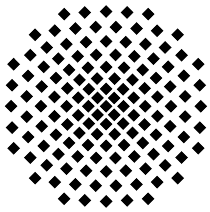

Universität Stuttgart
Institut für Theorie der Elektrotechnik
Prof. Dr. techn. Wolfgang M. Rucker

Bachelorarbeit

# Entwicklung eines Softwareagenten zur Darstellung der Prozessparameter numerischer Simulationen

## Development of an Software Agent for Presenting Numerical Simulation Process Parameters

Fabian Diener

| | |
|---|---|
| Betreuer: | Dipl.-Ing. Matthias Jüttner |
| Beginn der Arbeit: | 01.04.2014 |
| Abgabe der Ausarbeitung: | 13.11.2014 |

# Erklärung

Hiermit erkläre ich,

- dass ich die vorliegende Arbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens ist,
- dass ich die Arbeit noch nicht veröffentlicht habe,
- dass das elektronische Exemplar mit diesem Exemplar übereinstimmt.

Stuttgart, den 13.11.2014

# Abkürzungsverzeichnis

| | |
|---|---|
| **ACL** | Agent Communication Language (siehe [2]) |
| **CA** | Berechnungsagent (CalculationAgent) |
| **CPU** | Central Processing Unit |
| **FIPA** | Foundation for Intelligent Physical Agents |
| **GPL** | General Public License |
| **GUI** | Graphical User Interface – Benutzeroberfläche |
| **ID** | IDentifikator |
| **ITE** | Institut für Theorie der Elektrotechnik |
| **JADE** | Java Agent DEvelopment framework |
| **LGPL** | Lesser General Public License |
| **PC** | Personal Computer |
| **PDF** | Portable Document Format |
| **RA** | Darstellungsagent (ReportAgent) |
| **RAM** | Random Access Memory – Arbeitsspeicher |
| **SIGAR** | System Information Gatherer And Reporter |
| **SMS** | Short Message Service |

# Zusammenfassung

In der folgenden Arbeit wird ein Programm zur Erfassung und Darstellung von Prozessparametern einer Rechner-Plattform zur Lösung numerischer Simulationen vorgestellt. Dieses soll möglichst einfach in der Bedienung sein, einen guten Überblick über die gesamte Plattform bieten und die erfassten Prozessparameter in geeigneter Weise darstellen.

Im Folgenden wird ein möglicher Lösungsansatz für diese Anforderungen beschrieben. Da das hier entwickelte Programm die Schnittstelle zwischen Benutzer und Rechner darstellt, wird auf eine gute Bedienbarkeit der Benutzeroberfläche besonderen Wert gelegt. Es gelingt, die Prozessparameter intuitiv interpretierbar zu präsentieren. Die Entwicklung des Programms geschieht nach dem Wasserfallmodell[1].

[1] lineares Vorgehensmodell in der Softwaretechnik (siehe [3])

# Inhaltsverzeichnis

# 1. Einleitung

Die Entwicklung technischer Geräte aller Art, beispielsweise Autos oder Mobiltelefone, geschieht heute mithilfe von Simulationen. Da zur Perfektionierung dieser Geräte möglichst präzise Informationen benötigt werden, steigt die Komplexität der dazugehörigen Simulationsmodelle. Hierdurch steigt auch der Rechenaufwand. Heutige Rechenleistung wird durch die Vernetzung einzelner Rechner zu sogenannten *Clustern*[1] generiert. Hierbei löst ein einzelner Rechner ein Teilproblem eines komplexen Gesamtsystems. Um in diesem Cluster die Kontrolle zu behalten, sind dedizierte Programme erforderlich. Ein solches Programm zur Überwachung und Kontrolle eines Rechner-Clusters in Form einer Agenten-Plattform für numerische Simulationen wird in dieser Arbeit vorgestellt. Zunächst werden die wesentlichen technischen Grundlagen erläutert, die für das Verständnis dieser Arbeit erforderlich sind. Da dies nur ein grober Überblick sein kann, wird für ein tieferes Verständnis eine präzisere Einarbeitung anhand der verwiesenen Referenzen empfohlen.

## 1.1. Software-Agenten

Bei einem (Software-)Agent handelt es sich um ein autonomes Programm, welches für eine bestimmte Aufgabe, die an die Software gestellt wird, benötigt wird. Der Zusammenhalt der einzelnen Agenten zur bedienbaren Software geschieht hierbei durch ein Nachrichtensystem (siehe dazu Kapitel 1.4.1). Hierdurch wird eine größere *Kapselung*[2] der einzelnen Teilbereiche der Software geschaffen, als es durch objektorientierte Programmierung möglich wäre. Dies erhöht wiederum das Abstraktionsniveau und wird daher auch als die logische Fortsetzung der bisherigen Programmierentwicklung gesehen[3].

[1] siehe [1]
[2] Abschottung von Softwareteilen in der objektorientierten Programmierung
[3] siehe [3]

## 1.2. Java

Java ist eine objektorientierte Programmiersprache. Sie wird für die Realisierung des Programms benötigt. Eine gute Einführung ist in [5] zu finden.

## 1.3. JADE

Das Java Agent DEvelopment framework[4] (JADE) implementiert ein Agentensystem, welches die Spezifikationen der Foundation for Intelligent Physical Agents (FIPA) als Java-Plattform zur Verfügung stellt. Durch die Einhaltung des Standards wird dabei eine Kompatibilität mit anderen Agenten gewährleistet, welche nicht unbedingt in Java implementiert sein müssen. Mit JADE können einzelne Agenten beliebig angehalten und fortgesetzt werden, sogar die Portierung auf andere Rechner zur Laufzeit wird gestattet. Letzteres erlaubt eine bessere Ressourcennutzung durch eine bessere Verteilbarkeit des Rechenaufwandes.

## 1.4. Struktur der existierenden Plattform

Das ITE betreibt eine Agenten-Plattform zur Berechnung elektrotechnischer Probleme. Der Agent zur Darstellung der Prozessparameter numerischer Simulationen ist in diese, bereits bestehende Plattform eingebettet. Diese ist, wie in Abbildung 1.1 dargestellt, aufgebaut.

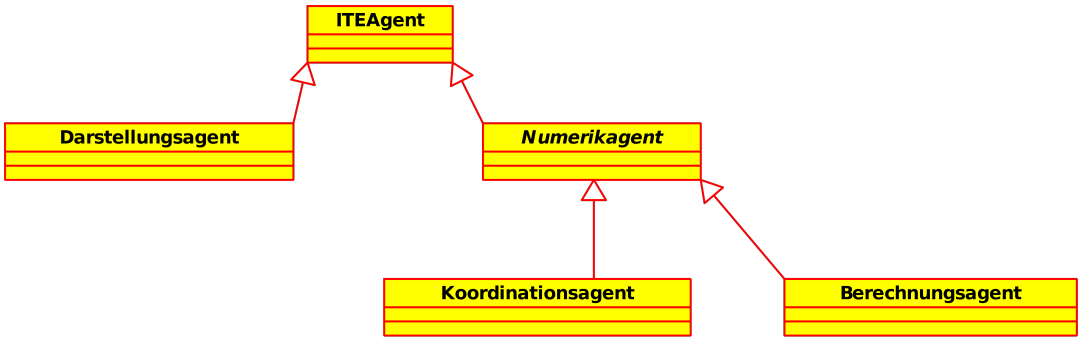


Abbildung 1.1.: Überblick über das bisherige Agenten-System, *nach Vorlage von Dipl.-Ing. Matthias Jüttner*

Der ITE-Agent ist die Basis aller weiteren Agenten. Der Numerikagent, der den Berechnungszweig beginnt und sich klar von dem Darstellungsagenten abgrenzt, erbt vom ITE-Agent und ist wiederum Grundlage für Berechnungsagent und

[4] siehe [2]

Koordinationsagent. Der Darstellungsagent erbt ebenfalls vom ITE-Agent und übernimmt die Darstellung der Prozessparameter. Der Berechnungsagent löst ein Teilproblem des Modells, während der Koordinationsagent die verschiedenen Berechnungsagenten koordiniert. Es ist im Allgemeinen davon auszugehen, dass eine Instanz[5] des Koordinationsagenten und Darstellungsagenten, jedoch mehrere Instanzen des Berechnungsagenten existieren. Alle Agenten nutzen zur Darstellung denselben Darstellungsagenten.

Da der Darstellungsagent allgemein eingesetzt werden kann, sei an dieser Stelle erwähnt, dass dieser vom ITE-Agent im Wesentlichen die Funktion erbt, andere Agenten in der Agentenplattform zu suchen. Soll der Darstellungsagent auf eine andere Plattform übertragen werden, so muss ihm diese Fähigkeit verliehen werden.

### 1.4.1. Nachrichtensystem

Die Kommunikation unter den Agenten geschieht über Nachrichten, sogenannte *ACLMessages* (Agent Communication Language Messages). Dieses in JADE implementierte Nachrichtensystem transportiert Textnachrichten in Form von *Strings* oder ganze Objekte[6]. Dabei trägt jede Nachricht eine *Conversation-ID*. Diese ist als *String* realisiert. Sie wird in der Kommunikation mit dem Darstellungsagenten durch das Protokoll (siehe Kapitel A.3) definiert und verwendet, um dem Darstellungsagenten die Bedeutung der Nachricht mitzuteilen, z. B. ob ein neuer Prozessparameter registriert werden soll.

[5] Im Allgemeinen ein Objekt, durch den agentenorientierten Ansatz jedoch eher als Teilprogramm zu verstehen

[6] Der Transport von Objekten ist nicht FIPA-konform und sollte daher vermieden werden. [2]

# 2. Prozessparameter

Prozessparameter, die in numerischen Simulationen zur Auswertung zur Verfügung stehen, existieren reichlich. Um die Übersichtlichkeit zu gewährleisten, ist eine Beschränkung auf die *wesentlichen* Prozessparameter ratsam. Dabei hängt der Begriff ‚wesentlich' jedoch vom jeweiligen Umfeld ab.
Die Prozessparameter, die hier an dieser Stelle erfasst werden, lauten:

- **Fortschritt**
  Der Berechnungsfortschritt des jeweiligen Teilproblems
- **PC-Status-Monitor**
  Auslastung von Prozessor (CPU) und Arbeitsspeicher (RAM) des jeweiligen Rechners, der den Berechnungsagenten trägt
- **Mesh[1]-Daten**
  Informationen über die Meshes des Modells
- **Konvergenzdaten**
  Daten, die den Iterationsfehler in Abhängigkeit des Interationsschritts beinhalten. Diese Daten sind damit ein Maß für die Konvergenz des iterativen Lösungsverfahrens.

