Architekturentwurf

Master Infrastructure Situation Display Observing Windows and Linux

Ein System zur Überwachung von vernetzten Rechnern

P. Brombosch, D. Krauss, F. Müller, Y. Noller, J. Scheurich

Universität Stuttgart
Studenten der Fachrichtung Softwaretechnik

Erstellt am: 10. Juli 2012
Freigegeben am: 24. März 2013
Version: Version 2.0

Inhaltsverzeichnis

1 Einleitung **11**

1.1 Überblick über das Projekt 11

1.2 Über dieses Dokument 11

1.3 Grundlagen für dieses Dokument 12

1.4 Leserkreis 12

1.5 Namenskonventionen für dieses Dokument 12

2 Grundsätzliche Entwurfsüberlegungen **13**

2.1 Entwurfsmuster 13

2.2 Architekturmuster 15

2.2.1 Prism 15

2.2.2 Model View ViewModel 15

3 Komponentenentwurf **19**

3.1 Server 20

3.1.1 Services 21

3.1.2 Scheduling 22

3.1.3 Manager 23

3.1.4 ClusterConnection 23

3.1.5 Database 23

3.2 Client 23

3.3 Workstation 23
3.3.1 ServerConnection 24
3.3.2 Plugins 24
3.3.3 Scheduling 24
3.4 Cluster 24

4 Feinentwurf Core **25**

4.1 Scheduling 25
4.1.1 SchedulerBase 25
4.1.2 TimerJobBase 26
4.2 States 27
4.2.1 MappingState 27
4.2.2 WorkstationState 27
4.3 DataType 28
4.4 ClusterConnection 28
4.5 IndicatorSettings 29
4.6 IPlugin 29
4.7 IPluginExtensions 30
4.8 LogType 30
4.9 Logger 31
4.10 Platform 31
4.11 PluginFile 32
4.12 PluginMetadata 32
4.13 StringExtensions 33
4.14 Layout 33
4.15 WorkstationInfo 33

5 Feinentwurf Server **35**

5.1 Bootstrapper . . . 35

5.2 Cluster . . . 35

5.2.1 BrightClusterConnection . . . 36

5.2.2 BrightClusterShell . . . 36

5.2.3 HpcClusterConnection . . . 37

5.3 Database . . . 37

5.3.1 Datenbank . . . 37

5.3.2 DataContextFactory . . . 38

5.3.3 PrecompiledQueries . . . 38

5.4 Email . . . 39

5.4.1 Tagesbericht Template . . . 39

5.4.2 WarningMailParser . . . 40

5.4.3 Mailer . . . 42

5.5 Manager . . . 42

5.5.1 ADManager . . . 43

5.5.2 OUManager . . . 43

5.5.3 ClusterMananger . . . 44

5.5.4 MetaClusterManager . . . 44

5.5.5 FilterManager . . . 45

5.5.6 MetricManager . . . 45

5.5.7 UpdatIntervalManager . . . 46

5.5.8 PluginManager . . . 47

5.5.9 UIConfigManager . . . 48

5.5.10 ValueManager . . . 48

5.5.11 WorkstationManager . . . 49

5.6 Scheduling . . . 50
5.6.1 CleanerJobScheduler . . . 50
5.6.2 CleanerJob . . . 50
5.6.3 GlobalScheduler . . . 51
5.6.4 GlobalTimerJob . . . 51
5.6.5 ClusterTimerJob . . . 52
5.6.6 MailScheduler . . . 52
5.6.7 DailyMailTimerJob . . . 53
5.6.8 MainScheduler . . . 53
5.6.9 MainRefreshTimerJob . . . 54
5.7 Services . . . 54
5.7.1 ClientWebService . . . 55
5.7.2 WorkstationWebService . . . 55

6 Feinentwurf Workstation **56**
6.1 ServerConnection . . . 56
6.2 WorkstationLogger . . . 57
6.3 Plugins . . . 57
6.3.1 PluginManager . . . 58
6.4 Scheduling . . . 58
6.4.1 Scheduler . . . 59
6.4.2 IndicatorTimerJob . . . 59
6.4.3 MainUpdateTimerJob . . . 60

7 Feinentwurf Client **61**
7.1 Model . . . 62
7.1.1 DataModel . . . 62

7.1.2 MenuState 63
7.2 ViewModel 63
7.2.1 ViewModel 64
7.2.2 Converter 64

8 Interaktion der Komponenten **71**
8.1 Interaktion von einem Client ausgehend 71
8.1.1 Layout-Verwaltung 71
8.2 Serverseitige Interaktion 74
8.2.1 Visualisierungs-Daten senden 74
8.2.2 Visualisierungs-Plugin senden 75
8.2.3 Eine Workstation zur Ignore-Liste hinzufügen 75
8.2.4 Eine Workstation in den Wartungszustand versetzen 76
8.2.5 Workstation anmelden/registrieren 76
8.2.6 Workstation abmelden 79
8.2.7 Kenngroessenwert empfangen 79
8.3 Interaktion von einer Workstation ausgehend 80
8.3.1 Anmelden beim Server 80
8.3.2 Abmelden vom Server 81
8.3.3 *Kenngrößenwerte* senden 82

9 Datenhaltung **84**
9.1 Datenbank 84
9.1.1 Überblick 84
9.1.2 Listen und Gruppen 85
9.1.3 Benachrichtigungen 86
9.1.4 GUI Einstellungen 87

9.1.5 Cluster Credential 87
9.1.6 Datenerfassung 88
9.2 XML-Dateien 90
9.2.1 Globale Einstellungen 90
9.2.2 Client Einstellungen 91
9.2.3 Plugineinstellungen 91
9.3 Speicherorte 92

10 Zustände **94**

11 Schnittstellen **96**

11.1 Datenstrukturen 96
11.2 Workstation Web Service 100
11.3 Client Web Service 104

A Anhang **117**

A.1 Begriffslexikon 117
A.2 Versionshistorie 131

Abbildungsverzeichnis

3.1 Komponentendiagramm 20

4.1 Klasse SchedulerBase 25
4.2 Klasse TimerJobBase 26
4.3 Die Enumeration MappingState 27
4.4 Die Enumeration WorkstationState 27
4.5 Die Enumeration DataType 28
4.6 Die abstrakte Klasse IClusterConnection 28
4.7 Klasse IndicatorSettings 29
4.8 Interface IPlugin 29
4.9 Die statische IPluginExtensions 30
4.10 Die Enumeration LogType 30
4.11 Klasse Logger 31
4.12 Die Enumeration Platform 31
4.13 Klasse PluginFile 32
4.14 Klasse PluginMetadata 32
4.15 Statische Klasse StringExtensions 33
4.16 Klasse Layout 33
4.17 Klasse WorkstationInfo 33

5.1 Klasse Bootstrapper 35

5.2 Klasse BrightClusterConnection 36
5.3 Klasse BrightClusterShell 36
5.4 Klasse HpcClusterConnection 37
5.5 Klasse DataContextFactory 38
5.6 Klasse PrecompiledQueries 38
5.7 Klasse DailyMailTemplate 39
5.8 Klasse DailyMailTemplateData 40
5.9 Klasse TemplateTag 40
5.10 Klasse WarningMailParser 41
5.11 Klasse Mailer 42
5.12 Klasse ADManager 43
5.13 Klasse OUManager 43
5.14 Klasse ClusterMananger 44
5.15 Klasse MetaClusterManager 44
5.16 Klasse FilterManager 45
5.17 Klasse MetricManager 45
5.18 Klasse UpdatIntervalManager 46
5.19 Klasse PluginManager 47
5.20 Klasse UIConfigManager 48
5.21 Klasse ValueManager 48
5.22 Klasse WorkstationManager 49
5.23 Klasse CleanerJobScheduler 50
5.24 Klasse CleanerJob 50
5.25 Klasse GlobalScheduler 51
5.26 Klasse GlobalTimerJob 51
5.27 Klasse ClusterTimerJob 52

5.28 Klasse MailScheduler 52
5.29 Klasse DailyMailTimerJob 53
5.30 Klasse MainScheduler 53
5.31 Klasse MainRefreshTimerJob 54
5.32 Klassen Web Services 54

6.1 Klasse ServerConnection 56
6.2 Klasse WorkstationLogger 57
6.3 Klasse PluginManager 58
6.4 Klasse Scheduler 59
6.5 Klasse IndicatorTimerJob 59
6.6 Klasse MainUpdateTimerJob 60

7.1 Klasse DataModel 62
7.2 Enum MenuState 63
7.3 Klasse ViewModel 64

8.1 Layout hinzufügen 72
8.2 Layout laden 73
8.3 Visualisierungs-*Plugins* laden 74
8.4 Visualisierungs-Daten senden 75
8.5 Visualisierungs-*Plugins* senden 75
8.6 Eine *Workstation* zur *Ignore-Liste* hinzufügen 76
8.7 Eine *Workstation* in den *Wartungszustand* versetzen 76
8.8 Workstation einchecken/registrieren 78
8.9 Workstation auschecken 79
8.10 *Kenngrößenwert* empfangen 80
8.11 Anmelden beim *Server* 81

8.12 Abmelden vom *Server* 82
8.13 *Kenngrößenwert* senden 83

9.1 ER-Diagramm: Überblick über die Datenbankstruktur 85
9.2 ER-Diagramm: Listen und Gruppen 86
9.3 ER-Diagramm: Benachrichtigungen 87
9.4 ER-Diagramm: Einstellungen 87
9.5 ER-Diagramm: Datenerfassung 89

10.1 Zustand einer *Workstation* 95

Kapitel 1

Einleitung

1.1 Überblick über das Projekt

Dieser Architekturentwurf beschreibt die Architektur des Projekts *MISD OWL* der Universität Stuttgart. Im Rahmen des Studienprojekts 2012 des Instituts VISUS soll ein System zur Überwachung von vernetzten Rechnern erstellt werden.

MISD OWL soll in der Lage sein, verschiedene Systeme (Workstations und Cluster) mithilfe von flexibel erweiterbaren *Plugins* zu überwachen. Diese Überwachung soll zentral verwaltet werden und anschließend sowohl auf *Desktop*-Systemen, als auch auf den *Powerwalls* des Instituts visualisiert werden.

1.2 Über dieses Dokument

Dieses Dokument beschreibt die Architektur der *Client*-Anwendung, des *Server-Diensten*, der *Dienste* auf den *Workstations* und die Datenbankarchitektur. Die einzelnen Architekturen werden mit Hilfe von Diagrammen erläutert. Die verwendeten Architekturmuster werden vorgestellt, bewertet und der Entscheidungsweg erläutert. Die Feinentwürfe werden im Laufe des Projekts in dieses Dokument eingefügt und dienen als Grundlage der Implementierung. Die Entwickler sorgen dafür, dass dieses Dokument jederzeit aktuell und konsistent gehalten wird.

1.3 Grundlagen für dieses Dokument

Grundlage für dieses Dokument ist die Spezifikation von MISD.

1.4 Leserkreis

Zum Leserkreis dieses Dokuments gehören:

- Die Entwickler des Systems
- Der Kunde
- Die Betreuer dieses Studienprojekts
- Die Gutachter des Reviews
- Personen, die dieses Projekt später weiterentwickeln, erweitern oder warten

1.5 Namenskonventionen für dieses Dokument

- Begriffe, die Referenzen auf das Begriffslexikon darstellen, werden *kursiv* geschrieben.
- Besonders wichtige Informationen oder hervorzuhebende Teile werden **fett** geschrieben.
- Verweise auf externe Informationen werden als Fußnoten dargestellt.

Die Klassendiagramme enthalten keine Parameter. Diese sind den Kopfkommentaren der Methoden zu entnehmen.

Kapitel 2

Grundsätzliche Entwurfsüberlegungen

Diesem Entwurf liegen einige Einschränkungen, welche bereits in der Spezifikation beschrieben wurden, zu Grunde. Diese sind zum Teil durch den Kunden als auch durch die Entwickler definiert worden. Auf diese Einschränkungen wird im Entwurf Rücksicht genommen. Im Folgenden werden verschiedene Entwurfsmuster erläutert, welche in der Entwicklung zum Einsatz kommen sollen.

2.1 Entwurfsmuster

In diesem Kapitel wird beschrieben welche Entwurfsmuster[1] benutzt werden und wie sie im *System* angewendet werden sollen. Folgende Entwurfsmuster sollen verwendet werden:

- Singleton:
  Mit dem Singleton-Pattern soll sichergestellt werden, dass es zu einer Klasse höchstens eine Instanz gibt. Die Singleton-Klassen sind threadsicher implementiert[2]. Auf dem *Server* werden Singleton-Objekte für die Manager (z.B. *Filter*, *Plugin* oder das *Active Directory* und Scheduler) verwendet.

- Composite:
  Mit dem Composite-Pattern sollen hierarchische Teil-Ganzes-Beziehungen modelliert werden. Einzelne Objekte (Primitive) und zusammengesetzte Objekte (Container) sollen für

[1] nach Gamma, E. u.a., "Design Patterns, Elemets of Reusable Object-Oriented Software", Addison-Wesley, 1995

[2] MSDN: Implementing Singleton in C# `http://msdn.microsoft.com/en-us/library/ff650316.aspx`

den Verwender die gleiche Schnittstelle haben. Dazu wird eine gemeinsame Oberklasse für Primitive und Container modelliert, die beide Eigenschaften in sich vereinigt. Dieses Entwurfsmuster passt besonders gut für die Repräsentation der *OrganisationalUnits* (siehe Abbildung 9.1) im Kernsystem.

- Strategie:
  Mit dem Strategie-Pattern sollen Klassen so entworfen werden, dass weitere Algorithmen für bestimmte Probleme noch ergänzt werden können. Es gibt für jedes Problem (mit nicht eindeutiger Lösung) eine Oberklasse, die eine abstrakte Methode besitzt, die das Problem lösen soll. Dann können beliebig viele Klassen von dieser Oberklasse erben und jeweils eine eigene Problemlösung bereitstellen. Dieses Pattern wird bei den TimerJobs der Scheduler verwendet. Die Implementierung der TimerJobs legt die wiederkehrende Aufgabe fest.

- Observer:
  Das Observer-Pattern löst folgende Problemstellung: Der Zustand eines Objektes ändert sich und als Folge davon sollen all die Objekte, die sich auf das geänderte Objekt beziehen, über die Änderung informiert werden. Die Observer registrieren sich bei dem zu beobachteten Objekt und sobald sich das Objekt ändert, führen alle Observer ihre Aufgabe durch. Mit einer Oberklasse oder einem Interface wird sichergestellt, dass jeder Observer diese update-Methode besitzt. Ein Anwendungsbeispiel ist die Visualisierung im *System*. Die Datenbank ändert sich, falls Änderungen in der Visualisierung durch Interaktion vorgenommen werden. Aber auch bei der Event-Überwachung auf den *Workstations* kann das Entwurfsmuster eingesetzt werden. Auf dem Server findet dieses Pattern Verwendung bei der Aktualisierung der Pluginmetadaten auf der Datenbank mittels den in *Plugin*-Assemblys gespeicherten Informationen.

- Factory
  Das Factory-Pattern stellt eine abstrakte Klasse zur Verfügung, die ein Produkt instanziert, dass von einer Unterklasse definiert wird.

Die genannten Entwurfsmuster sollen im jeweiligen Feinentwurf nochmals auf die spezielle Eignung hin geprüft werden und anschließend, je nach Resultat der Prüfung, in die Klassenstruktur eingearbeitet werden.

2.2 Architekturmuster

In diesem Abschnitt werden die zum Einsatz kommenden Architekturmuster erläutert.

2.2.1 Prism

Die MISD-OWL Entwickler haben sich entschieden zur Umsetzung einiger Architekturmuster das externe Projekt Prism[3] zu verwenden. Prism ist ein Projekt, das sich an Entwickler richtet, die WPF oder Silverlight Programme schreiben. Es soll bei der Verwendung verschiedener Design Patterns eine Hilfestellung sein. Prism bietet außer einer Klassenbibliothek umfangreiche Beispiel- und Referenzimplementierungen sowie eine ausführliche Dokumentation. Prism soll in der Version 4.0 als Richtlinie zur Realisierung des MVVM Musters und zur Modularisierung der MISD-OWL Software beitragen. Die Prism Dokumentation hilft dabei mit der Entscheidungsfindung in den meisten relevanten Fragen.

2.2.2 Model View ViewModel

Dieser Abschnitt erläutert das Architekturmuster *Model View ViewModel (MVVM)*, das besonders für WPF-Anwendungen genutzt wird und bei der *Client*-Anwendung zum Einsatz kommt. *MVVM* trennt die grafische Oberfläche von der Datenhaltung und von der Präsentations- und Geschäftslogik.

View

Die View legt die grafische Oberfläche fest und ändert gegebenfalls grafische Elemente ab. Die View enthält keine Verarbeitungs-Logik.

Die View beinhaltet beispielsweise Frames, Controls, Data Binding oder Data Templates. Das ViewModel wird von der View als DataContext referenziert. Die View ist dem ViewModel dagegen nicht bekannt. Die Controls der View werden an die passenden Eigenschaften im ViewModel oder im Model gebunden. Die View nutzt die Methoden des ViewModels zur Dateneinund Ausgabe.

[3] http://compositewpf.codeplex.com/

ViewModel

Das ViewModel kapselt die Präsentations-Logik von der grafischen Oberflächen-Logik ab. Dem ViewModel ist das Model bekannt, nicht aber die View. Das ViewModel implementiert die Funktionen der grafischen Oberfläche und koordiniert die Interaktion von View und Model. Dazu bereitet das ViewModel Daten für die Oberfläche auf, bearbeitet Einstellungen, die das Model nicht betreffen, verwaltet den Status der grafischen Oberfläche und validiert und bearbeitet Eingaben, so dass diese vom Model genutzt werden können.

An das ViewModel können mehrere Models gebunden werden, die Geschäfts-Logik und Datenhaltung zur Verfügung stellen.

Grundregel View oder ViewModel

Alle visuellen Funktionen, die später bei einem Restyle (grafische Überarbeitung) der Oberfläche geändert werden, werden in der View implementiert. Alle Funktionen, die Bezug zur Geschäftslogik und der Datenhaltung haben, werden im ViewModel implementiert.

Beispiel: Farbe eines markierten Elements ist Teil der View. Die Auswahlmöglichkeiten eines DropDown werden im ViewModel verwaltet.

Model

Das Model verwaltet die Datenhaltung und die Geschäftslogik, wie zum Beispiel Überwachung der Datenkonsistenz, Validieren von Daten. Um die Wiederverwendbarkeit zu sichern, sind die Methoden des Models nicht auf einen UseCase oder eine Aufgabe beschränkt. Das Model stellt Methoden zur Verfügung um auf Änderungen einzelner Daten (**INotifyPropertyChanged**[4]) oder Collections von Daten (**INotifyCollectionChanged**[5]) per Data-Binding, zu reagieren. Models die Collections von Daten verwalten, beinhalten meist eine **ObservableCollection<T>**[6]. Meist werden die Daten selbst zusätzlich in Datenbanken oder weiteren Diensten gekapselt.

Das Model unterstützt das ViewModel in der Validierung durch die Interfaces **IDataErrorIn-**

[4] `http://msdn.microsoft.com/de-de/library/system.componentmodel.inotifypropertychanged.aspx`

[5] `http://msdn.microsoft.com/de-de/library/system.collections.specialized.inotifycollectionchanged.aspx`

[6] `http://msdn.microsoft.com/de-de/library/ms668604.aspx`

fo[7] und **INotifyDataErrorInfo**[8].

Das Model nutzt keine Methoden von View oder ViewModel.

Data Binding

Mittels unidirektionalem Data Binding ist es möglich Controls der View an das ViewModel zu binden, um Informationen zum Rendern der View vom ViewModel zu erhalten. Bidirektionales Data Binding ermöglicht zusätzlich die Weitergaben von Informationen aus der View zum ViewModel, um weitere Verarbeitungen zu ermöglichen.

Um die View-Klasse über Änderungen im ViewModel zu informieren müssen die Eigenschaften, die gebunden werden sollen, das **INotifyPropertyChanged** Interface implementieren. Soll eine Collection von Daten gebunden werden, muss diese Collection das **INotifyCollectionChanged** Interface implementieren oder von **ObservableCollection<T>** erben. Diese Interfaces feuern ein Event, wenn deren Eigenschaften geändert wurden. Die gebundenen Controls werden daraufhin automatisch aktualisiert.

Commands

Commands stellen die Aktionen und Operationen der Präsentations-Logik des ViewModel zur Verfügung. Diese können wiederum an Controls der View-Klasse gebunden werden.

Ein Command wird meist durch eine User-Interaktion wie zum Beispiel ein Mausklick, einem ShortCut, oder einem anderem Event aufgerufen. Commands können auch bidirektional implementiert werden, sodass ein Command, beispielsweise durch eine User-Interaktion, die grafische Oberfläche beeinflusst.

Die View-Klasse kann Commands durch **Command Methods** oder durch Implementierungen des Interfaces **ICommand**[9] implementieren. Es können auch Commands ohne Events direkt im code-behind-file implementiert werden.

[7] http://msdn.microsoft.com/de-de/library/system.componentmodel.idataerrorinfo.aspx

[8] http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifydataerrorinfo(v=vs.95).aspx

[9] http://msdn.microsoft.com/de-de/library/system.windows.input.icommand.aspx

Data Validation und Error Reporting

Oft müssen Interaktionen des Users vom ViewModel oder der Model-Klasse validiert werden und resultierende Fehlermeldungen an den User zurückgegeben werden. WPF unterstützt die Validierung von gebundenen Eigenschaften. Ist die **ValidatesOnExceptions**[10] Eigenschaft eines Data Binding *true*, kann WPF auf diese Exception reagieren und dem User Fehlermeldungen anzeigen.

Alternativ können die Klassen auch das **IDataErrorInfo** oder **INotifyDataErrorInfo** Interface implementieren. Diese Implementierung ermöglicht die Validierung eines ganzen Datensatzes mit mehreren Werten.

[10] `http://msdn.microsoft.com/de-de/library/system.windows.data.binding.validatesonexceptions.aspx`

Kapitel 3

Komponentenentwurf

Das System *MISD OWL* besteht aus mehreren einzelnen Komponenten, die in diesem Kapitel beschrieben werden. Zur besseren Übersicht folgt zuerst das Komponentendiagramm (siehe Abbildung 3.1) und anschließend die Beschreibung der einzelnen Komponenten.

