Technische Universität Darmstadt

Fachgebiet Simulation, Systemoptimierung und Robotik

Bachelorarbeit

Blackopt - effiziente Ankopplung entfernter Black Box Auswertungen an Optimierungs-Infrastrukturen

Florian Nöll

Betreuer:
Prof. Dr. Oskar von Stryk
Dr.-Ing. Thomas Hemker

Abgabedatum: 16. November 2009

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 16. November 2009

.....

(Unterschrift des Kandidaten)

Abstract

Mathematische Probleme können oft nicht lokal optimiert werden, da Teilauswertungen auf entfernten Rechnern ausgeführt werden müssen. Damit der Benutzer mit den entfernten Rechnern kommunizieren kann, muss ein Framework zur Verfügung gestellt werden, das die Übertragung der mathematischen Probleme, die Berechnung und deren Auswertungen unterstützt.

Die bisherigen Ansätze solcher Tools haben gezeigt, dass es nur möglich ist, vollständig mathematisch modellierbare Optimierungsprobleme zu verarbeiten. Dies motiviert die Annahmen in dieser Arbeit zur Erstellung eines solchen Systems.

Die Annahmen sind zum Einen eine einfache Verwaltungsoberfläche für bestehende und neue Probleme, die mittels eines Webservices mit den Endgeräten kommunizieren kann. Zum Anderen eine zuverlässige und koordinierte Verbindung zwischen den Blackboxen, die die Teilauswertungen liefern und dem Optimierungsserver.

Die Evaluation des erstellten Systems hat gezeigt, dass der Ansatz ein einfaches und intuitives Werkzeug zur Optimierung sein kann. Die Benutzbarkeit der momentanen Umsetzung deckt jedoch nur einen kleinen Teil der Möglichkeiten zur entfernten Optimierung ab. Die Erweiterungen des Systems stellen aber lösbare Probleme dar, deren Umsetzung Teil von zukünftigen Arbeiten sein wird.

Inhaltsverzeichnis

1 Einleitung

Die Aufgabe dieser Einleitung ist es dem Leser die Gründe für diese Arbeit zu vermitteln, zu motivieren und einen kurzen Überblick über die anschließenden Kapitel zu verschaffen.

1.1 Motivation

Viele Probleme in der Wirtschaft und Industrie sind mathematische Optimierungsprobleme [1], wie beispielsweise das an den Menschen angepasste Laufen eines Roboters, die Minimierung von Produktionskosten in der industriellen Herstellung oder auch die Verkehrssteuerung auf unseren Straßen. Generell besteht das Modell eines Optimierungsproblems aus einer Zielfunktion und den Nebenbedingungen. Gesucht ist bei einem Optimierungsproblem [1] diejenige Variablenkonstellation, die alle Nebenbedingungen erfüllt und in die Zielfunktion eingesetzt einen möglichst großen (kleinen) Wert ergibt. Die Modellierung dieser Probleme wird mit Hilfe einer mathematischen Modellierungssprache [2] erzeugt. Das Problem wird in der Regel durch die Modellierung nur angenähert [2]. Gelöst wird das Optimierungsproblem durch den Einsatz eines Solvers, welcher ein Verfahren zum Lösen des Problems implementiert. Es gibt eine Vielzahl an frei verfügbaren Solvern im Internet.

1.2 Problemstellung

Diese mathematischen Optimierungsprobleme können häufig nicht lokal optimiert werden und der Benutzer muss dafür einen erhöhten Aufwand betreiben. Die Gründe hierfür sind zum einen spezielle für die Optimierung eingesetzte entfernt stehende Systeme. Auf diesen Systemen sollen für das Optimierungsproblem relevante Teilauswertungen ausgeführt werden. Zum anderen braucht die Auswertung Zeit und Ressourcen. Weiterhin kann die entfernte Anbindung direkt aus einem Optimierungsverfahren heraus kann nur schwer angesteuert werden. Bereits existierende Systeme (Webservices) bieten nur eine Lösung für vollständig mathematisch modellierbare Optimierungsprobleme, wie beispielsweise $minf(x)$ mit $min\{c^T x \mid Ax \leq b, x \geq 0\}$ mit $c^T x$ als Zielfunktion und unter den Nebenbedingungen $Ax \leq b$. Nicht aber für Probleme der

[1] Optimierungsproblem - siehe Abschnitt 2.1
[2] Modellierungssprache - siehe Abschnitt 2.1.2

Form $minf(x, \mathbf{y}(x))$, mit $\mathbf{y}$ als Blackbox, die nur durch Auswertung eines Computerprogramms, Simulation oder eines Experiments bestimmt werden kann. Die Ergebnisse der Blackbox werden während der Iterationsphase auf Seiten der Optimierung gebraucht. Aus diesem Grund soll für diese Problemstellungen eine einfache Übermittlung an geeignete Optimierungsverfahren untersucht werden. Weiterhin soll die Erstellung eines Webinterfaces der intuitiven Verwaltung der entfernten Optimierung genügen. Optimierungsprobleme werden oft in verschiedenen mathematischen Programmiersprachen formuliert. Um dem Benutzer ein möglichst universelles Tool zur Seite zu stellen, soll eine Lösung für die Integration dieser Sprachen erstellt werden. Hierbei sollen alle expliziten Probleminformationen der Modellierungssprache erhalten bleiben, und an Optimierungsverfahren direkt weitergegeben werden, die diese nutzen können.

Ein solches System, welches im Rahmen dieser Bachelorarbeit erstellt wurde, wird in Abbildung 1.1 gezeigt. Das Blackopt System gibt dem Benutzer ein Webinterface zum Verwalten entfernter Blackboxen. Grundlage des Webinterfaces ist ein Webservice[3], der die Kommunikation zwischen Blackbox, Webinterface und Optimierungsserver leitet. Innerhalb des Webservices werden Informationen der Teilauswertungen gesendet und in einer Datenbank gespeichert. Beispielhaft wurde die Modellierung der Optimierungsprobleme in Matlab gewählt. Allerdings lässt dieser Ansatz noch Verbesserungsmöglichkeiten zu. Matlab bietet jedoch die Möglichkeit Java Programme

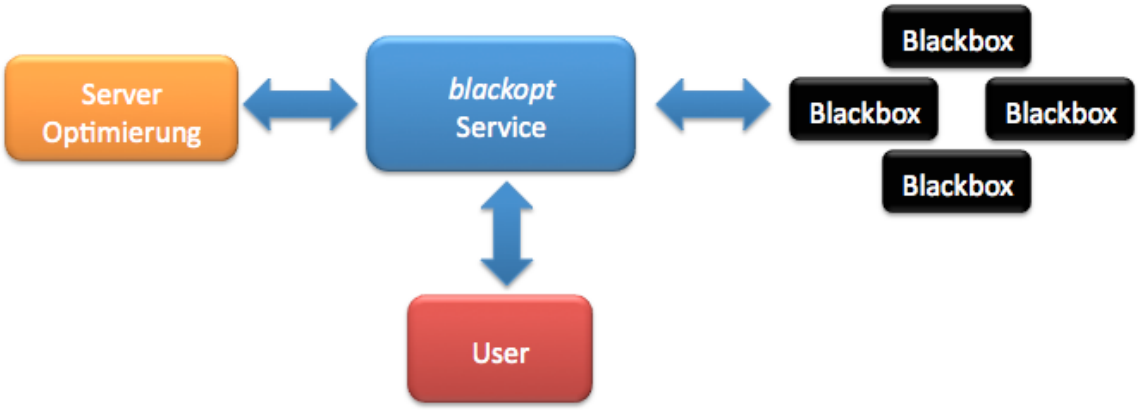

Abbildung 1.1: Zusammenspiel der Systemkomponenten

zu integrieren, welche Bestandteile des Blackopt Systems sind. Der Ansatz zum Verwalten der Blackboxen basiert auf einer einfachen Annahme. Durch die Eingabe von blackbox-spezifischen Daten erstellt der Benutzer eine Blackbox im System. Mittels der nun vorhandenen Blackbox erstellt er ein Problem, welches eine Blackbox für seine Teilauswertungen benötigt. Der eigentliche Prozess im Webinterface wird als Job bezeichnet. Ein Job besteht aus einem Problem und kann von einem Benutzer angelegt werden. Alle Auswertungen werden unter einer JobID in der Datenbank gespeichert.

[3]Webservice - siehe Abschnitt 3.3.1 auf Seite 17

Der Benutzer kann somit jederzeit bereits bekannte Ergebnisse der Auswertungen anschauen.

Das Blackopt System besteht aus den Blackboxen, dem Webinterface und einem Tool zur Anbindung von Modellierungssprachen. Die Blackbox ist ein entfernter Computer auf dem eine mathematische Auswertung ausgeführt werden kann. Speziell auf Seite der Blackbox wurde ein weiterer Webservice erstellt, der Eingabewerte entgegennimmt, das lokale Programm zur Auswertung startet und die Auswertungen an den Blackopt Webservice zurücksendet.

1.3 Überblick über die Kapitel

Dieser Überblick über die nachfolgenden Kapitel soll dem Leser eine kurze Zusammenfassung der Inhalte geben, die ab jetzt folgen.

Kapitel 2 Stand der Forschung
Hier werden aktuelle Systeme näher diskutiert. Ziel ist es, die Anforderungen des Blackopt Systems klar zu formulieren.

Kapitel 3 Grundlagen
Hier werden Grundlagen, die für das weitere Verständnis dieser Arbeit notwendig sind, vermittelt und im Detail beschrieben. Dabei soll klar werden, warum die eingesetzten Programmiersprachen, Programmiertechniken und Programme verwendet wurden. Weiterhin werden Anwendungsfälle beschrieben, die die Grundlagen des entwickelten Systems bilden abgeleitet aus den Anforderungen, die in Kapitel 2 definiert worden sind.

Kapitel 4 Umsetzung
Nach der Analyse sollen hier einige Implementierungsdetails des Blackopt Systems wiedergegeben werden. Speziell werden hier die Komponenten des Systems beschrieben.

Kapitel 5 Exemplarischer Einsatz
Hier wird das entwickelte System getestet. Die Bedienung wird anhand von Bildschirmmasken näher erläutert.

Kapitel 6 Ausblick Im letzten Kapitel werden die Ergebnisse der Evaluation diskutiert, auf die Schwachstellen eingegangen und einige Erweiterungsmöglichkeiten für die Zukunft vorgestellt.

2 Aktueller Stand der Forschung

Der aktuelle Stand der Forschung soll einen tieferen Einblick in die bereits bestehenden Systeme geben. Ein besonderes Augenmerk wird dabei auf die Anbindung von Blackboxen an die existierenden Systeme sowie die Funktionalität des Systems gelegt. Im Folgenden steht das Optimierungsproblem im Mittelpunkt um die Problemstellung und den Stand der Forschung besser zu verstehen. Der Leser bekommt eine allgemeine Einführung in Optimierungsprobleme und den vorhandenen Modellierungssprachen.

2.1 Optimierungsproblem

Aus der Modellbildung folgen Systeme mit vielen Unbekannten, die in einem Vektor $\boldsymbol{x}$ passender Dimension zusammengefasst werden. Weiterhin gilt es Nebenbedingungen in Form von Gleichungen und Ungleichungen zu formulieren, welchen $\boldsymbol{x}$ genügen muss. Die Zielfunktion f eines Systems hängt von $\boldsymbol{x}$ ab und wird durch eine geeignete Wahl von $\boldsymbol{x}$ optimiert.