[1] Polygonnetz zur Diskretisierung mehrdimensionaler Modelle

# 3. Analyse

Der Agenten zur Darstellung von Prozessparameter numerischer Simulationen ist ein Agent, der ausschließlich für die Interaktion mit dem Benutzer vorgesehen ist. Daher hat er vor allem nachfolgende Aufgaben zu bewältigen

A1: Einfache, möglichst *intuitive Bedienbarkeit*

A2: Darstellung der Prozessparameter in adäquater Weise. Diese hat nach Möglichkeit aufgrund der einfacheren Erfassbarkeit in Form von Diagrammen zu erfolgen.

A3: *Robustheit* gegenüber Programmierfehlern. Ein Abstürzen, Einfrieren oder sonstiges von der Norm abweichendes Verhalten ist für den Darstellungsagenten nicht tolerierbar.

Der Agent beschränkt sich also auf die Darstellung der Prozessparameter. Eine Auswertung dieser in Kombination mit einem autonomen Verhalten stellen interessante Erweiterungsmöglichkeiten dar und werden in Kapitel 7 näher beleuchtet.

## 3.1. Benutzeroberfläche

Bei der Entwicklung der Benutzeroberfläche (GUI) wird nach oben erwähnten Anforderungen vor allem auf folgende Punkte Wert gelegt:

- *Übersichtlichkeit*
- *Einfache Bedienbarkeit*
- *Einfache Vergleichbarkeit* der Daten
- *Dynamik*: Die GUI muss in der Lage sein, neue Daten zur Laufzeit und während des Berechnungsprozesses anzuzeigen bzw. zu aktualisieren. Ist sie dies nicht, so erweckt sie schnell den Eindruck, die gesamte Agentenplattform wäre untätig.

# 4. Entwurf des Agenten

In diesem Kapitel wird der Entwurf des Agenten vorgestellt. Kapitel 5 beschreibt die Implementierung des Agenten und setzt die in diesem Kapitel gewonnenen Erkenntnisse um.

## 4.1. Einsatz von Komponenten

Beim Entwurf des Agenten, spätestens jedoch bei dessen Implementierung, muss entschieden, ob Software-Komponenten eingesetzt werden oder nicht. An dieser Stelle fällt die Entscheidung für die Software-Komponenten, da diese die Software-Qualität verbessern und den Implementierungsaufwand verringern[1].

## 4.2. Benutzeroberfläche

Der Entwurf der GUI wird in diesem Kapitel vorgestellt. Die Anforderungen hierzu sind in Kapitel 3.1 aufgeführt. Diese legen hierbei eine hierarchische Struktur nach Abbildung 4.1 nahe.

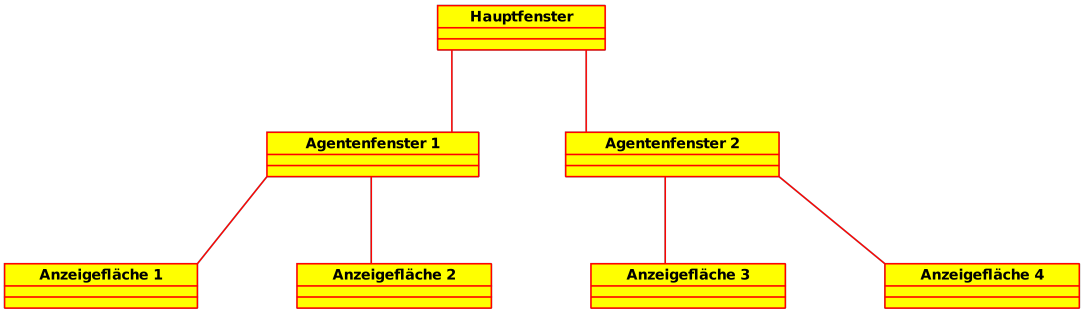


Abbildung 4.1.: Hierarchische Struktur der Benutzeroberfläche

Die *Vergleichbarkeit* fordert, dass mehrere Graphen über-, in- oder zumindest nebeneinander dargestellt werden können. Die *Übersichtlichkeit* wird insofern berücksichtigt, als jedem Agenten ein eigenes Teilfenster zugewiesen wird. Dieser Ansatz trennt die einzelnen Agenten klar voneinander ab und lässt sofort

[1] siehe [3], [4]

Rückschlüsse zu, welcher Prozessparameter zu welchem Agenten gehört. Durch separierte Fenster wird auch die *Vergleichbarkeit* von Prozessparameter zwischen verschiedenen Agenten gewährleistet. Die *Bedienbarkeit* und *Dynamik* werden hier nicht durch einen bestimmten Ansatz verfolgt, werden jedoch bei der Implementierung in Kapitel 5 beachtet. Die Zuweisung der jeweiligen Diagramme zu den Agentenfenstern ist dabei notwendig und wird in einfacher Form vorgenommen.

## 4.3. Datenerhebung

Die Datenerhebung, also das Ermitteln der Prozessparameter, geschieht nicht im Darstellungsagent selbst, sondern erfolgt außerhalb – in der in Kapitel 1.4 vorgestellten Plattform im Berechnungsagent. Sie ist der Vollständigkeit halber dennoch erwähnt, wobei eine Beschränkung auf die Implementierungsdetails (Kapitel 5) stattfindet.

## 4.4. Prozessparameter-Verwaltung

Verwaltet werden die Prozessparameter, die in Kapitel 2 erwähnt sind. Dabei geschieht die Verwaltung eines Prozessparameters prinzipiell wie in Abbildung 4.2 dargestellt. Zunächst wird ein Prozessparameter beim Darstellungsagenten

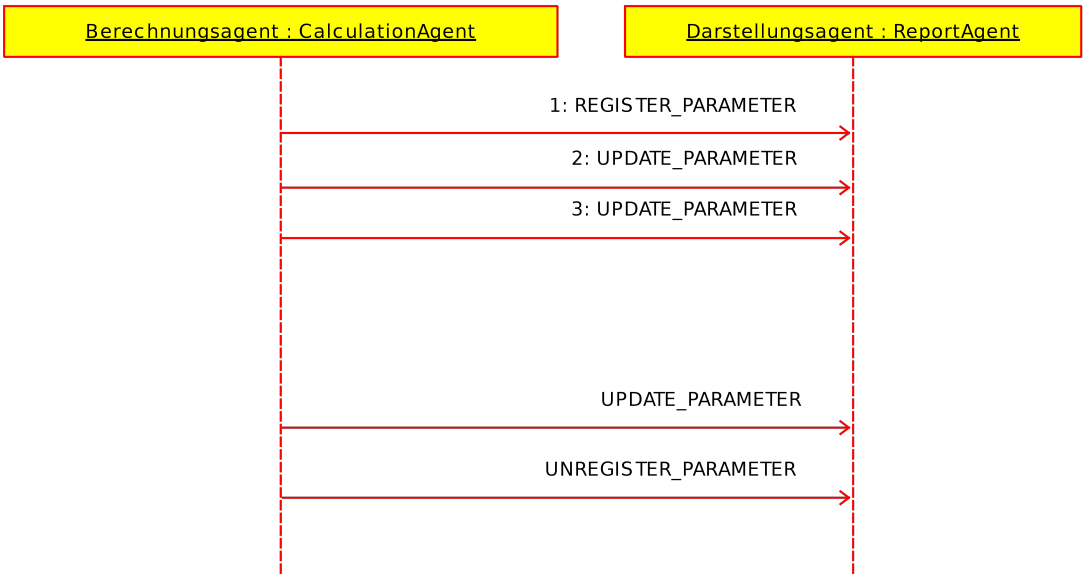


Abbildung 4.2.: Prinzipielle Prozessparameterverwaltung

*registriert*. Anschließend besteht die Möglichkeit durch *Updates* einem bestehenden Prozessparameter Daten hinzuzufügen. Wird der Prozessparameter nicht mehr benötigt, so kann er durch *Deregistrierung* wieder entfernt werden. Durch diesen Entwurfsansatz ist die einfache Erweiterbarkeit durch neue Prozessparameter gegeben. Es ist zu betonen, dass jeder Agent lediglich in der Lage ist, die Prozessparameter zu manipulieren, die er selbst registriert hat. Dies fördert die *Robustheit* des Programms, da sich eventuelle Fehler eines Berechnungsagenten nicht auf den gesamten Darstellungsagenten ausdehnen können und dürfen.

# 5. Implementierung des Agenten

Im Folgenden wird die Entwicklung des Darstellungsagenten beschrieben. Dabei werden die Erkenntnisse aus Kapitel 4 umgesetzt. Die hier vorgestellte Implementierung ist dabei nur eine Möglichkeit von vielen.

## 5.1. Eingesetzte Komponenten

Im Entwurf in Kapitel 4 wurde die Entscheidung für den Einsatz von Komponenten gefällt. Im Folgenden werden die eingesetzten Komponenten vorgestellt.