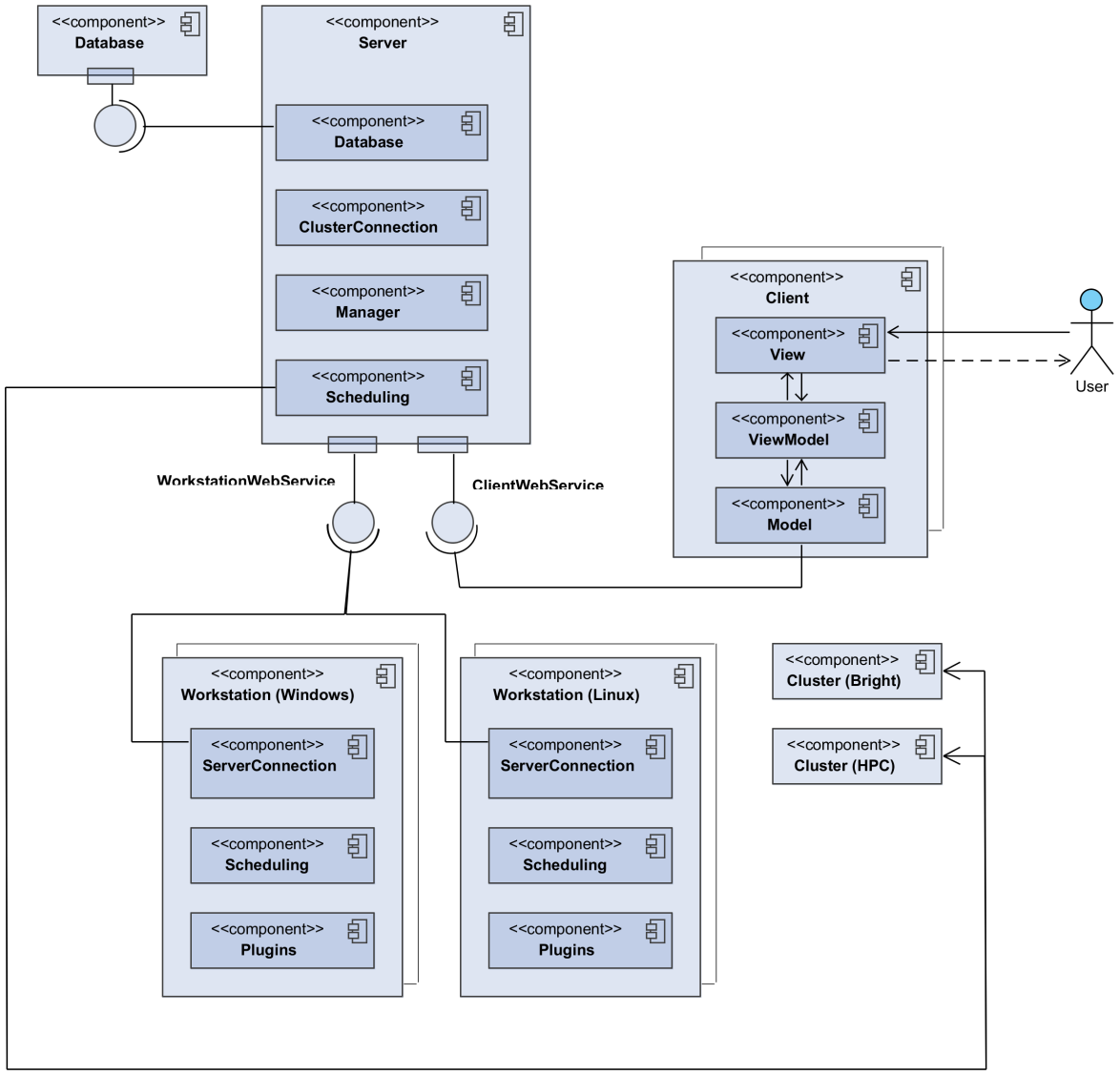

Abbildung 3.1: Komponentendiagramm

Eine allgemeine Beschreibung von *Server*, *Client* und *Workstation* befindet sich im Begriffslexikon.

3.1 Server

Die zentrale Komponente des *Systems* ist der *Server*. Dort werden alle Daten der *zu überwachenden Rechner* gesammelt, die anschließend von den *Client*-Anwendung abgefragt werden

können. Die *Server*-Komponente gibt es, wie im Komponentendiagramm (siehe Abbildung 3.1) gezeigt, nur einmal im gesamten *System*. Alle *Server*-Komponenten von *MISD OWL* sollen auch wirklich nur auf dem zentralen *Server* laufen und nicht im *System* verteilt sein. In der *Server*-Komponente befinden sich fünf Komponenten: Services, Scheduling, Manager, Cluster-Connection und Database. Die Komponente Services beinhaltet die beiden Web Services für *Clients* und *zu überwachende Rechner*. Alle zeitabhängigen wiederkehrenden Aufgaben werden von der Scheduler-Komponente angestoßen und verwaltet. Sämtliche Datenverwaltungs und Datenmanipulationsaufgaben und die *Plugin*-Verwaltung wird von der Manager-Komponente übernommen. Die Verwaltung und Datenerhebung der Cluster führt die Komponente Cluster aus. Die Komponente Database stellt den Datenbankzugriff mittels LINQ Technologie zur Verfügung und wird von einer Factory Klasse verwaltet.

Die Server-*Plugins* der Komponente Scheduling greifen auf die Cluster zur *Kenngrößenwerte*-Erhebung zu.

3.1.1 Services

WorkstationWebService

Der Web Service WorkstationWebService wird ausschließlich von der Komponente *Workstation* in den Ausprägungen Windows und Linux benutzt. Er bietet folgende Funktionen:

- *Aktualisierungsintervall* der *Dienste* abfragen
- am *System* an- und abmelden
- Upload der ermittelten *Kenngrößenwerte*
- Synchronisation der Windows/Linux *Plugins*
- Synchronisation der *Aktualisierungsintervalle*
- Synchronisation der *Filterbedingungen*
- Loggen von Fehlermeldungen

ClientWebService

Der Web Service ClientWebService wird ausschließlich von der Komponente Client verwendet. Folgende Funktionalitäten stehen zur Verfügung:

- Plugins aktualisieren
- Synchronisation der Visualisierungs-*Plugins*
- *Aktualisierungsintervalle* verändern
- *Filterbedingungen* verändern
- *Metriken* verändern
- *Layouts* der einzelnen *Benutzer* hoch- und herunterladen, ändern und löschen
- *zu überwachende Rechner* auf die *Ignore-Liste* setzen und herunternehmen
- *Wartungszustand* eines *zu überwachenden Rechners* aktivieren und deaktivieren
- *Organisationseinheiten* verwalten
- Speicherdauern der *Kenngrößen* verändern
- Gültigkeitsdauer des kritischsten Mappings verändern
- *Kenngrößenwerte* der *zu überwachenden Rechner* erhalten
- *Cluster* verwalten.

3.1.2 Scheduling

Die Komponente Scheduling stößt zeitabhängige Aufgaben an. Auf dem *Server* implementieren die Scheduler das Singleton und das Strategie Pattern (siehe Kapitel 2.1). Scheduler werden zur Datenerhebung, Email-Versand, Cleaning der Datenbank und Aktualisierung von Intervallen verwendet. Die Scheduler Instanzen arbeiten dabei immer mit der Komponente die gesteuert wird zusammen.

3.1.3 Manager

Die Manager Komponente auf dem *Server* implementiert das Singleton Pattern (siehe Kapitel 2.1) und ist für die komplette Datenmanipulation und Verwaltung zuständig. Konkret fallen in den Aufgabenbereich des Manager das *Active Directory*, *Filter*, *Organisationseinheiten*, *Plug-ins*, erfasste Daten und Intervalle.

3.1.4 ClusterConnection

Die Cluster Komponente des Servers stellt die jeweiligen Verbindungen zu einem HPC oder Bright *Cluster* bereit.

3.1.5 Database

Die Komponente Database ist eine externe Komponente des *Systems* und enthält eine MSSQL-Datenbank. Der Zugriff erfolgt mittels der SQL-Pass-Through-Technologie und wird durch die Komponente Database in der Komponente *Server* gesteuert. Die einzelnen Instanzen werden durch eine Factory verwaltet, die für die jeweilige Aufgabe optimierte Verbindungsklassen erstellt.

3.2 Client

Die Visualisierung der gesammelten Daten erfolgt in der Komponente *Client*. Dabei wird im Komponentendiagramm nicht zwischen *Desktop* und *Powerwall* unterschieden. Von dieser Komponente kann es mehrere Instanzen geben, je nach dem wie viele *Clients* im *System* vorhanden sind. Eine Instanz der *Clients*-Komponente setzt sich nach dem Architekturmuster *MVVM* zusammen (siehe Kapitel 2.2). Die Komponente *Client* interagiert in ihrer Komponente Model über den ClientWebService mit dem *Server*.

3.3 Workstation

Die Komponente *Workstation* ist in zwei Ausprägungen unterteilt: die Komponente *Workstation* für Windows und die Komponente *Workstation* für Linux. Im Folgenden wird aber nicht

zwischen diesen beiden unterschieden. Von der Komponente *Workstation* kann es mehrere Instanzen geben, je nach dem wie viele *Workstations* sich im *System* befinden. Diese Komponente interagiert mit dem *Server* über die Schnittstellen WorkstationWebService (siehe Kapitel 3.1.1).

3.3.1 ServerConnection

Die Komponente ServerConnection verwaltet die Verbindung zwischen *Workstation* und *Server*

3.3.2 Plugins

Die Komponente Plugins verwaltet die *Plugins* auf der *Workstation*.

3.3.3 Scheduling

Die Komponente Scheduling verwaltet zeitabhängig Aufgaben auf der *Workstation*. Insbesondere die Datenerfassung.

3.4 Cluster

Die Komponente *Cluster* ist momentan in zwei Ausprägungen vorgesehen: für einen Bright-Cluster und für einen HPC-Cluster. Weitere Ausprägungen können in das System aber eingefügt werden. Die Cluster Komponente stellt die jeweiligen Verbindungen zu einem *Cluster* bereit.

Kapitel 4

Feinentwurf Core

Die Assembly Core stellt Klassen zur Verfügung die von allen Komponenten *Server*, *Client* und *Workstation* genutzt werden. Dies sind vorrangig Interfaces, abstrakte Superklassen und komplexe Datentypen.

4.1 Scheduling

4.1.1 SchedulerBase

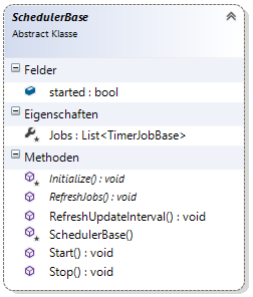

Abbildung 4.1: Klasse SchedulerBase

Die Klasse SchedulerBase dient als Grundlage für alle in *MISD OWL* verwendeten Schedulern zur Verwaltung von zeitabhängigen Aufgaben.

Diese Klasse besitzt eine Liste von Timerjobs des Typ TimerJobBase. Diese Timerjobs werden von diesem Scheduler verwaltet.

Die abstrakte Klasse stellt die abstrakten Methoden 'Initialize' und 'RefreshJobs'. Die Methode 'Initialize' wird im Konstruktor der Superklasse SchedulerBase aufgerufen und kann dazu verwendet werden Timerjobs zu erstellen.

Die abstrakte Methode 'RefreshJobs' dient dazu Timerjobs individuell mit aktualisierten Informationen zu versorgen. Diese Methode wird regelmäßig vom MainScheduler (siehe Kapitel 5.6.9) aufgerufen.

Desweiteren können die Timerjobs gestartet und gestoppt werden.

4.1.2 TimerJobBase

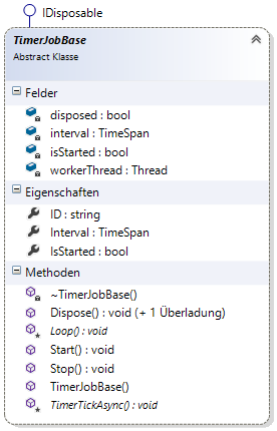

Abbildung 4.2: Klasse TimerJobBase

Die abstrakte Klasse TimerJobBase verwaltet einen TimerJob. Dieser besitzt grundsätzlich eine ID und ein Ausführungsintervall als Timespan. Ausführungsintervalle unter einer Sekunde sind nicht möglich.

Abhängig von dem Ausführungsintervall wird die abstrakte Methode 'TimerTickAsync' aufgerufen, die die Aufgabe des Timerjobs implementiert.

Zum Zerstören einer Timerjob-Instanz wird die 'Dispose' Methode implementiert.

4.2 States

4.2.1 MappingState

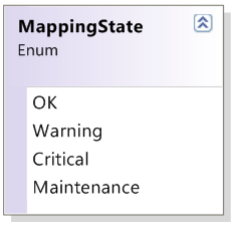

Abbildung 4.3: Die Enumeration MappingState

Die Enumeration MappingState repräsentiert das Mapping einer *Workstation*.

4.2.2 WorkstationState

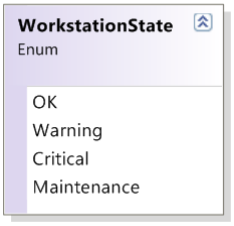

Abbildung 4.4: Die Enumeration WorkstationState

Die Enumeration WorkstationState beinhaltet die *Status*.

4.3 DataType

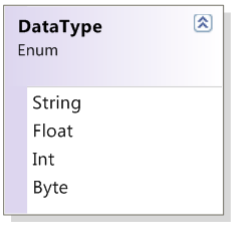

Abbildung 4.5: Die Enumeration DataType

Die Enumeration DataType repräsentiert die verschiedenen Datentypen, die als eine *Kenngröße* erfasst werden können. Diese werden zusätzlich von einem Byte repräsentiert.

- String (0)
- Float (1)
- Integer (2)
- Byte (3)

4.4 ClusterConnection

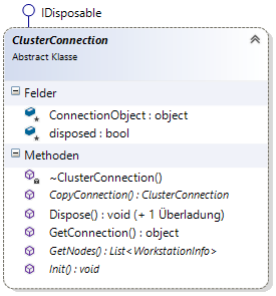

Abbildung 4.6: Die abstrakte Klasse IClusterConnection

Die abstrakte Klasse ClusterConnection enthält die Verbindung eines HPC oder Bright-*Clusters*. Die Klasse kann die Verbindung erstellen, die Verbindung duplizieren und die Namen der Knoten eines *Clusters* zurückgeben.

4.5 IndicatorSettings

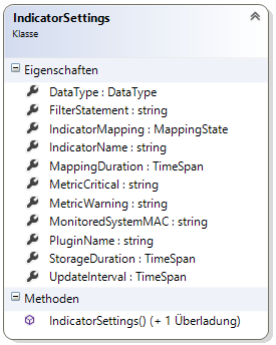

Abbildung 4.7: Klasse IndicatorSettings

Die Klasse IndicatorSettings dient zum Transport der Einstellungen einer *Kenngröße*.

4.6 IPlugin

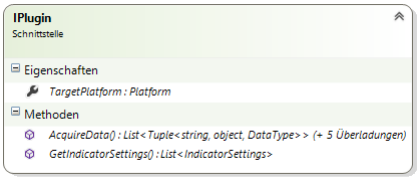

Abbildung 4.8: Interface IPlugin

Das Interface IPlugin wird von allen *Plugins* implementiert, die im *System* verfügbar sind.

Das Interface stellt verschiedenen Überladungen der Methode 'AcquireData' zur Verfügung, die für *Cluster*, *Server* oder *Workstations* optimiert sind.

4.7 IPluginExtensions

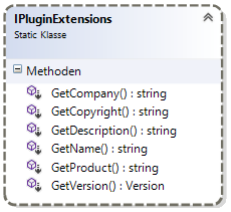

Abbildung 4.9: Die statische IPluginExtensions

Die statische Klasse IPluginExtensions liest aus *Plugins* die Assemblyinformationen aus:

- Company
- Copyright Informationen
- Beschreibung des Plugins
- Name des Plugins
- Produkt (z.B. 'MISD OWL')
- Version

4.8 LogType

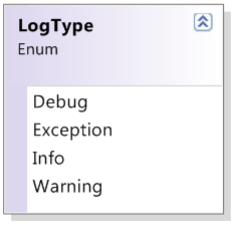

Abbildung 4.10: Die Enumeration LogType

Die Enumeration LogType beschreibt die verschiedenen Typen der Logeinträge von *MISD OWL*.

4.9 Logger

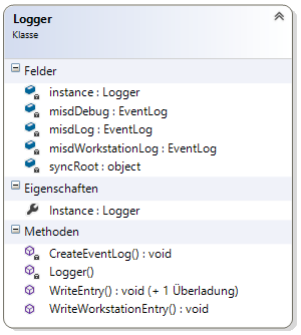

Abbildung 4.11: Klasse Logger

Der Logger bietet die Funktionalitäten zum Eintragen von Serverlogs bei Fehlern, Ausnahmen, Warnungen, Informationen oder Debug-Nachrichten.

Alle Log Einträge werden in das Windows Event Log des Servers geschrieben. Für *Server*-, *Workstation*- und Debug Logs existieren separate Event Sourcen.

4.10 Platform

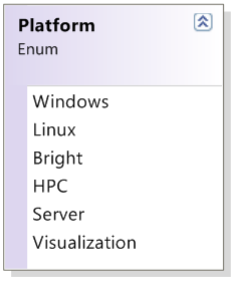

Abbildung 4.12: Die Enumeration Platform

Die Enumeration Platform repräsentiert die verschiedenen Plugintypen als Ausführungsgrundlage. Diese sind zusätzlich einem Byte zugeordnet.

- Windows (0)

- Linux (1)
- Bright Cluster (2)
- HPC Clsuter (3)
- Server (4)
- Visualisierung (5)

4.11 PluginFile

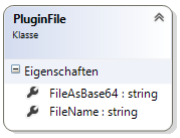

Abbildung 4.13: Klasse PluginFile

Die Klasse PluginFile dient zur Übertragung einer *Plugin*-Assembly. Eine PluginFile-Instanz enthält den Dateinamen und die Datei als Base64 String.

4.12 PluginMetadata

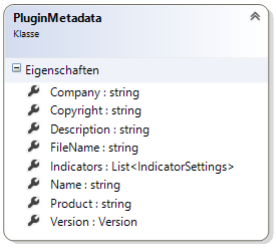

Abbildung 4.14: Klasse PluginMetadata

Die Klasse PluginMetadata dient zur Übertragung sämtlicher Informationen über ein *Plugin* die zum Betrieb und Auswahl eines *Plugin* nötig sind.

4.13 StringExtensions

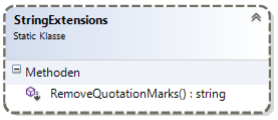

Abbildung 4.15: Statische Klasse StringExtensions

Diese Klasse dient zur Bearbeitung von Strings. Die Methode 'RemoveQuotationMarks' entfernt von Dateipfaden führende oder abschließende '\'.

4.14 Layout

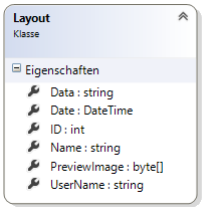

Abbildung 4.16: Klasse Layout

Die Klasse Layout dient zur Übertragung von *Layouts* zwischen *Server* und *Client*.

4.15 WorkstationInfo

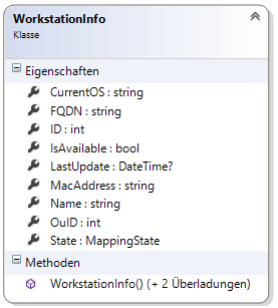

Abbildung 4.17: Klasse WorkstationInfo

Die Klasse WorkstationInfo dient zur Übertragung von Informationen über eine *Workstation* zwischen den Komponenten von *MISD OWL*.

Kapitel 5

Feinentwurf Server

5.1 Bootstrapper

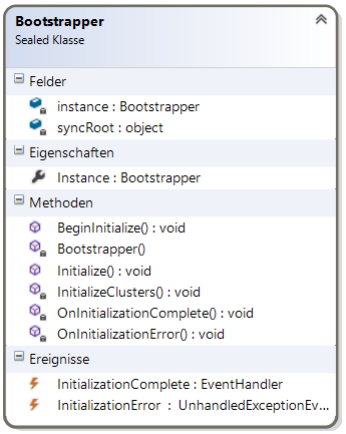

Abbildung 5.1: Klasse Bootstrapper

Der Bootstrapper startet den *Server* und initialisiert alle Scheduler sowie die Web Services. Er lädt zum Start des *Server* die *Plugins* und initialisiert die bekannten Cluster.

5.2 Cluster

Die *Cluster*-Klassen kümmern sich um den Aufbau einer Verbindung zum jeweiligen *Cluster* sowie das Hinzufügen von Nodes und Verbindungen der *Clustern*.

5.2.1 BrightClusterConnection

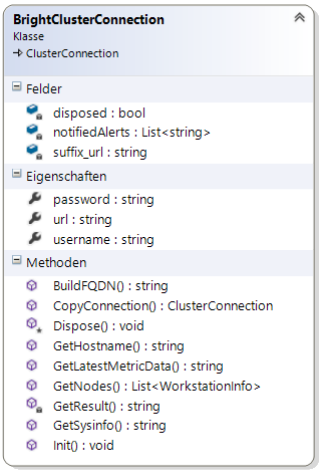

Abbildung 5.2: Klasse BrightClusterConnection

Die BrightClusterConnection baut mit Hilfe der BrightClusterShell eine Verbindung zum Bright-*Cluster* auf. Zudem kann von hier aus auf die Daten des Bright-*Cluster* zugegriffen werden.

5.2.2 BrightClusterShell

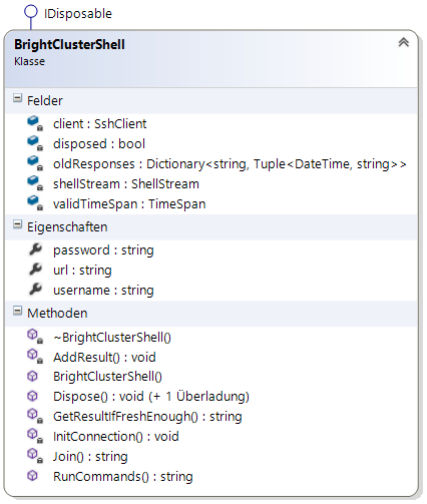

Abbildung 5.3: Klasse BrightClusterShell

Die BrightClusterShell baut eine SSH-Shell auf, um auf die Daten des Bright-*Cluster* zugreifen zu können. Diese Klasse führt die Commands aus und formatiert die Antworten.

5.2.3 HpcClusterConnection

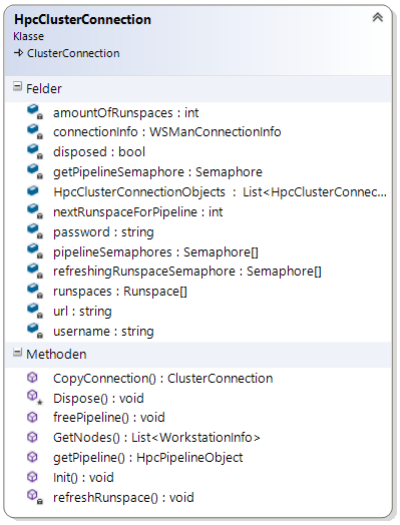

Abbildung 5.4: Klasse HpcClusterConnection

Die HpcClusterConnection baut eine Verbindung zum HPC auf, verwaltet die Namen der Knoten.

5.3 Database

Die Database besteht aus der MISD.dbml, einer DataContext Factory und den Precompiled Querries für die Linq Statements.

5.3.1 Datenbank

Die MISD.dbml beinhaltet das Datenbankschema und bietet Möglichkeiten auf die Datenbank zuzugreifen.