2.1.1 Minimierungsproblem

Ein Minimierungsproblem ist das mathematische Problem, welches den Funktionswert $f(\boldsymbol{x})$ unter allen $\boldsymbol{x}$ minimiert und den Nebenbedingungen der Form

$$\begin{aligned} \mathbf{a}(\boldsymbol{x}) = 0, \quad & a : \Re^{n_x} \rightarrow \Re^{n_a} \\ \mathbf{b}(\boldsymbol{x}) \leq 0, \quad & b : \Re^{n_x} \rightarrow \Re^{n_b} \\ & \boldsymbol{x} \in \Re^{n_x} \end{aligned}$$

genügt. Die reelle Funktion $\mathbf{a}(\boldsymbol{x})$ beschreibt die Gleichungsbedingungen, $b(\boldsymbol{x})$ die Ungleichungsbedingungen und $\Re^{n_x}$ ist der Bereich auf dem f, $\mathbf{a}$ und $\mathbf{b}$ als reelle Funktionen definiert sind. Jeder Vektor $\boldsymbol{x}$, der die Nebenbedingungen erfüllt, ist eine zulässige Lösung. Die Optimallösung des Problems ist diejenige zulässige Lösung, deren Funktionswert minimal ist.

2.1.2 Modellierungssprache

Die Aspekte der Modellbildung und das Lösen von Real-World Optimierungsproblemen bringen das formale Modellieren dieser Probleme unweigerlich mit sich. Zur Ver-

deutlichung soll ein systematischer Überblick über die aktuell präsenten Modellierungssprachen gegeben werden. In Kapitel 2 wird der Leser dann einen Einblick in die aktuelle Software erfahren, die benutzt wird um mathematische Optimierungsprobleme zu lösen. Die bekanntesten Modellierungssprachen [5] sind:

- *AIMMS* (Jan Bishops, Paragon Decision Technology B.V, Haarlem)
- *AMPL* (Bob Fourer, Northwestern University; David Gay, Bell-Laboratories)
- *GAMS* (Alex Meeraus, Michael Bussieck, Steven Dirkse, GAMS, Washington)
- *Mosel* (Yves Colombani, Susanne Heipcke, Dash Optimization, Blisworth, UK)
- *MPL* (Bjarni Kristjansson, Maximal Software, Arlington, VA)
- *NOP* (Arnold Neumaier, Vienna University)
- *OPL* (Gregory Glockner, ILOG Inc.).

Im Folgenden soll ein einfaches Beispiel der Modelliersprache AMPL[1] gezeigt werden, welches aus dem offiziellen AMPL Tutorial [7] stammt. Eine Firma stellt goldene ($PaintG$) und blaue Farbe ($PaintB$) her. Die blaue Farbe wird für 10$ verkauft und die goldene für 15$. Die Firma hat eine Fabrik und kann 40 Einheiten der Farbe Blau, aber nur 30 Einheiten der goldenen Farbe pro Stunde herstellen. Außerdem besteht die Einschränkung , dass man höchstens 860 Einheiten goldener Farbe und 1000 Einheiten blauer Farbe verkaufen kann. Wenn eine Woche 40 Stunden hat und die Farbe nicht gelagert werden kann, wieviel muss die Firma an blauer und goldener Farbe produzieren, damit der Gewinn maximal ist.

$$
\begin{aligned}
&max && 10\,PaintB + 15\,PaintG \\
&s.t. && \tfrac{1}{10}\,PaintB + \tfrac{1}{30}\,PaintG \leq 40 \\
& && 0 \leq PaintB \leq 1000 \\
& && 0 \leq PaintG \leq 860
\end{aligned}
$$

Abbildung 2.1 zeigt nun wie dieses Problem in AMPL Code umgesetzt wird. Der Code wir in dieser Form einem Solver übergeben..

```
1 var PaintB; # amount of blue
2 var PaintG; # amount of gold
3
4 maximize profit: 10*PaintB + 15*PaintG;
5 subject to time: (1/40)*PaintB + (1/30)*PaintG <= 40;
6 subject to blue_limit: 0 <= PaintB <= 1000;
7 subject to gold_limit: 0 <= PaintG <= 860;
```

Abbildung 2.1: AMPL Code eines Optimierungsproblems

[1] AMPL - A Modeling Language for Mathematical Programming

2.2 Blackbox - Optimierungsproblem

Optimierungsprobleme werden als Blackbox-Probleme [11] bezeichnet, wenn über die Teilfunktionen keine Informationen vorliegen und sie sich nicht analytisch beschreiben lassen. Im Besonderen existieren keine Angaben zu Definitions- und Wertebereich der Teilfunktionen.

Das mathematische Problem $f(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x}))$ wird im Folgenden als Blackbox Optimierungsproblem bezeichnet. Die Teilfunktion $\mathbf{y}(\boldsymbol{x})$ ist eine Blackbox und wird für die Auswertung der Funktion $f(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x}))$ während der Iterationsphase benötigt. Eine Blackbox kann ein lokales oder entferntes Programm sein. Meistens ist die Blackbox nicht lokal vorhanden und es muss eine Verbindung zur Blackbox hergestellt werden. Diese Verbindung kann ein Webservice sein, der per Email oder ein Webinterface angesprochen wird oder eine Person, die die Auswertung vor Ort durchführt.

Das Blackbox - Optimierungsproblem ist das mathematische Problem, welches den Funktionswert $f(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x}))$ unter allen $\boldsymbol{x}$ minimiert und den Nebenbedingungen der Form

$$\begin{aligned} \mathbf{a}(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x})) = 0, \quad & a : \Re^{n_x} \rightarrow \Re^{n_a} \\ \mathbf{b}(\boldsymbol{x}), \mathbf{y}(\boldsymbol{x})) \leq 0, \quad & b : \Re^{n_x} \rightarrow \Re^{n_b} \\ \mathbf{y}(\boldsymbol{x}) \in \Re^{n_y}, \quad & y : \Re^{n_x} \rightarrow \Re^{n_y} \\ & \boldsymbol{x} \in \Re^{n_x} \end{aligned}$$

genügt. Die reelle Funktion $\mathbf{a}(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x}))$ beschreibt die Gleichungsbedingungen, $\mathbf{b}(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x}))$ die Ungleichungsbedingungen und $\Re^{n_x}$ ist der Bereich auf dem f, $\mathbf{a}$ und $\mathbf{b}$ als reelle Funktionen definiert sind. Jeder Vektor $\boldsymbol{x}$ der die Nebenbedingungen erfüllt, ist eine zulässige Lösung. Die Optimallösung des Problems ist diejenige zulässige Lösung, deren Funktionswert minimal ist.

2.3 Anforderungen

Abbildung 2.2 zeigt die Grenzen bestehender Systeme. Das modellierte Blackbox - Optimierungsproblem wird dem Solver übergeben. Dieser startet die Berechnung und ruft die Blackbox $\mathbf{y}(\boldsymbol{x})$ auf. Diese liefert die notwendigen Teilauswertungen und schickt diese an den Solver zurück. Die roten Linien zeigen die aktuellen Probleme bestehender Systeme auf. (1) soll zeigen, dass es keine Modellsprache gibt, die ein Blackbox Problem so modellieren kann, dass der Solver Teilauswertungen auf entfernten Rechnern ausführen kann. (2) beschreibt die Voraussetzung, dass entweder der Solver auf die Blackbox gebracht werden muss oder die Blackbox in die Infrastruktur des Solvers integriert werden muss. Nun werden diese Ansätze analysiert und klar formuliert.

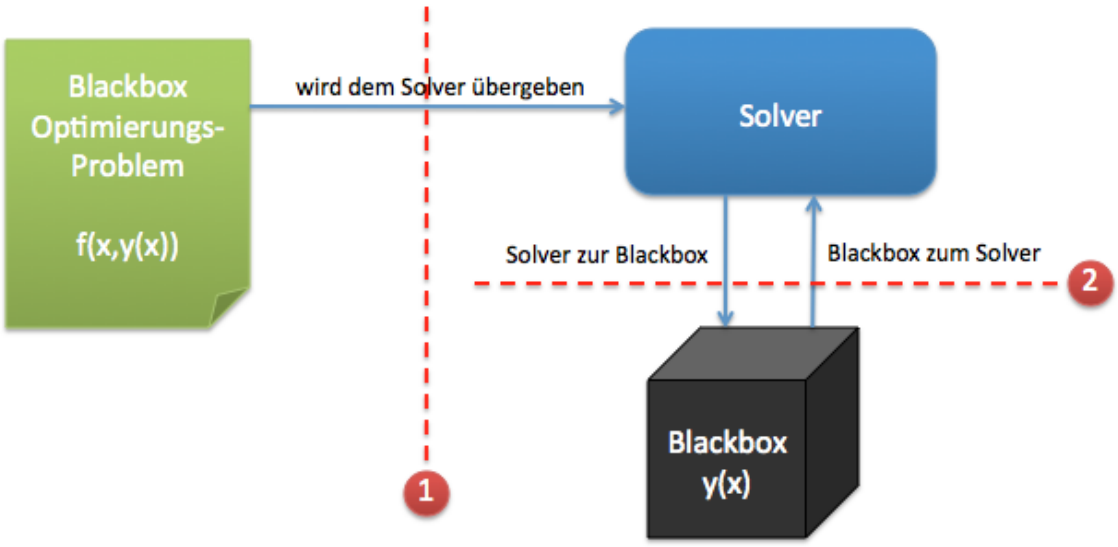

Abbildung 2.2: Grenzen bestehender Systeme

2.3.1 Modellierung eines Blackbox - Problems

Das erste Problem, das in Abbildung 2.2 beschrieben wurde, betrifft die Modellierung des Blackbox - Problems. Aktuelle Modellierungssprachen lassen es nicht zu Blackbox-Probleme zu modellieren, da die Anbindung von Blackboxen nicht möglich ist. Die Lösung besteht hier darin, eine mathematische Programmiersprache so zu modifizieren, so dass die Verbindung mit einer Blackbox während der Berechnung des Optimierungsproblems aufgebaut wird. Dabei soll auf der Blackbox ein Service gestartet werden, der Eingaben annimmt, die Auswertung durchführt und das Ergebnis an den Solver zurücksendet.

2.3.2 Anpassung des Solvers auf Blackbox - Probleme

An diesem Punkt gibt es zwei Ansätze zur Lösung von Blackbox - Problemen. Der erste Ansatz integriert die Blackbox in die Infrastruktur des Solvers. Bestehende Systeme müssen derart verändert werden, dass der Aufruf aus dem Solver heraus auf die Blackbox zugreifen kann. Die zweite Variante bringt den Solver zur Blackboxumgebung. Beide Varianten lösen das Problem der entfernten Kommunikation derart, dass sie lokal zusammengeführt werden. Das Ziel dieser Bachelorarbeit soll es jedoch sein, auf entfernte Blackboxen zugreifen zu können. Die rotgestrichelte Linie mit der Kennzeichnung 2 aus Abbildung 2.2 muss durch geeignete Verfahren zur Herstellung einer Verbindung zwischen den Komponenten überbrückt werden.