### 5.1.1. DockingFrames

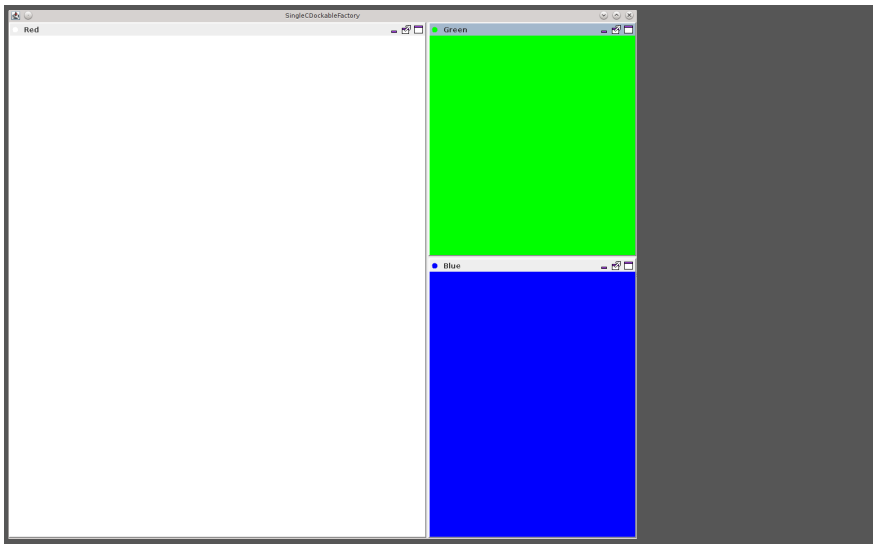


Abbildung 5.1.: DockingFrames-Beispiel

DockingFrames[1] stellt Fenster als sogenannte *Dockables* dar. Ein Dockable ist dabei ein Unterfenster, welches eine beliebige Positionierung auf dem Bildschirm gestattet. Es ist vom Hauptfenster lösbar und lässt sich jederzeit wieder in dieses

[1] Projektseite: [6]

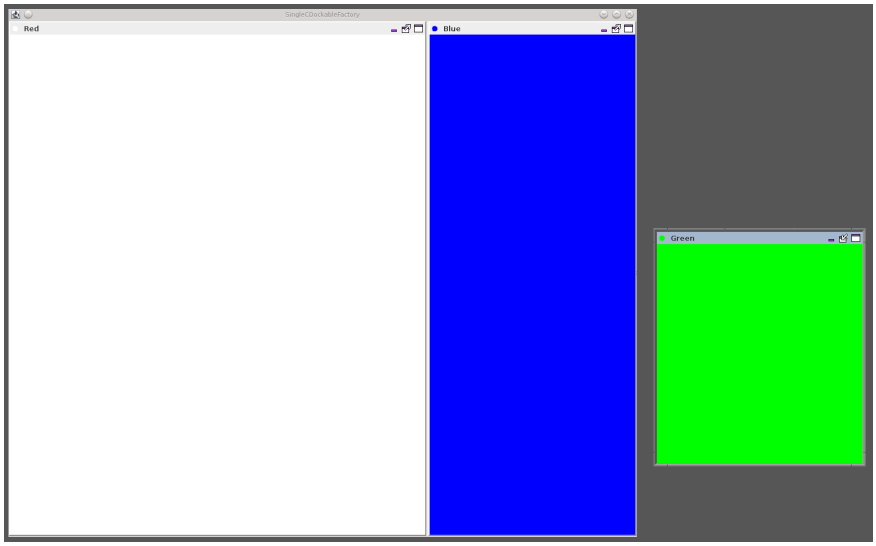


Abbildung 5.2.: DockingFrames-Beispiel mit ausgedocktem Fenster

integrieren, wie in Abbildung 5.1 und 5.2 dargestellt. Dieses Projekt erfüllt damit viele Anforderungen, die im Entwurf der GUI gefordert wurden. Das Projekt ist unter der Lesser General Public License (LGPL) veröffentlicht.

### 5.1.2. JFreeChart

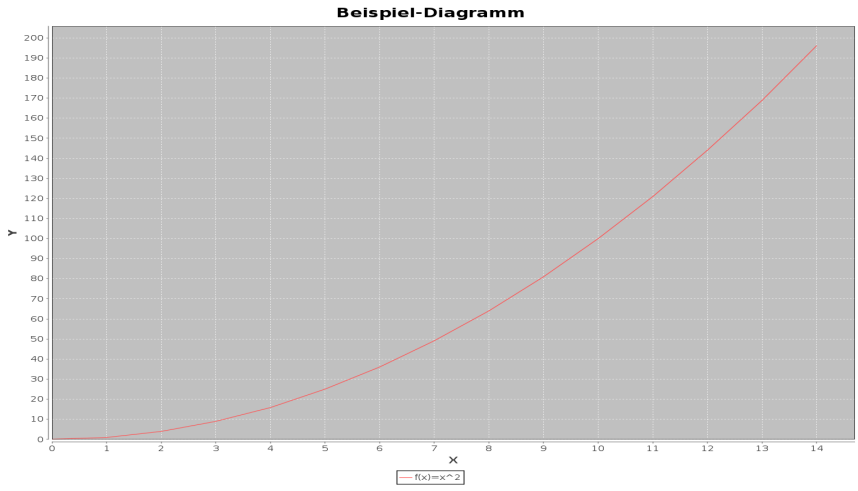


Abbildung 5.3.: Beispiel-Diagramm, mit JFreeChart gezeichnet

JFreeChart[2] ist ein Projekt, mit dem Diagramme gezeichnet werden können.

[2]Projektseite: [7]

Es werden die verschiedensten Diagrammtypen unterstützt, beispielsweise Balkendiagramme, Kreisdiagramme oder – von dem Darstellungsagenten häufig benötigt – XY-Diagramme. Ein Beispiel-Diagramm ist in Abbildung 5.3 abgebildet. JFreeChart ist ebenfalls unter der LGPL veröffentlicht.

### 5.1.3. SIGAR

Der System Information Gatherer And Reporter (SIGAR)[3] stellt eine betriebssystemunabhängige Schnittstelle für Systemressourcen-Informationen unter Java zur Verfügung. Dieses Paket wird dazu verwendet, CPU- und RAM-Auslastung anzuzeigen und findet bei dem später vorgestellten PC-Status-Monitor Verwendung. Das Projekt ist unter der Apache-Lizenz Version 2 veröffentlicht.

## 5.2. Benutzeroberfläche

Im Folgenden wird die Umsetzung der Erkenntnisse für die GUI aus Kapitel 4 vorgestellt.

### 5.2.1. Hauptfenster

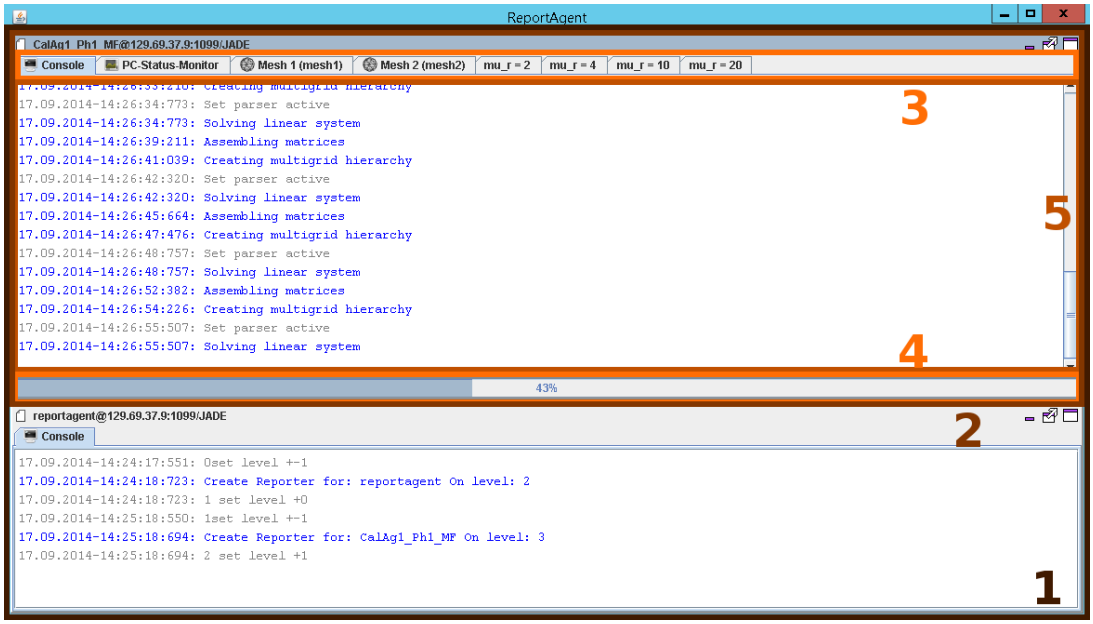


Abbildung 5.4.: GUI des Darstellungsagenten (Hauptfenster)

[3]Projektseite: [8]

Das Hauptfenster ist durch die Swing-Klasse *JFrame* realisiert und in Abbildung 5.4 abgebildet. Dieses bildet die Basis der GUI des Darstellungsagenten.

Das Hauptfenster (1) beinhaltet alle Komponenten, wobei einzelne Agentenfenster (2) auch in der Lage sind, dieses zu verlassen. Pro Agent existiert ein Agentenfenster mit mehreren Tabs (3). Diese Tabs listen einzelne Prozessparameter eines Agenten auf – in Abbildung 5.4 sind es beispielsweise der PC-Status-Monitor, zwei Meshes, sowie mehrere Konvergenzdiagramme. Der Fortschrittsbalken (4) ist nur dann dargestellt, wenn er benötigt wird; er wird ausgeblendet, sobald die Berechnung vollständig abgeschlossen ist. Die Anzeigefläche (5) stellt den momentan gewünschten Prozessparameter dar. Die Konsole ist nicht als Prozessparameter geführt und ist stets der erste Tab. Darauf wird in Kapitel 5.2.3 eingegangen.

### 5.2.2. Agentenfenster

Das Agentenfenster wird durch einen Fenster-Typ des DockingFrames-Projektes (siehe Kapitel 5.1.1) realisiert. Dieser Fenster-Typ heißt *Dockable*, da sie die Fähigkeit besitzen, sich in das Hauptfenster „ein- oder auszudocken“, also die Fähigkeit, sich flexibel zu lösen und neu anzuordnen. Dies veranschaulicht Abbildung 5.1 und 5.2 aus Kapitel 5.1.1. Hierdurch wird ein flexibler Vergleich von Prozessparametern verschiedener Agenten ermöglicht. Darüber hinaus sind die Agentenfenster in sich geschlossen, was die Wahrnehmung der Autonomie einzelner Agenten stärkt.