5.3.2 DataContextFactory

Abbildung 5.5: Klasse DataContextFactory

Die effiziente Verwaltung der Instanzen der Klasse MISDDataContext wird von der Factory Klasse DataContextFactory übernommen. Diese Klasse stellt eine eine Instanz mit Leserechten auf der Datenbank und mit Schreib- und Leserechten zur Verfügung. Die Performanz der MISDDataContext-Instanz ohne Schreibrechte ist höher.

Verwendet werden die Instanzen von MISDDataContext jeweils in using Blöcken:

```
using (var dataContext = DataContextFactory.CreateReadOnlyDataContext())
  {
    //linq statements here.
  }
```

5.3.3 PrecompiledQueries

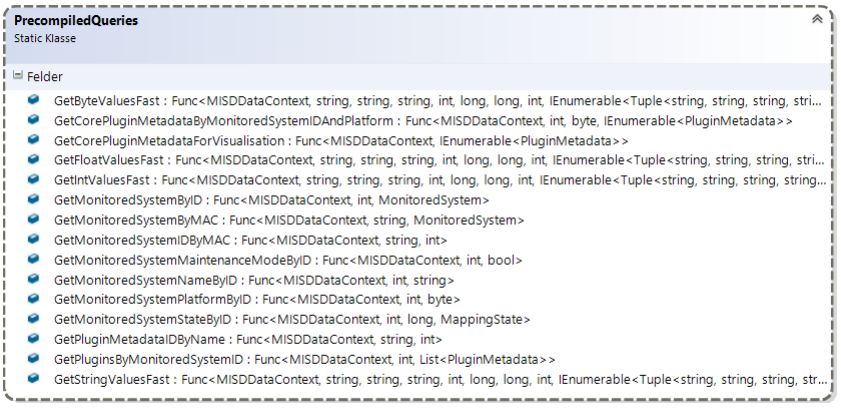

Abbildung 5.6: Klasse PrecompiledQueries

Die statische Klasse PrecompiledQueries stellt besonders häufig genutze Datanbankabfragen durch performante precompiled Linq Statements zur Verfügung.

5.4 Email

Die Komponente Email stellt alle Funktionalitäten zum Erstellen und Versenden der Emails von *MISD OWL* zur Verfügung.

5.4.1 Tagesbericht Template

Die Tagesberichte werden mit T4-Templates des .NET erzeugt. Änderungen an dem T4-Template erfordern eine Neuerstellung des Projekts.

DailyMailTemplate

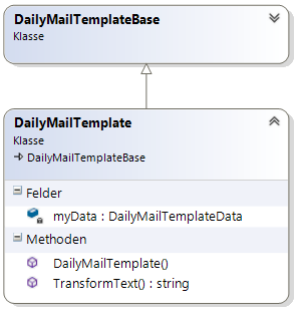

Abbildung 5.7: Klasse DailyMailTemplate

Die Klasse DailyMailTemplate wird automatisch erzeugt. Änderungen sind nur über die DailyMailTemplate.tt Datei möglich in der das T4-Template[1] gespeichert ist. Das Template enthält HTML und C# Code.

[1]Generieren von Text zur Laufzeit mithilfe von vorverarbeiteten T4-Textvorlagen: `http://msdn.microsoft.com/de-de/library/vstudio/ee844259%28v=vs.100%29.aspx`

DailyMailTemplateCode

Diese Klasse erweitert die Klasse DailyMailTemplate um ein Feld der Klasse DailyMailTemplateData und einen Konstruktor, der ein Objekt der Klasse DailyMailTemplateData übergeben bekommt. Dieses Objekt steht im Template zur Verarbeitung zur Verfügung.

DailyMailTemplateData

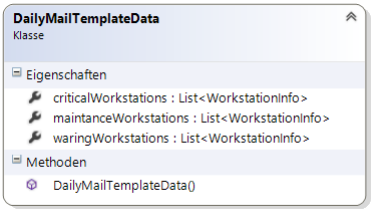

Abbildung 5.8: Klasse DailyMailTemplateData

Die Klasse DailyMailTemplateData enthält jeweils eine Liste für *zu überwachender Rechner* im *Status WARNUNG*, *KRITISCH* und *Wartungszustand* vom Typ WorkstationInfo (siehe Kapitel 4.15). Diese stehen im Template zur Tabellenerstellung zur Verfügung.

5.4.2 WarningMailParser

Der Parser wird für Warn-Emails verwendet und unterstützt spezielle Tags. Das Template selbst in eine .txt-Datei.

TemplateTag

Abbildung 5.9: Klasse TemplateTag

Diese Klasse definiert einzelne Tags die im Template zur Verfügung stehen. Folgende Tags stehen für *MISD OWL*-Warnung-Templates zur Verfügung:

- '[%WSName%]' (Workstationname)
- '[%Date%]' (Timestamp der Erfassung des *Kenngrößenwert*)
- '[%PluginName%]' (*Plugin* der *Kenngröße*)
- '[%Indicator%]' (*Kenngröße*)
- '[%Value%]' (*Kenngrößenwert*)

WarningMailParser

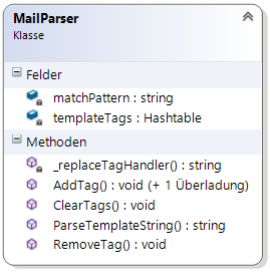

Abbildung 5.10: Klasse WarningMailParser

Die Klasse WarningMailParser verwaltet die Template-Tags, das Template und erzeugt mit Hilfe der übergebenen Daten die Email-Nachricht.

5.4.3 Mailer

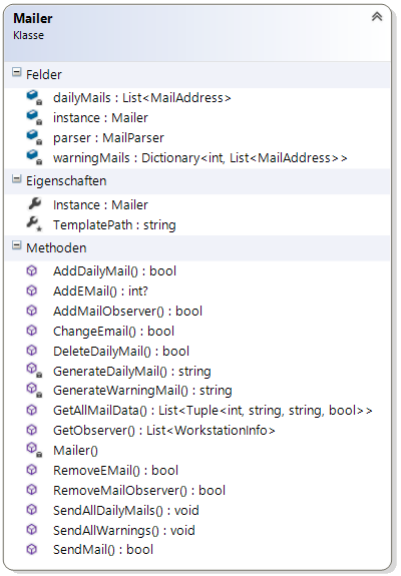

Abbildung 5.11: Klasse Mailer

Die Klasse Mailer erstellt und versendet Tagesberichte sowie Warnungs-Emails an die in der Datenbank hinterlegten Email-Adressen. Des weiteren werden Methoden zur Verfügung gestellt, um die Email-Adressen der Datenbank zu verwalten.

5.5 Manager

Die Manager-Klassen sind für alle internen Berechnungen und Abfragen des Servers zuständig. Je nach Art der Aufgabe ist ein anderer Manager zuständig.

5.5.1 ADManager

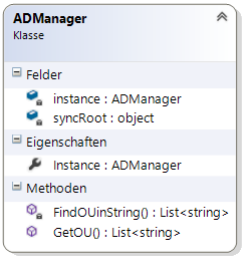

Abbildung 5.12: Klasse ADManager

Der ADManager erfasst Informationen aus dem *Active Directory*, formatiert diese in ein passendes Format und stellt die Informationen zur weiteren Verarbeitung zur Verfügung.

5.5.2 OUManager

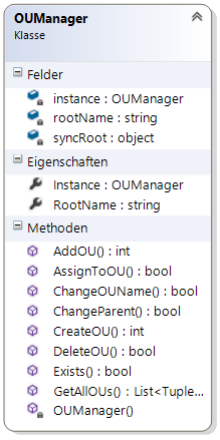

Abbildung 5.13: Klasse OUManager

Der OUManager kümmert sich um die Verwaltung der *Organisationseinheiten* in der Datenbank. Er bietet Methoden zum Erstellen, Hinzufügen, Ändern und Löschen von *Organisationseinheiten*. Zudem kann überprüft werden, ob eine *Organisationseinheit* existiert und eine *Workstation* zu einer *Organisationseinheit* hinzugefügt werden.

5.5.3 ClusterMananger

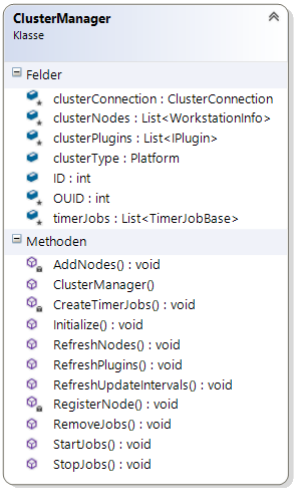

Abbildung 5.14: Klasse ClusterMananger

Der ClusterManager verwaltet für einen Cluster die einzelnen Nodes, Plugins und die Timerjobs, die von *Clustern Kenngröße* erfassen.

5.5.4 MetaClusterManager

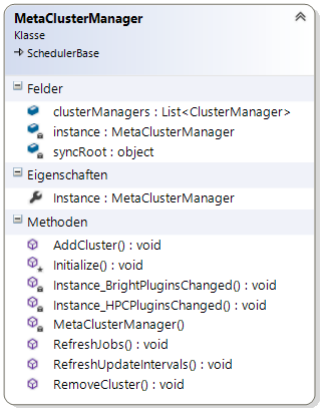

Abbildung 5.15: Klasse MetaClusterManager

Der MetaClusterManager erbt von der SchedulerBase im Core. Er verwaltet die einzelnen ClusterManager. Er bietet zudem dem Client Methoden um ein Cluster zu löschen, oder hinzuzu-

fügen.

5.5.5 FilterManager

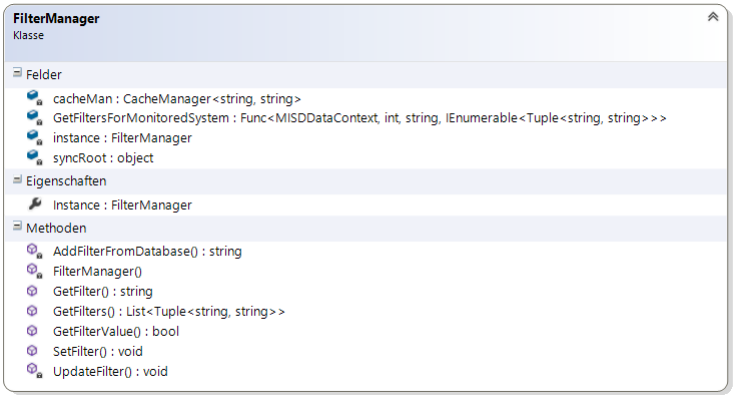

Abbildung 5.16: Klasse FilterManager

Der Filtermanager bietet Methoden, um die *Filter* je *Workstation* und je *Kenngröße* aus der Datenbank abzurufen, neu zu setzen, oder auch einen Wert darauf zu prüfen, ob er seinen *Filter* passiert.

5.5.6 MetricManager

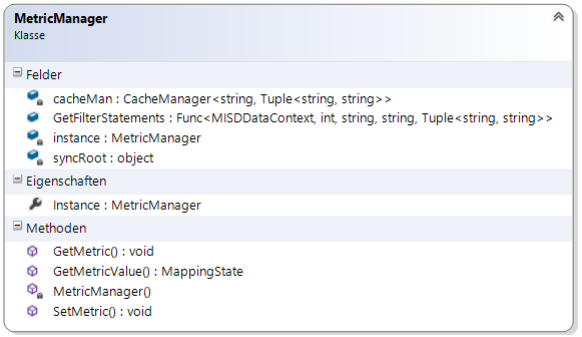

Abbildung 5.17: Klasse MetricManager

Der MetricManager bietet Methoden, um die *Metriken* je Workstation und je *Kenngröße* aus der Datenbank abzurufen, neu zu setzen, oder auch einen Wert darauf zu prüfen, welches Mapping er annimmt.

5.5.7 UpdatIntervalManager

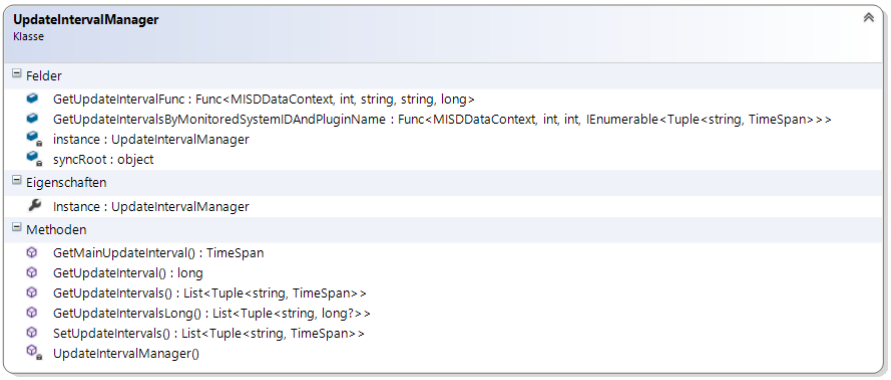

Abbildung 5.18: Klasse UpdatIntervalManager

Der UpdateIntervallManager bietet verschiedenen Methoden zum Abrufen und Setzen der unterschiedlichen UpdateIntervalle unter verschiedenen Parametern, wie je *Workstation*, oder je *Kenngröße* und *Workstation*.

5.5.8 PluginManager

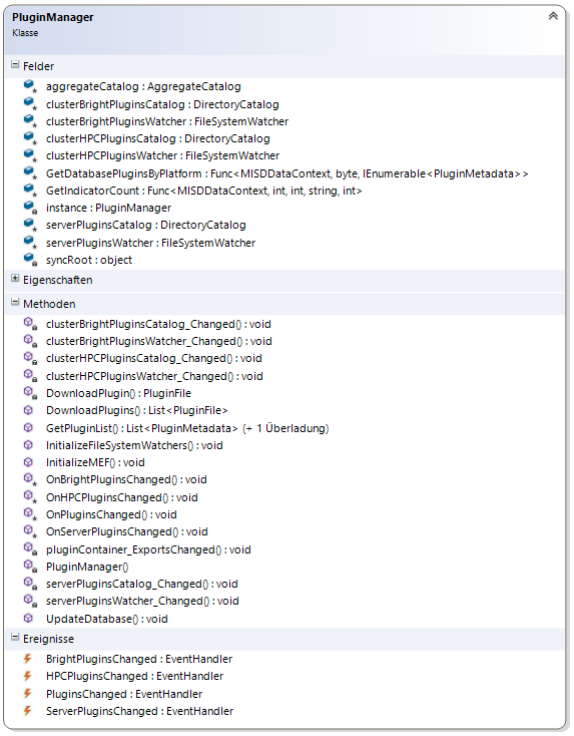

Abbildung 5.19: Klasse PluginManager

Der PluginManager verwaltet die *Plugin*-Metadaten in der Datenbank sowie der *Plugin*-Assemblys im Verzeichnis. Hier werden die Pluginfiles überwacht und die Daten der Datenbank damit synchron gehalten. Weiterhin managed der PluginManager die Verteilung von *Plugins* an *Workstation* und *Client*, sowie das Aktualisieren der Pluginmetadaten.

5.5.9 UIConfigManager

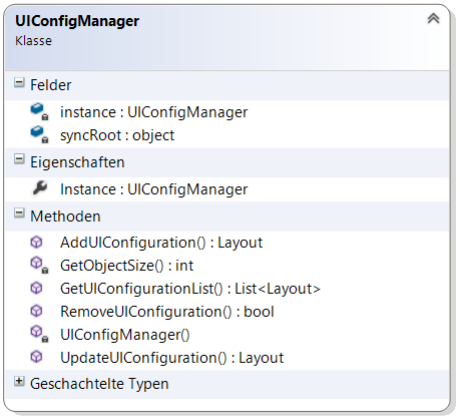

Abbildung 5.20: Klasse UIConfigManager

Der UIConfigManager verwaltet die UIKonfiguratíonen. Hier werden dem Client Methoden zum Erstellen, Ändern und Löschen einer Konfiguration geboten. Zudem kann hier eine Liste aller verfügbaren *Layouts* angefragt werden.

5.5.10 ValueManager

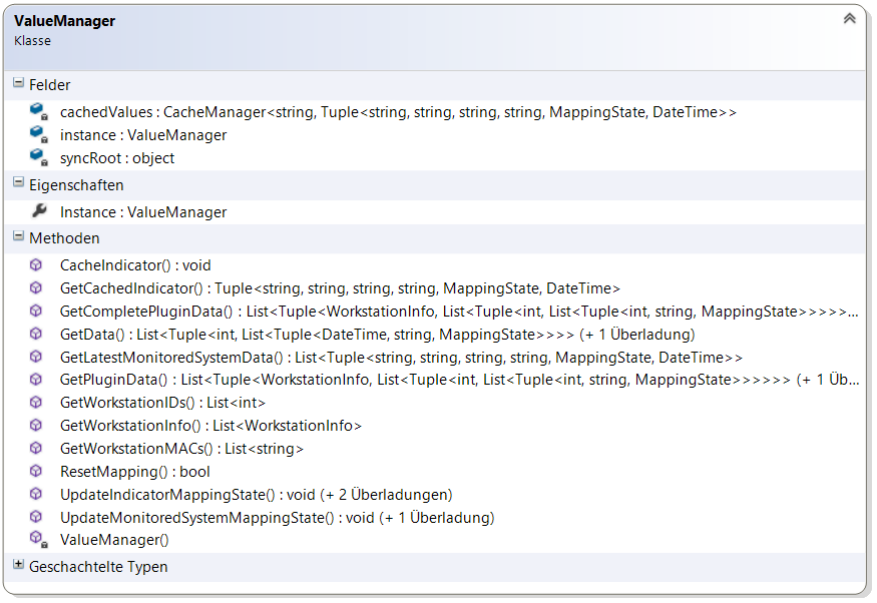

Abbildung 5.21: Klasse ValueManager

Der ValueManager bietet dem Client alle Funktionen zum Abrufen der erfassten Daten. Eintragen von *Kenngrößenwerte* und dem Senden von *Kenngrößenwerte* eines Zeitraums an einen *Client*.

5.5.11 WorkstationManager

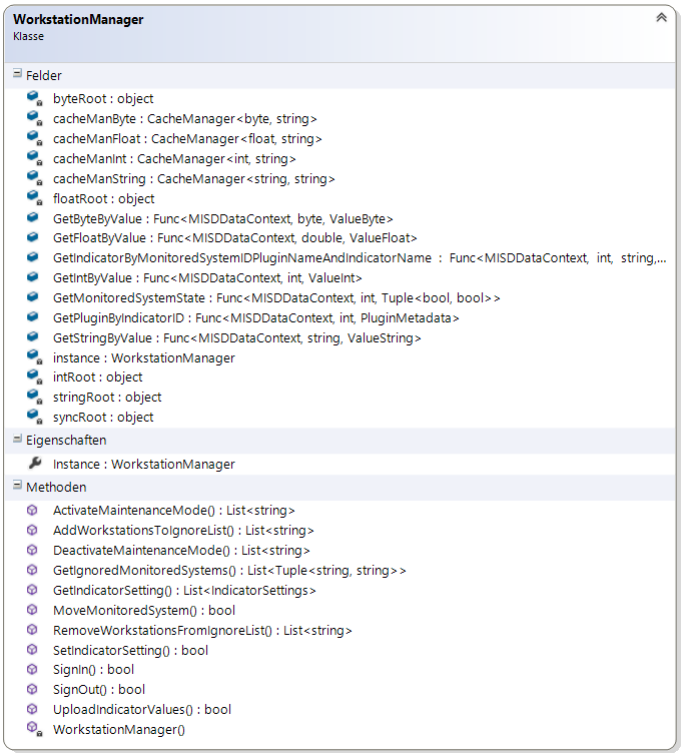

Abbildung 5.22: Klasse WorkstationManager

Der WorkstationManager bietet alle Methoden zur Verwaltung der im System überwachten Rechner. Dieser Manager bietet Methoden zum Ein- und Ausloggen von *Workstations*, Speicherung von *Kenngrößenwerte*. Es können die *Kenngrößen* Settings einer *Workstation* gesetzt und abgerufen werden. Außerdem werden *Wartungszustände* und *Ignore-Liste* gesetzt.

Wird ein neuer *Kenngrößenwert* in die Datenbank eingetragen wird zunächst in der entsprechenden *ValueX*-Tabelle nach einem gleichen Wert gesucht und in den neuen IndicatorValue Eintrag ein Verweis auf den bereits bestehenden *ValueX*-Eintrag erstellt. Wurde ein gleicher *Kenngrößenwert* noch nicht erfasst wird ein neuer *ValueX*-Eintrag erstellt.

5.6 Scheduling

5.6.1 CleanerJobScheduler

Die Klasse CleanerJobScheduler erbt von der abstrakten Klasse SchedulerBase im Namespace Core. Der CleanerJobScheduler erstellt neue Cleanerjobs für alle *Kenngrößen* und aktualisiert die Intervalle.

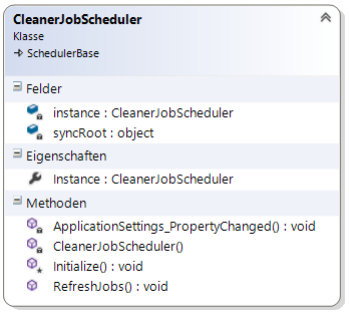

Abbildung 5.23: Klasse CleanerJobScheduler

5.6.2 CleanerJob

Die Klasse CleanerJob erbt von der abstrakten Klasse TimerJobBase im Namespace Core. Der Cleanerjob entfernt alle veralteten *Kenngrößen* aus der Datenbank.

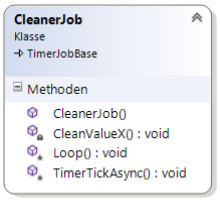

Abbildung 5.24: Klasse CleanerJob

Bei einem Cleaning-Vorgang werden die Datenbank-Tabelle 'IndicatorValue' sowie die *ValueX*-Tabellen bearbeitet. Grundlage ist die 'StorageDuration', die für jede *Kenngröße* einer *Workstation* festgelegt ist. Alle Einträge die älter wie die festgelegte 'StorageDuration' sind werden entfernt.

Abschließend werden in den *ValueX*-Tabellen alle Einträge entfernt, die von keinem erfassten

Kenngrößenwert (Datenbanktabelle 'IndicatorValue') referenziert werden. Von diesem Vorgang ist die Datenbank-Tabelle 'ValueByte' ausgenommen, da diese nur maximal 255 Einträge enthalten kann und ein Cleaning-Vorgang an dieser Stelle auf Grund des geringen Wertebereiches überflüssig ist.

5.6.3 GlobalScheduler

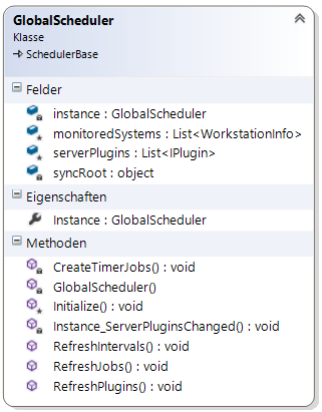

Abbildung 5.25: Klasse GlobalScheduler

Der GlobalScheduler erbt von der SchedulerBase im Namespace Core. Die Klasse erstellt GlobalTimerJobs für alle *Kenngrößen* eines Serverplugins für einen*zu überwachender Rechner* und hält die Intervalle aktuell.