2.4 Marktüberblick

Zahlreiche Universitäten und Einrichtungen bieten Software zur Optimierung an. Beispiele für existierende Systeme sind COIN-OR [2], Sandia Acro [13] und Neos Server [14]. COIN-OR ist eine Open Source Community, die unter dem Projektnamen OS ein Set an Services erstellt hat, welche eine Kommunikation zwischen Clienten und Solvern mit Hilfe von Web Services herstellt. Es stehen Module und Bibliotheken für die Anbindung aller relevanten mathematischen Modellierungssprachen zur Verfügung. Sandia Acro stellt ebenfalls ein Optimierungs-Framework zur Verfügung, über das der Benutzer Zugriff zu Optimierungsbibliotheken und Solvern hat. Acro verwendet parallel mehrere Ressourcen um die Auswertung eines Optimierungsproblems schneller zu berechnen. Abbildung 2.3 gibt einen Überblick über die aktuellen Systeme und Methoden hinsichtlich einer Blackbox - Auswertung. Dabei sind folgende Merkmale relevant für den Vergleich: Lokale Installation,. Webinterface, Parallelität und Blackbox - Auswertungen. COIN-OR OS als Beispiel muss lokal installiert werden. Es bietet kein Webinterface, an das man die Probleme schicken kann. Weiterhin bietet es die Möglichkeit parallel Probleme zu bearbeiten, jedoch nicht die Möglichkeit Blackbox - Auswertungen einzubinden. Genauer betrachtet wird im Folgenden der Neos Server,

Typ / Art	Lokale Installation	Webinterface	Parallelität	Blackbox Auswertungen
COIN-OR OS	✓	✗	✓	✗
Sandia Acro	✓	✗	✓	✗
Neos Server	✗	✓	✓	✗
Blackbox und Solver auf einem System	✓	✗	✗	✓
Gemeinsames Dateisystem	✓	✗	✓	✓
Email	✗	✗	✗	✓
Ziel dieser Bachelorarbeit	✗	✓	✓	✓

Abbildung 2.3: Überblick existierender Ansätze

da dieser als einziger ein Webinterface bietet. Der Neos Server liefert mittels einer Schnittstelle einen Zugang zu diversen Optimierungsverfahren. Ohne die einzelnen Solver lokal installieren zu müssen, hat man mit diesem Tool die Möglichkeit Optimierungsprobleme via Internet zu senden und lösen zu lassen. Neos bietet die Möglichkeit

[2]COIN - COmputational INfrastructure for Operations Research [12]

Optimierungsprobleme via E-Mail zu schicken oder durch ein Webinterface einzugeben. Hauptsächlich unterscheidet Neos zwischen zwei Sprachen, AMPL und GAMS. Eine kurze Einleitung und ein Beispiel zu AMPL befindet sich unter Punkt 2.1.2. Es ist jedoch auch möglich in MATLAB oder anderen Modellierungssprachen geschriebene Probleme am Server lösen lassen. Durch die im Moment vorhandene Schnittstelle können jedoch nur AMPL und GAMS zum Server abgeschickt werden. Die Anbindung von Blackboxen in den drei vorgestellten Systemen ist nicht vorgesehen und fehlt im Moment noch. Es können somit direkt während der Optimierung keine Auswertungen einer Blackbox ohne Weiteres eingebunden werden. Um dies zu verdeutlichen werden nun die Möglichkeiten näher betrachtet, wie Probleme modelliert und an einen Server geschickt werden können. Zum einen kann dies so gelöst werden, dass Solver und Blackbox auf einem System gekoppelt sind. Man umgeht damit das Problem der entfernten Kommunikation. Steht jetzt die Blackbox entfernt, existiert eine weitere aufwendige Variante, das Senden der Eingabeparameter der Blackbox per Email an die Blackbox. Die lokale Auswertung auf der Blackbox wird durch eine Person manuell durchgeführt und per Mail zurückgesendet.

Mit den aktuellen Methoden und Werkzeugen ist ein verteiltes Lösen von Blackbox - Optimierungsproblemen noch nicht möglich, sondern nur ein auf die jeweiligen Probleme speziell angepasstes Verfahren würden dies zulassen. Damit ist aber eine leichte Austauschbarkeit von Solvern oder die Anwendung eines Solvers auf unterschiedliche Probleme immer mit sehr hohen Aufwand verbunden.

2.5 Konzept

In dieser Arbeit soll ein System entwickelt werden, welches sich in bestehende Optimierungsinfrastrukturen integriert sowie dem Benutzer eine intuitive Oberfläche zum Verwalten von mathematischen Optimierungsproblemen bietet. Der Unterschied zu bereits existierenden Systemen besteht darin, dass ein Optimierungsproblem Teilauswertungen benötigt, welche auf Blackboxen durchgeführt werden. Diese Blackboxen sollen mit Hilfe eines Webservices in das System integriert werden. Ziel hierbei ist es auch eine lose Kopplung zwischen den Bestandteilen des Systems zu gewährleisten, so dass die Auswahl an Modellierungssprachen und der Anbindung weiterer Blackboxen möglich ist. Weiterhin soll das Webinterface eine intuitive Verwaltung der Blackboxen unterstützen. Auf der Ebene der Programmierung soll gewährleistet werden, dass ein Optimierungsproblem 1 bis n Blackboxen ansprechen kann, um das schnellst verfügbare Ergebnis zu bekommen. Die Auslastung und Bearbeitungszeiten der Blackboxen spielen hier eine Rolle für die zukünftigen Auswertungen um eine optimale Verbindung zu einer Blackbox zu finden. Eine Kombination all der vorgestellten Funktionen und eine Automatisierung der Kommunikation stellt eine optimale Lösung und das Ziel dieser Bachelorarbeit da.

3 Grundlagen

In diesem Kapitel werden die verwendeten Grundlagen wie Software, Programmiersprachen, Hilfsmittel und Programmiertechniken näher beschrieben.

Aus der Problemstellung in Kapitel 2 ergibt sich, das dies effizient nur über eine bestehende Infrastruktur geht. Um dies zu erreichen müssen bekannte effiziente Tools angepasst werden. Eine leichte Ankopplung des Blackopt Systems an Blackboxen soll Voraussetzung für die Entwicklung sein. Dadurch ist es relevant darauf zu achten, dass eine Blackbox auf unterschiedlichen Betriebssystemen in unterschiedlichen Programmiersprachen implementiert sein kan. Ebenso müssen die Programme auf Seiten des Optimierers angebunden werden, hier ist ebenfalls darauf zu achten, dass der Optimierer in unterschiedlichen Sprachen und Laufumgebungen angekoppelt werden muss. Aus diesen Grund wurde mit Java eine universell einsetzbare Programmiersprache gewählt, die auf jedem Betriebssystem heutzutage läuft. Die nahe Verbindung zu Java erlaubt es JSP zu benutzen um das Webinterface zu modellieren. Für den Einsatz von JSP wird ein Apache Applikationsserver[1] verwendet, der die Webseiten darstellt und die Verbindung zwischen der in Java implementierten Logik und der JSP Webseiten herstellt. In Bezug auf die Datenverwaltung des Blackopt Systems wurde MySQL als Datenbanklösung gewählt. Um weiterhin eine einfache und schnelle Installation des Blackopt Systems zu gewährleisten wurde eine virtuelle Maschine in Virtual Box aufgesetzt. Das Betriebssystem der virtuellen Maschine ist eine Server Variante der Linux Distribution Ubuntu. Genaueres zur Ubuntu Server Installation befindet sich in Abschnitt 3.1.4 auf Seite 12.

3.1 Entwicklungsumgebung

Das Blackopt System wurde mit Hilfe der folgenden Software entwickelt:

- Eclipse 3.5 Galileo
- Java SE Development Kit (JDK) 6
- MySQL 5.1.37
- Apache Tomcat 6.0 Server
- Virtual Box 3.0.8
- Ubuntu Server 9.04

[1] Applikationsserver - ein Softwareframework zum effizienten Ausführen von Prozeduren

Die Entwicklungsumgebung Eclipse unterstützt das Erstellen von dynamischen Web Projekten mit JSP und Apache Tomcat. Das fertige Webprojekt wird dabei auf dem Server ausgeführt und ist unter einer voreingestellten Adresse aufrufbar. Innerhalb des Webprojektes kann Java Code implementiert werden, welcher mit Hilfe von Java Servlets mit dem JSP Code kommunizieren kann. Einzelheiten dazu findet man unter 3.2.4.

3.1.1 Eclipse

Eclipse ist ein freies Programmierwerkzeug zur Entwicklung von Software verschiedenster Art. Besonders gut geeignet ist es für die Programmiersprache Java. Eclipse unterstützt dynamische Webprojekte sowie eine einfach Anbindung an den Apache Tomcat Server. Die aktuelle Version nennt sich Galileo, trägt die Versionsnummer 3.5 und ist verfügbar unter http://www.eclipse.org/.

3.1.2 Apache Tomcat

Apache Tomcat ist eine Open Source Software Implementierung der Java Servlet und Java Server Pages (JSP) Technologien. Apache Tomcat stellt eine Umgebung zur Ausführung von Java-Code auf Webservern bereit. Es handelt sich um einen in Java geschriebenen Servlet-Container, der mithilfe des JSP-Compilers Jasper auch Java-Server Pages in Servlets übersetzen und ausführen kann. Dazu kommt ein kompletter HTTP-Server. Nähere Information findet man unter http://tomcat.apache.org/.

3.1.3 VirtualBox

Sun VirtualBox von Sun Microsystems ist eine Virtualisierungssoftware für Linux, Windows, Mac OS X. Als Gastsystem wird in unseren Fall eine Linuxdistribution installiert. Für die Installation von Ubuntu Server wird in VirtualBox eine neue dynamisch wachsende Partition angelegt(siehe dazu Hilfeseite von VirtualBox).

Das portable Image, das bei der Installation des Gastsystems erstellt wird, erlaubt es Ubuntu Server überall dort auszuführen, wo Virtualbox vorhanden ist.

Die aktuelle Version kann unter http://www.virtualbox.org/wiki/Downloads heruntergeladen werden.

3.1.4 Ubuntu Server

Die Ubuntu Server Edition kann auch ohne grafische Benutzeroberfläche installiert werden. Darüber hinaus bietet er optional eine integrierte, zeitsparende Installation

des häufig genutzten LAMP-Pakets (Linux, Apache, MySQL und PHP). Das LAMP Paket dient als Voraussetzung des Blackopt Systems. Die Ubuntu Server Installation wird durch einen Assistenten begleitet. Nachfolgend werden chronologisch aufgetretene Ereignisse und die jeweiligen Einstellungen erläutert.

Zuerst die Ubuntu Server 9.04 (x86) downloaden und als ISO Abbild in die virtuelle Maschine einbinden. Ubuntu wird unter http://www.ubuntu.com/GetUbuntu/download zur Verfügung gestellt.

Das automatische Booten vom DVD Laufwerk muss eingestellt sein (voreingestellt). Die Installation von Ubuntu Server startet. Man wählt als Sprache deutsch und als Land ebenfalls Deutschland.

Der nächste Schritt ist das Einrichten eines Netzwerkes. Folgende Einstellungen wurden hier getroffen.

- DNS Server: keine
- Rechnername: blackopt

Nun wird die Festplatte partitioniert. Hier wählt man die geführte Partition sowie gesamte Platte verwenden und LVM einrichten. LVM steht für Logical Volume Manager. Dieser fasst Festplatten bzw. Partitionen zu einem Volume zusammen, aus dem dynamisch "Partitionen" (die Logical Volumes, LV) erstellt werden.

Man wird nun aufgefordert Benutzer und Passwörter einzurichten.

- Name: Blackopt
- Benutzername: blackopt
- Passwort: born2opt
- Daten verschlüsseln: ja

Weitere Schritte:

- HTTP Proxy: nein
- Software Updates: keine automatischen Installationen

Nun kommt der wichtigste Schritt. Hier werden die nötigen Pakete installiert, die als Voraussetzung für den Blackopt Webservice dienen.

- Lamp
- PostgressSQL
- APACHE
- MySQL Root Passwort: born2opt

3.2 Verwendete Programmiersprachen

Die Wahl der Programmiersprache fiel relativ schnell auf Java, da diese eine objektorientierte Programmiersprache und die Programme sind plattformunabhängig, das heißt sie laufen in aller Regel ohne weitere Anpassungen auf verschiedenen Computern und Betriebssystemen, für die eine Java-VM existiert.