Das Agentenfenster ist aufgeteilt in eine Anzeigefläche, die durch Tabs verändert werden kann, sowie eine Fortschrittsanzeige am unteren Bildschirmrand. Diese Aufteilung hat den Vorteil, dass der aktuelle Fortschritt der Berechnung stets eingesehen werden kann.

### 5.2.3. Anzeigefläche

Das Agentenfenster beinhaltet eine oder mehrere Anzeigeflächen. Die einzelnen Anzeigeflächen werden in demselben Agentenfenster angezeigt. Das Umschalten erfolgt über Registerkarten, *Tabs* genannt. Die einzelnen Tabs sind – mit Ausnahme der Konsole – durch einen Rechtsklick schließbar. Das gezielte Schließen einzelner, nicht benötigter Tabs erhöht die Übersichtlichkeit der GUI. Die Möglichkeit, dies ohne Menü durch einen einfachen Rechtsklick zu erledigen, erleichtert die Bedienbarkeit.

### Konsole

Der erste Tab beinhaltet die Konsole des jeweiligen Agenten. Dies ist auch durch ein entsprechendes Icon (siehe Abbildung 5.5) dargestellt.

Abbildung 5.5.: Konsole-Icon (*Quelle:* [9])

**Bedeutung** Die Konsole nimmt eine herausragende Bedeutung ein. Sie ist unverzichtbar, da sie folgende Aufgaben erfüllt:

- Ausgabe der Informationen während der numerischen Lösung
- Anzeige von Status- und Fehlermeldungen
- Ausgabesystem im Fehlerfall der GUI

Da diese Aufgaben von hoher Bedeutung sind, ist es nicht möglich, die Konsole zu schließen. Zur besseren Übersichtlichkeit werden die einzelnen Meldungen ihrer *Wichtigkeit* nach formatiert. Die *Wichtigkeit* ist hierbei in den Text-Levels (siehe Anhang A.1) eindeutig definiert. Die Formatierung sieht dabei aus wie Abbildung in 5.6 dargestellt.

```
Fatal
Error
Warning
Information important
Information casual
Debug
Unknown
```

Abbildung 5.6.: Formatierung der verschiedenen Text-Level

### Inhalt der weiteren Tabs

Die weiteren Tabs beinhalten Prozessparameter des Computers oder des numerischen Berechnungsprogramms. Diese werden im Folgenden vorgestellt.

**PC-Status-Monitor** Der *PC-Status-Monitor* stellt die CPU- und Arbeitsspeicher-Auslastung in einem Diagramm graphisch dar. Die Daten hierzu stellt das SIGAR-Projekt zur Verfügung (siehe Kapitel 5.3.2). Er benutzt zur Darstellung dieser Daten das Projekt *JFreeChart* (siehe Kapitel 5.1.2) und ist durch das Icon aus Abbildung 5.7 klar identifizierbar.

Abbildung 5.7.: Icon des PC-Status-Monitors (*Quelle:* [10])

Ein Beispiel des Status-Monitors ist in Abbildung 5.8 dargestellt. Hier erkennt man, dass die Berechnung nach ca. 135 s beendet wurde.

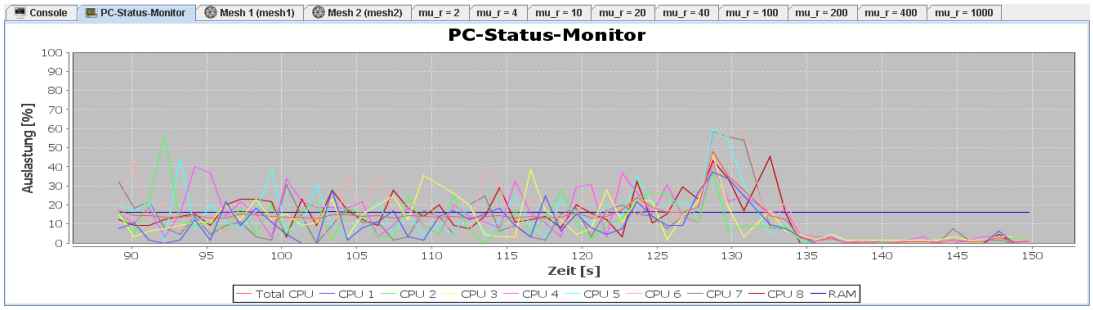


Abbildung 5.8.: PC-Status-Monitor

Es ist zu beachten, dass die Systemressourcen nicht einfach durch einen *Task-Manager* oder ein vergleichbares Tool überwacht werden können, da davon auszugehen ist, dass sich die Agenten auf verschiedenen Rechnern befinden. Der Status-Monitor stellt die Daten der Rechner, auf denen die Berechnungsagenten arbeiten, übersichtlich dar, wohingegen vergleichbare Tools lediglich den Rechner überwachen, auf dem sie selbst laufen.

Der Status-Monitor zeigt, wie bei vergleichbaren Tools üblich, die aktuellsten Messwerte an, hier in etwa die letzten 60 Sekunden. Überwacht wird in der ITE-Beispielplattform jeder Agent ausgenommen dem Darstellungsagenten selbst. Dieser ist für diese Anwendung wenig interessant. Er ließe sich gegebenenfalls durch ein externes Tool überwachen, da der Darstellungsagent stets auf dem Rechner, auf den der Benutzer zugreifen kann, läuft.

**Mesh-Daten** Da sich die Mesh-Daten nicht sinnvoll in einem Diagramm darstellen lassen, werden diese textuell ausgegeben. Dabei werden weitere nützliche Informationen wie Autor oder Bearbeitungsdatum angezeigt. Das Icon in der Tab-Leiste ist in Abbildung 5.9 dargestellt, ein Beispiel für eine Mesh-Ausgabe ist in Abbildung 5.10 zu sehen.

Abbildung 5.9.: Icon der Mesh-Statistik

Console | PC-Status-Monitor | Mesh 1 (mesh1) | Mesh 2 (mes

**Mesh-Statistik für Mesh 1 (mesh1):**

| | |
|---|---|
| Max. Growth Rate: | **3.278** |
| Max. Volume: | **5.425E-5** |
| Mean Growth Rate: | **1.416** |
| Mean Quality: | **0.801** |
| Min. Quality: | **0.2676** |
| Number of elements: | **19480** |
| Volume: | **0.219** |
| isActive: | **true** |
| isComplete: | **true** |
| isEmpty: | **false** |
| Model: | **mod1** |
| Tag: | **mesh1** |
| Name: | **Mesh 1** |
| Scope: | **root.mod1** |
| Date created: | **Sep 18, 2014 12:26:52 PM** |
| Date modified: | **Sep 18, 2014 12:26:52 PM** |
| Help: | **file:///C:/Program Files/COMSOL/COMSOL43b/doc/html/comsol/helpdesk.htm** |

Abbildung 5.10.: Mesh-Statistik

**Konvergenzdiagramme** Die Konvergenzdiagramme stellen den Iterationsfehler über die einzelnen Iterationsschritte dar (siehe Abbildung 5.11). Sie sind damit ein Indiz für den Fortschritt des Lösungsprozesses. Sie werden ebenfalls von *JFreeChart* dargestellt, im Unterschied zum PC-Status-Monitor allerdings mit logarithmischer $y$-Achse.

Der Titel eines Konvergenzdiagrammes lautet „<Solver-Typ> – Konvergenzplot“, wobei <*Solver-Typ*> die Art des Lösungsverfahrens angibt, beispielsweise „LinearSolver“[4]. Werden mehrere Parameter berechnet, so beschreibt der Titel den dargestellten Parameter – wie in Abbildung 5.11. Hierbei sind die einzelnen Iterationsschritte diskret, zur besseren Übersichtlichkeit ist es jedoch üblich, die einzelnen Punkte durch lineare Interpolation zu einer Kurve zu verbinden.

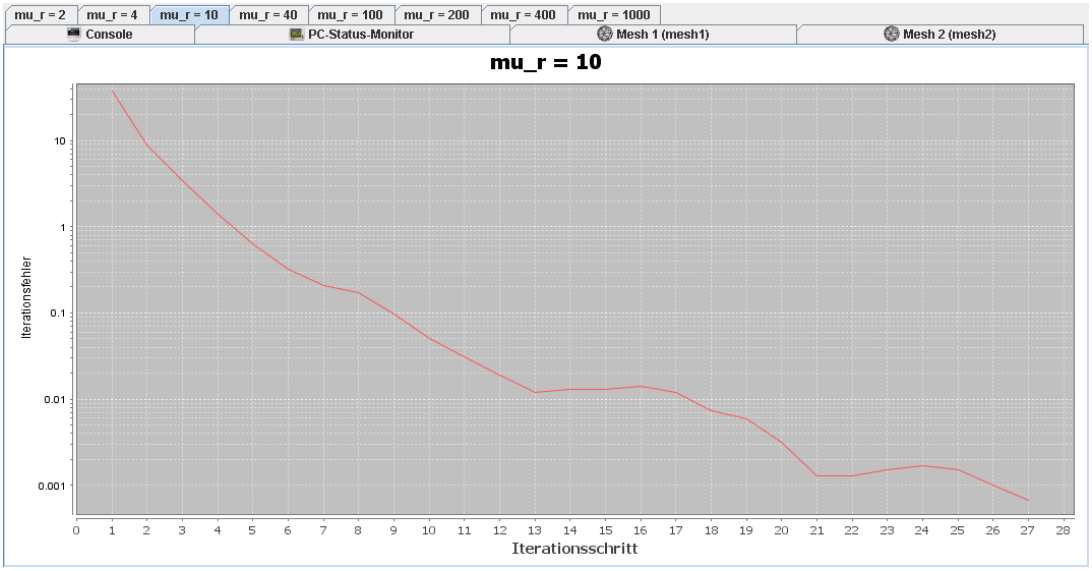


Abbildung 5.11.: Beispiel eines Konvergenzdiagrammes

## 5.3. Datenerhebung

Im Folgenden werden die Wege beschrieben, mit denen der Berechnungsagent aus der in Kapitel 1.4 vorgestellten Plattform die Prozessparameter ermittelt, die er an den Darstellungsagenten weiterleitet. Dabei werden die Prozessparameter thematisiert, welche nach Kapitel 2 erfasst wurden. Die Datenerhebung ist nicht Teil des Darstellungsagenten selbst, ferner ist sie direkt von der numerischen Berechnungsumgebung abhängig.

