöpfen.
- **Wrapper** - Schnittstelle zwischen zwei Programmen. Dient als Übersetzer zwischen verschiedenen Sprachen.

Index

3D-Welt
    Subsystem, 36

Dokumentenphase
    Arbeitspakete, 117
    Meilensteine, 117

Energiequellen, 9
Erste Iteration
    Ablauf, 110
    Arbeitspakete, 109
    Meilensteine, 108
    Planung, 18

Funktionstest, 101

Graphen, 18

IEC 61400-25, 12
IEEE 802.15.4, 91
IEEE 829-1998, 99
Integrationstest, 100

Java Micro Edition, 91

LDFSPOT, 92
Leistungstest, 101
LNSPOT, 91

Odysseus, 9
    Subsystem, 36

Seminarphase, 108
Squawk Java Virtual Machine, 91
SunSpot, 9
    Subsystem, 36

Tabellen, 18

View
    Subsystem, 36

Zweite Iteration
    Ablauf, 115
    Arbeitspakete, 114
    Meilensteine, 113
    Planung, 21

Literaturverzeichnis

[abo09] aboutpixel.de: *Aboutpixel AGB*. http://www.aboutpixel.de/agb.php4, 2009. – letzter Zugriff: 24.09.2009

[FF05] Fernandez, John D. ; Fernandez, Andres E.: SCADA systems: vulnerabilities and remediation. In: *J. Comput. Small Coll.* 20 (2005), Nr. 4, S. 160–168. – ISSN 1937–4771

[IEC06] IEC: *IEC61400-25 - Wind turbines - Part 25-2: Communications for monitoring and control of wind power plants - Information models*, 2006

[Mar08] Mark Barnes and Ellen Levy Finch and Sony Computer Entertainment Inc.: *COLLADA - Digital Asset Schema Release 1.4.1 Specification*. http://www.khronos.org/files/collada_spec_1_4.pdf, 2008

[Pow09] Powel, Mark: *jMonkeyEngine.com*. http://www.jmonkeyengine.com, 2009. – letzter Zugriff: 24.09.2009

[SCC+06] Simon, Doug ; Cifuentes, Cristina ; Cleal, Dave ; Daniels, John ; White, Derek: *Java(TM) on the Bare Metal of Wireless Sensor Devices – The Squawk Java Virtual Machine*. http://research.sun.com/projects/squawk/docs/vee06-squawk.pdf, 2006

[SM09] Sun Microsystems, Inc.: *Java ME: the Most Ubiquitous Application Platform for Mobile Devices*. http://java.sun.com/javame/index.jsp, 2009. – letzter Zugriff: 30.09.2009

[Soc06] Society, IEEE C.: *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*. http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf, 2006

[Sun09] Sun Inc.: *Sun's Copyright Usage Requirements*. http://www.sun.com/aboutsun/copyright/usage.jsp, 2009. – letzter Zugriff: 24.09.2009

[TS07] Tanenbaum, Andrew S. ; Steen, Maarten van: *Verteilte Systeme: Prinzipien und Paradigmen*. Addison Wesley Verlag, 2007

[Zel04] Zelkowitz, Marvin: *Advances in Computers: Architectural Issues*. Bd. 61. ACADEMIC PR INC, 2004