Die Verwendung von PHP und HTML als Programmiersprache für das Webinterface wurde frühzeitig verworfen, da bereits eine vorhandene Verbindung zwischen Java und Java Server Pages (JSP) ideal für die vorgegeben Zwecke ist. Java wird dabei den Part der Logikimplementierung übernehmen und Java Server Pages dient zur dynamischen Erzeugung von HTML- und XML-Ausgaben eines Webservers, der es erlaubt Java-Code und spezielle JSP-Aktionen in statischen Inhalt einzubetten. Dies entspricht dem Programmierprinzip des Model-View-Controllers. Das Datenmodell und die Darstellung der Daten werden möglichst unabhängig voneinander programmiert.

Weiterhin wird eine MySql Datenbank verwendet, in der alle Daten gespeichert werden, die zwischen den Komponenten des Blackopt Systems ausgetauscht werden. Der Zugriff zur MySql Datenbank kann aus Java heraus oder aus JSP geschehen.

3.2.1 Extensible Markup Language (XML)

Extensible Markup Language [15] , abgekürzt XML, ist eine Auszeichnungssprache, die es erlaubt hierarchisch strukturierte Daten in Form von Textdaten zu halten und deren Bedeutung zu beschreiben. Den meisten Einsatz findet Extensible Markup Language (XML) im Internet, um Datenaustausch zwischen den Computersystemen zu realisieren. Das World Wide Web Consortium (W3C) veröffentlichte am 10. Februar 1998 die erste Spezifikation dieser Metasprache. Die beiden Konzepte Elemente und Attribute bilden die Basis jeder Extensible Markup Language (XML) Datei. Der Anfang der Elemente wird mit $< .. >$ und ihr Ende mit $< /.. >$ gekennzeichnet, während die Attribute diese beschreiben. Bei einem Element kann es sich um ein einfaches oder ein komplexes Element handeln. Während die einfachen Elemente lediglich Datenwerte enthalten, sind die komplexen Elemente in der Lage auch andere Elemente verschachtelt in sich einzuschließen. Ein Extensible Markup Language (XML) Dokument ist wohlformiert. Sein Aufbau folgt einem bestimmten Schema. Die erste Zeile ist eine Deklaration, die auf die verwendete Version, sowie weitere Attribute hinweist, wie das in Abbildung 3.1 auf Seite 15 zu sehen ist. Anschließend folgt ein Root Element, dass alle anderen in sich einschließt.

```
<?xml version=\"1.0\" encoding=\"UTF?8\" standalone=\"yes\"?>
<verzeichnis >
    <titel>Automobilhersteller</titel>
    <eintrag>
        <name>VolksWagen</name>
        <beschreibung>VolksWagen ist ...</beschreibung>
    </eintrag>
    <eintrag>
        <name >Mercedes </name>
        <beschreibung>Mercedes ist ...</beschreibung>
    </eintrag>
</verzeichnis>
```

Abbildung 3.1: Beispiel XML

3.2.2 Document Object Model (DOM)

Um einen komfortablen Zugriff auf die Daten innerhalb von Extensible Markup Language (XML) Dokumenten zu erhalten, wurde das Document Object Model (DOM) [16] spezifiziert. Hierbei handelt es sich um ein sprach- und plattformunabhängiges Application Programming Interface (API). Es erlaubt den Programmen ganz im Sinne der objektorientierten Programmierung die Inhalte oder die Darstellung des Extensible Markup Language (XML) Baums zu manipulieren.

3.2.3 Java

Für den Fall einer Entwicklung eines Webservices fiel die Wahl relativ schnell auf Java. Was nun genau macht einen Webservice aus. Vereinfacht ausgedrückt stellt ein Anbieter über einen Webservice einen Dienst bereit, der von externen Anwendungen angesprochen werden kann. Über die Web Services Description Language (WSDL), einer XML-Spezifikation, wird angebundenen Anwendungen bekannt gegeben, welche Funktionen der Dienst genau bereitstellt. Die Kommunikation findet über das Simple Object Access Protocol (SOAP), einem XML-basierten Protokoll, statt. Im Falle des Blackopt Services dienen die Funktionen der Übertragung von Eingabedateien, die eine Blackbox konsumiert und der Ausgabedateien, die zurück an den Optimierungsserver gesendet werden. Der Vorteil dieser Gesamtkombination ist, dass die Dienste, als auch die sie benutzenden Anwendungen, in grundverschiedenen Programmiersprachen und Plattformen implementiert sein können.

3.2.4 Java Server Pages

JSP-Dateien sind HTML-Dateien, die eingebundenen Java Code enthalten. Somit können dynamische Inhalte erzeugt werden. Die Grundlage von JSP ist die J2EE-Spezifikation von Sun Microsystems, die von verschiedenen Applikationsservern, wie auch dem Apache Tomcat, implementiert wird. Apache Tomcat ist nur ein Programm, das JSP und Servlets ausführen kann.

Die Java Servlets dienen zur Steuerung und zum Aufrufen der Logik der Web-Anwendung. Servlets stellen Java-Programme dar, die auf einem Web-Server laufen. Sie erlauben dem Entwickler dynamische Web-Seiten mit Java zu erstellen.

Ein Servlet hat dabei folgende Aufgaben. Daten, die ein Benutzer beispielsweise in ein HTML-Formular auf einer Web-Seite eingegeben hat, werden gelesen und verarbeitet. Es wird Logik entweder direkt im Servlet ausgeführt oder eine andere Klasse aufgerufen, welche die Logik enthält oder die z.B. eine Datenbankabfrage durchführt. Der Zugriff auf die MySQL Datenbank wird mit einem MySQL Connector durchgeführt. Die Ergebnisse werden formatiert. Erwartet der Browser eine Antwort im HTML-Format, so müssen die Ergebnisse gemäß dem Standard formatiert werden. Das Zurücksenden des Dokuments an den Browser in dem Format, das in den Antwortparametern angekündigt wurde.

3.2.5 MySQL

Die Verwaltung der Daten im Blackopt System wird mittels einer MySQL Datenbank realisiert. Der Zugriff auf die Datenbank von Java und JSP ist bereits in vorhandenen Bibliotheken implementiert. Den aktuellen MySQL Treiber steht unter folgenden Link zum Download, `http://dev.mysql.com/downloads/connector/j/3.0.html`. Die Einbindung erfolgt über den Java Build Path.

3.3 Programmiertechniken

Da die Implementierung des Blackopt Systems auf Webservices ausgelegt sein wird, werden hier die Grundtechniken eines Webservices und dessen Kommunikation bei ein- und ausgehenden Verbindungen aufgezeigt. Die wichtigsten Begriffe sind zum einem WSDL eine Beschreibungssprache für den Webservice, welche unter Abschnitt 3.3.1 beschrieben wird und SOAP, ein Netzwerkprotokoll zur Übertragung von Daten unter Abschnitt 3.3.2.

3.3.1 Web Services Description Language (WSDL)

WSDL [17] beschreibt einen Web-Service bzgl. seines Interfaces und gehört demnach zu den Interface Definition Languages (IDL's). Dabei unterteilt man zum einen in abstrakte Beschreibungen und konkrete Beschreibungen. Zu den abstrakten Beschreibungen gehören die Datentypen, Daten und die Funktionen, die ein Webservice implementiert. Die konkreten Beschreibungen beinhalten Bindings, Ports und Services. Es werden im Wesentlichen die Operationen definiert, die von außen zugänglich sind, sowie die Parameter und Rückgabewerte dieser Operationen. WSDL spezifiziert somit lediglich die syntaktischen Elemente eines Web Services, d. h. die Art und Weise, wie ein Client auf den entsprechenden Web Service zugreifen kann.

```
<definitions>

// Abstrakter Part
        <types>definition of types</types>
        <message>definition of message</message>
        <portType>definition of interface
                <operation>
                        definition of operation
                </operation>
        </portType>

// Konkreter Part
        <binding>definition of binding</binding>
        <service>definition of endpoint
                <port>definition of location</port>
        </service>
</definitions>
```

Abbildung 3.2: WSDL - Struktur Beschreibung

Über das Messages Tag werden die eingehenden und ausgehenden Messages beschrieben. Es wird der Name und der Typ definiert. Der Port Type definiert die Adresse eines Bindings und stellt den Endpunkt einer Kommunikation dar. Ein Service kann mehrere Ports haben. Das Binding definiert das Message Format und beschreibt die Protokoll Details. Die Spezifikation beinhaltet Bindings für SOAP, HTTP und MIME.

3.3.2 Simple Object Access Protocol (SOAP)

SOAP [18] ist ein Netzwerkprotokoll, das auf XML basiert und Daten zwischen Systemen austauscht. Außerdem können so genannte Remote Procedure Calls [2] ausgeführt

[2]Remote Procedure Call - Aufruf von Funktionen in entfernten Adressräumen

werden. SOAP regelt, wie Daten zu übertragen und interpretieren sind und gibt eine Konvention für entfernte Aufrufe vor. Die Struktur einer SOAP Nachricht besteht aus einem "Envelope", einen Element dem ein lokaler Name zugewiesen wird. Es dient als Hülle für die gesamte Nachricht. Im Header können Meta-Informationen, beispielsweise zum Routing, zur Verschlüsselung oder zu Transaktionsidentifizierung, untergebracht werden. Im Body-Element sind die eigentlichen Nutzdaten untergebracht. SOAP wird beispielsweise von eBay und Amazon zur Datenbankabfrage für Suchanfragen genutzt.

SOAP Webservice

Prinzipiell kann man in Java jede Klasse als Webservice exportieren. Alles was man dazu tun muss, sind Java Annotations zu verwenden, die die jeweilige Klasse als Webservice deklariert.

```
1 import javax.jws.WebService;
2 import javax.jws.soap.SOAPBinding;
3 import javax.jws.soap.SOAPBinding.Style;
4
5 @WebService
6 @SOAPBinding(style=Style.RPC)
```

Abbildung 3.3: Java Annotations für einen Webservice

Über die Angabe *@SOAPBinding(style=Style.RPC)* geben wir an, dass diese Klasse per SOAP-Protokoll gebunden werden soll und die Kommunikation prozedurorientiert (RPC = Remote Procedure Call) stattfindet. Das zeigt an, dass Aufrufe Parameter enthalten und Werte zurückgegeben werden.

Webservice Server

Um eine Klasse von Java nach außen zu veröffentlichen muss man einen Webservice Server implementieren. Dies geschieht über das Einbinden eines integrierten HTTP Servers. Folgender Beispielcode zeigt einen einfachen Server.

Endpoint.publish("http://localhost:8080/example", server)
gibt die Adresse an, unter welcher der Webservice erreichbar ist. Die Klasse Example ist diesem Fall ein Webservice, der aus einer Java Klasse heraus erstellt wurde.

Ruft man nun *http://localhost:8080/example?wsdl*
im Browser auf, bekommt man eine WSDL-Datei angezeigt mit allen Definitionen des Webservices.

```
1 import javax.xml.ws.Endpoint;
2
3 public class Server {
4   public static void main (String args[]) {
5     Example server = new Example();
6     Endpoint endpoint =
7       Endpoint.publish(''http://localhost:8080/example'', server);
8   }
```

Abbildung 3.4: Beispielimplementierung eines Webservice Servers

Webservice Client

Für den Zugriff des Webservices benötigt man einen Client. Diesen Client können wir mittels der publizierten WSDL-Datei automatisch generieren lassen. Dazu ruft man in der Shell eines neuen Java Projekts folgenden Befehl auf: *wsimport -keep http://localhost:8080/example?wsdl*.

Über den keep-Parameter werden die generierten Java-Klassen gespeichert und nicht nach dem Kompilieren gelöscht. Wenn der Befehl fehlerlos durchgelaufen ist findet man folgende vier generierten, bzw. kompilierten, Dateien vor:

- Example.java
- Example.class
- ExampleService.java
- ExampleService.class

Der aus der WSDL generierte ExampleService übernimmt die Kommunikation zum Server. Eine Instanz auf unseren Example Webservice erhalten wir über den Aufruf von getExamplePort(), welches wir anschließend für Zugriffe auf unsere Klasse nutzen.