### 5.3.1. Fortschritt

Der Berechnungsfortschritt ist dem Berechnungsagenten selbst unbekannt, er leitet diesen vom zugrunde liegenden numerischen Berechnungsprogramm (hier Comsol) an den Darstellungsagenten weiter.

[4]Mehr dazu in Kapitel 5.3.4

### 5.3.2. PC-Status-Monitor

Die Daten über CPU- und RAM-Auslastung werden in regelmäßigen Intervallen[5] vom Berechnungsagenten mithilfe des SIGAR-Projektes abgefragt und an den Darstellungsagenten versendet. Da es ein wiederkehrendes Event ist, ist es intern als *TickerBehaviour*[6] implementiert.

Diese Daten sind dabei von dem zugrundeliegenden numerischen Berechnungsprogramm unabhängig. Durch diesen zweiten Weg der Datenbeschaffung werden die gesamten Daten besser überprüfbar. Stürzt beispielsweise das numerische Berechnungsprogramm ab, so werden die Daten des PC-Status-Monitor weiter geliefert. Mithilfe dieser Daten ist dann unter Umständen das Problem erkennbar.

### 5.3.3. Mesh-Daten

Die Mesh-Daten werden dem numerischen Berechnungsprogamm entnommen. In diesem Beispiel geschieht das in Comsol mithilfe der Klasse *com.comsol.model.Model*. Hieraus wird eine Liste alle zur Verfügung stehenden Meshes ausgelesen. Anschließend werden jedem Mesh die verfügbaren statistischen Daten entnommen und an den Darstellungsagenten gesendet. Dabei wird ein Statistik-Objekt, welches alle verfügbaren Daten enthält, versendet. Hierbei ist jedoch die FIPA-Spezifikation verletzt (vgl. Kapitel 1.4.1). Dies wird in Kauf genommen, da zum einen die Flexibilität mehrerer einsetzbarer Programmiersprachen nicht benötigt wird, zum anderen ein bereits existierendes Serialisierungssystem aus Gründen der Software-Qualität der manuellen Serialisierung vorzuziehen ist[7]. Da die Mesh-Daten nach Laden des Numerik-Modells vollständig sind, erfolgt kein *Update*[8] dieses Parameters.

### 5.3.4. Konvergenzdaten

Die Konvergenzdaten werden ebenfalls über das numerische Berechnungsprogramm ermittelt. Diese sind jedoch durch folgende Faktoren komplexer zu interpretieren:

– Die Daten hängen vom numerischen Lösungsverfahren ab. So unterscheidet sich beispielsweise im Allgemeinen die Datenstruktur zwischen linearem

[5] Voreinstellung: 1 s
[6] in konstanten Zeitintervallen ausgeführte Routine. Die Zeitüberwachung übernimmt dabei die JADE-Agentenplattform.
[7] siehe [5], [4]
[8] für genauere Informationen hierzu siehe Kapitel 5.4, insbesondere Kapitel 5.4.2

und nichtlinearem Lösungsverfahren.

– Die Daten können durch weitere Variationen, beispielsweise Parametern, zusätzlich verändert werden

Um diesem Problem zu begegnen, instanziiert der Berechnungsagent einen *LogParser*, der die Aufgabe der Interpretation für den Agenten übernimmt. Diese Klasse selbst ist abstrakt. Für jedes numerische Lösungsverfahren existiert eine eigene Parser-Klasse, die genau definiert, wie die Informationen zu lesen sind. Durch dieses Modellierungskonzept ist eine Funktionsfähigkeit bei guter Übersichtlichkeit des Quelltextes gewährleistet. Das zu dieser Struktur gehörende Klassendiagramm ist in Abbildung 5.12 zu finden.

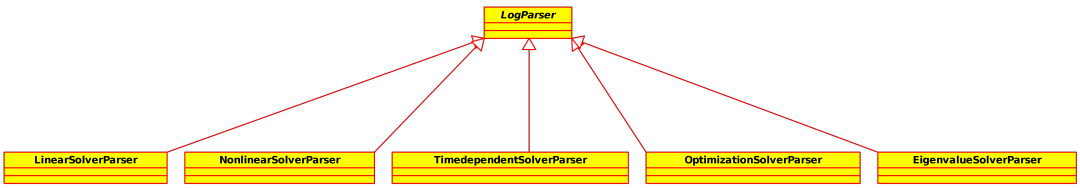


Abbildung 5.12.: Parser-Struktur des Berechnungsagenten

Der Berechnungsagent leitet die gelesenen Daten an den Darstellungsagenten zur Darstellung weiter.

## 5.4. Prozessparameter-Verwaltung

Der Entwurf der Prozessparameter-Verwaltung wurde in Kapitel 4.4 beschrieben. Die Umsetzung wird in den folgenden Kapiteln erläutert. Dabei wird beleuchtet, wie der Darstellungsagent seine Daten erhält, die Darstellung dieser wird in Kapitel 5.2 beschrieben. Die konkrete Implementierung des Datentransfers ist in Anhang A.4 skizziert.

### 5.4.1. Registrierung

Die darzustellenden Prozessparameter werden dem Darstellungsagenten in einem Registrierungsprozess mitgeteilt. Hierzu empfängt er eine *ACLMessage*[9] mit einer *Conversation-ID* mit Inhalt *REGISTER_PARAMETER*[10]. Durch diesen Empfang wird ein neuer Prozessparameter registriert, welcher im Inhalt

[9] siehe Kapitel 1.4.1
[10] für eine vollständige Erklärung des Übertragungsprotokolls siehe Anhang A.3

der Nachricht spezifiziert wird. Der Darstellungsagent erwartet ein *Parameter*-Objekt (siehe Anhang A.2). Hier wird, wie bereits in Kapitel 5.3.3, die FIPA-Spezifikation verletzt. Die Begründung ist analog. Das übergebene Parameter-Objekt kann bei der Registrierung bereits darzustellende Daten enthalten.

### 5.4.2. Update

Einem Prozessparameter können Daten zur Laufzeit hinzugefügt werden. Der Darstellungsagent hat dabei die Aufgabe, die über das Netzwerk empfangenen Daten in seine Datenhaltung zu integrieren. Die GUI des Agenten ist für eine entsprechende Aktualisierung selbst verantwortlich.

### 5.4.3. Deregistrierung

Jedes Parameter-Objekt enthält eine für den Prozessparameter kennzeichnende ID. Wird diese in Verbindung mit der ACL-Message-Conversation-ID[11] *UNREGISTER_PARAMETER* erhalten, so wird das dazu gehörige Parameter-Objekt aus der Datenhaltung des Darstellungsagenten entfernt.

[11] siehe Kapitel 1.4.1

# 6. Anwendungsbeispiel

In diesem Kapitel soll die vorhandene Agenten-Plattform dazu genutzt werden, um ein beispielhaftes elektrotechnisches Problem zu lösen. Dabei wird der Darstellungsagent mit all seinen Funktionen dargestellt und dessen Funktionsfähigkeit überprüft. Das Problem ist dabei sehr einfach gehalten, um eine Konzentration auf den Darstellungsagenten zu ermöglichen.

## 6.1. Problem

Gesucht ist im Folgenden das sich einstellende elektrische Feld zweier Kugelelektroden, welche im Abstand von 5 cm voneinander entfernt eine Potentialdifferenz von 1 V aufweisen. Beide Elektroden weisen einen Radius von 1 cm auf. Sie werden als perfekte elektrische Leiter angenommen. Eine Kugelelektrode ist geerdet.

## 6.2. Modell

Modelliert wird mit Comsol Multiphysics[1] Version 4.3. Das Modell wird dabei nach den Anforderungen des beschriebenen Problems erstellt. Das entstandene Modell ist in Abbildung 6.1 zu sehen.

## 6.3. Berechnung

Die Berechnung selbst geschieht nach Start der Agenten-Plattform. Zunächst wird das Modell geladen, anschließend wird die Berechnung durchgeführt. Beides zusammen benötigt ca. 60 Sekunden (siehe Abbildung 6.2). Das reine Berechnen dauert 16 Sekunden, Comsol berechnet das Modell in ca. 13 Sekunden. Um über die Effektivität des Systems eine Aussage treffen zu können, müsste die Berechnung genauer untersucht werden. Dies ist jedoch nicht Gegenstand dieser Arbeit.