5.6.4 GlobalTimerJob

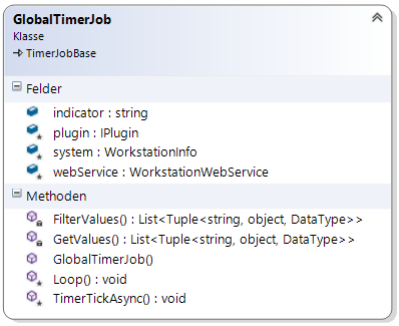

Abbildung 5.26: Klasse GlobalTimerJob

Der GlobalTimerJob erbt von der TimerJobBase im Namespace Core. Die Klasse enthält einen Thread, der für eine *Workstation* und eine *Kenngröße* das Erfassen und Filtern von *Kenngrößenwerte* übernimmt.

5.6.5 ClusterTimerJob

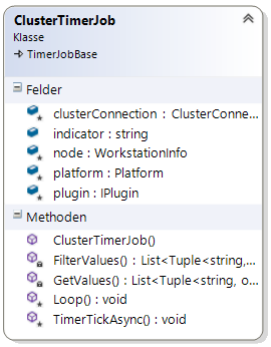

Abbildung 5.27: Klasse ClusterTimerJob

Der GlobalTimerJob erbt von der TimerJobBase im Namespace Core. Die Klasse enthält einen Thread, der für einen *Cluster* und eine *Kenngröße* das Erfassen und Filtern von *Kenngrößenwerte* übernimmt.

5.6.6 MailScheduler

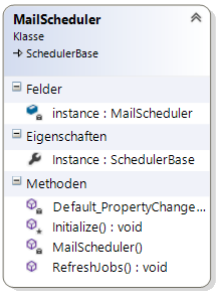

Abbildung 5.28: Klasse MailScheduler

Der MailScheduler erbt von der SchedulerBase im Core. Er erzeugt für die Tagesmail einen DailyMailTimerJob und hält diesen aktuell.

5.6.7 DailyMailTimerJob

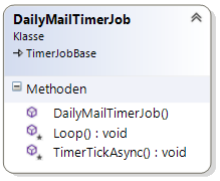

Abbildung 5.29: Klasse DailyMailTimerJob

Der DailyMailTimerJob erbt von der TimerJobBase im Namespace Core. Die Klasse aktiviert regelmäßig die Versendung des Tagesberichts.

5.6.8 MainScheduler

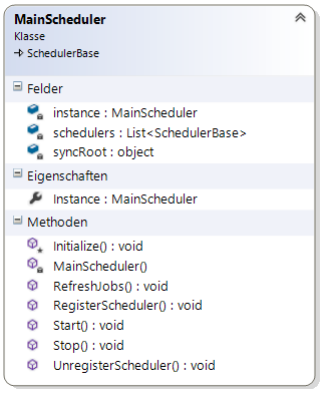

Abbildung 5.30: Klasse MainScheduler

Der MainScheduler erbt von der SchedulerBase im Core. Die Klasse verwaltet das Timing und das Hinzufügen und das Beenden der anderen Scheduler auf dem *Server*.

5.6.9 MainRefreshTimerJob

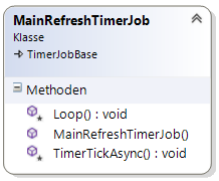

Abbildung 5.31: Klasse MainRefreshTimerJob

Der MainRefreshTimerJob erbt von der TimerJobBase im Core. Er ist ein Thread, welcher das Updateintervall und dadurch das Timing der Threads verwaltet.

5.7 Services

Die beiden Web Services dienen als Schnittstelle zum *Client* bzw. zur *Workstation*. In dieser Komponente werden alle nötigen Funktionalitäten des *Servers* für *Workstations* und *Clients* angeboten.

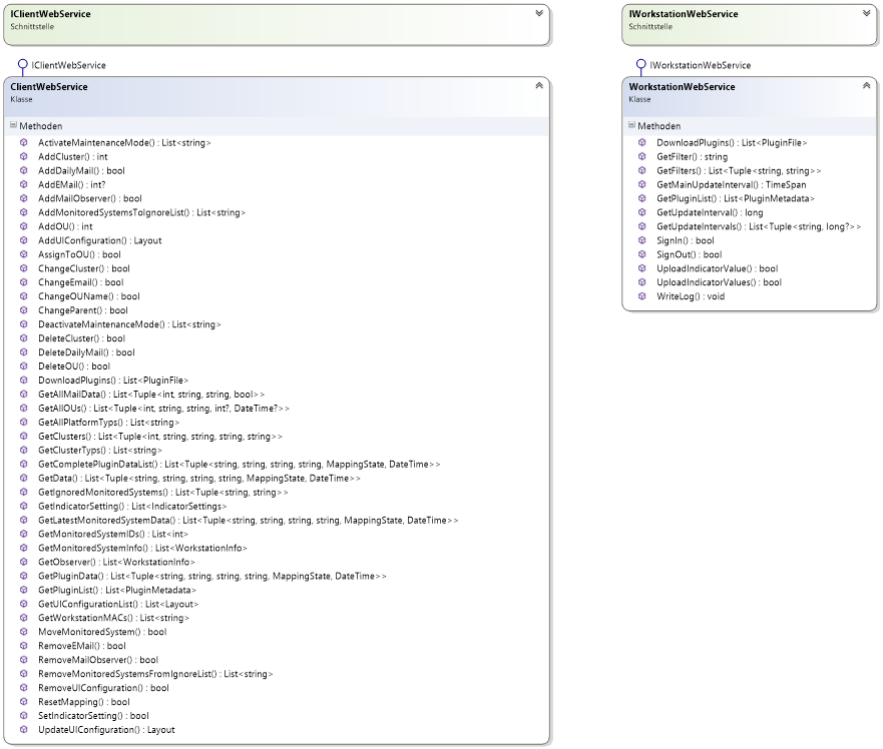

Abbildung 5.32: Klassen Web Services

5.7.1 ClientWebService

Der ClientWebService implementiert das Interface IClientWebService. Er bietet dem *Client* alle Methoden zum Aufbau der Visualisierung. Die einzelnen Methoden des Web Services rufen Funktionalitäten des Servers in den Managern auf.

5.7.2 WorkstationWebService

Der WorkstationWebService implementiert das Interface IWorkstationWebservice. Er bietet den *Workstations* alle Methoden für den laufenden Betrieb. Die einzelnen Methoden des Web Services rufen Funktionalitäten des *Servers* in den Managern auf.

Kapitel 6

Feinentwurf Workstation

6.1 ServerConnection

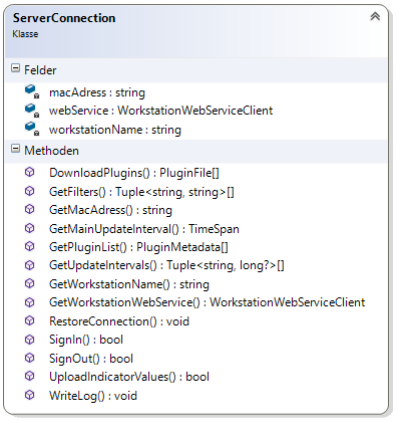

Abbildung 6.1: Klasse ServerConnection

Diese Klasse verwaltet die Verbindung zum Web Service des *Servers*. Zur Kommunikation stellt diese Klasse das Webservice-Objekt und den Namen der *Workstation* zur Verfügung. Für den WorkstationWebService ist die Klasse ServerConnection eine Facade, die alle Web Service-Methoden intern zur Verfügung stellt. Die Facade handelt alle Fehler, die bei einem serverseitigem Webs Service-Ausfall auftreten. Die SignIn-Methode des Web Service wird solange wiederholend ausgeführt, bis die Methode ausgeführt werden konnte. Ist der Web Service nicht verfügbar, wird die SignIn-Methode nach 5 Sekunden wieder ausgeführt.

6.2 WorkstationLogger

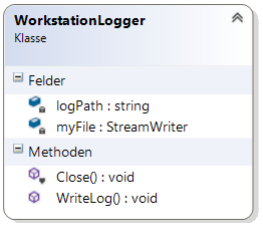

Abbildung 6.2: Klasse WorkstationLogger

Die Klasse WorkstationLogger erstellt eine Log-Datei (log.txt) im Programmverzeichnis von *MISD OWL*. In dieser Datei werden alle Meldungen gespeichert.

Unter Windows wird die Log-Datei automatisch erstellt.

Unter Linux muss der *Daemon* mit den Parameter 'log' gestartet werden um die Log-Datei zu erstellen. Wird *MISD OWL* mit dem Parameter 'console' gestartet werden die Meldungen im Terminal ausgegeben. Nach der Installation von *MISD OWL* unter Linux wird unter Standarteinstellungen keine Log-Datei angelegt.

6.3 Plugins

Die Plugins-Komponente stellt eine Klasse zu Verwalten von *Plugins* zur Verfügung.

6.3.1 PluginManager

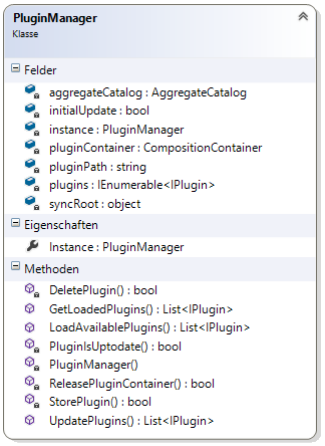

Abbildung 6.3: Klasse PluginManager

Der PluginManager des *Workstation-Dienstes* bindet *Plugins* aus dem Programmverzeichnis ein und lädt neue oder aktualisierte *Plugins* vom *Server* ins Programmverzeichnis.

Werden neue oder aktualisierte *Plugins* vom *Server* geladen, wird eine Liste mit Plugin-Metadaten mit dem vorhandenen *Plugin* Bestand verglichen. Müssen neue *Plugins* vom *Server* bezogen werden, werden alle IndicatorTimerJobs liquidiert und die eingebundenen *Plugins* entfernt. Die benötigten *Plugins* werden vom *Server* heruntergeladen und alle *Plugins* neu eingebunden. Das Erstellen der IndicatorTimerJobs übernimmt der Scheduler

Für diese Aufgaben stellt der PluginMananger Methoden zum Vergleich (nach den Versionsnummern) und Löschen von *Plugin*-Files zur Verfügung.

6.4 Scheduling

Die Komponente Scheduling beinhaltet den Scheduler der *Workstation* und alle TimerJobs.

6.4.1 Scheduler

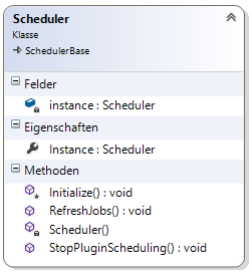

Abbildung 6.4: Klasse Scheduler

Die Klasse Scheduler erbt von SchedulerBase (siehe Kapitel 4.1.1) und verwaltet alle zeitabhängigen Aktionen der *Workstation* aus.

Der Scheduler lädt inital alle verfügbaren *Plugins* und startet die IndicatorTimerJob und den MainUpdateTimerJob.

6.4.2 IndicatorTimerJob

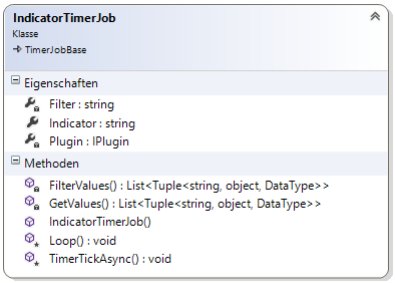

Abbildung 6.5: Klasse IndicatorTimerJob

Die Klasse IndicatorTimerJob erbt von TimerJobBase (siehe Kapitel 4.1.2) und führt alle zeitabhängigen Aktionen der Datenerfassung auf der *Workstation* aus.

Eine Instanz der Klasse IndicatorTimerJob verwaltet eine *Kenngröße* eines *Plugins* und erfasst regelmäßig den *Kenngrößenwert*, filtert den *Kenngrößenwert* und senden den *Kenngrößenwert* via ServerConnection (siehe Kapitel 6.1) an den *Server*.

6.4.3 MainUpdateTimerJob

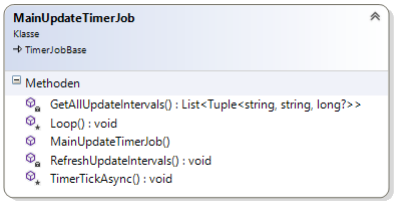

Abbildung 6.6: Klasse MainUpdateTimerJob

Die Klasse MainUpdateTimerJob erbt von TimerJobBase (siehe Kapitel 4.1.2) und aktualisiert das MainUpdateInterval auf der *Workstation* und erstellt gegebenenfalls neue IndicatorTimer-Jobs.

Eine Instanz der Klasse MainUpdateTimerJob bezieht via ServerConnection (siehe Kapitel 6.1) das aktuelle MainUpdateInterval vom *Server* und aktualisiert die *Plugins*. Wurden neue *Plugins* vom *Server* bezogen werden die IndicatorTimerJobs über den Scheduler neu erstellt.

Kapitel 7

Feinentwurf Client

7.1 Model

7.1.1 DataModel

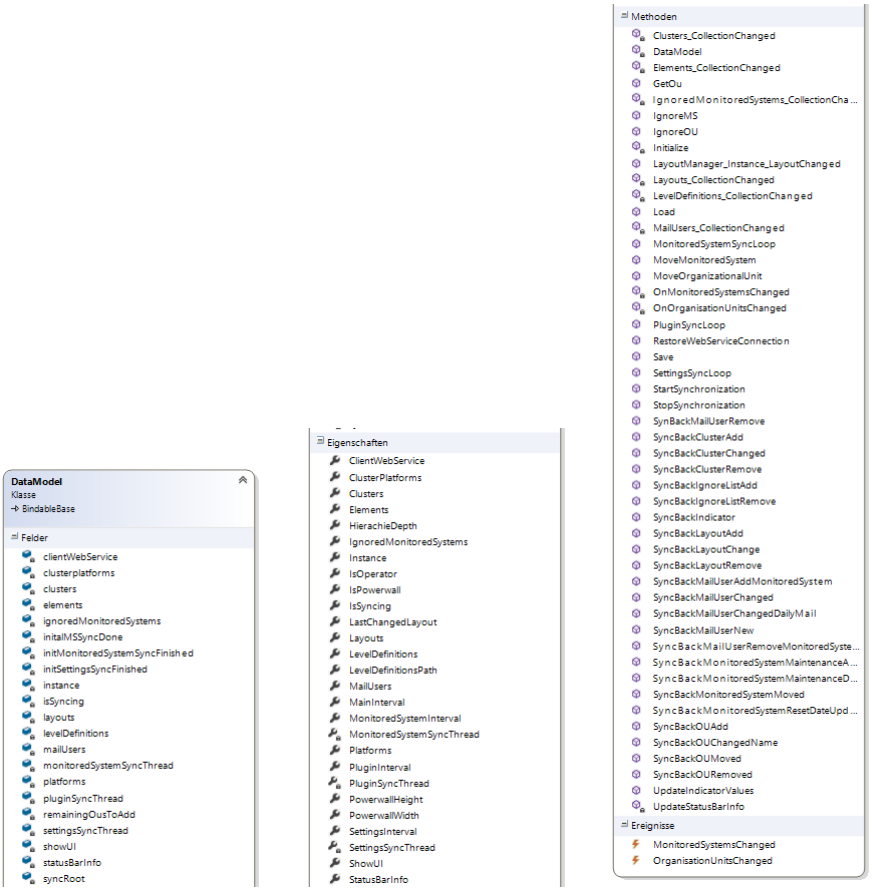

Abbildung 7.1: Klasse DataModel

Die Klasse DataModel enthält alle vom *Server* erhaltenen Informationen und wickelt die komplette Synchronisation mit dem *Server* ab.

7.1.2 MenuState

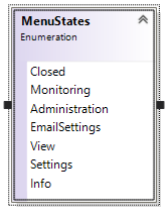

Abbildung 7.2: Enum MenuState

Das Enum MenuState dient dazu um programmatisch einzelne Tabs des ApplicationMenu zu öffnen.

7.2 ViewModel

Die Klassen der Komponente ViewModel enthalten alle Daten, die ausschließlich zum Betrieb der graphischen Oberfläche nötig sind.

7.2.1 ViewModel

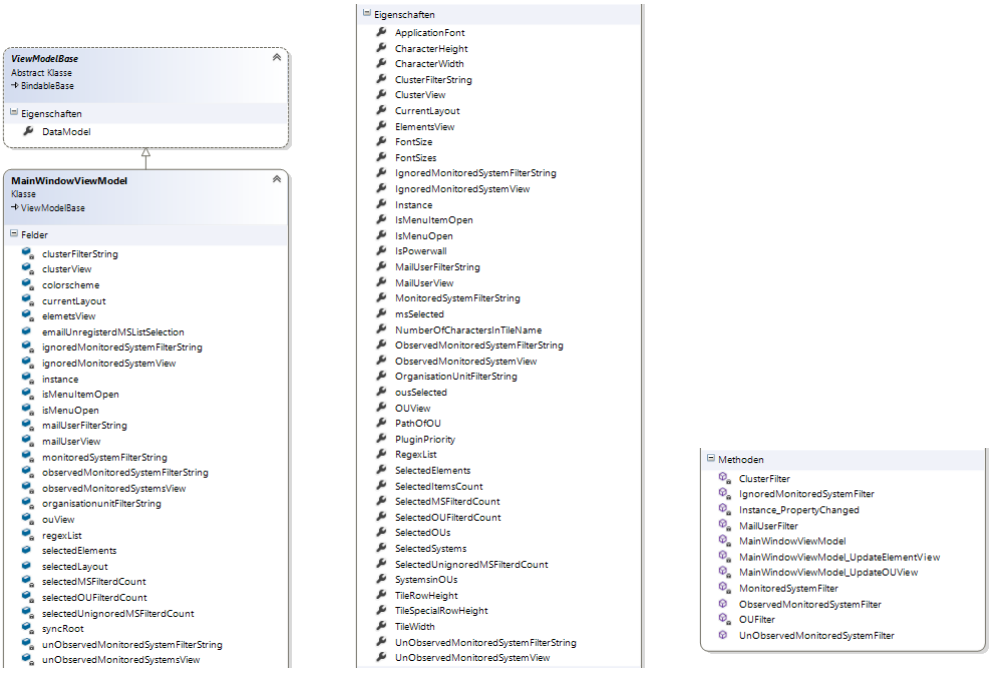

Abbildung 7.3: Klasse ViewModel

Kapitel 8

Interaktion der Komponenten

Dieses Kapitel beschreibt die Interaktionen ausgehend von *Workstation* und *Client* mit dem *Server*.

8.1 Interaktion von einem Client ausgehend

Der *Client* kommuniziert zur *Plugin*-Verwaltung, Konfigurations-Verwaltung und Visualisierung mit dem *Server* über den ClientWebService.

8.1.1 Layout-Verwaltung

Der *Benutzer* von *MISD OWL* kann die Anordnung der Elemente auf dem *Graphical User Interface (GUI)* in ein *Layout* speichern und dieses *Layout* zu späteren Zeitpunkten wieder laden. Die *Layouts* werden zentral vom *Server* verwaltet und werden komplett auf Anfrage des *Client* vom *Server* versendet.

Dieses Kapitel zeigt den Umgang mit *Layouts* aus *Client*-Sichtweise. *Layouts* können vom *Benutzer* erstellt, verändert, entfernt und auf das *GUI* angewendet werden. Zur genauen Erläuterung wird auf die Spezifikation verwiesen.

Layout hinzufügen

Möchte der *Benutzer* ein neues *Layout* erstellen, muss ein Name angeben werden. Das Vorschaubild wird selbstständig vom *Client* generiert. Dieser Beschreibungssatz wird abschließend

mit den *Layout*-Einstellungen zum *Server* übermittelt.

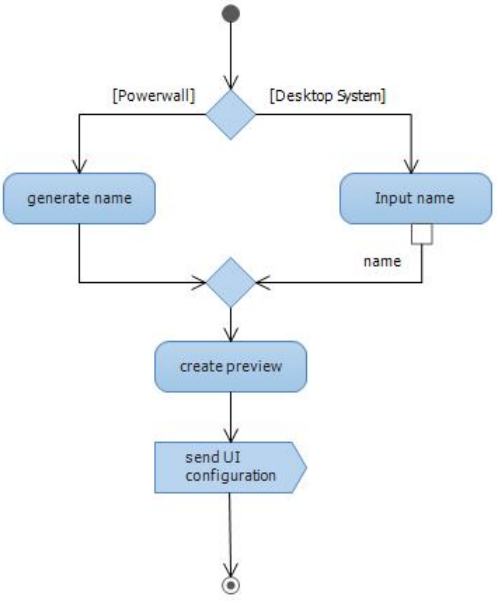

Abbildung 8.1: Layout hinzufügen

Layout laden

Soll ein neues *Layout* angezeigt werden, sendet der *Server* alle verfügbaren *Layouts* und zeigt diese mit Name und Vorschaubild an. Nach der Auswahl des *Benutzers* wird das *GUI* aktualisiert. *Kacheln* oder *Organisationseinheiten*, die nicht Bestandteil des angewendeten *Layout* sind, werden gemäß ihrer Repräsentation in der Datenbank angeordnet.

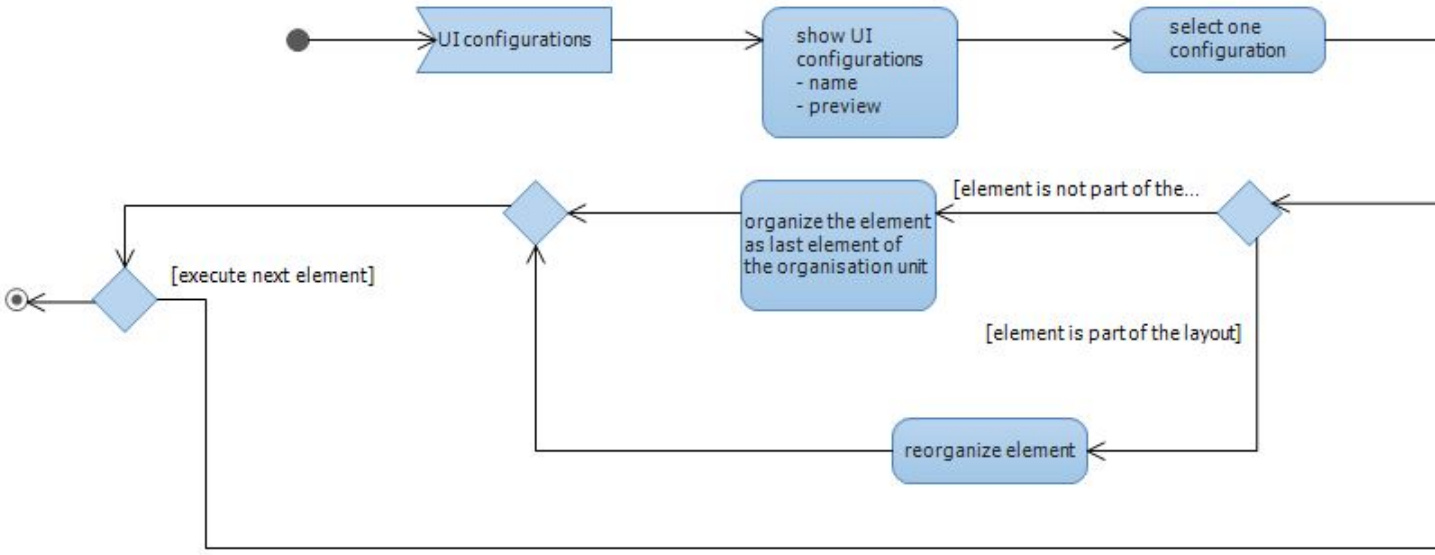

Abbildung 8.2: Layout laden

Visualisierungs-Plugin laden

Beim Start des *Clients* werden die Visualisierungs-*Plugins* aktualisiert. Der *Client* fordert vom *Server* eine Liste mit den Metadaten der verfügbaren Visualisierungs-*Plugins* an und vergleicht diese mit den bereits vorhandenen *Plugins* und fordert die nicht vorhandenen oder veralteten *Plugins* vom *Server* an und speichert die erhaltenen *Plugins* in das dafür vorgesehene Verzeichnis. Abschließend werden die *Plugins* geladen und bei der nächsten Oberflächenaktualisierung eingebunden.