Der Client sieht wie folgt aus:

```
1 public class ExampleClient {
2     public static void main(String args[]) {
3         ExampleService service = new ExampleService();
4          Example example = service.getExamplePort();
5     }
6 }
```

Abbildung 3.5: Implementierung eines Webservice Clients

Nun kann man alle gewünschten Methoden auf der Instanz *example* aufrufen und erhält die Ergebnisse direkt zurück, wie bei einem lokalen Methodenaufruf.

3.4 Anwendungsfälle

Ein Anwendungsfall beschreibt eine Interaktion zwischen einer Person und dem System. Die in Abbildung 3.6 dargestellten Anwendungsfälle demonstrieren alle Abläufe, die das zukünftige System durch seine Architektur unterstützen soll. Wir gehen davon aus, dass das System von einem Benutzer bedient wird, der alle Schritte zum Erstellen eines Jobs durchführt und das System überwacht. Deshalb ist er im Rahmen der nachfolgenden Anwendungsfälle der einzige Akteur.

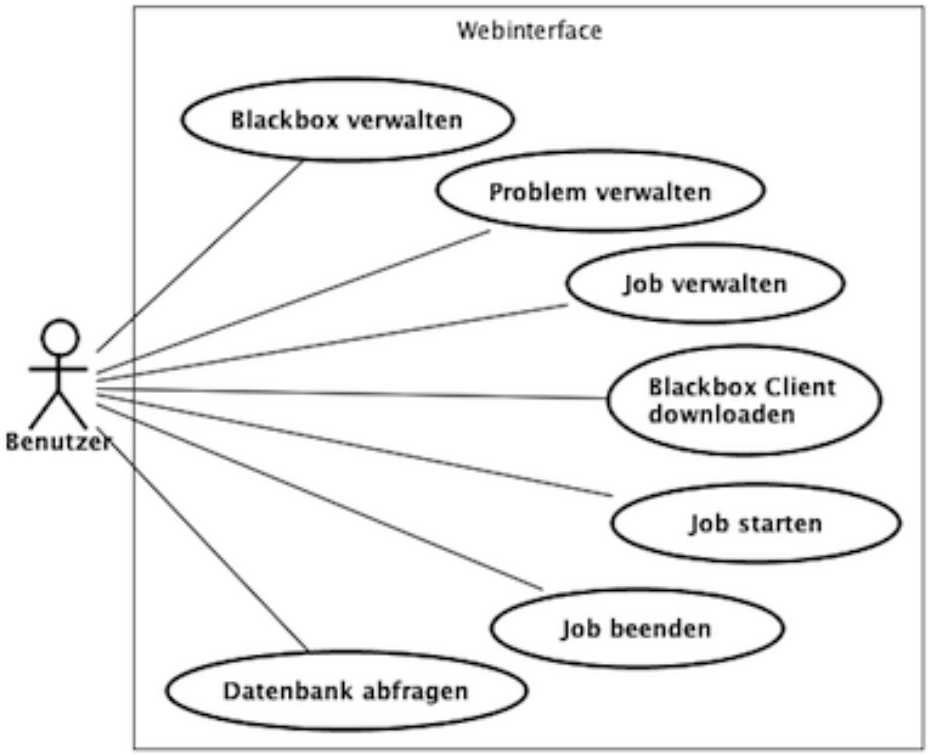

Abbildung 3.6: Diagramm der Anwendungsfälle

3.4.1 Datenverwaltung

Der Aktionsradius des Benutzer beschränkt sich auf die Verwaltung der Daten des Blackopt Systems und der Durchführung eines Jobs. In diesem Abschnitt wird gezeigt, welche Aktionen der Benutzer auf den Datenstrukturen ausführen kann.

Blackbox verwalten

Der Benutzer des Systems möchte eine neue Blackbox hinzufügen, eine bestehende Blackbox einsehen, editieren oder löschen. Beim Hinzufügen oder Editieren einer neuen Blackbox werden spezifische Blackbox Daten an das System übergeben und in die Datenbank eingetragen.

Akteur(en): Benutzer

Vorbedingung(en): Das System wurde gestartet und es besteht eine Verbindung zur MySQL Datenbank.

Nachbedingung(en): Das System kennt die neu eingetragene Blackbox und kann auf diese zugreifen. Der Download des Blackbox Programms ist nun möglich. Weiterhin kann nun ein Problem erstellt werden. Wird eine Blackbox gelöscht, ist diese nicht mehr in der Datenbank verfügbar. Löschen ist nur möglich, wenn die Blackbox nicht von einem Problem verwendet wird.

Problem verwalten

Der Benutzer des Systems möchte ein neues Problem hinzufügen, eine bestehendes Problem einsehen, editieren oder löschen. Beim Hinzufügen oder Editieren eines neuen Problems werden spezifische Problem Daten an das System übergeben und in die Datenbank eingetragen.

Akteur(en): Benutzer

Vorbedingung(en): Das System wurde gestartet und es besteht eine Verbindung zur MySQL Datenbank.

Nachbedingung(en): Das System kennt das neu eingetragene Problem und kann auf dieses zugreifen. Das Problem kann nun mit Blackboxen verknüpft werden. Weiterhin kann nun ein Job erstellt werden. Wird ein Problem gelöscht, ist diese nicht mehr in der Datenbank verfügbar. Löschen ist nur möglich, wenn das Problem nicht von einem Job verwendet wird.

Job verwalten

Der Benutzer des Systems möchte einen neuen Job hinzufügen, einen bestehenden Job einsehen, editieren oder löschen. Beim Hinzufügen oder Editieren eines neuen Jobs werden spezifische Job

Daten an das System übergeben und in die Datenbank eingetragen.

Akteur(en): Benutzer

Vorbedingung(en): Das System wurde gestartet und es besteht eine Verbindung zur MySQL Datenbank. Es wurde eine Problem erstellt, welches optional eine Blackbox verwendet.

Nachbedingung(en): Das System kennt den neu eigetragenen Job und kann auf diesen zugreifen. Der Job kann nun gestartet werden.

3.4.2 Durchführung eines Jobs

Der zweite Abschnitt der Anwendungsfälle beschreibt die Schritte zur Durchführung eines Jobs.

Blackbox Client downloaden

Nachdem eine Blackbox erstellt wurde, kann der Benutzer die Software für die Blackbox herunterladen. Inhalt dieser Software ist ein Service, der Anfragen annimmt und die Ergebnisse der Auswertung zurückschickt an den Server.

Akteur(en): Benutzer

Vorbedingung(en): Es besteht eine Verbindung zur MySQL Datenbank. Eine Blackbox wurde erstellt.

Nachbedingung(en): Der Benutzer kann nun die Software auf dem entfernten Rechner installieren.

Job starten

Der im System existierende Job wird durch den Benutzer durch einen Aufruf von der Seite des Optimierers aus gestartet.

Akteur(en): Benutzer

Vorbedingung(en): Es besteht eine Verbindung zur MySQL Datenbank und ein Job wurde erstellt. Der Optimierer läuft in der Iterationsphase und sendet eine Anfrage an das Blackopt System. Die Anfrage startet den Job innerhalb des Systems und sendet die Daten an eine passende Blackbox.

Nachbedingung(en): Der Job ist gestartet und das Blackopt System wartet darauf, dass die Auswertungen der Blackbox zurückgeschickt werden.

Job beenden

Der Benutzer beendet einen Job abrupt ohne weitere Ergebnisse zu erwarten. Das Blackopt System sendet einen Stopp Befehl an die Blackbox und beendet die Ausführung des lokalen Programms. Die Blackbox ist wieder frei für andere Anfragen.

Akteur(en): Benutzer

Vorbedingung(en): Ein Job wurde gestartet.

Nachbedingung(en): Der Job ist beendet und es werden keine Daten mehr zwischen Blackbox und System ausgetauscht.

Datenbank abfragen

Der Benutzer möchte alle Eingaben und die dazugehörigen Auswertungen der Blackbox bezüglich eines Jobs einsehen.

Akteur(en): Benutzer

Vorbedingung(en): Im System ist ein Job vorhanden. Außerdem sind die Ergebnisse der Blackbox Auswertungen in der Datenbank gespeichert.

Nachbedingung(en): Die job-spezifischen Ergebnisse werden ausgegeben.

4 Umsetzung

In diesem Kapitel werden die Bestandteile des Blackopt Systems exakt beschrieben. Dabei wird die Architektur des Systems aufgezeigt und auf das Zusammenspiel der einzelnen Komponenten eingegangen.

Die Software besteht aus vier Teilen, die unabhängig voneinander ausgeführt werden:

- Blackopt Webservice
- Blackopt Webinterface
- Blackbox Client
- Blackopt Matlab Connector

Der Blackopt Webservice ist der Kernbestandteil des Blackopt Systems. Hier werden alle eingehenden und ausgehenden Verbindungen verwaltet. Auf der einen Seite ist der Server mit den Blackbox Clienten verbunden und auf der anderen Seite mit dem Optimierungsserver. Er bildet somit die Schnittstelle für die entfernte Optimierung. Abschnitt 4.2 erläutert den Aufbau des Webservices auf programmiertechnischer Ebene. Alle Bestandteile des Webservices werden auf ihre Funktionsweise und Zweck untersucht. Der Blackopt Matlab Connector bietet dem Optimierungsserver die Möglichkeit auf den Blackopt Webservice zuzugreifen.

4.1 Systemüberblick

Abbildung 4.1 zeigt einen Einblick in das Blackopt System. Das Zusammenspiel der einzelnen Komponenten wird anhand der Pfeile erläutert. Die vier Bestandteile sind farblich markiert, lediglich die zum Blackopt Webservice gehörende Datenbank ist zusätzlich farblich aufgezeigt. Der User hat die Möglichkeit eine Blackbox, ein Problem oder einen Job zu verwalten. Die Pfeile zwischen Blackopt - Webservice und dem Optimierungsserver sowie den Blackboxen zeigen den Datenverlauf. Wird ein Job gestartet, findet folgender Ablauf statt. Der Optimierer sendet eine Anfrage an den Blackopt Service. Falls in der Datenbank bereits eine passende Auswertung der Blackbox zur Verfügung steht, wird das Ergebnis direkt an den Server zurückgeschickt. Existiert kein passender Eintrag in der Datenbank, wird die Anfrage zur ausgewählten Blackbox weitergeleitet. Sobald die Blackbox mit der Auswertung fertig ist, wird das Ergebnis an den Blackopt Service zurückgesendet, in der Datenbank gespeichert und an den Optimierer weitergeleitet.

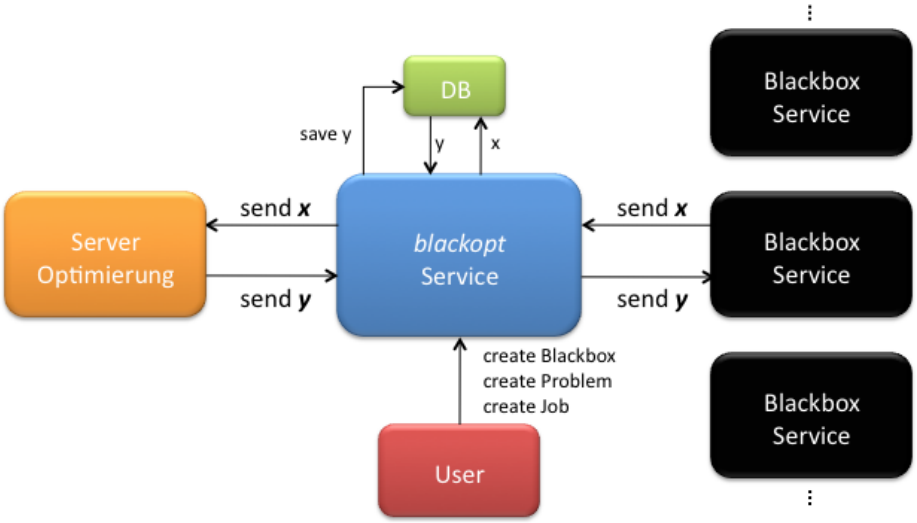

Abbildung 4.1: Blackopt Systemüberblick

4.2 Blackopt Webservice

Der Blackopt Webservice in Abbildung 4.2 besteht aus Receiver und jeweils einem Delivery Client zur Blackbox und zum Optimierungsserver. Der Receiver dient zum Empfang der Daten von Optimierungsserver und Blackbox, der Delivery Client zum Senden an die jeweiligen Komponenten.