Abbildung 6.2 zeigt die CPU- und RAM-Auslastung im Vergleich zwischen dem

[1] Für mehr Informationen: [11]

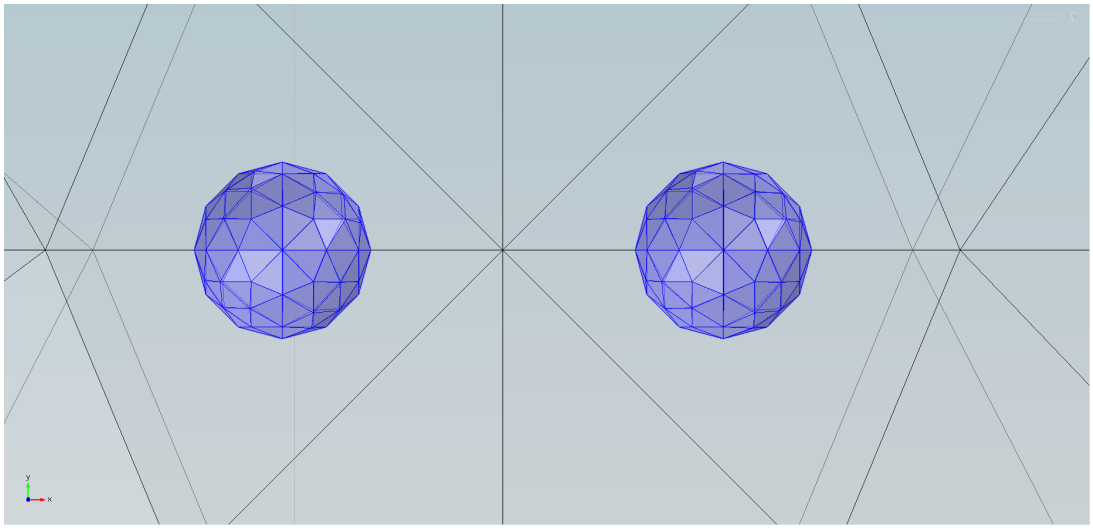

Abbildung 6.1.: Modell des Anwendungsbeispiels

in dieser Arbeit implementierten *PCStatusMonitor* und dem Windows-Bordmittel *TaskManager*. Hier ist erkennbar, dass die CPU-Auslastung in etwa denselben Verlauf besitzt. Die RAM-Auslastung variiert zwischen StatusMonitor und TaskManager um ca.

$$17\,\% - 14\,\% = 3\,\%,$$

was einer absoluten Differenz von

$$0{,}03 \cdot 32\,\mathrm{GB} \approx 1\,\mathrm{GB}$$

entspricht. Der Rechner verfügt dabei über 32 GB Arbeitsspeicher. Grund hierfür kann ein Fehler in *SIGAR* sein. Dennoch ist eine Tendenz ablesbar, was die Information für den StatusMonitor brauchbar macht. Hier ist auch in Abbildung 6.2 ablesbar, dass während des Ladens und Berechnens der Arbeitsspeicher um ca. 1,5 % zunimmt, was einen effektiven Arbeitsspeicherverbrauch von ca.

$$0{,}015 \cdot 32\,\mathrm{GB} \approx 500\,\mathrm{MB}$$

bedeutet. Im Rahmen der Ablesbarkeit ist dies in etwa die von Comsol angegebene Arbeitsspeicherbelegung.

Die Statistik des Meshes aus dem Anwendungsbeispiel ist in Abbildung 6.3 dargestellt. Dieses ist weitgehend vollautomatisch generiert, daher sind nur wenige Informationen dargestellt. Zur Erinnerung: Felder ohne Belegung, beispielsweise das Feld *Autor*, werden vom Darstellungsagenten ignoriert.

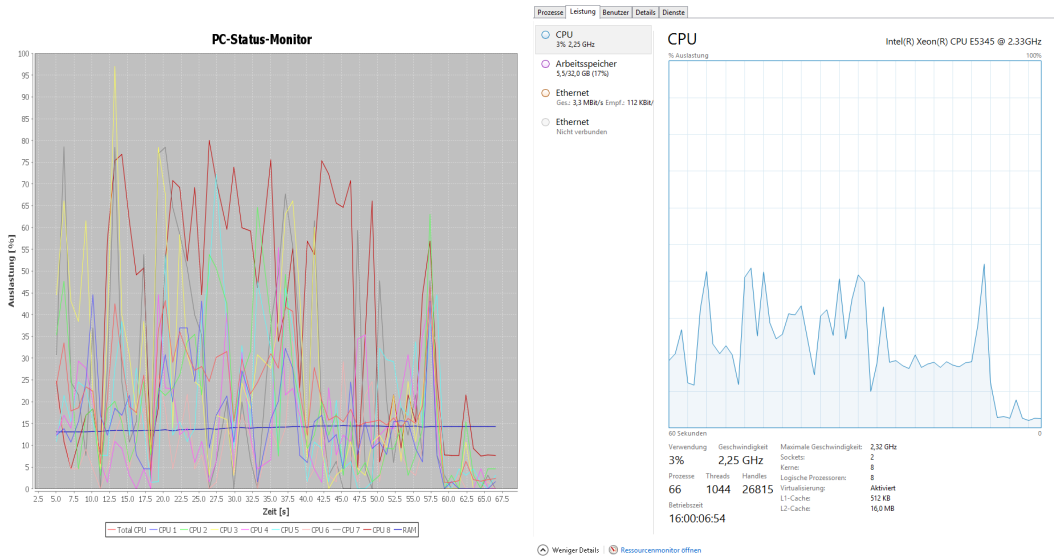


Abbildung 6.2.: Rechner-Auslastung bei der Berechnung des Anwendungsbeispiels – links: PCStatusMonitor, rechts: TaskManager

Das zugehörige Konvergenzdiagramm ist in Abbildung 6.4 zu finden. Er wurde vom Darstellungsagenten gezeichnet. Abbildung 6.5 zeigt das Konvergenzdiagramm von COMSOL, welcher dieses bei der Berechnung desselben Modells erzeugt. Man sieht eine sehr starke Ähnlichkeit, somit kann das vom Agenten gezeichnete Diagramm als äquivalent betrachtet werden.

## 6.4. Ergebnis

Die Agenten-Plattform, mit der an dieser Stelle gearbeitet wurde, ist eine vereinfachte Version, daher kann das Simulationsergebnis nicht dargestellt werden. Der Darstellungsagent ist davon jedoch nicht betroffen, er findet die gleichen Schnittstellen vor wie in der eigentlichen Version der ITE-Agentenplattform.

**Mesh-Statistik für Mesh 1 (mesh1):**

| | |
|---|---|
| Max. Growth Rate: | **3.621** |
| Max. Volume: | **20.46** |
| Mean Growth Rate: | **1.632** |
| Mean Quality: | **0.7687** |
| Min. Quality: | **0.1481** |
| Number of elements: | **76136** |
| Volume: | **521800.0** |
| isActive: | **true** |
| isComplete: | **true** |
| isEmpty: | **false** |
| Model: | **mod1** |
| Tag: | **mesh1** |
| Name: | **Mesh 1** |
| Scope: | **root.mod1** |
| Date created: | **Nov 2, 2014 1:54:15 PM** |
| Date modified: | **Nov 2, 2014 1:54:15 PM** |
| Help: | **file:///C:/Program Files/COMSOL/COMSOL43b/doc/html/comsol/helpdesk.htm** |

Abbildung 6.3.: Mesh-Statistik des Anwendungsbeispiels

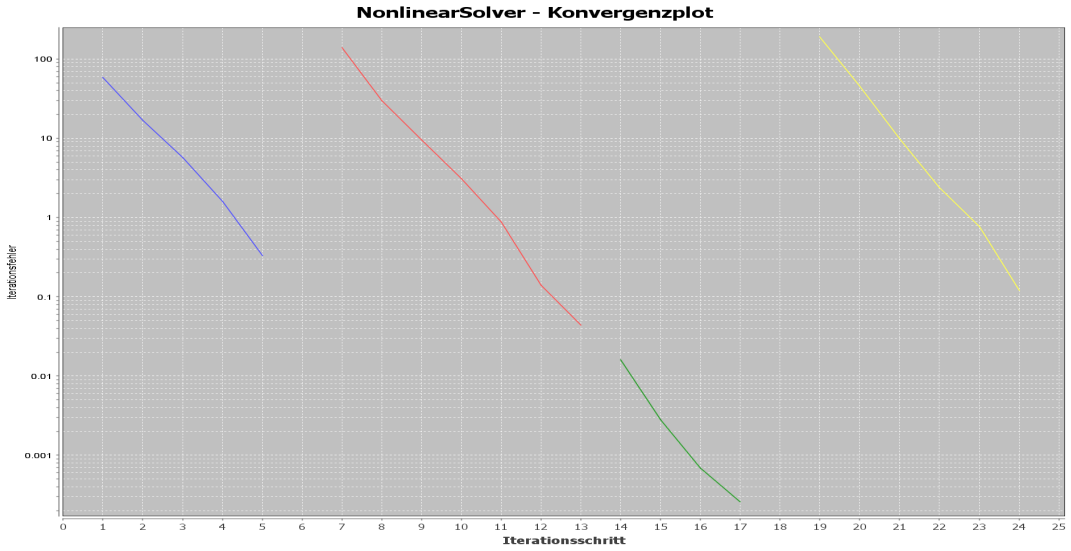


Abbildung 6.4.: Konvergenzdiagramm des Anwendungsbeispiels

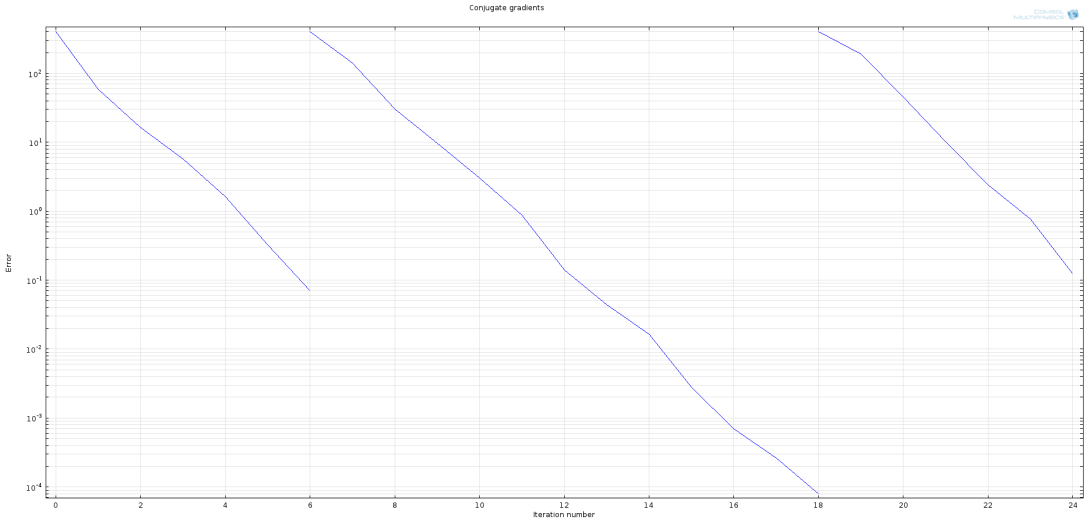


Abbildung 6.5.: Konvergenzdiagramm des Anwendungsbeispiels bei der Berechnung mit Comsol

# 7. Zusammenfassung und Ausblick

Das Hauptziel dieser Arbeit besteht darin, ein Programm zu schaffen, welches in einfacher Weise Prozessparameter aus einer Agentenplattform zur Simulation numerischer Probleme extrahieren und darstellen kann. Dabei soll die Benutzeroberfläche intuitiv, einfach gestaltet und leicht zu bedienen sein. Das Design der Oberfläche soll ansprechend und in sich geschlossen wirken.