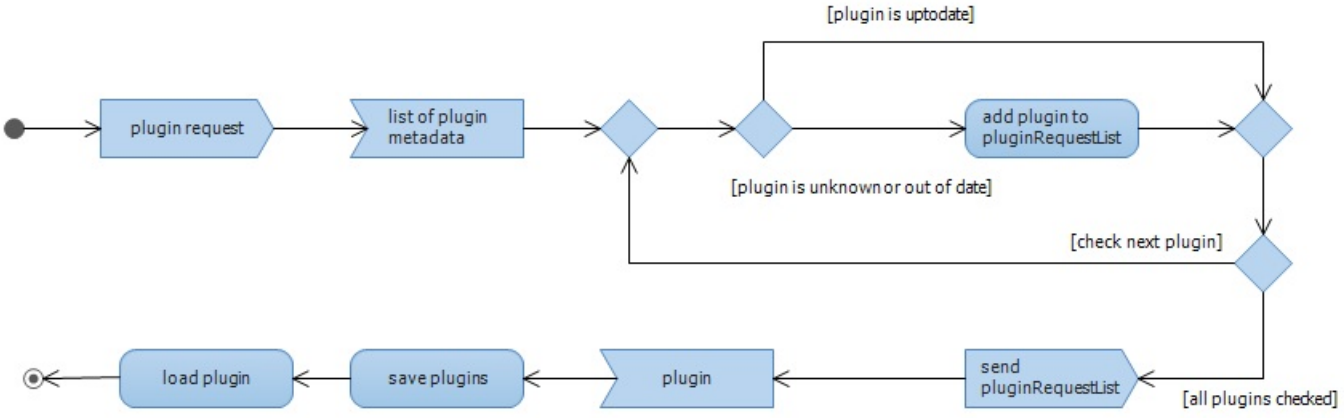

Abbildung 8.3: Visualisierungs-*Plugins* laden

8.2 Serverseitige Interaktion

Die Aktionen des *Servers* werden in diesen Kapitel mittels Aktivitätendiagrammen erläutert.

8.2.1 Visualisierungs-Daten senden

Der *Client* kann die jeweils aktuellsten *Kenngrößenwerte* oder *Kenngrößenwerte* aus der Vergangenheit vom *Server* abfragen. Der *Server* sendet den gewünschten *Kenngrößenwert*, falls vorhanden zurück. Ist der gewünschte *Kenngrößenwert* nicht in der Datenbank vorhanden wird eine leere Antwort zum *Client* zurück gesendet.

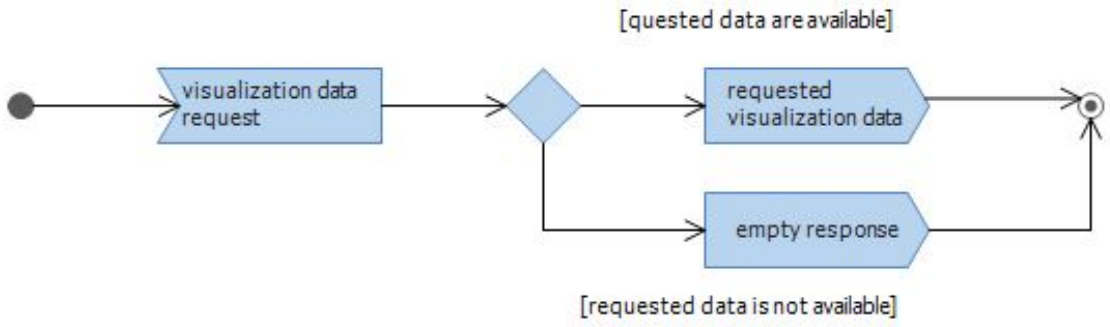

Abbildung 8.4: Visualisierungs-Daten senden

8.2.2 Visualisierungs-Plugin senden

Der *Server* sendet dem anfragenden *Client* die zur Verfügung stehenden Metadaten der Visualisierungs-*Plugins* an den anfragenden *Client* und erhält daraufhin vom *Client* eine Liste von *Plugins*, die dieser *Client* benötigt. Diese werden zusammen mit den Metadaten vom *Server* an den *Client* gesendet.

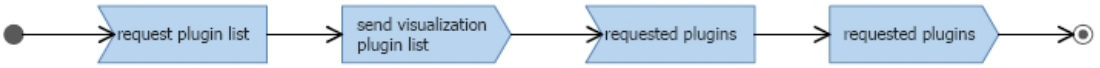

Abbildung 8.5: Visualisierungs-*Plugins* senden

8.2.3 Eine Workstation zur Ignore-Liste hinzufügen

Soll eine *Workstation* in Zukunft ignoriert werden, wird diese der *Ignore-Liste* hinzugefügt. Es werden alle bestehenden *Kenngrößenwerte* und Einstellungen (z.B *Filter*) gelöscht.

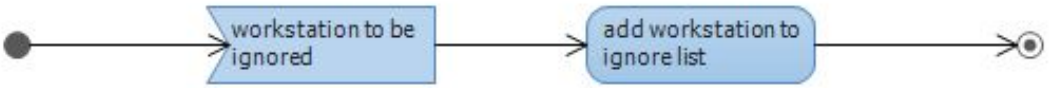

Abbildung 8.6: Eine *Workstation* zur *Ignore-Liste* hinzufügen

Weitere Informationen zur *Ignore-Liste* befinden sich in der Spezifikation unter dem Kapitel "Funktionale Anforderungen"

8.2.4 Eine Workstation in den Wartungszustand versetzen

Wird eine *Workstation* in den *Wartungszustand* versetzt, wird dies mit dem Eintragen des Eintrittdatums in die Datenbank gespeichert. Während sich eine *Workstation* im *Wartungszustand* befindet, werden alle von ihr während dieser Zeit gesendeten *Kenngrößenwerte* verworfen.

Wird eine *Workstation* wieder aus dem *Wartungszustand* herausgenommen, wird dieser Vorgang in der Datenbank gespeichert.

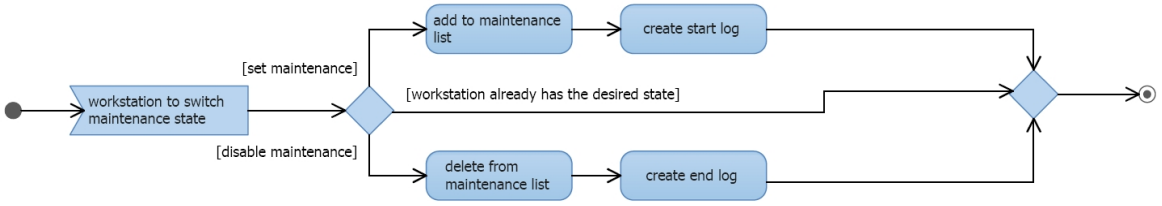

Abbildung 8.7: Eine *Workstation* in den *Wartungszustand* versetzen

Weitere Informationen zum *Wartungszustand* befinden sich in der Spezifikation unter dem Kapitel "Funktionale Anforderungen"

8.2.5 Workstation anmelden/registrieren

Sobald der *Server* eine Anfrage zum Einchecken einer *Workstation* empfängt, schaut der *Server* in der Datenbank nach, ob ihm diese *Workstation* schon bekannt ist. Ist das nicht der Fall, initialisiert der *Server* die *Workstation* mit Hilfe des *Fully Qualified Domain Name (FQDN)*, der

MAC-Adresse und Daten aus dem Active Directory. Schließlich wird noch die Information in die Datenbank geschrieben, dass die Workstation eingecheckt ist. Der Vorgang des erstmaligen Eincheckens wird Registrierung genannt. Die MAC-Adresse wird zur weiteren Kommunikation verwendet. Der *Workstation* wird der Erfolg oder Misserfolg des Eincheckens über einen Wahrheitswert zurückgegeben. Sollte die *Workstation* dem Server bereits bekannt sein, wird die Registrierung übersprungen.

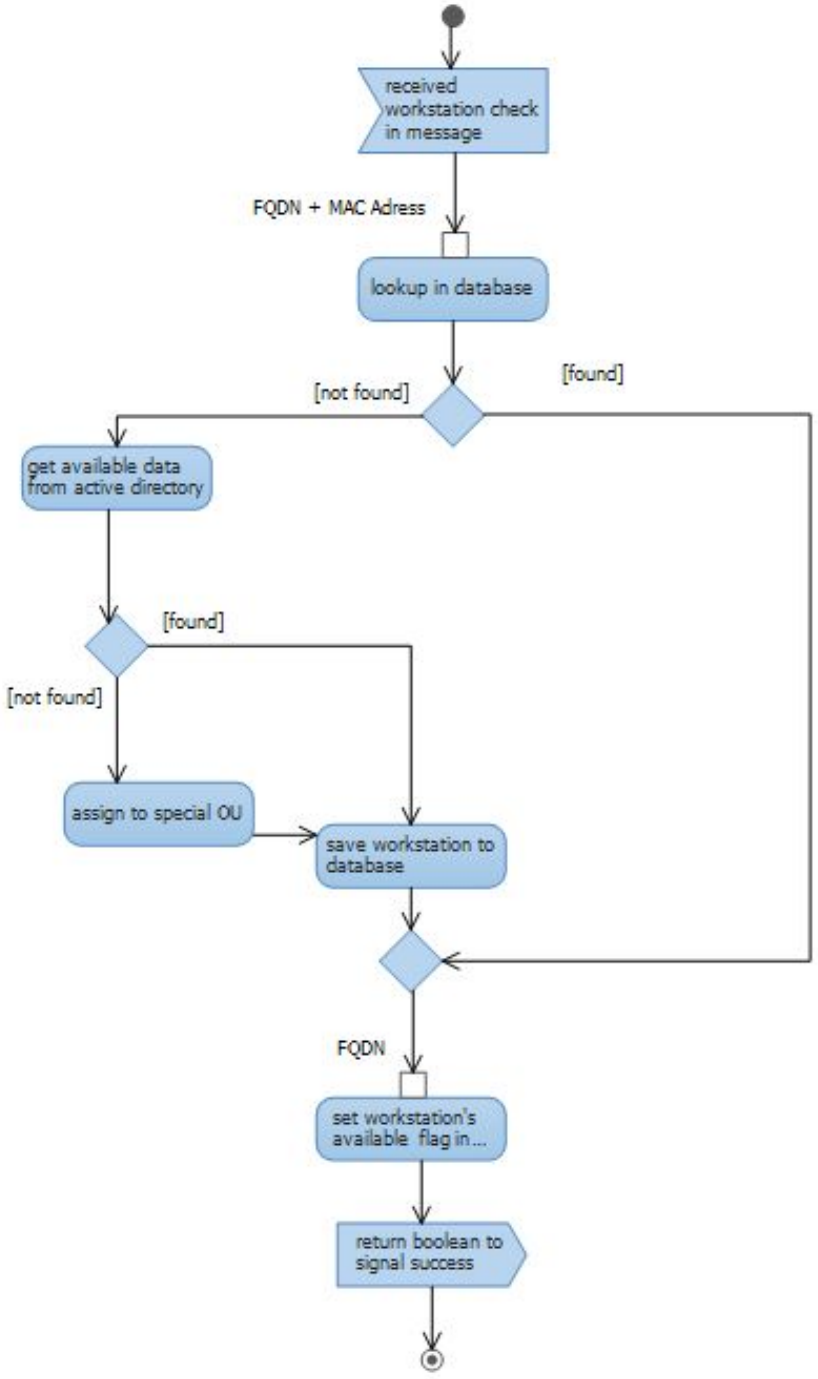

Abbildung 8.8: Workstation einchecken/registrieren

8.2.6 Workstation abmelden

Wenn der *Server* eine Anfrage zum Abmelden einer *Workstation* empfängt, setzt der *Server* diese Information in der Datenbank.

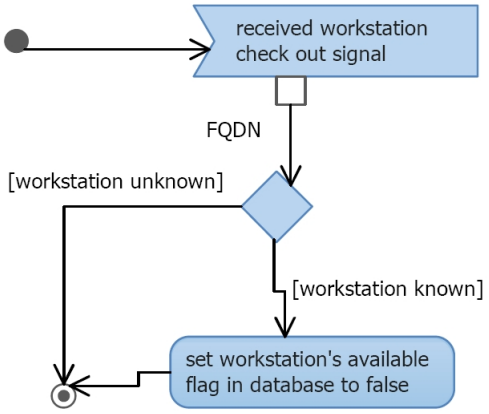

Abbildung 8.9: Workstation auschecken

8.2.7 Kenngroessenwert empfangen

Wenn der Server *Kenngrößenwerte* empfängt, wird zunächst überprüft, ob die Workstation bekannt ist, ob sie ignoriert wird oder ob der Wert entgegen genommen werden kann. Wenn das *Plugin* oder die *Workstation* von dem der *Kenngrößenwert* stammt unbekannt ist, wird der Workstation ein Fehler signalisiert. Andernfalls wird der *Kenngrößenwert* durch seine *Metrik* auf einen der drei Zustände *OK*, *WARNUNG*, oder *KRITISCH* abgebildet. (siehe Kapitel 10)

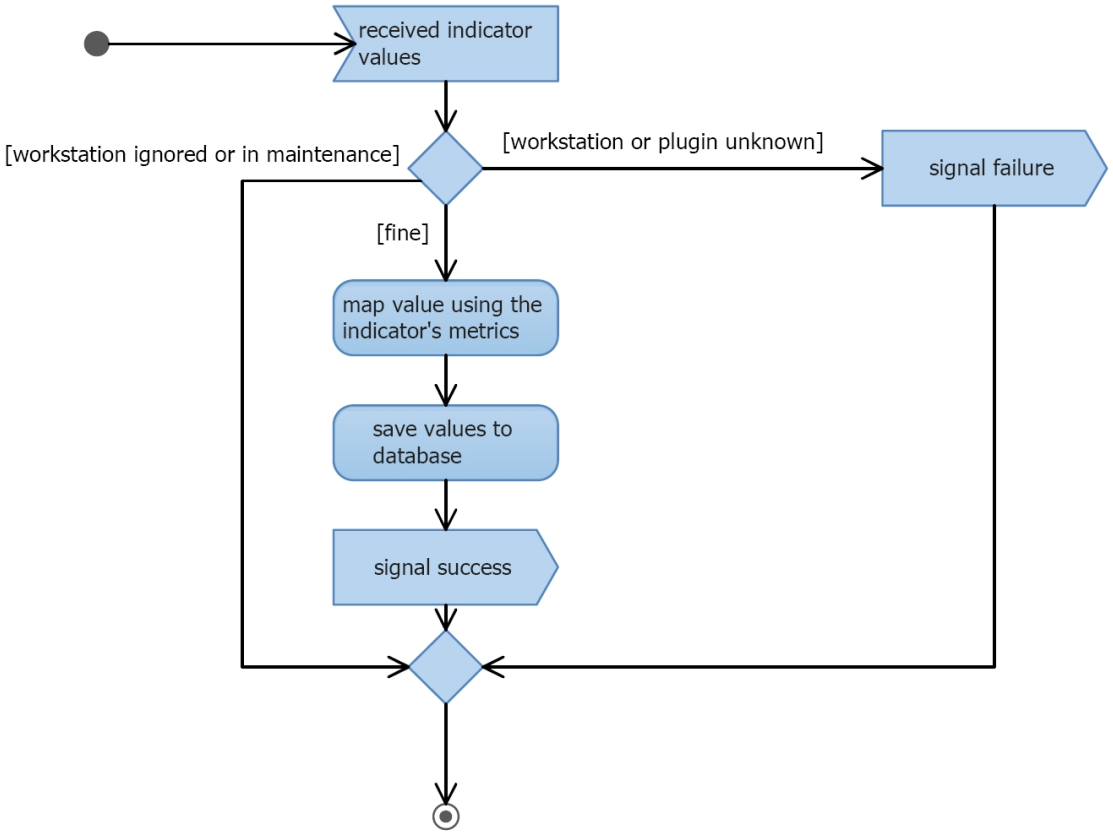

Abbildung 8.10: *Kenngrößenwert* empfangen

8.3 Interaktion von einer Workstation ausgehend

Eine *Workstation* meldet sich beim Start des Dienstes am *Server* an, sendet *Kenngrößenwerte* und meldet sich beim Beenden des Dienstes wieder ab. Diese Sequenzen werden im Folgenden jeweils als Aktivitätendiagramm dargestellt und erklärt.

8.3.1 Anmelden beim Server

Die *Workstation* sendet beim Start des Dienstes den *FQDN* und die MAC-Adresse der *Workstation* an den Server. Die MAC-Adresse ist unter allen *Workstations* eindeutig und wird für die nachfolgende Kommunikation verwendet. Anschließend fragt die *Workstation* als nächstes die Liste der Plugins ab und wählt diese zum Download aus, die noch nicht lokal auf der *Workstation* liegen. Plugins die zwar lokal auf der *Workstation* liegen, aber nicht in der Liste enthalten

sind, werden auf der *Workstation* gelöscht.

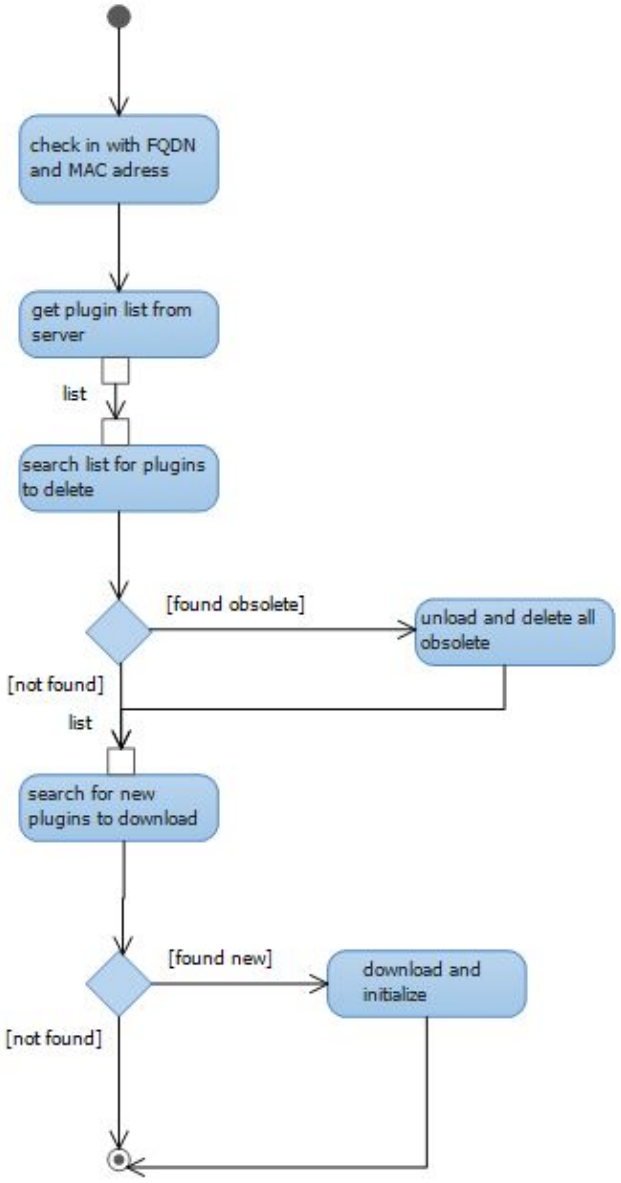

Abbildung 8.11: Anmelden beim *Server*

8.3.2 Abmelden vom Server

Bevor der Dienst auf einer Workstation beendet wird, beispielsweise wegen Neustart oder Herunterfahren des Systems, meldet sich dieser beim *Server* ab. Um sich vom *Server* abzumelden ruft eine *Workstation* den entsprechenden Befehl mit der MAC-Adresse der *Workstation* auf.

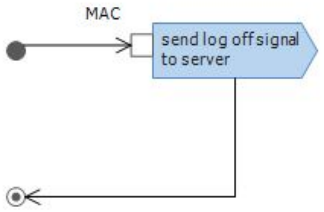

Abbildung 8.12: Abmelden vom *Server*

8.3.3 *Kenngrößenwerte* senden

Sobald das Intervall einer *Kenngröße* auf einer Workstation abläuft, wird der *Kenngrößenwert* erhoben. Dieser wird anschließend mit dem Filter dieser *Kenngröße* verglichen. Wenn der *Kenngrößenwert* nicht herausgefiltert wird, wird er schließlich an den Server übertragen. Ist eine Verbindung zum Web Service nicht möglich, wird die erfasste *Kenngröße* nicht übertragen.

