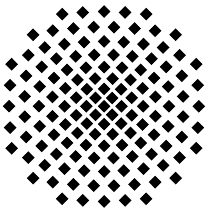

Universität Stuttgart
Institut für Theorie der Elektrotechnik
Prof. Dr. techn. Wolfgang M. Rucker

Master thesis

Web based Visualization

Nan Zhao

Betreuer:	Dr.-Ing. Matthias Jüttner
Beginn der Arbeit:	01.02.2017
Abgabe der Ausarbeitung:	24.07.2017

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 24.07.2017

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. Introduction

1.1. Motivation

1.2. Outline of this thesis

2. Visualization system

2.1. System architecture

2.2. Work flow

3. System design

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 **??** 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.

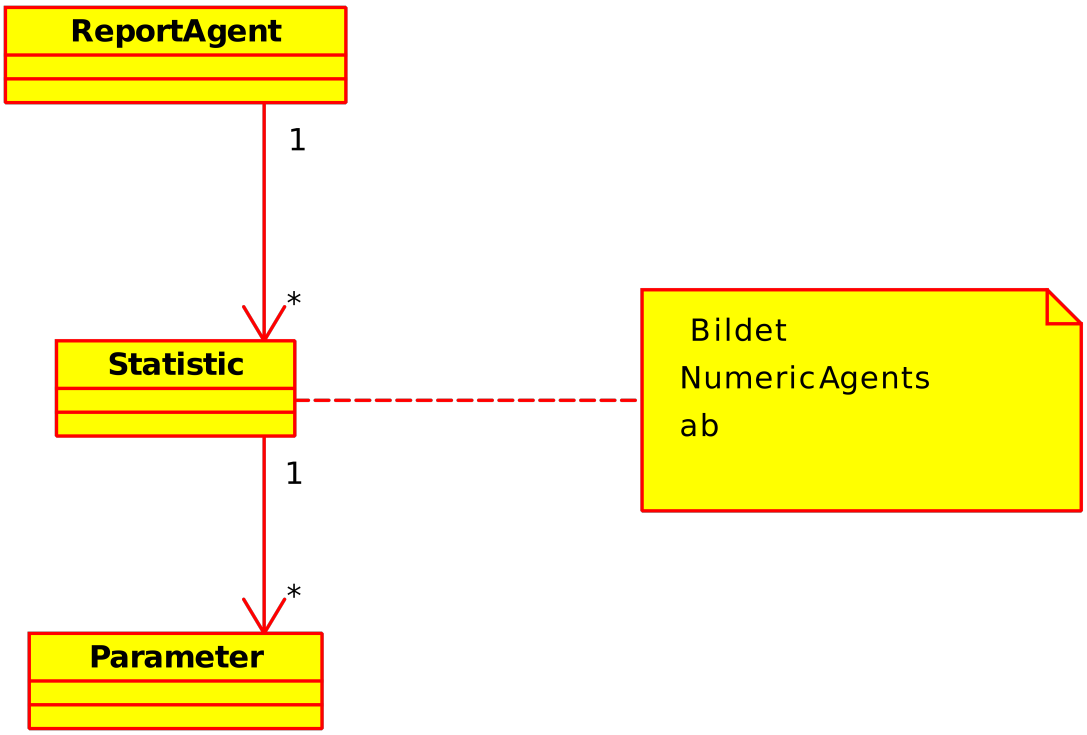

Abbildung A.1.: Parameterverwaltung des Darstellungsagenten

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();
}
```

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