Der in dieser Arbeit entwickelte Agent besitzt die Fähigkeit, Prozessparameter zu empfangen, zu speichern und graphisch darzustellen. Obzwar die Daten statistisch nicht ausgewertet werden, wird dies durch die Diagramme wesentlich erleichtert. Die Benutzeroberfläche verzichtet auf Menüs, hierdurch lässt sich die Oberfläche einfach und ohne Vorkenntnisse bedienen. Die meisten Registerkarten sind dabei mit Icons versehen, welche eine Bedienung ohne das Lesen von Text ermöglichen. Damit wird eine intuitive Bedienbarkeit ermöglicht. Die grafische Benutzeroberfläche visualisiert das Agenten-System durch die Fensteraufteilung in geeigneter Weise. Dies ermöglicht auch den direkten Vergleich von Prozessparametern verschiedener Agenten.

Der hier entwickelte Agent erfüllt damit die an ihn gestellten Anforderungen. Allerdings gehen die theoretischen Möglichkeiten des Agenten über die in dieser Arbeit entwickelten weit hinaus. Diese Erweiterungen werden im Folgenden erläutert.

## 7.1. Auswerten der Daten

Der Darstellungsagent könnte die Daten, die er von den anderen Agenten bekommt, auswerten und entsprechend reagieren. Hierdurch ließe sich beispielsweise eine höhere Stabilität oder eine höhere Effizienz des Agenten-Systems erreichen. Durch den PC-Status-Monitor kann der jeweilige Rechner überwacht werden. Dies lässt in Grenzen Rückschlüsse auf die „Gesundheit“, also die Funktionsfähigkeit, des Systems zu. Wird ein Defekt erkannt, so kann versucht werden, diesen zu beheben. Sollten die Grenzen des Agenten erreicht sein, so können Mitteilungen in Form von SMS, E-Mail oder Ähnlichem erfolgen. Hierdurch kann wertvolle Zeit gespart werden, die zwischen dem Auftreten des Fehlers und dessen Entdeckung vergeht. Mögliche Beispiele für Anwendungen sind:

- Neustart des Agenten, wenn diese sich nicht rückmelden
- Portierung des Agenten auf einen anderen Rechner, um vorhandene Rechner-Kapazitäten möglichst effizient auszunutzen
- Warnmeldung oder eine geeignete Ereignisbehandlung, falls einzelne Rechner nicht genutzt werden
- Warnmeldung, E-Mail-Mitteilung oder Ähnliches, wenn der Rechenvorgang unerwartet beendet wurde

## 7.2. Weitere Funktionen

Der Agent kann um zusätzliche Funktionalität erweitert werden. Einige Möglichkeiten werden im Folgenden kurz vorgestellt.

### 7.2.1. Ausgliederung der Tabs

Der Vergleich von Prozessparametern verschiedener Agenten ist zurzeit problemlos möglich. Doch ein Vergleich verschiedener Prozessparameter desselben Agenten gestaltet sich schwierig, da stets zwischen verschiedenen Tabs gewechselt werden muss. Es ist daher empfehlenswert, die GUI so anzupassen, dass dieses Problem gelöst werden kann. Ein möglicher Ansatz besteht dabei, die Tabs auf Mausklick ebenfalls als *Dockable* auszugliedern. Hierdurch wird die Flexibilität vergrößert, allerdings ist dann die Zuordnung eines Prozessparameters zu einem Agenten nicht zweifelsfrei möglich. Abhilfe kann hier der Einsatz von Farben schaffen.

### 7.2.2. Filterfunktion der Konsole

Die Konsole ist momentan in der Lage, Text – je nach Wichtigkeit verschieden formatiert – darzustellen. Für eine bessere Übersichtlichkeit, ist ein Filter, mit dem unwichtigere Nachrichten ausgeblendet werden können, vorteilhaft. Dieser würde Betrieb und Fehlersuche gleichermaßen erleichtern.

### 7.2.3. Suchfunktion der Konsole

Eine weitere Erweiterung für die Konsole stellt eine Suchfunktion dar. Diese Funktion wird voraussichtlich weniger gebraucht, ist jedoch heute nahezu überall zu finden (Browser, Dokumentenbetrachter, Dateimanager, ...). Sie kann im

Einzelfall enorm Zeit sparen und ist mit nicht allzu hohem Aufwand implementierbar, daher wird sie dennoch als sinnvoll eingestuft. In Kombination mit der vorgestellten Filterfunktion (Kapitel 7.2.2) kann die Suchfunktion treffende Ergebnisse liefern.

### 7.2.4. Export der Konsole

Eine weitere nützliche Funktion ist der Export des Konsoleninhaltes in eine Datei. Hierdurch könnten verschiedene Anwendungsfälle gespeichert und verglichen werden. Auch die Fehlersuche wäre damit wesentlich leichter, vor allem im produktiven Einsatz des Systems.
Ein Export ist bereits durch die Tastenkombination [Strg]+[C] in die Zwischenablage möglich, jedoch geht hierbei die Formatierung des Textes verloren. Ein sehr effektives System wäre der Export der Konsole in Kombination mit der Filterfunktion (Kapitel 7.2.2), denn so kann der Export des Konsoleninhaltes auf den entscheidenden Inhalt beschränkt werden.

### 7.2.5. Export der Diagramme

Die Speicherung der Diagramme ist bislang nur über einen *Screenshot*[1] möglich. Diese Fähigkeit zur Speicherung von Diagrammen wird häufig benötigt, beispielsweise für Präsentationen. Der Screenshot muss nachbearbeitet werden, da er nicht nur das Diagramm enthält. Daher wäre die gezielte Speicherung einzelner Diagramme per Knopfdruck als Bild oder PDF evtl. mit den Datensätzen eine äußerst nützliche Funktion.

[1] Bildschirmaufnahme in Form eines Bildes

# A. Datenmanagement

Dieses Kapitel erläutert einige bedeutende Aspekte der Implementierung des Darstellungsagenten. Diese sind sehr speziell und werden daher nur benötigt, wenn der Quelltext verstanden werden will.

## A.1. Text-Level

Da die Konsole eine formatierte Ausgabe erlaubt, wurden sogenannte *Text-Level* eingeführt. Diese lauten (von wichtig zu unwichtig):

1. *Fatal*: Das Problem, welches diese Nachricht verursacht, bringt das Programm zum Absturz.
2. *Error*: Dieses Problem kann nicht ignoriert werden. Das Programm läuft dennoch fort.
3. *Warning*: Das aufgetretene Problem kann ignoriert werden.
4. *Information Important*: Der Text beinhaltet eine wichtige Information für den Benutzer.
5. *Information Casual*: Der Text beinhaltet eine beiläufige Information für den Benutzer, welche dieser nicht unbedingt benötigt.
6. *Debug*: Der Text dient ausschließlich als Debug-Information. Der Benutzer sollte diese Information nicht sehen.
7. *Unknown*: Falls kein Text-Level definiert ist oder sonstige Fehler bei dessen Dekodierung auftreten.

## A.2. Wichtige Klassen

**reportagent.stats.Parameter** Repräsentiert einen Datensatz mit sämtlichen Informationen, die zur Darstellung eines Prozessparameters erforderlich sind, z. B. den Diagramm-Typ. Das zur Klasse zugehörige Attribut *parameterID* definiert klar den Typ des Prozessparameters, z. B. Fortschritt.

*Hinweis:* Dieses Attribut darf keinesfalls mit dem Attribut *type* verwechselt werden, welches den *Diagramm-Typ* spezifiziert, also z. B. ein XY-Chart.

**reportagent.stats.ParameterMap** Abstrakte Klasse, die Informationen über einzelne Parametertypen enthält, beispielsweise das X-Achsen-Label des Parameters *PCStatus*.

**reportagent.ProtocolRA** Interface, welches Konstanten beinhaltet. Diese definieren das Übertragungsprotokoll (siehe Kapitel A.3).

## A.3. Das Übertragungsprotokoll

Übertragen werden müssen folgende Kommandos:

**REGISTER_PARAMETER** Befehl zur Registrierung eines neuen Prozessparameters beim Darstellungsagenten. Es wird erwartet, dass in der ACLMessage als *ContentObject* ein Objekt des Typs *Parameter* vorliegt.

**UNREGISTER_PARAMETER** Befehl zum Löschen eines Prozessparameters. Wieder wird als *ContentObject* ein Objekt vom Typ Parameter erwartet.

**UPDATE_PARAMETER** Befehl zum Update eines Prozessparameters. Die Syntax lautet:

Quelltext A.1: Syntax UPDATE_PARAMETER

```
<parameterID>;<Daten>
```

wobei der Tag *Daten* frei spezifiziert werden darf. Der Code zur Interpretation dieser Zeile findet sich dabei in der Klasse *reportagent.behaviours.UpdateMsgHandler*.

## A.4. Austausch der Daten

Dieser Teil beschreibt die Implementierung des Datentransfers zwischen den Agenten. Diese ist sehr spezifisch und bezieht sich direkt auf die in Kapitel 1.4 vorgestellte Beispielplattform, in die der Darstellungsagent eingebettet ist.

### A.4.1. Sendeseite

Für die Verwaltung der Daten ist der Numerikagent mit der Klasse *numerikagent.InformationManager* ausgestattet. Die Klasse *InformationManager* besitzt die Methode *addParameter(...)*, die den Darstellungsagenten über einen neuen Prozessparameter informiert. Ferner besitzt der *InformationManager* die Methode *update(...)*, die zuerst *addParameter(...)* aufruft, wenn nicht bereits geschehen. Anschließend wird der Darstellungsagent über die neuen Daten informiert.