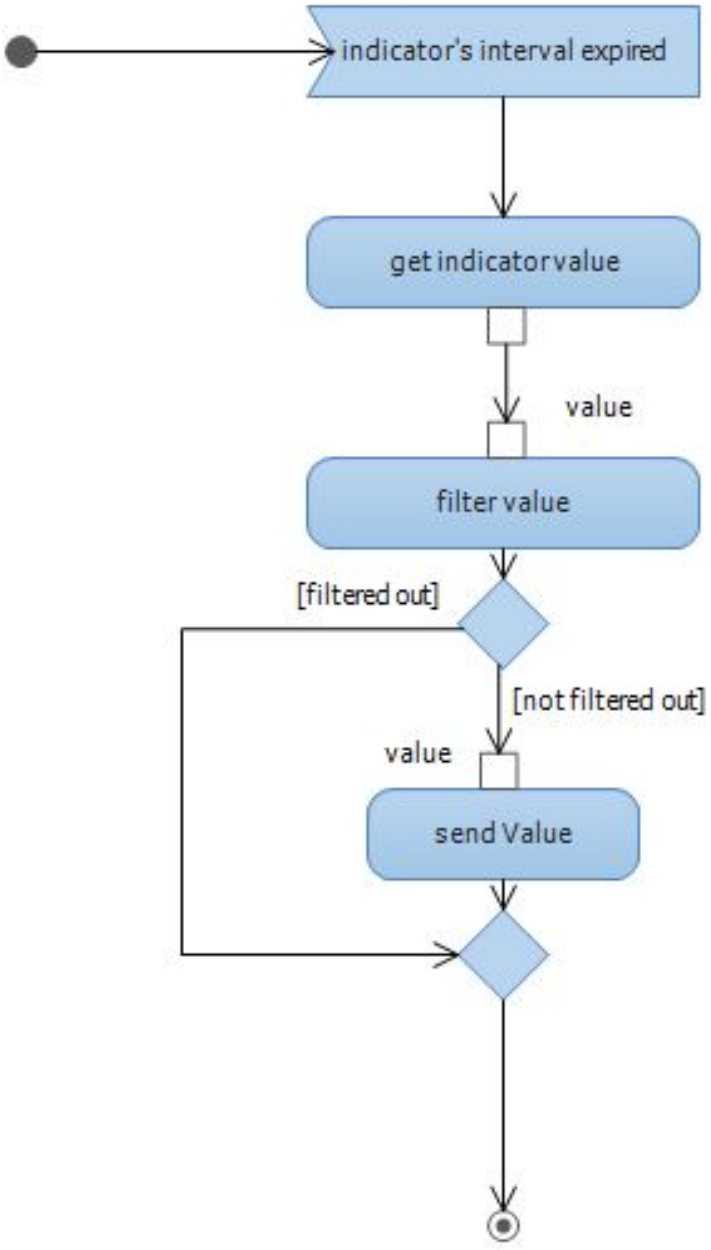

Abbildung 8.13: *Kenngrößenwert* senden

Kapitel 9

Datenhaltung

In diesem Kapitel wird beschrieben, wie das *System* Daten dauerhaft speichert und verwaltet. Dazu werden C#-Settings und eine Datenbank eingesetzt. Globale Einstellungen werden auf dem Server als C#-Settings im Benutzerbereich sowie im Anwendungsbereich hinterlegt, *Plugin*-Informationen, Informationen über *zu überwachende Rechner*, *Kenngrößen*, *Kenngrößenwerte*, Oberflächeneinstellungen und Benachrichtigungen werden in der Datenbank abgelegt. Die genaue Struktur und Art der gespeicherten Informationen wird im Folgenden genauer betrachtet.

9.1 Datenbank

In diesem Abschnitt wird der Aufbau der Datenbank mit Hilfe von ER-Diagrammen genauer erläutert. Dazu wird zunächst ein Überblick über alle beteiligten Entitäten gegeben. Anschließend werden zusammengehörige Gruppen von Entitäten und derer Relationen untereinander genauer beschrieben.

9.1.1 Überblick

In Abbildung 9.1 werden alle in der Datenbank enthaltenen Entitäten und ihre Relationen dargestellt. Diese Abbildung dient dazu einen Überblick über die gesamte Datenbank zu vermitteln. Eine genauere Beschreibung ist in den folgenden Abschnitten zu finden.

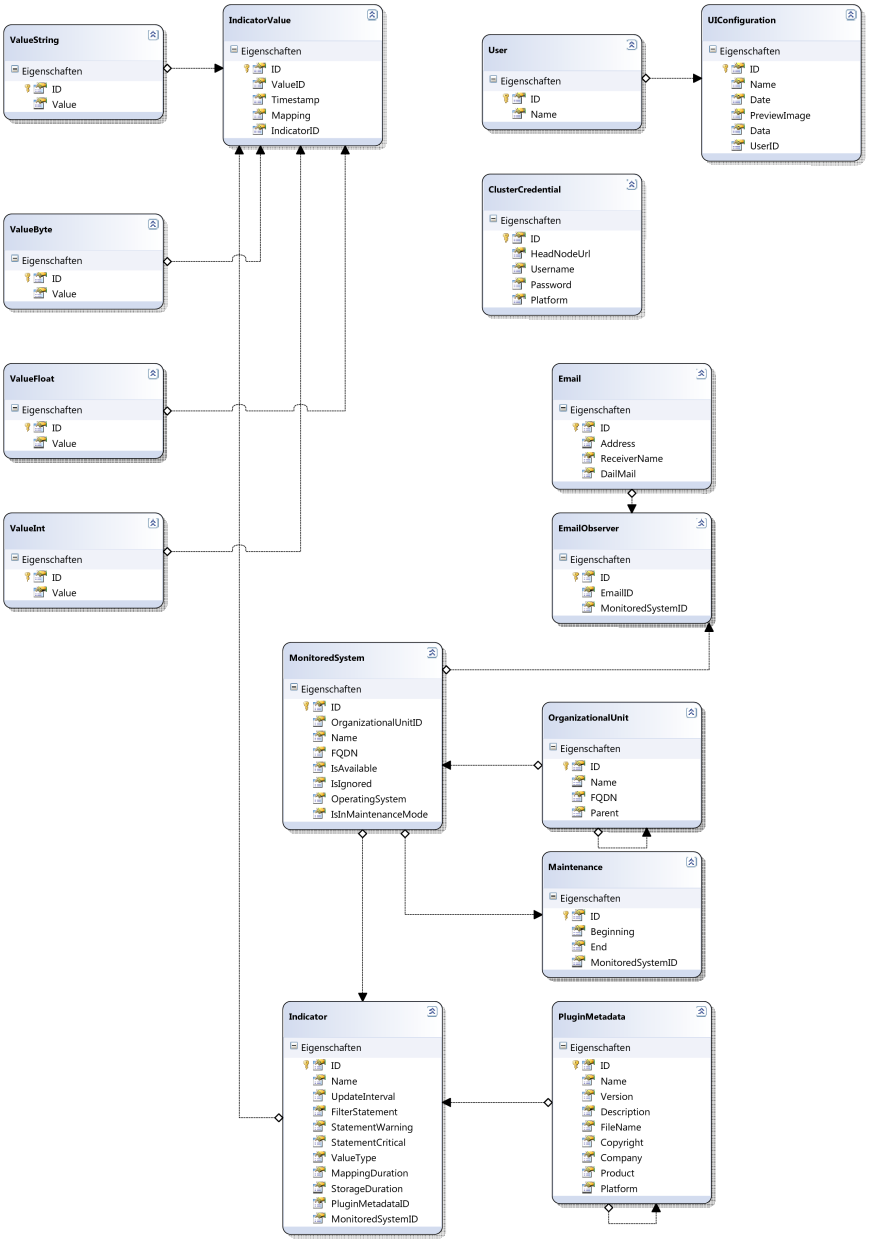

Abbildung 9.1: ER-Diagramm: Überblick über die Datenbankstruktur

9.1.2 Listen und Gruppen

Die in Abbildung 9.2 zu sehenden Tabellen enthalten Informationen über Gruppen- und Listenzugehörigkeiten aller *zu überwachender Rechner*. In der Tabelle 'Maintenance' sind alle *zu*

überwachenden Rechner enthalten, die einmal in den Zustand 'Wartung' versetzt wurden. Neben dem *zu überwachenden Rechner* wird zusätzlich das Ein- und Austrittsdatum gespeichert. Wenn sich ein *zu überwachender Rechner* im Wartungszustand befindet, enthält das Feld 'End' der Wert 'null'. Die Tabelle 'OrganisationalUnit' ordnet die *zu überwachenden Rechner* den im *System* vorhandenen *Organisationseinheiten* zu. Dabei kann eine *Organisationseinheit* weitere *Organisationseinheiten* enthalten. Eine übergeordnete *Organisationseinheit* wird über das Feld 'Parent' referenziert. Die 'IgnoreList' wird durch die boolsche Markierung 'Ignored' in der Tabelle 'MonitoredSystem' repräsentiert.

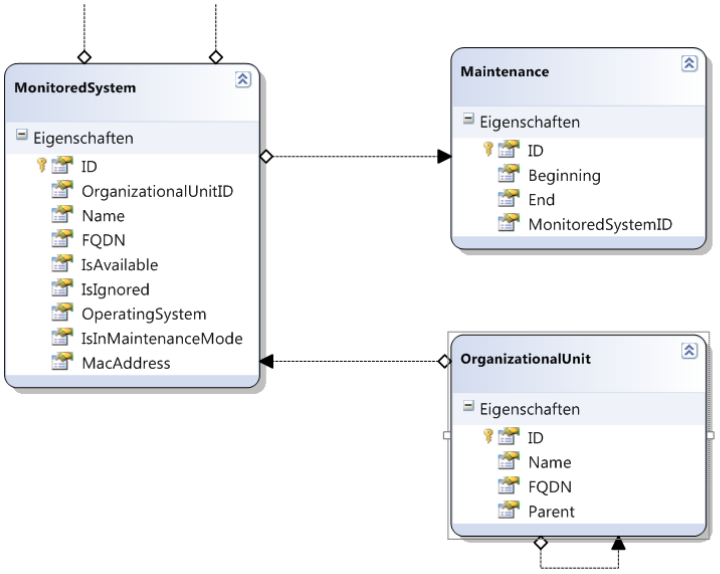

Abbildung 9.2: ER-Diagramm: Listen und Gruppen

9.1.3 Benachrichtigungen

Alle Daten die zum Versenden von E-Mail-Benachrichtigungen benötigt werden, werden in den Tabellen in Abbildung 9.4 gespeichert. Alle E-Mail-Adressen, die in der Tabelle 'Email' enthalten sind, erhalten E-Mail-Warnungen aller *zu überwachender Rechner*, die ihr über die Tabelle 'MailObserver' zugeordnet wurden. Der Tagesbericht wird an alle E-Mail-Adressen, bei denen der Flag 'DailyMail' auf 'true' gesetzt wurde, versendet.

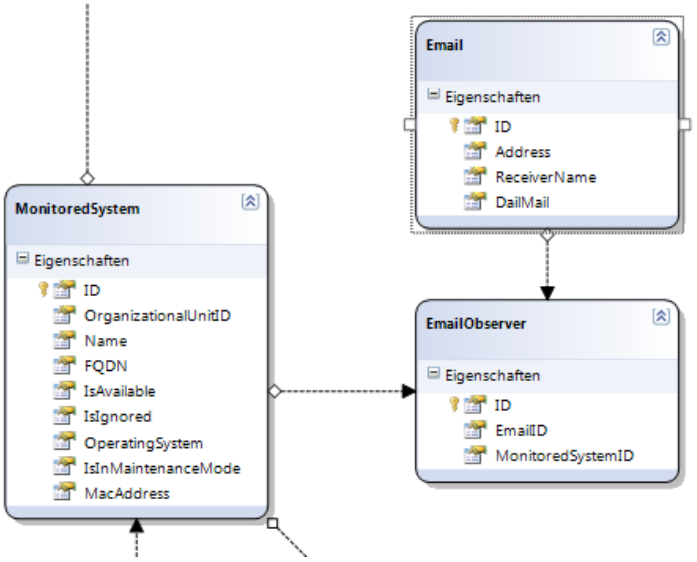

Abbildung 9.3: ER-Diagramm: Benachrichtigungen

9.1.4 GUI Einstellungen

Die Tabelle 'UIConfiguration' enthält alle *Layouts*, die im *System* vorhanden sind. Jedem *Layout* wird der Name des Uploaders zugeordnet.

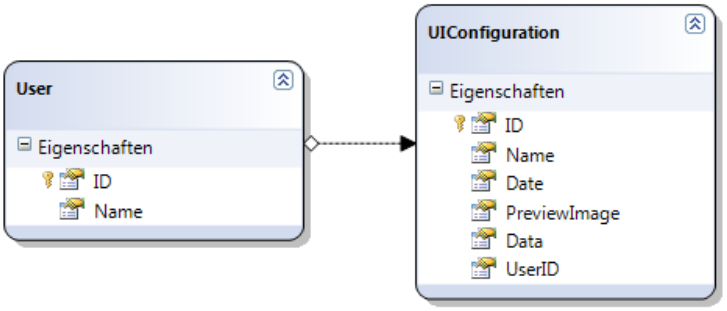

Abbildung 9.4: ER-Diagramm: Einstellungen

9.1.5 Cluster Credential

Für jedes *Cluster* sind in dieser Tabelle die Headnode-URL, die Plattform und die Zugangsdaten gespeichert.

9.1.6 Datenerfassung

Die Tabellen in Abbildung 9.5 enthalten sowohl Daten, die für die Beschaffung und Bewertung der *Kenngrößenwerte* benötigt werden als auch die *Kenngrößenwerte* selbst. *Plugin*-Informationen werden in den Tabellen 'PluginMetaData' und 'Indicator' gespeichert. In der Tabelle 'Indikator' werden zusätzlich Einstellungen für jeden *zu überwachenden Rechner* pro *Kenngröße* hinterlegt. Dazu zählen Metriken, Filter, Updateintervalle, Speicherdauern und die Gültigkeit des kritischsten Zustandes. Die Plugin-DLLs selbst werden nicht in der Datenbank gespeichert (siehe Kapitel 9.3). Die Tabellen 'IndicatorValue' ordnet jedem *zu überwachenden Rechner* pro *Kenngröße* mehrere *Kenngrößenwerte* inklusive Timestamp und Mapping zu. Die *Kenngrößenwerte* befinden sich je nach Datentyp in einer der folgenden Tabellen: 'ValueFloat', 'ValueString', 'ValueByte' oder 'ValueInt'. Welche *Kenngröße* mit welchen Datentyp abgespeichert ist, wird in der Tabelle 'IndicatorValue' angegeben.

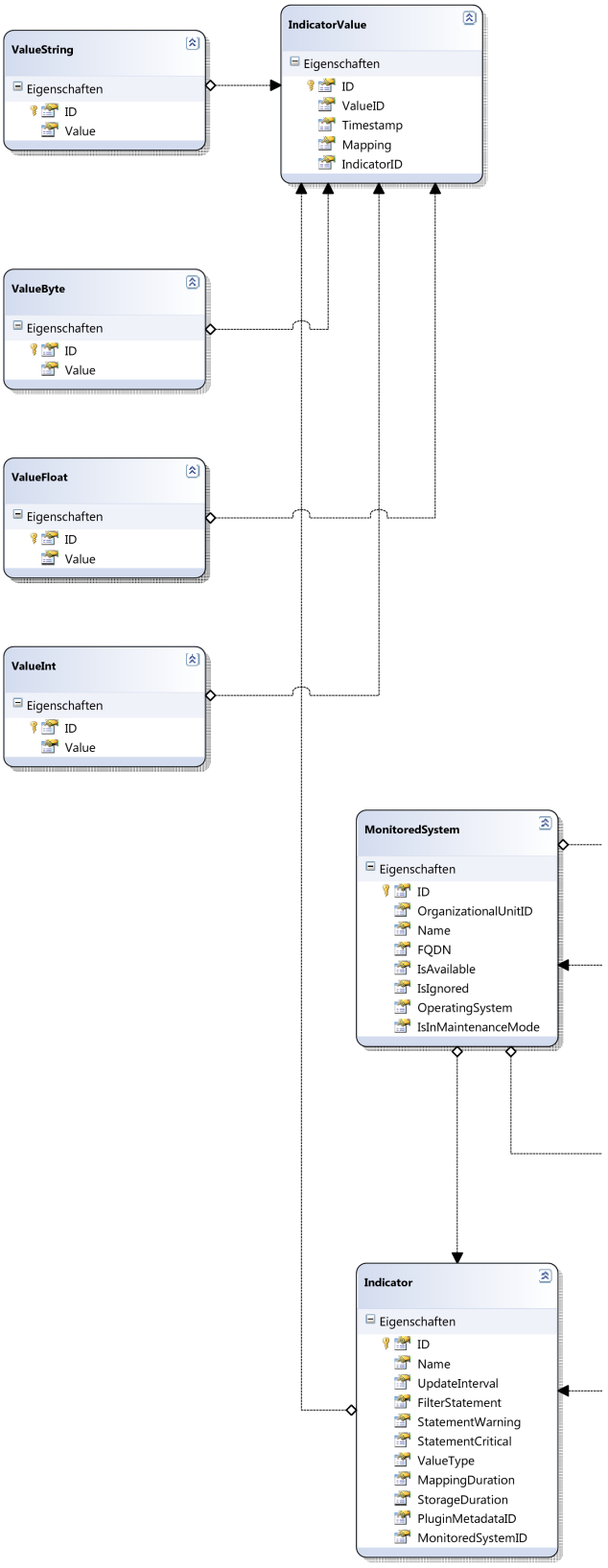

Abbildung 9.5: ER-Diagramm: Datenerfassung

9.2 XML-Dateien

In diesem Abschnitt wird beschrieben, wie *Client*-Einstellungen und globale Einstellungen mit Hilfe von C#-Settings gespeichert werden.

9.2.1 Globale Einstellungen

In den C#-Settings des Servers werden folgende Einstellungen im Benutzerbereich oder im Anwendungsbereich gespeichert:

- MISDConnectionString (String)
- Programmverzeichnis (String)
- Plugin Pfad im Verzeichnis (String)
- Plugin Pfad Windows im Verzeichnis (String)
- Plugin Pfad Linux im Verzeichnis (String)
- Plugin Pfad Bright Cluster im Verzeichnis (String)
- Plugin Pfad HPC Cluster im Verzeichnis (String)
- Plugin Pfad Server im Verzeichnis (String)
- Template Pfad Server im Verzeichnis (String)
- Mail SMTP Host (String)
- Mail SMTP Port (Integer)
- Absender Mail Adresse (String)
- Absender Mail Adresse Passwort (String)
- Debug Modus (Boolean)
- Cleaner Intervall (TimeSpan)
- Tagesbericht Intervall (TimeSpan)
- Name der Default *Organisationseinheit*

- Client Web Service URL
- Workstation Web Service URL
- Anmeldename Bright Cluster
- Anmeldepasswort Bright Cluster

9.2.2 Client Einstellungen

Auf jedem *Client* werden die folgenden Anzeigeeinstellungen in den C#-Einstellungen im Benutzerbereich oder Anwendungsbereich gespeichert.

- Aktualisierungsrate der Oberfläche (TimeSpan)
- Länge des angezeigten Namens, bevor er abgeschnitten wird (Zeichen als Integer)
- Schriftart
- Schriftgröße

9.2.3 Plugineinstellungen

Die Plugins stellen selbst inital einige Informationen zur Verfügung:

AssemblyInfo

Diese Assemblyinformationen müssen ausgefüllt sein, damit das Plugin korrekt verarbeitet werden kann.

- Titel (String)
- Beschreibung (String)
- Company (String)
- Product (String)
- Copyright (String)
- Version (String, automatisch)

IndicatorSettings

Die IndicatorSettings sind als Liste pro Indicator in dem Plugin definiert und umfassen:

- Pluginname (String)
- Indicatorname (String)
- MAC-Adresse des *zu überwachender Rechner* (leerer String)
- Filterstatement als RegEx (String)
- Update Intervall (Timespan)
- Storage Duration (Timespan)
- Mapping Duration (Timespan)
- Datentyp (Core.DataTyp)
- Metric Warnung als RegEx (String)
- Metric Kritisch als RegEx (String)

9.3 Speicherorte

Dieser Abschnitt beschreibt die Speicherorte auf dem *Server*.

Plugins

Verzeichnis der Plugins: '/MISD/Plugins/'
Verzeichnis der Windows Plugins: '/MISD/Plugins/Workstation/Windows'
Verzeichnis der Linux Plugins: '/MISD/Plugins/Workstation/Linux'
Verzeichnis der Server Plugins: '/MISD/Plugins/Server'
Verzeichnis der Bright-Cluster Plugins: '/MISD/Plugins/Cluster/Bright'
Verzeichnis der HPC-Cluster Plugins: '/MISD/Plugins/Cluster/HPC'
Verzeichnis der Visualisierungs-Plugins: '/MISD/Plugins/Client'

Die Plugins sind nach dem Schema 'MISD.Plugins.[Platform].[Pluginname].dll' benannt.

Templates

Verzeichnis der Templates: '/MISD/Templates'

Der Name des Warnungmail-Template ist 'WarningMailTemplate.txt'

Kapitel 10

Zustände

Dieses Kapitel erläutert die Zustände im System *MISD OWL*. Eine *Workstation* kann nach dem Hinzufügen der Informationen, unter Anderem aus dem *Active Directory*, fünf verschiedene Zustände annehmen die auch die *Status* beinhalten. Die *Status* sind also eine echte Teilmenge der Zustände von *MISD OWL*:

- *OK*
- *WARNUNG*
- *KRITISCH*
- *Wartungszustand*
- Element der *Ignore-Liste*

Die *Status OK*, *WARNUNG* und *KRITISCH* sind reine visuelle Status. Die *Status* der *Workstation* werden automatisch aus dem Mapping der *Kenngrößen* bestimmt. Wurde in einem festgelegtem Zeitraum eine *Kenngröße* auf *WARNUNG* oder *KRITISCH* abgebildet, befindet sich die *Workstation* in diesem Zustand, wobei immer der kritischste Zustand angenommen wird. Die *Workstation* ist also im Zustand *KRITISCH* sobald eine *Kenngröße* in dem festgelegtem Zeitraum (Gültigkeitsdauer des kritischen Zustands) auf *KRITISCH* abgebildet wurde.

Setzt der *Benutzer* den *Status* zurück, werden kritische *Kenngrößenwerte* im vergangenen Zeitraum der Gültigkeitsdauer des kritischen Zustands ignoriert und stattdessen der aktuelle Zustand angezeigt.

Aus jedem *Status* kann eine *Workstation* in den Zustand *Wartungszustand* oder Element der *Ignore-Liste* versetzt werden. Diese beiden Funktionen werden in der Spezifikation erläutert.

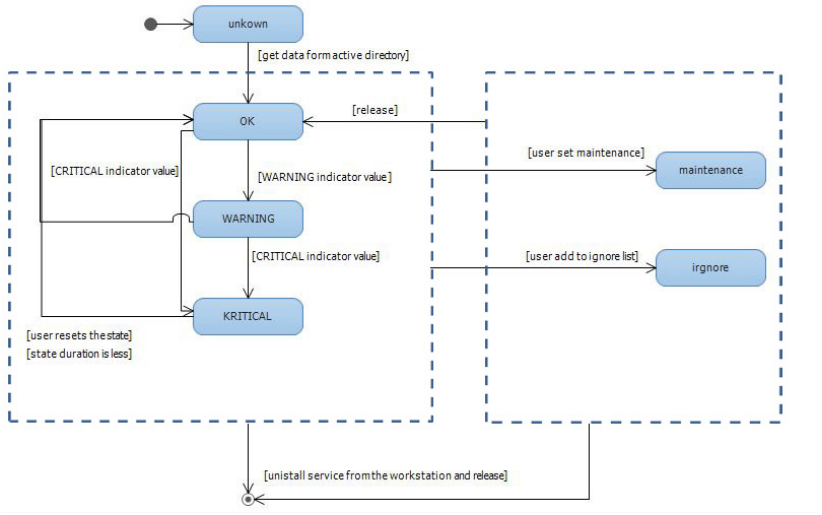

Abbildung 10.1: Zustand einer *Workstation*

Kapitel 11

Schnittstellen

Dieser Abschnitt behandelt die Web Service Schnittstellen des Servers.