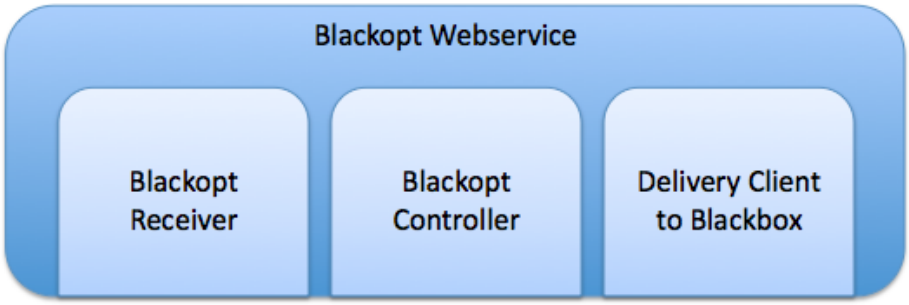

Abbildung 4.2: Blackopt Webservice

4.2.1 Blackopt Receiver

Der Blackopt Receiver ist die zentrale Kontrolleinheit über alle Anfragen, die den Blackopt Webservice erreichen. Der Receiver ist multithreadfähig. Es wird für jede eingehende Verbindung eine eigene Receive-Routine gestartet. Im folgenden werden die wichtigsten Klassen und deren Interaktionen beschrieben.

- **BoReceiveRoutine.java**
  Initiiert einen neuen Thread zum Empfangen der Daten
- **BoReceiveRoutineThread.java**
  Tatsächlich aufgerufene Thread Klasse. Bei jeder Anfrage wird ein neuer Thread gestartet um parallelen Zugriff zu gewährleisten
- **ConfigBoReceiver.java**
  Beinhaltet die Konfiguration des Receivers (IP-Adresse und Port)
- **StartBoReceiver.java**
  Der Receiver wird unter der Adresse gestartet, die in der Konfigurationsdatei angegeben ist und wartet auf Anfragen

BoReceiveRoutine

Die BoReceiveRoutine wird entweder durch den Optimierungsserver oder die Blackbox gestartet. Da die jeweiligen Daten unterschiedlich sind, die gesendet werden, muss zwischen Optimierer und Blackbox unterschieden werden.

Anfrage Optimierer an Blackopt Webservice

Sendet der Optimierer eine Anfrage an den Webservice, wird die Methode *initOpt(String text, int jobid)* aufgerufen. Die Eingabeparameter sind zum einen ein String der die Konfigurationsdatei für die Blackbox beinhaltet, sowie die JobID, um die Konfigurationsdatei dem Job zuzuordnen.

Nach dem Aufruf dieser Methode werden auf Seite des Blackopt Werbservices alle Voraussetzungen getroffen, um die Anfrage zu bearbeiten und an eine Blackbox weiterzuleiten. Dazu wird ein neuer BoReceiveRoutineThread gestartet. Dieser wird später noch näher erläutert.

Anfrage Blackbox an Blackopt Webservice

Sendet eine Blackbox Daten an den Blackopt Webservice, wird die Methode *init(QueueObject qObj)* von derBlackbox aufgerufen. Der Inhalt ist ein QueueObject. Dieses QueueObject beinhaltet alle Daten einer Auswertung und wird im nächsten Unterpunkt näher erläutert.

Weitere Methoden, die vom Optimierer aufgerufen werden, sind *getY(int jobid, int version)* und *isJobReady(int jobid, int version)*. Letztere Abfrage gibt "true" zurück, falls eine Anfrage auf der Blackbox abgearbeitet wurde und ein Ergebnis vorliegt. Dieses Ergebnis kann mit der Funktion *getY(int jobid, int version)* abgerufen werden.

Queue

Die Queue dient zur Organisation der eingehenden Anfragen. Damit diese Anfragen korrekt abgearbeitet werden und es nicht zu inkonsistenten Daten kommt, werden alle Anfragen in einer Queue gespeichert. Diese Queue beinhaltet speziell angepasste QueueObjects.

Die komplexe Datenstruktur wird durch die folgenden Eigenschaften näher beschrieben. In einem QueueObject werden folgende Informationen gespeichert:

- **inputFile** Konfigurationsdatei gesendet vom Optimierer
- **outputFile** Ausgabedatei gesendet von der Blackbox
- **version** Aktuelle Version der Auswertung
- **jobid** Job ID
- **status** In Bearbeitung auf einer Blackbox (Eintrag auf "active" gesetzt) oder auf keiner Blackbox (Eintrag auf "inactive")

BoReceiveRoutineThread

Jede Anfrage erzeugt einen neuen Thread. Zuerst wird überprüft, von welcher Komponente die Anfrage kam, entweder vom Optimierer oder der Blackbox. Die passende Routine zum Speichern der Daten wurde in Abschnitt 4.2.1 erläutert. Nun wird der Inhalt Anfrage dem passenden Job zugeordnet und lokal in der Datenbank gespeichert. Kommt die Anfrage vom Optimierer, wird diese an den Blackopt Controller weitergeleitet. Dieser kümmert sich um das Senden der Daten an die Blackbox. Kommt die Anfrage von einer Blackbox, dann wird das QueueObject vervollständigt und der Status auf inaktiv gesetzt. Der Optimierer kann nun beim nächsten Abholversuch das Ergebnis abrufen.

4.2.2 Blackopt Controller

Um viele Anfragen parallel abarbeiten zu können, ist es wichtig eine Kontrolleinheit zu haben, die threadfähig ist und die Anfragen an die Blackboxen koordiniert.

Der Blackopt Controller ist zuständig für die Vorbereitung der Dateiübertragung an die Blackbox. Zuerst wird eine Verbindung zur Blackopt MySQL Datenbank hergestellt um alle zur JobID gehörenden Blackboxen zu ermitteln und in einer Liste zu speichern.

Da alle Blackboxen in dieser Liste die gleiche Auswertung durchführen, wird lediglich zu einer Blackbox aus der Blackboxliste die Anfrage gesendet (die Blackbox wird

anhand der Auswertungszeit der vergangenen Anfragen ermittelt).

Die letzte Aufgabe des Controllers ist das Starten eines Delivery Clients, der die Anfrage an die zuvor ausgewählte Blackbox weiterleitet.

4.2.3 Delivery Client to Blackbox

Der Delivery Client sendet die Anfrage an die Blackbox. Die Blackbox benutzt hierzu einen eigenen kleinen Webservice, welcher auf Anfragen des Delivery Clients reagiert.

Der Delivery Client besteht aus folgenden drei Bestandteilen:

- Deliver Service
- Deliver Routine
- Start Delivery Client

Deliver Service

Hier wird die Verbindung zum Blackbox Webservice hergestellt. Dabei werden alle nötigen Schritte der notwendigen Informationen zum Zugriff auf den WSDL Service vorbereitet.

Deliver Routine

Die Deliver Routine an sich definiert lediglich die Operationen, die von außen zugänglich sind, sowie die Parameter und Rückgabewerte dieser Operationen. Der Blackopt Webservice greift hierdurch auf die Methoden des Blackbox Clients zu.

Start Delivery Client

Um den Delivery Client zu Starten muss diesem eine Blackbox zugewiesen werden. Die IP-Adresse, sowie der Port können aus dem Blackbox Datenmodel ausgelesen werden. Ein neuer Deliver Service wird gestartet und die Daten an die Blackbox gesendet. Die Übertragung sollte bei bestehender Verbindung zur Blackbox innerhalb weniger Sekunden durchgeführt sein.

4.2.4 Blackopt Webservice Interaktion

Schaubild 4.3 zeigt die Interaktion zwischen den Bestandteilen des Blackopt Systems auf. Auf Seite des Optimierungsservers wird eine Anfrage an das Blackopt System

gesendet. Diese Anfrage beinhaltet eine JobID und die Eingabewerte in Form eines Strings. Der Blackopt Receiver nimmt diese Daten an und erzeugt ein neues Queue-Object. Das nächste zu bearbeitende Objekt wird aus der Queue entfernt und wird mittels des Blackopt Controllers an den Delivery Service weitergereicht. Dort wird die Anfrage an die zuvor durch den Controller ausgewählte Blackbox gesendet.

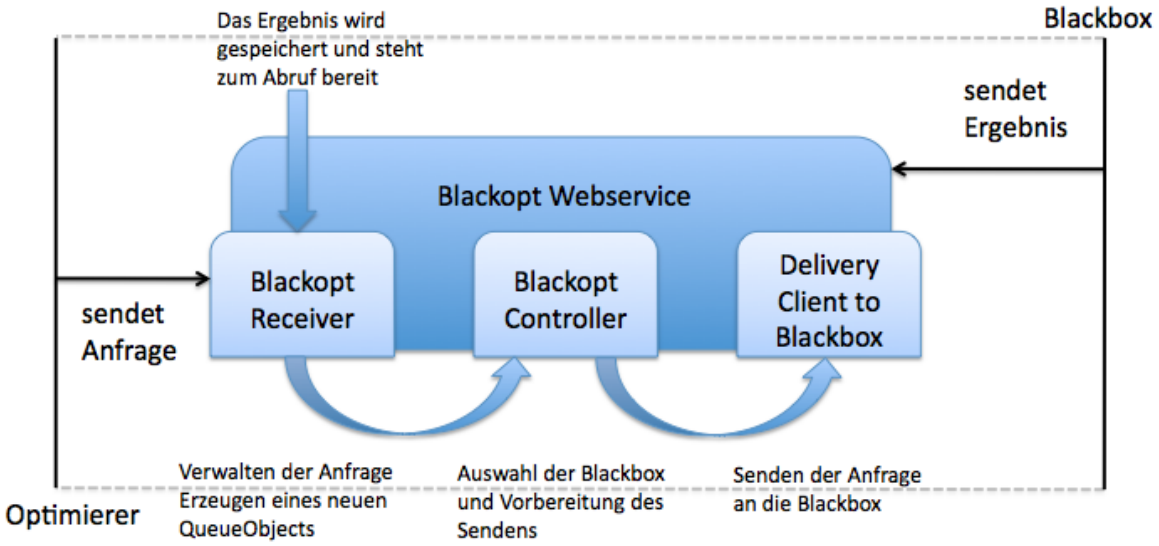

Abbildung 4.3: Blackopt Webservice Interaktion

Die Blackbox wertet nun die Eingabe aus und sendet ihr Ergebnis zurück an den Blackopt Webservice. Die detaillierte Funktion des Blackbox Clients wird in 4.3 erklärt. Der Blackopt Receiver erkennt, dass Daten ankommen, speichert diese ab und ändert den Status des Jobs auf "inactive". Durch die Statusänderung bemerkt der Optimierungsserver [1], dass die Ergebnisse zur Verfügung stehen und die Ausgabewerte abgeholt werden können.