Da der Berechnungsagent von Numerikagent erbt, steht ihm diese Klasse ebenfalls zur Verfügung. Zu beachten ist, dass im Design davon ausgegangen wurde, dass es von dieser Klasse nur ein Objekt gibt, dies jedoch nicht sichergestellt wurde. Hintergrund ist, dass es mehrere Instanzen des Darstellungsagenten geben kann. Weiter läuft man bei der Programmierung keine Gefahr, mehrere Instanzen der Klasse *InformationManager* zu erzeugen.

### A.4.2. Empfangsseite

Die Parameter, die neu registriert werden, werden in dem Darstellungsagenten hinterlegt. In der GUI wird ein neuer Tab mit dem entsprechenden Parameter angelegt. Zu beachten ist, dass ein Darstellungsagent mehrere Numerikagenten kennen kann und jeder Numerikagent mehrere Parameter besitzen kann. Dies ist in der Datenstruktur zur Speicherung der Parameter berücksichtigt, es ergibt sich das Klassendiagramm aus Abbildung A.1.

Wird nun ein neues Datum empfangen, so wird ein neuer Parameter registriert oder ein bestehender aktualisiert. Die GUI aktualisiert sich in diesem Fall automatisch.

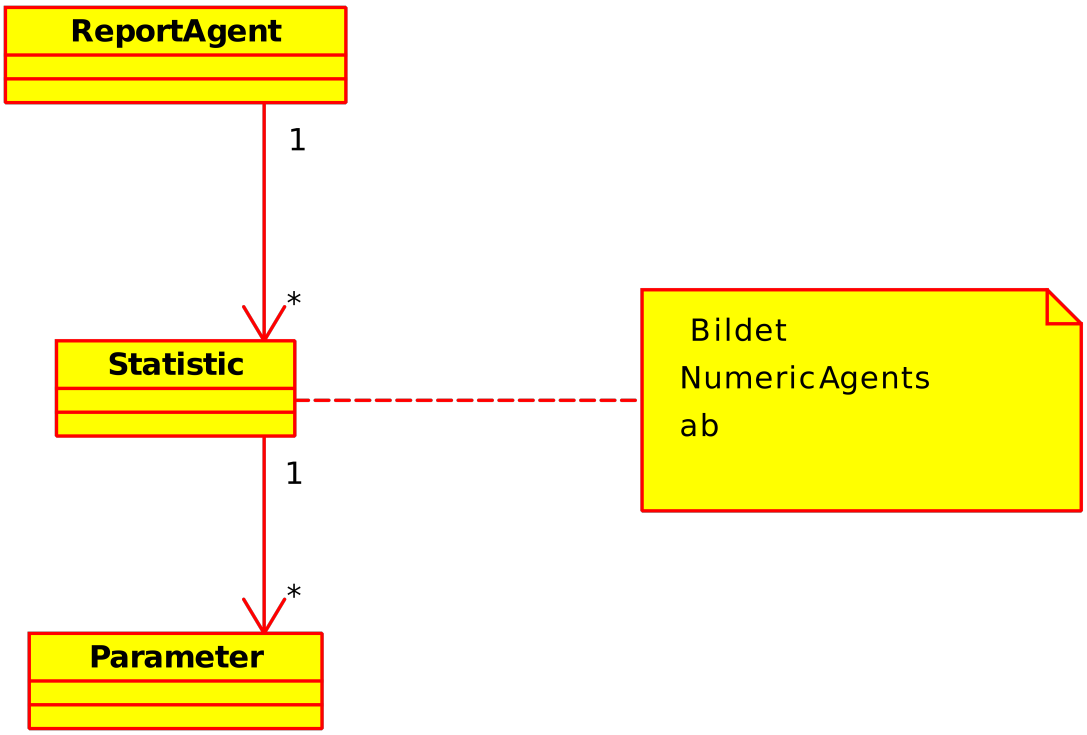


Abbildung A.1.: Parameterverwaltung des Darstellungsagenten

# B. Hinzufügen von Prozessparametern

Für das nachträgliche Hinzufügen von Parametern in das System müssen lediglich wenige Stellen im Programmtext bearbeitet werden. Diese werden im Folgenden kurz vorgestellt. Da dieses Kapitel als Programmieranleitung gedacht ist, wird für dessen Verständnis Zugang und Verständnis des Quelltextes vorausgesetzt.

## B.1. Weiterleitung an den Darstellungsagenten

Es wird davon ausgegangen, dass die Daten bereits abrufbar sind. Diese müssen an den Darstellungsagenten übergeben werden. Dies geschieht durch die Klasse *numericagent.InformationManager* mit der Methode *addParameter(...)*. Diese ist überladen (siehe Quelltext B.1).

Quelltext B.1: Auszug aus numericagent.InformationManager

```
public void addParameter(Parameter p) {
        ...
}

public void addParameter(int paramID) {
        addParameter(new Parameter(paramID));
}
```

In der einen Version erwartet sie einen Parameter vom Typ *reportagent.stats.Parameter*, in der anderen einen Parameter vom primitiven Typ *Integer*. Dabei erzeugt die Methode mit dem primitiven Parameter ein neues *Parameter*-Objekt und übergibt dieses an die andere Methode. Obwohl ein Fehlerfall nicht überprüfbar ist, kann davon ausgegangen werden, dass nach Aufruf dieser Methode das *Parameter*-Objekt korrekt an Darstellungsagent übergeben wurde. Aufgetretene Fehler wird die *update(...)*-Methode beheben.

## B.2. Parameter

Um die *addParameter(...)*-Methode des *InformationManagers* aufrufen zu können, wird mindestens eine *parameterID* benötigt (siehe Quelltext B.1). Diese sollte in der Klasse *Parameter* als Konstanten definiert werden. Dabei ist der Wert in Grenzen frei wählbar – er sollte zwischen 1 und 2000 liegen. Ein Beispiel ist die Konstante *Parameter.PROGRESS*, die einen Wert von 100 aufweist.

Die genauere Definition des Parameters (Diagramm-Typ, Achsen-Beschriftungen,...) werden in der Klasse *reportagent.stats.ParameterMap* vorgenommen. Dabei sollten mindestens die Methoden *getParameterType(...)*, *getParameterTitle(...)* und *getParameterDataset(...)* erweitert werden, andernfalls wird eine Warnung ausgegeben. Die Methoden sind alle nach ähnlichem Muster Aufgebaut. Ein Beispiel ist in Quelltext B.2 zu finden.

Quelltext B.2: reportagent.stats.ParameterMap.getParameterTitle()

```
public static String getParameterTitle(int parameterID) {
        try {
                switch (parameterID) {
                case Parameter.PROGRESS:
                        return "Fortschritt";

                case Parameter.MEMORY:
                        return "Speicherauslastung";

                case Parameter.PCSTATUS:
                        return "PC-Status-Monitor";

                default:
                        handleUnknownParameterError();
                        return null;
                }
        } catch (Exception e) {
                return null;
        }
}
```

Hier ist zu erkennen, dass die Methoden als *static* deklariert sind. Die Differenzierung der Parameter erfolgt in einer die Methode bestimmenden *switch*-Verzweigung. Hier muss durch ein weiteres *case*-Statement lediglich der neue Prozessparameter definiert werden. Hierbei ist die *ID* des Prozessparameters, definiert in *reportagent.stats.Parameter*, anzugeben.

## B.3. Update-Messages

Sofern der Prozessparameter aktualisiert werden muss, muss auch die Update-Message *UPDATE_PARAMETER* definiert werden (siehe dazu Kapitel A.3). Dies geschieht sendeseitig durch einen entsprechenden Aufruf der Methode *update(...)* des *InformationManagers*. Der übergebene Parameter *updateMsg* vom Typ String muss dabei die korrekte Definition des *Daten*-Tags wahren (siehe Quelltext A.1). Empfangsseitig muss die Methode *action()* der Klasse *reportagent.behaviours.UpdateMsgHandler* erweitert werden (siehe Quelltext B.3). In der Methode wird zuerst geprüft, ob die Nachricht korrekt ist. Die Interpretation des *Daten*-Tags findet in der großen *switch*-Anweisung am unteren Ende der Methode statt.

Quelltext B.3: reportagent.behaviours.UpdateMsgHandler.action()

```
try {
        // Validierung und Vorverarbeitung der Nachricht

        switch(p.getParameterID()) {
                // Verarbeitung des Daten-Tags
                // Erweiterung hier!
        }
} catch (Exception e) {
        e.printStackTrace();
}
```

# Abbildungsverzeichnis

# Quelltextverzeichnis

# Literaturverzeichnis


[1] Bauke, Heiko ; Mertens, Stephan: *Cluster Computing*. Springer, 2006. – ISBN 978–3–540–42299–0

[2] Bellifemine, Fabio ; Caire, Giovanni ; Greenwood, Dominic: *developing multi-agent systems with JADE*. John Wiley & Sons, 2007 (Wiley Series in Agent Technology). – ISBN 978–0–470–05747–6

[3] Göhner, Prof. Dr.-Ing. Dr. h. c. Peter: *Grundlagen der Softwaretechnik*. 2012

[4] Hoffmann, Dirk W.: *Software-Qualität*. 2. Springer Vieweg, 2013. – ISBN 978–3–642–35699–5

[5] Krüger, Guido ; Stark, Thomas: *Handbuch der Java-Programmierung*. 5. Addison-Wesley, 2009. – ISBN 978–3–8273–2815–1

# Verzeichnis der Webadressen


[6] Projektseite DockingFrames. http://dock.javaforge.com/. Eingesehen am 12.11.2014

[7] Projektseite JFreeChart. http://www.jfree.org/jfreechart/. Eingesehen am 12.11.2014

[8] Projektseite Sigar. https://support.hyperic.com/display/SIGAR/Home. Eingesehen am 12.11.2014

[9] Icon der konsole. http://upload.wikimedia.org/wikipedia/commons/8/82/Konsole-icon.png. Eingesehen am 12.11.2014

[10] Icon des status-monitors. http://findicons.com/files/icons/2166/oxygen/128/utilities_system_monitor.png. Eingesehen am 12.11.2014

[11] Projektseite Comsol. http://www.comsol.com/comsol-multiphysics. Eingesehen am 12.11.2014