11.1 Datenstrukturen

Folgende Datenstrukturen werden zur Kommunikation mit dem Web Service benötigt.

```
1
  /// <summary>
    /// Contains information about a plugin without containing the plugin
       file itself.
    /// </summary>
    [DataContract]
6   public class PluginMetadata
    {
        [DataMember]
        public Version Version { get; set; }

11      [DataMember]
        public string Name { get; set; }

        [DataMember]
        public string Description { get; set; }
16
        [DataMember]
        public string FileName { get; set; }
```

```
        [DataMember]
        public string Company { get; set; }

        [DataMember]
        public string Copyright { get; set; }

        [DataMember]
        public string Product { get; set; }

        [DataMember]
        public List<IndicatorSettings> Indicators { get; set; }
    }

    /// <summary>
    /// Contains a plugin's DLL file.
    /// </summary>
    [DataContract]
    public class PluginFile
    {
        [DataMember]
        public string FileName { get; set; }

        [DataMember]
        public string FileAsBase64 { get; set; }
    }

/// <summary>
    /// Contains information about a single indicator for a monitored
       system.
    /// It belongs to a plugin and a monitored system and contains the
       filterStatement, the update intervall, the storage duration, the
       data type and the metric settings.
    /// Each Plugin defines this IndicatorSettings for its indicators.
    ///
    /// </summary>
    [DataContract]
    public class IndicatorSettings
    {
        public IndicatorSettings() { }

        public IndicatorSettings(string pluginName, string indicatorName,
           string monitoredSystemMAC, string filterStatement,
```

```
                                TimeSpan updateInterval, TimeSpan
                                    storageDuration, TimeSpan
                                    mappingDuration, DataType dataType,
                                string metricWarning, string
                                    metricCritical)
        {
61          this.PluginName = pluginName;
            this.IndicatorName = indicatorName;
            this.MonitoredSystemMAC = monitoredSystemMAC;
            this.FilterStatement = filterStatement;
            this.UpdateInterval = updateInterval;
66          this.StorageDuration = storageDuration;
            this.MappingDuration = mappingDuration;
            this.DataType = dataType;
            this.MetricWarning = metricWarning;
            this.MetricCritical = metricCritical;
71      }

        [DataMember]
        public string PluginName { get; set; }

76      [DataMember]
        public string IndicatorName { get; set; }

        [DataMember]
        public string MonitoredSystemMAC { get; set; }
81
        [DataMember]
        public string FilterStatement { get; set; }

        [DataMember]
86      public TimeSpan UpdateInterval { get; set; }

        [DataMember]
        public TimeSpan StorageDuration { get; set; }

91      [DataMember]
        public TimeSpan MappingDuration { get; set; }

        [DataMember]
        public DataType DataType { get; set; }
96
```

```
        [DataMember]
        public string MetricWarning { get; set; }

        [DataMember]
        public string MetricCritical { get; set; }

        [DataMember]
        public MappingState IndicatorMapping { get; set; }
    }

    /// <summary>
    /// Defines the two different supported operating systems as well as
       the two supported cluster manager.
    /// </summary>
    [DataContract]
    public enum Platform
    {
        Windows = 0,
        Linux = 1,
        Bright = 2,
        HPC = 3,
        Server = 4,
        Visualization = 5
    }

    /// <summary>
    /// Specifies the datatypes of indicator values.
    /// This is used for Database Storage.
    /// </summary>
    [DataContract]
    public enum DataType
    {
        [EnumMember]
        String=0,
        [EnumMember]
        Float=1,
        [EnumMember]
        Int=2,
        [EnumMember]
        Byte=3
    }
```

11.2 Workstation Web Service

```
    /// <summary>
    /// Defines the methods that are exposed via the Workstation Web
       Service.
    /// </summary>
    [ServiceContract]
    public interface IWorkstationWebService
    {
        #region Main Intervall Update

        /// <summary>
        /// Gets the update interval for services, for the indicators and
           for the filters.
        /// </summary>
        /// <returns>The main update interval.</returns>>
        [OperationContract]
        TimeSpan GetMainUpdateInterval();

        #endregion

        #region Server Login

        /// <summary>
        /// This method registers a workstation on the server.
        /// </summary>
        /// <param name="monitoredSystemFQDN">The FQDN of the Workstation,
           it must match the name that is listed in the
           ActiveDirectory.</param>
        /// <param name="monitoredSystemMAC">The MAC of the
           Workstation.</param>
        /// <param name="operatingSystem">The operating system of the
           workstation.</param>
        /// <returns>A value that indicates, whether the method call was
           successful or not.</returns>
        [OperationContract]
```

```
        bool SignIn(string monitoredSystemFQDN, string monitoredSystemMAC,
            byte operatingSystem);

        /// <summary>
        /// This method tells the server that the workstation is shutting
            down.
        /// </summary>
        /// <param name="monitoredSystemMAC">The MAC of the
            workstation.</param>
        /// <returns>A value that indicates, whether the method call was
            successful or not.</returns>
        [OperationContract]
        bool SignOut(string monitoredSystemMAC);

        #endregion

        #region Plugin Management

        /// <summary>
        /// Gets a list of all plugins, that are currently available on the
            server.
        /// </summary>
        /// <param name="monitoredSystemMAC">The MAC of the workstation
            that is requesting.</param>
        /// <returns>The metadata of all plugins that are available on the
            server, individually created for the specified
            workstation.</returns>
        [OperationContract]
        List<PluginMetadata> GetPluginList(string monitoredSystemMAC);

        /// <summary>
        /// Downloads the plugins that match the given names.
        /// </summary>
        /// <remarks>
        /// This method gets called only by workstations.
        /// </remarks>
        /// <param name="monitoredSystemMAC">The MAC of the workstation
            that wants to download the plugins.</param>
        /// <param name="pluginNames">The names of the plugins that shall
            be downloaded.</param>
        /// <returns>The plugin files that are specific for the given
            workstation.</returns>
```

```
59          [OperationContract]
            List<PluginFile> DownloadPlugins(string monitoredSystemMAC,
                List<string> pluginNames);

            /// <summary>
            /// This method transfers indicator values to the server.
64          /// </summary>
            /// <param name="monitoredSystemMAC">The MAC of the
                workstation.</param>
            /// <param name="pluginName">The name of the plugin.</param>
            /// <param name="indicatorValues">A list containing tuples of:
                Indicator | IndicatorValue | Datatype | DateTime.
            /// These do not have to be all plugin values of the given plugin.
69          /// The DateTime is the time when the value was acquired. </param>
            /// <returns>A value that indicates, whether the method call was
                successful or not.</returns>
            [OperationContract]
            bool UploadIndicatorValues(string monitoredSystemMAC, string
                pluginName, List<Tuple<string, Object, MISD.Core.DataType,
                DateTime>> indicatorValues);

74          /// <summary>
            /// This method transfers a single indicator value to the server.
            /// </summary>
            /// <param name="monitoredSystemMAC">The MAC of the
                workstation.</param>
            /// <param name="pluginName">The name of the plugin.</param>
79          /// <param name="indicatorValueName">The name of the
                indicator.</param>
            /// <param name="value">The value itself.</param>
            /// <param name="valueDataType">The datatype of the value.</param>
            /// <param name="aquiredTimestamp">The timestamp of the
                value.</param>
            /// <returns>A value that indicates, whether the method call was
                successful or not.</returns>
84          [OperationContract]
            bool UploadIndicatorValue(string monitoredSystemMAC, string
                pluginName, string indicatorValueName, object value, DataType
                valueDataType, DateTime aquiredTimestamp);

            /// <summary>
            /// Gets all filters for a certain plugin of a given workstation.
```

```
89          /// </summary>
            /// <param name="monitoredSystemMAC">The MAC of the
                workstation.</param>
            /// <param name="pluginName">The name of the plugin.</param>
            /// <returns>A list containing tuples of: IndicatorName |
                FilterStatement.</returns>
            [OperationContract]
94          List<Tuple<string, string>> GetFilters(string monitoredSystemMAC,
                string pluginName);

            /// <summary>
            /// Gets the filters for a certain plugin of a given workstation
                and an indicator name.
            /// </summary>
99          /// <param name="monitoredSystemMAC">The MAC of the
                workstation.</param>
            /// <param name="pluginName">The name of the plugin.</param>
            /// <param name="indicatorName">the name of the indicator</param>
            /// <returns>the FilterStatement as string..</returns>
            [OperationContract]
104         string GetFilter(string monitoredSystemMAC, string pluginName,
                string indicatorName);

            /// <summary>
            /// Gets all update intervals for a certain plugin of a given
                workstation.
            /// </summary>
109         /// <param name="monitoredSystemMAC">The MAC of the
                workstation.</param>
            /// <param name="pluginName">The name of the plugin.</param>
            /// <returns>A list containing tuples of: IndicatorName |
                Duration.</returns>
            [OperationContract]
            List<Tuple<string, long?>> GetUpdateIntervals(string
                monitoredSystemMAC, string pluginName);
114
            /// <summary>
            /// Gets update interval for a certain plugin of a given
                workstation and a given indicatorname.
            /// </summary>
            /// <param name="monitoredSystemMAC">The MAC of the
                workstation.</param>
```

```
        /// <param name="pluginName">The name of the plugin.</param
        /// <param name="indicatorName">the name of the indicator</param>
        /// <returns>the duration of the UpdateIntervall.</returns>
        [OperationContract]
        long GetUpdateInterval(string monitoredSystemMAC, string
            pluginName, string indicatorName);

        /// <summary>
        /// Logs an event that happened on a workstation to the server's
            windows event log.
        /// Debug messages are written into a text file.
        /// </summary>
        /// <param name="message">event description</param>
        /// <param name="type">type of the event</param>
        /// <returns></returns>
        [OperationContract]
        void WriteLog(string message, LogType type);

        #endregion
    }
```

11.3 Client Web Service

```
    /// <summary>
    /// Defines the methods that are exposed via the Client Web Service.
    /// </summary>
    [ServiceContract]
    public interface IClientWebService
    {
        #region Plugin Management

        /// <summary>
        /// Gets a list of all plugins, that are currently available on the
            server.
        /// </summary>
        /// <returns>The metadata of all plugins that are available on the
            server.</returns>
        [OperationContract]
```

```
        List<PluginMetadata> GetPluginList(string plattform);

        /// <summary>
        /// Downloads the plugins that match the given names.
        /// </summary>
        /// <remarks>
        /// This method gets called only by workstations.
        /// </remarks>
        /// <param name="monitoredSystemMAC">The MAC of the workstation
            that wants to download the plugins.</param>
        /// <param name="pluginNames">The names of the plugins that shall
            be downloaded.</param>
        /// <returns>The plugin files that are specific for the given
            workstation.</returns>
        [OperationContract]
        List<PluginFile> DownloadPlugins(List<string> pluginNames);

        #endregion

        #region Filter, Metriken, Mapping und Aktualisierungsintervalle

        /// <summary>
        /// Gets all indicator-settings by plugin for the given workstation.
        /// </summary>
        /// <param name="monitoredSystemMAC">The MAC of the workstation to
            get the settings of.</param>
        /// <returns>PluginName | IndicatorSetting</returns>
        [OperationContract]
        List<IndicatorSettings> GetIndicatorSetting(string
            monitoredSystemMAC);

        /// <summary>
        /// Updates the given indicator-settings.
        /// </summary>
        /// <param name="settings">The new settings.</param>
        /// <returns>True, if the update was successfull, false
            otherwise.</returns>
        [OperationContract]
        bool SetIndicatorSetting(List<IndicatorSettings> settings);

        #endregion
```

```
        #region MonitoredSystem

        /// <summary>
        /// Resets the mapping of a workstation
        /// </summary>
        /// <param name="macList">Tuple: mac | updateTime</param>
        /// <returns></returns>
        [OperationContract]
        bool ResetMapping(List<Tuple<string, DateTime>> macList);

        /// <summary>
        /// Activates the maintenance mode for several workstations.
        /// </summary>
        /// <param name="monitoredSystemMACAddresses">Tuple: mac |
            updateTime</param>
        /// <returns></returns>
        [OperationContract]
        List<String> ActivateMaintenanceMode(List<Tuple<string, DateTime>>
            monitoredSystemMACAddresses);

        /// <summary>
        /// Deactivates the maintenance mode for several workstations.
        /// </summary>
        /// <param name="monitoredSystemMACAddresses">Tuple: mac |
            updateTime</param>
        /// <returns>A list containing the names of the workstations that
            have been put out of maintenance mode successfully.</returns>
        [OperationContract]
        List<String> DeactivateMaintenanceMode(List<Tuple<string,
            DateTime>> monitoredSystemMACAddresses);

        /// <summary>
        /// Changes the ouIDs of the given monitoredSytems.
        /// </summary>
        /// <param name="monitoredSystems">List of tuples: MAC | ouID |
            updateTime </param>
        /// <returns>boolean</returns>
        [OperationContract]
        bool MoveMonitoredSystem(List<Tuple<string, int, DateTime>>
            monitoredSystems);

        #endregion
```

```
        #region GUI-Configuration

88      /// <summary>
        /// Gets a list containing all UI configurations , that are
            available on the
        /// server.
        /// </summary>
        /// <returns>A list containing the UI configurations.</returns>
93      [OperationContract]
        List<Layout> GetUIConfigurationList();

        /// <summary>
        /// Adds a UI configuration of a certain user to the server 's UI
98      /// configurations list.
        /// </summary>
        /// <param name="name">The name of the UI configuration.</param>
        /// <param name="userName">The bane of the user.</param>
        /// <param name="previewImageAsBase64">A preview image as base64 -
            String.</param>
103     /// <param name="data">The main window view model.</param>
        /// <returns>The server version of the UI configuration.</returns>
        [OperationContract]
        Layout AddUIConfiguration(string name, string userName, byte[]
            previewImageAsBase64, object data, DateTime Date);

108     /// <summary>
        /// Removes a UI configuration from the server.
        /// </summary>
        /// <param name="id">The ID of the UI configuration.
        /// <returns>A value that indicates , whether the UI configuration
            has been removed successfully , or not</returns>
113     [OperationContract]
        bool RemoveUIConfiguration(int id);

        /// <summary>
        /// Updates a UI configuration with the specified values .
118     /// Remarks : Parameters that are null will not be changed in the
            database .
        /// </summary>
        /// <param name="configurationID">The ID of the UI configuration
            that is about to be updated.</param>
```

```
        /// <param name="name">The new name of the UI configuration.</param>
        /// <param name="userName">The name of the UI configuration.</param>
        /// <param name="previewImageAsBase64">The new preview image of the
            UI configuration.</param>
        /// <param name="date">The new UI configuration data.</param>
        /// <returns>The server version of the UI configuation.</returns>
        [OperationContract]
        Layout UpdateUIConfiguration(int configurationID, string name,
            string userName, byte[] previewImageAsBase64, object data,
            DateTime Date);

        #endregion

        #region Ignore List

        /// <summary>
        /// Adds several workstations to the ignore list.
        /// </summary>
        /// <param name="monitoredSystemMACAddresses">List of tuples: MAC |
            updateTime</param>
        /// <returns>A list containing the names of the workstations that
            have been added to the ignore list successfully.</returns>
        [OperationContract]
        List<String> AddMonitoredSystemsToIgnoreList(List<Tuple<string,
            DateTime>> monitoredSystemMACAddresses);

        /// <summary>
        /// Removes several workstations from the ignore list .
        /// </summary>
        /// <param name="monitoredSystemMACAddresses">List of tuples: MAC |
            updateTime</param>
        /// <returns>A list containing the names of the workstations that
            have been removed from the ignore list successfully.</returns>
        [OperationContract]
        List<String>
            RemoveMonitoredSystemsFromIgnoreList(List<Tuple<string,
            DateTime>> monitoredSystemMACAddresses);

        /// <summary>
        /// Gets all workstations , that are currently ignored by the
            server.
        /// </summary>
```

```
        /// <returns>A list containing a tuple of: mac | name</returns>
153     [OperationContract]
        List<Tuple<string, string>> GetIgnoredMonitoredSystems();

        #endregion

158     #region Email Berichte und Warnungen

        /// <summary>
        /// Adds a email - adress to the adress - list.
        /// </summary>
163     /// <param name="emailAdress">A string containing a email -
            adress.</param>
        /// <param name="userName">A string containing the full name of the
            user.</param>
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
        int? AddEMail(string emailAdress, string userName);
168
        /// <summary>
        /// Remove a email - adress form the system
        /// </summary>
        /// <param name="emailAdress">A string containing a email -
            adress.</param>
173     /// <param name="userName">A string containing the full name of the
            user.</param>
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
        bool RemoveEMail(int userID);

178     /// <summary>
        /// Add a email - adress to the observer - list of several
            workstations .
        /// </summary>
        /// <param name="emailAdress">A string containing a email -
            adress</param>
        /// <param name="mac">A list containing the MACs of the monitored
            systems.</param>
183     /// <returns>true if the execution was done without errors
            .</returns>
```

```
        [OperationContract]
        bool AddMailObserver(int userID, List<string> mac);

        /// <summary>
        /// Remove a email - adress from the observer - list of several
            workstations .
        /// </summary>
        /// <param name="emailAdress">A string containing a email -
            adress</param>
        /// <param name="monitoredSystemMACs">A list containing the MACs of
            the monitored systems.</param>
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
        bool RemoveMailObserver(int userID, List<string>
            monitoredSystemMACs);

        /// <summary>
        /// Add a email - adress to the daily - mail list .
        /// </summary>
        /// <param name="mailID">A int containing a email ID</param>
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
        bool AddDailyMail(int userID);

        /// <summary>
        /// Remove a email - adress from the daily - mail list .
        /// </summary>
        /// <param name="emailAdress">A string containing a email -
            adress</param>
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
        bool DeleteDailyMail(int userID);

        /// <summary>
        /// Gets all email user datas
        /// </summary>
        /// <returns>List of Tuple with data: ID | username | user mail
            adress | daily mail </returns>
        [OperationContract]
```

```
        List<Tuple<int, string, string, bool>> GetAllMailData();
218
        /// <summary>
        /// Retuns all observer of an monitored system
        /// </summary>
        /// <param name="userID">User ID of the email adress</param>
223     /// <returns>List of Tuples with user name | email adress</returns>
        [OperationContract]
        List<WorkstationInfo> GetObserver(int userID);

        /// <summary>
228     /// Changes the mail adress and the username of a given mail adress
        /// </summary>
        /// <param name="userID">User ID of the email adress</param>
        /// <param name="userNameNew"></param>
        /// <param name="mailAdressNew"></param>
233     /// <returns>true if the execution was done without
            errors.</returns>
        [OperationContract]
        bool ChangeEmail(int userID, string userNameNew, string
            mailAdressNew);

        #endregion
238
        #region Organisationseinheiten

        /// <summary>
        /// Changes the displayed name of a orginisational unit .
243     /// </summary>
        /// <param name="ouID">ID of the orginisational unit .</param>
        /// <param name="newName">The new name of the orginisational unit
            .</param>
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
248     bool ChangeOUName(int ouID, string newName, DateTime updateTime);

        /// <summary>
        /// Delete a given orginisational unit and adds the containing
            workstations to the ignore - list .
        /// </summary>
253     /// <param name="ouID">ID of the orginisational unit .</param>
```

```
        /// <returns>true if the execution was done without errors
            .</returns>
        [OperationContract]
        bool DeleteOU(int ouID);

        /// <summary>
        /// Adds a new orginisational unit .
        /// </summary>
        /// <param name="name">String containing the name of the new
            orginisational unit .</param>
        /// <param name="fatherOU">The orginisational unit which contains
            the new orginisational unit . NULL for a initial orginisational
            unit .</param>
        /// <returns>ID of the new orginisational unit .</returns>
        [OperationContract]
        int AddOU(string name, int? fatherOU, DateTime updateTime);

        /// <summary>
        /// Assigns a workstation to a new cluster .
        /// </summary>
        /// <param name="monitoredSystemMAC">The MAC of the workstation
            that is assigned to a new organiational unit.</param>
        /// <param name="newOUID">ID of the new organisational unit.</param>
        /// <returns>True if the execution was done withour
            errors.</returns>
        [OperationContract]
        bool AssignToOU(string monitoredSystemMAC, int newOUID);

        /// <summary>
        /// Gets all stored ous at the server.
        /// </summary>
        /// <returns></returns>
        [OperationContract]
        List<Tuple<int, string, string, int?, DateTime?>> GetAllOUs();

        /// <summary>
        /// Changes the parent ou of a given ou.
        /// </summary>
        /// <param name="ouID"></param>
        /// <param name="ouIDParent"></param>
        /// <returns></returns>
        [OperationContract]
```

```
        bool ChangeParent(int ouID, int? ouIDParent, DateTime updateTime);

        #endregion

        #region Aktuelle Kenngroessen

        /// <summary>
        /// Create's a List of all platform typse
        /// </summary>
        /// <returns>List of strings wiht all plattforms</returns>
        [OperationContract]
        List<string> GetAllPlatformTyps();

        /// <summary>
        /// Gets the IDs of all workstations that are known to the server .
        /// </summary>
        /// <returns>A list of workstation names .</returns>
        [OperationContract]
        List<int> GetMonitoredSystemIDs();

        /// <summary>
        /// Gets the IDs of all workstations that are known to the server .
        /// </summary>
        /// <returns>A list of workstation names .</returns>
        [OperationContract]
        List<string> GetWorkstationMACs();

        /// <summary>
        /// Gets infomation about workstations , containing the workstation
            's names and states .
        /// This method can be used to retrieve Level 1 data .
        /// </summary>
        /// <param name="monitoredSystemMACAddresses">The IDs of the
            workstations and the time of the last reset.</param>
        /// <returns> A list containing the workstation infos .</returns>
        [OperationContract]
        List<WorkstationInfo> GetMonitoredSystemInfo(List<Tuple<int,
            TimeSpan>> monitoredSystemIDsWithResetTime);

        /// <summary>
        /// Gets the latest indicator data for each indicator.
        /// </summary>
```

```
        /// <param name="macs">A list containing the macs of the monitored
            systems</param>
        /// <returns>List of: MAC | Pluginname | Indicatorname | Value |
            Mapping | Timestamp</returns>
        [OperationContract]
        List<Tuple<string, string, string, string, MappingState, DateTime>>
            GetLatestMonitoredSystemData(List<string> macList);

        /// <summary>
        /// Gets the complete data for serveral plugins of certain
            workstations.
        /// This method can be used to retrieve Level 2 data .
        /// </summary>
        /// <param name="macAndPluginName"> A list containing tuples of:
            MAC | PluginName</param>
        /// <returns>A list containing tuples of: MAC | PluginName |
            IndicatorName | Value | Mapping | Time</returns>
        [OperationContract]
        List<Tuple<string, string, string, string, MappingState, DateTime>>
            GetPluginData(List<Tuple<string, string>> macAndPluginName);

        /// <summary>
        /// Gets the data for all plugins of the given workstations .
        /// This method can be used to retrieve Level 3 data .
        /// </summary>
        /// <param name="mac">A list containing tuples of:  MAC | | Maximum
            numer of results? per inticator</param>
        /// <returns>A list containing tuples of: MAC | PluginName |
            IndicatorName | Value | Mapping | Time</returns>
        [OperationContract]
        List<Tuple<string, string, string, string, MappingState, DateTime>>
            GetCompletePluginDataList(List<string> macList, int?
            numberOfIndicators);

        #endregion

        #region Alte Kenngroessenwerte

        /// <summary>
        /// Gets the complete data for serveral plugins of certain
            workstations and a certain timespan.
        /// </summary>
```

```
        /// <param name="macAndProperties"> A list containing tuples of:
            MAC | PluginName | IndicatorName | LowerBound? | UpperBound? |
            Maximum numer of results?</param>
        /// <returns>A list containing tuples of: MAC | PluginName |
            IndicatorName | Value | Mapping | Time</returns>
        [OperationContract]
        List<Tuple<string, string, string, string, MappingState, DateTime>>
            GetData(List<Tuple<string, string, string, DateTime?, DateTime?,
            int?>> macAndProperties);

        #endregion

        #region Cluster verwalten
        /// <summary>
        /// Creates a list of all supported cluster types
        /// </summary>
        /// <returns></returns>
        [OperationContract]
        List<string> GetClusterTyps();

        /// <summary>
        /// Adds a new cluster to the misd owl system
        /// </summary>
        /// <param name="headnodeAddress">The name for the cluster as well
            as for the organisational unit for it's nodes.</param>
        /// <param name="username">The adress of the headnode of the
            clustermanager.</param>
        /// <param name="password">True if it's a HPC Cluster, false if
            it's a Bright Cluster.</param>
        /// <param name="database">database url for indicator datas</param>
        /// <param name="platform">Platform of the cluster</param>
        /// <returns>Cluster credential ID</returns>
        [OperationContract]
        int AddCluster(string headnodeAddress, string username, string
            password, string platform);

        /// <summary>
        /// Changes the cluster credentials of a cluster
        /// </summary>
        /// <param name="id">ID of the cluster credentials</param>
        /// <param name="data">A tuple containing headnode url | username |
            password |  platform</param>
```

```
        /// <returns></returns>
        [OperationContract]
        bool ChangeCluster(int id, Tuple<string, string, string, string>
            data);

        /// <summary>
        /// Removes a cluster from the system .
        /// </summary>
        /// <param name="clusterID">The ID of the cluster to be
            removed.</param>
        /// <returns>True if the execution was done withour
            errors.</returns>
        [OperationContract]
        bool DeleteCluster(int clusterID);

        /// <summary>
        /// Gets the cluster credentials
        /// </summary>
        /// <returns>A list containing tuples of ID | headnode url |
            username | password | platform </returns>
        [OperationContract]
        List<Tuple<int, string, string, string, string>> GetClusters();

        #endregion
    }
```