4.3 Blackbox Client

Der Aufbau der Blackbox ist ähnlich dem des Blackopt Webservices. Der Blackbox Client ist ein Webservice, der Anfragen vom Blackopt Webservice erhalten kann. Die Kommunikation wird über den Blackbox Receiver sowie über den Delivery Client zum Blackopt Webservice abgehandelt. Dabei dient der Receiver als eigenständiger Webservice, der auf Eingaben wartet. Die zentrale Einheit hier bildet der Blackbox Executor. Dieser startet das lokale Programm, übergibt die Eingabedatei und sorgt dafür, dass die Ausgabedatei an den Delivery Client übergeben wird. In der folgenden Abbildung 4.4 ist die Struktur des Blackbox Clients zu sehen.

[1] Methode *isJobReady(int jobid)* wird in Abschnitt 4.2.1 beschrieben

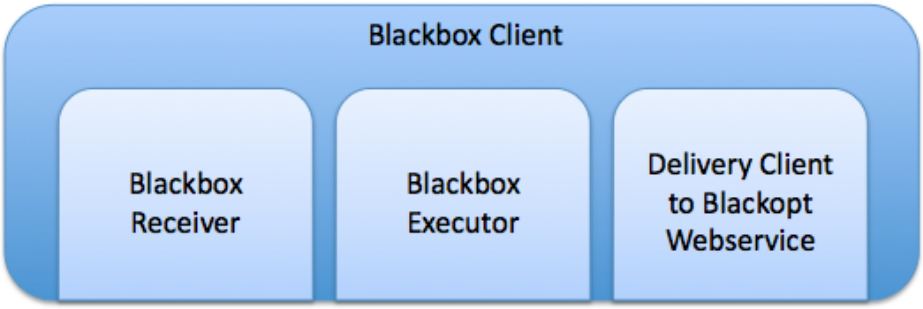

Abbildung 4.4: Blackbox Client

4.3.1 Blackbox Receiver

Nach dem Start des Blackbox Receivers, läuft dieser als Webservice und wartet auf Anfragen. Erhält der Receiver eine Anfrage, wird der Status der Blackbox auf belegt gesetzt. Die Blackbox ist nun für alle anderen Anfragen gesperrt. Dies ist wichtig für den Controller des Blackopt Webservices, der mittels den Statusmitteilungen die Jobs an die zugehörigen Blackboxen koordiniert. Somit wird außerdem gewährleistet, dass es nicht zu falschen Ergebnissen kommt, da die Blackbox für weitere Anfragen gesperrt ist. Der Receiver startet eine Receive-Routine, mit welcher der Eingabestring lokal in einer Textdatei abgesichert wird. Der Receiver besteht aus den folgenden Klassen:

- **ReceiveRoutine.java**
  Initiiert einen neuen Thread
- **ReceiveRoutineThread.java**
  Bei jeder Anfrage wird ein Thread gestartet, die Blackbox ist nun aktiv und für weitere Anfragen gesperrt
- **StartReceiver.java**
  Nach dem Start ist der Receiver für den Empfang von Eingabedateien bereit

ReceiveRoutine

Die Receive Routine beinhaltet die Methode *init(int jobid, int version, String inputFile)*. Diese Methode wird vom Blackopt Webservice aufgerufen, wenn eine Anfrage an die Blackbox gesendet wird. Im Folgenden wird der ReceiveRoutineThread beschrieben, der im Laufe einer Ausführung auf dem Blackbox Client gestartet wird. Weiterhin hat der Blackopt Webservice die Möglichkeit über zwei weitere Methoden die aktuellen Status der Blackbox *getStatus()* und die Verfügbarkeit der Blackbox *isAvailable()* zu überprüfen.

ReceiveRoutineThread

Durch die ReceiveRoutine initiiert, führt dieser Thread folgende Schritte aus. Die Eingabedatei wird lokal gespeichert und eine Instanz des Executors wird gestartet. Zu Statistikzwecken und zur Koordination für den Blackopt Controller [2] wird die Zeit gemessen, die die Ausführung vom Start bis zum Ende benötigt. Anhand dieser Roundtriptime wird die Blackbox eingestuft. Der Blackopt Controller benutzt diese Messungen um die schnellste Blackbox für die Auswertung auszuwählen. Die Ergebnisse werden mit den Daten der Auswertung an den Blackopt Webservice zurückgeschickt. Zukünftige Anfragen können somit durch den Blackopt Webservice effizienter koordiniert werden. Dies ist dann hilfreich, wenn mehrere Blackboxen die gleiche Arbeit verrichten und die schnellste ausgewählt werden soll. Nachdem die Auswertung durch den Executor beendet ist und die Ergebnisse vorliegen, werden diese mit der Hilfe eines String Readers eingelesen. Der String Reader ist eine Hilfsklasse, die es erlaubt Textdateien einzulesen. Zum Schluss wird ein Delivery Client gestartet, der die Ausgabedatei, die Statistiken sowie weitere Informationen des Jobs zurück an den Blackopt Webservice schickt.

StartReceiver

Der Receiver, der zum Empfang der Daten vom Blackopt Webservice dient, wird mittels dieser Klasse gestartet. Wichtig hierbei ist folgender Aufruf:

Endpoint.publish(address, server);

address ist der Eingabeparameter unter der Blackbox Receiver erreichbar ist. Eine beispielhafte Eingabe könnte folgende Adresse sein: *http://localhost:8088/blackboxreceiver*. Der Receiver wäre nun unter der IP-Adresse des localhost und dem Port 8088 abrufbar.

4.3.2 Blackbox Executor

Um das lokale Programm auf der Blackbox zu Starten, wurde der Executor entwickelt. Der Executor der Blackbox verwaltet die Ausführung des lokalen Programmaufrufs. Dabei wird die Konfiguration des Programmaufrufs durch die Parameter vorgenommen, die zuvor beim Erstellen der Blackbox im Webinterface deklariert wurden. Die Bestandteile sind wie folgt:

Client

Die Client Instanz wird beim ersten Starten der Blackbox initialisiert und liegt auf Abruf bereit. Die Klasse Client wurde mit dem Singleton Pattern programmiert, wel-

[2] Blackopt Controller - Steuereinheit bzgl. der Zuteilung von Blackboxen zu einem Job, siehe Abschnitt 4.2.2

- **Client.java**
  Startet den Executor Client
- **Configuration.java**
  Beinhaltet alle Informationen, die die Blackbox zur Ausführung benötigt
- **XmlReader.java**
  Wird benötigt um die Konfigurationsdatei einzulesen
- **ExecutorRoutine.java**
  Die Ausführung geschieht in dieser Klasse

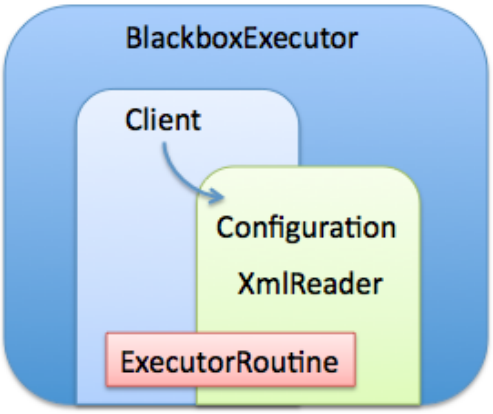

Abbildung 4.5: Blackbox Executor

ches gewährleistet, dass nur eine Instanz der Klasse Client ausgeführt wird. Wird die Client Instanz mit der Methode *start()* aufgerufen, beginnt die Ausführung innerhalb der Blackbox. Die aktuellen Konfigurationseinstellungen werden aus der Konfigurationsdatei geladen und die Executor Routine wird gestartet.

Configuration

Der Aufbau des Konfigurationsmodells sieht folgendermaßen aus.

- absolutePath: lokaler Pfad des auszuführenden Programms auf der Blackbox
- shellCommand: Shell Aufruf, mit dem das Programm gestartet wird, unterscheidet sich möglicherweise bei verschiedenen Betriebssystemen
- dataName: Datei Präfix der Ein- und Ausgabedatei
- bbid: Die Blackbox ID der Blackbox
- receiverAdress: Die Adresse der Blackbox
- os: Das Betriebssystem unter der die Blackbox läuft

Diese Werte werden mittels des XML Readers aus einer Konfigurationsdatei ausgelesen, die bei der Installation des Blackbox Clients auf der Blackbox hinterleget wurde. Die Konfigurationsdatei kann im Webinterface unter dem Eintrag der jeweiligen Blackbox runtergeladen werden.

XMLReader

Die Konfigurationsdatei liegt im XML Format vor. Abbildung 4.6 beschreibt eine Blackbox mit den Tags, die in der Konfiguration erklärt worden sind. Die Konfigurationsdatei wird durch den Blackopt Webservice erstellt, nachdem eine neue Blackbox

```
<Blackbox>
    <ReceiverAdress>Beispieladresse</ReceiverAdress>
    <AbsolutePath>Beispielpfad</AbsolutePath>
    <ShellCommand>Beispielcommand</ShellCommand>
    <Bbid>Beispiel ID</Bbid>
    <DataName>Beispielname</DataName>
</Blackbox>
```

Abbildung 4.6: XML Konfigurationsdatei des Blackbox Executors

ins System eingegeben wurde.

ExecutorRoutine

Die Executor Routine startet das lokale Programm mit den Befehlen aus der Konfigurationsdatei. Dabei unterscheidet sich die Ausführung unter unterschiedlichen Betriebssystemen.

Ein Programm unter Windows wird beispielsweise mit folgendem Befehl aufgerufen.

- Windows: Runtime.getRuntime().exec(new String[]"My Application.exe");

Die Klasse Runtime besitzt die statische Methode getRuntime(). Diese Methode liefert als Ergebnis das Runtime-Exemplar der virtuellen Maschine.

Hier die weiteren Befehle für Mac OS X und Linux.

- Mac OS X: Runtime.getRuntime().exec(new String[]"open", "My Application.app");
- Linux: Runtime.getRuntime().exec(new String[]"My Application");

4.3.3 Blackbox Delivery Client

Der Delivery Client sendet die Auswertungen und die Roundtriptime zurück an den Blackopt Webservice. Im Vergleich ist der Delivery Client dem des Blackopt Webservices sehr ähnlich, unterscheidet sich sich jedoch in einigen kleinen Feinheiten. Die Daten, die an den Blackopt Webservice zurückgeschickt werden, sind vollständige Datensätze mit Ein- und Ausgabedateien bezüglich einer Job ID.

Der Delivery Client besteht aus folgenden drei Bestandteilen:

- Blackbox Delivery Service
- Blackbox Delivery Routine

- Start Blackbox Delivery Client

Deliver Service

Hier wird die Verbindung zum Blackopt Webservice hergestellt. Dabei werden alle nötigen Schritte der notwendigen Informationen zum Zugriff auf den WSDL Service vorbereitet.

Deliver Routine

Die Delivery Routine an sich definiert lediglich die Operationen, die von außen zugänglich sind, sowie die Parameter und Rückgabewerte dieser Operationen.

Start Delivery Client

Um den Delivery Client zu starten, muss diesem alle nötigen Daten übergeben werden. Die IP-Adresse, sowie der Port sind dem Blackbox Clienten bekannt. Diese wurden bei der Installation der Blackbox gespeichert. Ein neuer Delivery Service wird gestartet und die Daten an den Blackopt Webservice gesendet.

4.3.4 Blackbox Client Interaktion

Schaubild 4.7 zeigt die Interaktion zwischen den Bestandteilen des Blackbox Clients auf.