Anhang A

Anhang

A.1 Begriffslexikon

Begriff	**Active Directory**
Bedeutung	Mit Active Directory ist der Microsoft Active Directory Verzeichnisdienst gemeint. In diesem befinden sich verschiedene Daten, die von MISD für die zu überwachenden Rechner übernommen werden kann. Aus der Hierarchie des Active Directory kann beispielsweise die Anordnung der Kacheln in Organisationseinheiten importiert werden. Dies geschieht beim erstmaligen Hinzufügen einer neuen Workstation.
Abgrenzung	Die Bezeichnung ist auf das Projekt bezogen und Aussagen gelten nur für die von der MISD-Software abgedeckten Active Directories.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Querverweise	Kachel, Organisationseinheit
Seiten	38

Begriff	**Aktualisierungsintervall**
Bedeutung	Das Aktualisierungsintervall wird für verschiedene Bereiche der MISD-Software festegelegt und konfiguriert. Für die Benutzerschnittstelle wird die Aktualisierungsrate der angezeigten Daten definiert. In Bezug auf die Dienste, wird ein Intervall zur Aktualisierung der Plugins und Einstellungen festegelegt. Bei den einzelnen Plugins wird außerdem ein Aktualisierungsintervall für das Erfassen der Kenngrößenwerte hinterlegt.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Querverweise	Kenngrößenwerte, Plugin, Benutzerschnittstelle, Dienst
Seiten	14, 15, 40

Begriff	**Benutzer**
Bedeutung	Ein Benutzer ist eine reale Person, die auf einem Client oder der Powerwall die MISD-Software zur Überwachung des Systemes nutzt.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Bezeichnung	Der Benutzer wird in der Datenbank über seinen Windows-Loginnamen identifiziert.
Querverweise	MISD, Workstation, Cluster, Client, Powerwall
Seiten	14, 15, 17, 18, 39

Begriff	**Client**
Bedeutung	Der Client ist ein Desktop oder eine Powerwall, welcher auf den Webservice der MISD-Software zugreift und die graphische Aufbereitung der Informationen anzeigt.
Abgrenzung	Ein Client kann gleichzeitig eine Workstation sein, muss das aber nicht.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Bezeichnung	Ein Client ist durch den Benutzer und den Rechner, von dem aus er benutzt wird, eindeutig gekennzeichnet.
Querverweise	Desktop, Powerwall, Server, Workstation
Seiten	6, 10, 14, 15, 17, 19–21, 28, 35

Begriff	**Cluster**
Bedeutung	Als Cluster wird eine Gruppe von zu überwachenden Rechnern bezeichnet, die entweder mit Hilfe eines Cluster-Managers wie dem HPC Cluster Manager oder dem Bright Cluster Manager betrieben wird.
Abgrenzung	Eine Workstation, die nicht über eine Cluster-Managment-Lösung überwacht wird, gehört nicht zu einem Cluster.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Bezeichnung	Jedes Cluster ist mit seinen Elementen in der Datenbank eindeutig benannt und gespeichert.
Querverweise	Workstation, Client, zu überwachender Rechner
Seiten	14, 16, 50

Begriff	**Desktop**
Bedeutung	Der Desktop ist ein Desktop-PC, welcher auf den Webservice der MISD-Software zugreift und die graphische Aufbereitung der Informationen anzeigt. Der Überbegriff ist Client.
Abgrenzung	Ein Desktop ist keine Powerwall. Ein Desktop kann gleichzeitig eine Workstation sein, muss das aber nicht.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Bezeichnung	Ein Desktop ist durch den Benutzer und den Rechner, von dem aus er benutzt wird, eindeutig gekennzeichnet.
Querverweise	Client, Powerwall, Server, Workstation
Seiten	6, 15

Begriff	**Dienst**
Bedeutung	Ein Dienst ist der Teil der Software, welcher auf den Windows- und Linux-Workstations läuft und dort mit Hilfe von Plugins Daten erhebt.
Abgrenzung	Die Software, die auf den Clients läuft, um die Oberfläche anzuzeigen, ist kein Dienst.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD-Software.
Bezeichnung	Ein Dienst wird über seinen Anwendungsbereich definiert. Je nach Systemumgebung, wird ein anderer Dienst genutzt.
Querverweise	zu überwachender Rechner, Workstation, Plugin
Seiten	6, 14, 40

Begriff	**Filter**
Bedeutung	Ein Filter stellt eine Funktionalität dar, die es ermöglicht, gewisse Kenngrößenwerte zu ignorieren und diese nicht zu übertragen und in der Datenbank abzuspeichern.
Abgrenzung	Ein Filter kann nicht auf ganze Kenngrößen oder Plugins angewandt werden. Nur auf die zu übertragenden Kenngrößenwerte einer Workstation.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Ein Filter wird eindeutig über die Kenngröße eines Plugins und eine Workstation identifiziert.
Querverweise	Kenngröße, Workstation, Kenngrößenwerte, Plugin
Seiten	21

Begriff	**Filterbedingung**
Bedeutung	Eine Filterbedingung legt fest, welche Werte einer Kenngröße von einem einem zu überwachenden Rechner gesendet werden und welche nicht. Beispielsweise könnte eine Filterbedingung der CPU-Auslastung auf Workstation 3 bei 80 liegen, was bedeutet, dass nur Kenngrößenwerte über 80 Prozent Auslastung überhaupt an den Server geschickt werden. Filterbedingungen können auch mit boolschen Operatoren verbunden werden.
Abgrenzung	Eine Filterbedingung muss nicht mit der Abbildung der Metrik zusammenhängen.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Eine Filterbedingung ist für jeden Filter eindeutig definiert.
Querverweise	Filter, Kenngröße, zu überwachende Rechner
Seiten	14, 15

Begriff	**Ignore-Liste**
Bedeutung	Die Ignore-Liste ist der Ersatz für das Löschen einer Workstation im MISD-System. Eine Workstation, die zwar gelöscht wurde, aber dennoch weiterhin den Dienst der MISD-Software installiert hat, wird weiterhin Daten senden. Diese werden nicht weiter verarbeitet sondern ignoriert. Eine solche Workstation kommt also auf die Ignore-Liste und kann von dort auch wieder hergestellt werden.
Abgrenzung	Die Ignore-Liste enthält nicht die zu überwachenden Rechner, welche im Wartungszustand sind.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Workstation, zu überwachende Rechner, Wartungszustand
Seiten	4, 15, 21, 38, 39

Begriff	**Kachel**
Bedeutung	Eine Kachel bezeichnet die grafische Repräsentation eines einzelnen zu überwachenden Rechners. Diese stellt das System in einem rechteckigen Rahmen in verschiedenen Detailstufen dar.
Abgrenzung	Eine Kachel repräsentiert nur einen Rechner, entspricht diesem jedoch nicht. Interaktion mit einer Kachel hat keine Einfluss auf den repräsentierten Rechner.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Cluster, Workstation, Visualisierungsplugin
Seiten	18

Begriff	**Kenngröße**
Bedeutung	Die Kenngröße eines Plugins ist eine Wertekategorie, die mit Hilfe des Plugins ermittelt werden. Eine Kenngröße als ein Name, eine Zahl (die beispielsweise die aktuelle Auslastung darstellt) oder ein komplexeres Objekt (wie ein ganzes Ereignis) gespeichert werden. Die erfassten Werte einer Kenngröße werden Kenngrößenwerte genannt.
Abgrenzung	Eine Kenngröße ist eine Bezeichnung für einen Teil der Daten, die mit Hilfe des Plugins erhoben werden, bezeichnet jedoch nicht die Daten selbst.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Die Kenngrößen sind in den Daten des Plugins eindeutig bestimmt. Auch in der Datenbank haben die Kenngrößen eindeutige Speicherbereiche für ihre erfassten Kenngrößenwerte.
Querverweise	Kenngrößenwerte
Seiten	26, 32, 34, 38, 39

Begriff	**Kenngrößenwert**
Bedeutung	Der Kenngrößenwert ist ein einzelner Messwert einer Kenngröße auf einer Workstation. Dieser wird als String in der Datenbank gespeichert. Auf den Wert werden vor der Speicherung Filter und die entsprechende Metrik zu Abbildung angewendet.
Abgrenzung	Der gemessene Wert ist nicht die Kenngröße sondern nur ein Eintrag in deren Werteverlauf.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Ein Kenngrößenwert ist eindeutig bestimmbar, durch eine Kenngröße, eine Workstation und einen Zeitpunkt.
Querverweise	Kenngröße, Workstation, Plugin, Filter, Metrik
Seiten	2, 4, 14, 15, 20, 21, 24, 26, 27, 32, 34, 39

Begriff	**KRITISCH**
Bedeutung	Der Zustand KRITISCH ist der höchste Zustand einer Kenngröße. KRITISCH bedeutet, dass der Wert sehr schlecht ist und evtl. das zugehörige System gefährdet. Der Zustand wird durch die entsprechende Metrik der Kenngröße eindeutig bestimmt.
Abgrenzung	Der Zustand KRITISCH ist scharf abzugrenzen gegenüber den Zuständen OK und WARNUNG.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Metrik, WARNUNG, OK
Seiten	24, 34, 38, 39

Begriff	**Layout**
Bedeutung	Ein Layout ist die Anordnung der Kacheln und Organisationseinheiten auf der Oberfläche sowie der Level der einzelnen Kacheln. Derarbtige Layouts können gespeichert und wieder geladen werden. Dadurch können Nutzungsmuster wiederholt angewendet werden, ohne erneut konfigurieren zu müssen.
Abgrenzung	Zum Layout gehören nicht die abgebildeten Farben der Kacheln und auch nicht die dargestellten Werte.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Kachel, Organisationseinheiten
Seiten	15, 17, 18, 31

Begriff	**Metrik**
Bedeutung	Eine Metrik ist eine für jede Kenngröße definierte Eigenschaft, welche den Wertebereich der Kenngröße in die Kategorien “KRITISCH“, “WARNUNG“ und “OK“ einordnet. Zusätzlich besitzt jede Kenngröße ein Aktualisierungsintervall, welches festlegt, wie oft neue Werte für die entsprechende Kenngröße in Erfahrung gebracht werden sollen.
Gültigkeit	Eine Metrik ist nur während ihres Einsatzes zur Abbildung gültig, Sobald eine Metrik geändert wird, ist die alte Metrik ungültig, alte Werte bleiben jedoch auf dem selben Zustand abgebildet.
Bezeichnung	Eine Metrik ist durch die zugehörige Kenngröße eines Plugins eindeutig identifiziert.
Querverweise	Kenngröße, Kenngrößenwerte, Plugin, Kritisch, Warnung, OK
Seiten	15, 24, 32

Begriff	**MISD-OWL**
Bedeutung	Master Infrastructure Situation Display-Observing Windows and Linux
Abgrenzung	MISD-OWL bezeichnet in diesem Dokument ausschließlich die im Studienprojekt 2012 der Universität Stuttgart entstandene Software.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Der Name MISD-OWL ist im Studienprojekt festgelegt.
Querverweise	System
Seiten	6, 13, 14, 17, 38, 67

Begriff	**OK**
Bedeutung	Der Zustand OK ist der niedrigste Zustand einer Kenngröße. OK bedeutet, dass der Wert der Kenngröße in Ordnung ist. Der Zustand wird durch die entsprechende Metrik der Kenngröße eindeutig bestimmt.
Abgrenzung	Der Zustand OK ist scharf abzugrenzen gegenüber den Zuständen WARNUNG und KRITISCH.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Metrik, WARNUNG, KRITISCH
Seiten	24, 34, 38, 39

Begriff	**Organisationseinheit**
Bedeutung	Eine Organisationseinheit ist eine Menge von Workstations, die einen technischen, lokalen, oder ähnlichen Zusammenhang haben. Jede Workstation kann Mitglied von genau einer Organisationseinheit sein. Die Organisationseinheiten sind entlang einer Hierarchie ineinander geschachtelt.
Abgrenzung	Eine Organisationseinheit muss nicht der Hierarchie des Active Directory entsprechen. Das Active Directory wird nur zur Initialisierung der zu überwachenden Rechner verwendet.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Jede Organisationseinheit muss über einen Namen oder eine ID eindeutig identifizierbar sein.
Querverweise	Workstation, Active Directory
Seiten	9, 15, 18, 30, 50

Begriff	**Plugin**
Bedeutung	Ein Plugin realisiert die Überwachungsfunktionalität für einen speziellen Bereich. Es kann aus den verschiedenen Datenerfassungsmodulen und einem Visualisierungsmodul bestehen. Zusätzlich wird dem Plugin ein Datensatz zugeordnet. Für das CPU-Plugin wären das alle Kenngrößen, Metriken und Standardwerte.
Abgrenzung	Ein Plugin besteht aus Kenngrößen, kann in Einzelfällen aber auch nur eine Kenngröße enthalten.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Jedes Plugin ist in der Datenbank eindeutig benannt.
Seiten	4, 6, 14, 15, 17, 19–21, 24, 28, 32, 37, 55

Begriff	**Powerwall**
Bedeutung	Eine Powerwall bezeichnet ein besonders hochauflösendes Display, auf welchem aufgrund der sehr hohen Auflösung sehr viele Informationen gleichzeitig angezeigt werden können. Der Überbegriff ist Client.
Abgrenzung	Im Falle von MISD wird die Anwendung auf die gegebenen Powerwalls angepasst. Eine Powerwall ist kein Desktop.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Client
Seiten	6, 15

Begriff	**Server**
Bedeutung	Der Server ist das zentrale Element der MISD Software. Auf ihm laufen die Webservices, werden die Daten der Workstations und Cluster gesammelt und in einer Datenbank gespeichert. Der Server stellt außerdem Daten für die Benutzerschnittstelle zur Verfügung.
Abgrenzung	Der Server bezieht sich immer auf den Server der MISD Software.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Der Server wird bei der Kommunikation mit den Workstations eindeutig erkennbar und zertifiziert sein.
Querverweise	*MISD OWL*
Seiten	4, 6, 8, 9, 14–26, 28, 33, 37, 40

Begriff	**Status**
Bedeutung	Jeder Kenngrößenwert wird durch seine Metrik auf die Status OK, WARNUNG und KRITISCH abgebildet.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Kenngrößenwert, Metrik, OK, WARNUNG, KRITISCH
Seiten	38, 39

Begriff	**System**
Bedeutung	Das Wort “System“ ist gleichzusetzen mit “MISD-OWL“ und wird als Synonym verwendet.
Querverweise	MISD
Seiten	8, 9, 14–16, 28, 30, 31

Begriff	**Systemeinstellung**
Bedeutung	Systemeinstellungen gelten für das gesamte System. Sie betreffen alle konfigurierbaren Einstellungen sowie die Einstellungen des Servers und der zu überwachenden Rechner.
Abgrenzung	Mit den Systemeinstellungen sind nicht die individuellen Benutzereinstellungen eingeschlossen, die pro Client definiert werden können.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Benutzereinstellung, Server, zu überwachender Rechner
Seiten	28

Begriff	**WARNUNG**
Bedeutung	Der Zustand WARNUNG ist der mittlere Zustand einer Kenngröße. WARNUNG bedeutet, dass der Wert der Kenngröße nicht in Ordnung ist, aber noch nicht systemkritisch ist. Der Zustand wird durch die entsprechende Metrik der Kenngröße eindeutig bestimmt.
Abgrenzung	Der Zustand WARNUNG ist scharf abzugrenzen gegenüber den Zuständen OK und KRITISCH.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Metrik, OK, KRITISCH
Seiten	24, 34, 38, 39

Begriff	**Wartungszustand**
Bedeutung	Der Wartungszustand eines zu überwachenden Rechners ist dazu gedacht, aus bekannten Gründen erreichte kritische Werte optisch zu deaktivieren. So können ungewollte Fehlermeldungen verhindert werden. Der Rechner kann jederzeit wieder aktiviert werden. Die Daten des Wartungszeit werden vom Server verworfen und graphisch abgegrenzt dargestellt.
Abgrenzung	Der Wartungszustand ist nicht die Ignore-Liste.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Querverweise	Ignore-Liste, zu überwachender Rechner
Seiten	4, 15, 21, 22, 38, 39

Begriff	**Workstation**
Bedeutung	Eine Workstation ist ein Rechner der von einem Dienst der MISD-Software überwacht wird.
Abgrenzung	Workstations müssen nicht gleichzeitig Clients sein, können dies jedoch. Außerdem werden die Cluster nicht als Workstations bezeichnet.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Jede Workstation ist im System über den Fully Qualified Domain Name (FQDN) gespeichert
Querverweise	Cluster
Seiten	4–6, 9, 14, 16, 17, 21–26, 38–40, 51, 53

Begriff	**zu überwachender Rechner**
Bedeutung	Die zu überwachenden Rechner sind die Gesamtmenge der über das System erfassten Geräte. Dazu zählen Workstations und Cluster-Nodes.
Abgrenzung	Ein zu überwachernder PC muss kein Client sein, kann dies jedoch. Ein PC, der nicht durch einen Dienst vom System überwacht wird, fällt nicht in diese Kategorie.
Gültigkeit	Bei der Entwicklung, Nutzung und Erweiterung der MISD Software.
Bezeichnung	Alle zu überwachenden PCs sind in der Datenbank eindeutig benannt und gespeichert.
Querverweise	Cluster, Workstation
Seiten	14, 15, 30, 32

A.2 Versionshistorie

Version 0.1

Datum	10.07.2012
Änderungen	Initiale Version
Bearbeiter	Jonas Scheurich

Version 0.2

Datum	14.07.2012
Änderungen	Kapitel Komponentenentwurf
Bearbeiter	Yannic Noller

Version 0.3

Datum	15.07.2012
Änderungen	Datenbankkapitel angelegt
Bearbeiter	Fabian Müller

Version 0.4

Datum	17.07.2012
Änderungen	Statusdiagramm
Bearbeiter	Jonas Scheurich

Version 0.5

Datum	19.07.2012
Änderungen	Aktivitätendiagramme Server/Client fertig gestellt
Bearbeiter	Jonas Scheurich

Version 0.6

Datum	19.07.2012
Änderungen	Datenbankkapitel ergänzt
Bearbeiter	Fabian Müller

Version 0.7

Datum	20.07.2012
Änderungen	Aktivitätendiagramme Wartungszustand, Ignore List hitzugefügt
Bearbeiter	Jonas Scheurich

Version 0.8

Datum	21.07.2012
Änderungen	Kapitel über Entwurfsmuster hinzugefügt
Bearbeiter	Yannic Noller

Version 0.9

Datum	21.07.2012
Änderungen	Kapitel Datenhaltung vervollständigt
Bearbeiter	Fabian Müller

Version 0.10

Datum	22.07.2012
Änderungen	Entwurfskorrektur
Bearbeiter	Paul Brombosch

Version 1.0

Datum	24.07.2012
Änderungen	Überarbeitung nach Review
Bearbeiter	Paul Brombosch und David Krauss

Version 1.1

Datum	26.07.2012
Änderungen	Schnittstellen
Bearbeiter	alle

Version 1.2

Datum	05.08.2012
Änderungen	Korrektur Kapitel 2
Bearbeiter	Yannic Noller

Version 1.3

Datum	01.10.2012
Änderungen	Überarbeitung des gesamten Dokuments
Bearbeiter	Fabian Müller, Arno Schneider

Version 1.4

Datum	24.11.2012
Änderungen	Kapitel Architekturmodell an die Entwicklung angepasst: Verwendung der Pattern hinz
Bearbeiter	Jonas Scheurich

Version 1.5

Datum	24.11.2012
Änderungen	Kapitel Komponenten an die Entwicklung angepasst: Weitere Komponenten des Servers
Bearbeiter	Jonas Scheurich

Version 1.6

Datum	25.11.2012
Änderungen	Kapitel Interaktion an die Entwicklung angepasst: Diagramme angepasst.
Bearbeiter	Jonas Scheurich

Version 1.7

Datum	26.11.2012
Änderungen	Komponentendiagramm aktualisiert.
Bearbeiter	Jonas Scheurich

Version 1.8

Datum	27.11.2012
Änderungen	Feinentwürfe erstellt
Bearbeiter	Jonas Scheurich, Hanna Schäfer

Version 1.9

Datum	3.12.2012
Änderungen	Klassendiagramme Feinentwürfe eingefügt
Bearbeiter	Jonas Scheurich

Version 1.10

Datum	21.01.2013
Änderungen	Ergänzung WorkstationManager und CleanerTimerJob
Bearbeiter	Jonas Scheurich

Version 1.11

Datum	27.01.2013
Änderungen	Aktualisierung Client Webservice
Bearbeiter	Jonas Scheurich

Version 2.0

Datum	22.03.2013
Änderungen	Aktualisierung Feinentwürfe
Bearbeiter	Jonas Scheurich