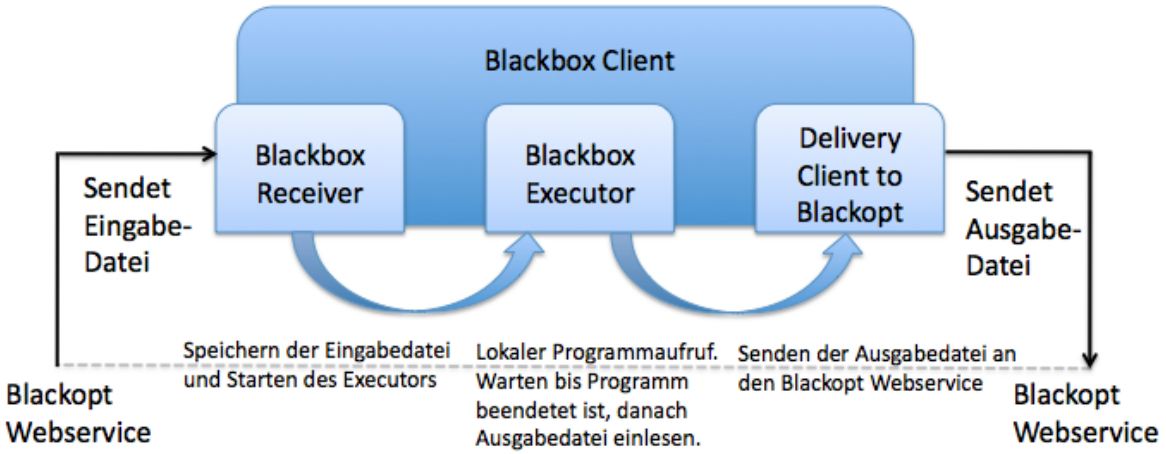

Abbildung 4.7: Blackbox Client Interaktion

Der Blackopt Webservice sendet die Eingabewerte eines Jobs an die Blackbox. Der Blackbox Receiver empfängt die Eingabedatei und speichert diese lokal ab. Im

nächsten Schritt wird das lokale Programm gestartet und die Auswertung kann beginnen. Der Executor überprüft wiederholt in einem Intervall, ob die Ergebnisse vorliegen. Nachdem die Ergebnisse vorliegen werden diese eingelesen und mittels des Delivery Clients wieder zum Blackopt Webservice zurückgeschickt.

4.4 Blackopt Matlab Connector

Die Anbindung des Blackopt Webservices in Matlab erfolgt durch die Implementierung von Java in Matlab. Das Aufrufen von Java Objekten von Matlab heraus wird von Mathworks unterstützt. Andersherum würde dies nicht funktionieren.
Der Matlab Connector beinhaltet lediglich zwei Klassen mit den nötigen Methoden zur Übertragung der Daten an den Blackopt Webservice sowie dem Abrufen gezielter Werte.

Connector

Die Connector Klasse umfasst die folgenden vier Funktionen:

- *sendData(Object jobid, Object elements, Object[] x)* sendet eine neue Anfrage an den Blackopt Webservice mit bestehender JobID
- *getResult(int jobid)*: Der Optimierungsserver kann mit Hilfe dieser Methode überprüfen, ob die Auswertung der Blackbox bereits beendet ist und die Resultate vorliegen.
- *sendBestValues(Double[] bestValues)*: Die besten Werte werden an den Blackopt Webservice geschickt und in der Datenbank abgespeichert.
- *getBestValues()*: Die besten Werte können aus der Blackopt Datenbank abgerufen werden.

Optimization Delivery Client

Dieser Übertragungsdienst bietet die Schnittstelle für die Methoden in der Connector Klasse. Der Blackopt Webservice bietet vorgefertigte Methoden, die mit dem Delivery Client entfernt aufgerufen werden können.

4.5 Blackopt Webinterface

Bei der Erstellung des Blackopt Webinterfaces wurde viel Wert darauf gelegt eine möglichst einfache und intuitive Oberfläche zu schaffen. Der Aktionsradius umfasst

drei grundlegende Einstellmöglichkeiten, das Verwalten von Blackboxen, Problemen und Jobs.

4.5.1 Datenmodell

Abbildung 4.8 zeigt das Datenmodell des Blackopt Systems. Dabei soll die Verbindung zwischen Job, Problem und Blackbox deutlich werden. Die kleinste Einheit ist die Blackbox ganz rechts. Sie besteht aus einer ID, welche eine Blackbox eindeutig identifiziert. Diese ID ist später wichtig für die Koordination der Anfragen auf dem Blackopt Webservice. Weiterhin kann man eine Beschreibung der Blackbox angeben. IP und Port ergeben zusammen die Adresse unter der die Blackbox erreichbar ist. Ein Problem wiederum besteht aus einer ID, einer Beschreibung, sowie dem mathematischen Code, der das Problem formuliert. Außerdem können jedem Problem 0 bis n Blackboxen zugeordnet werden. Zuletzt betrachten wir den Job als oberstes Element. Wird ein neuer Job angelegt, wird automatisch eine JobID vergeben. Der Benutzer wird aufgefordert eine Beschreibung der Blackbox und seinen Namen einzutragen. Ein Zeitstempel der Erstellung des Jobs wird ebenfalls hinzugefügt. Jedem Job wird ein Problem zugeordnet, welches nun gestartet werden kann. Der Status des Jobs gibt Auskunft darüber, ob der Job gerade das Problem bearbeitet, es bearbeitet hat oder zurzeit nichts macht.

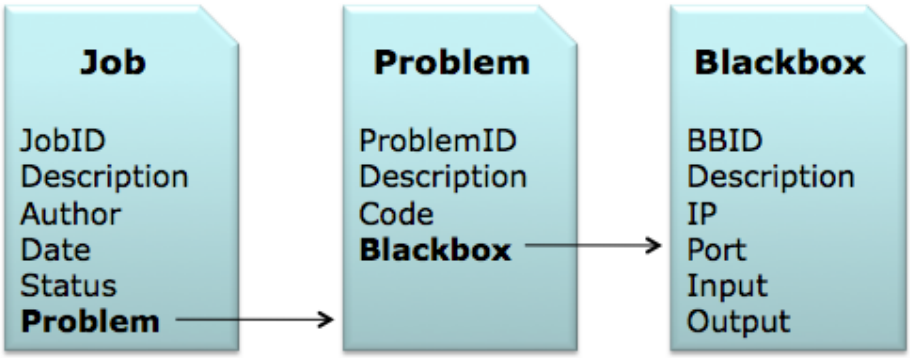

Abbildung 4.8: Datenmodell Abhängigkeiten

4.5.2 Startseite

Das Webinterface steht für eine intuitive Konfiguration der Blackboxen, Probleme und Jobs. Die Startseite gibt eine Übersicht der Verwaltungsmöglichkeiten sowie einen schnellen Blick in die Statistik des Blackopt Webservice. Folgende Statistiken sind verfügbar:

- ***Jobs:*** Anzahl der Jobs im System
- ***Active Jobs:*** Gibt die Anzahl der aktuell aktiven Jobs an. Ein Job ist aktiv, wenn eine Auswertung auf einer Blackbox vorgenommen wird.
- ***Blackboxes:*** Anzahl der eingetragenen Blackboxen
- ***Problems:*** Anzahl der erstellten Probleme

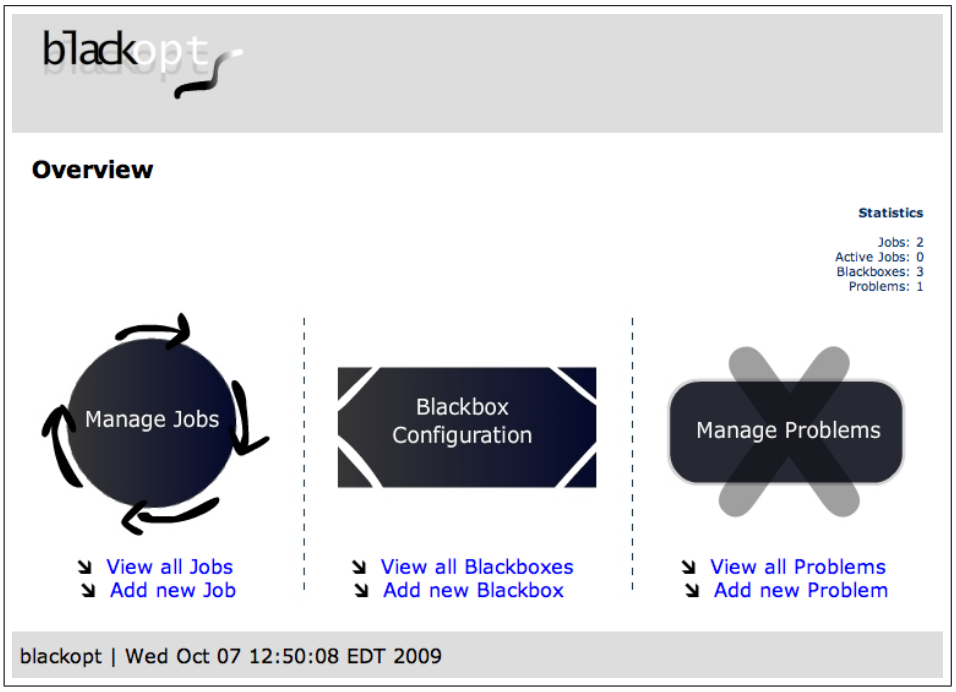

Abbildung 4.9: Startseite des Blackopt Webinterfaces

Abbildung 4.9 zeigt die Startseite des Webinterfaces. Die weiteren Ebenen des Webinterfaces werden in Kapitel 5 anhand eines exemplarischen Einsatzes aufgezeigt.

5 Expemplarischer Einsatz

Das Kapitel exemplarischer Einsatz stellt die entwickelte Lösung zur gestellten Problemstellung vor und greift die Frage auf, wie gut die vorgestellte Lösung das ursprüngliche Problem abdeckt. Dabei werden die im Abschnitt 4.1 auf Seite 25 vorgestellten Komponenten genauer unter die Lupe genommen. Anwendungsfälle aus dem Abschnitt 3.4 auf Seite 20 und das dazu gehörige Diagramm 3.6 bilden die Szenarien für den exemplarischen Einsatz. Da es sich bei der Lösung um eine GUI[1]-basierte Anwendung handelt, werden im Laufe der Evaluierung einige Bildschirmaufnahmen gezeigt, die sowohl Funktionsweise des Systems als auch die GUI-Bedienung näher bringen sollen.

5.1 Datenverwaltung mit dem Webinterface

Das Blackopt - Webinterface beschäftigt sich mit dem Verwalten der unterschiedlichen Datenstrukturen des Blackopt Systems. Abbildung 4.8 auf Seite 37 zeigt die Abhängigkeiten der verschiedenen Datenstrukturen. Jedes Element des Datenmodells hat eine eigene Verwaltungsoberfläche. Dabei kann der Benutzer eine Blackbox, ein Problem oder ein Job verwalten. Die Auswahl wird in Abbildung 5.1 aufgezeigt.

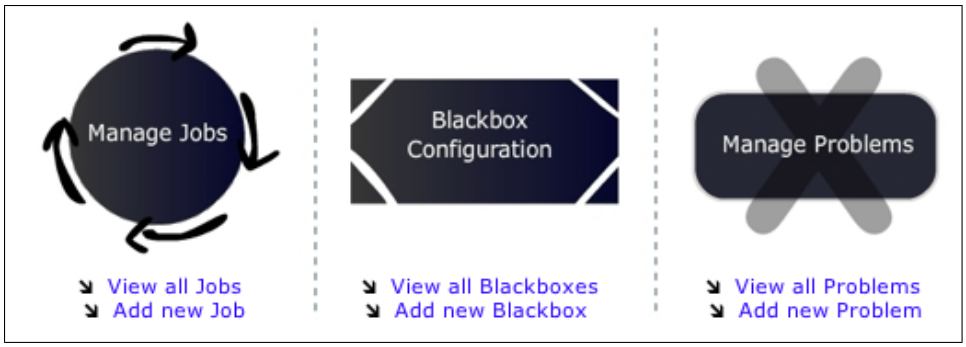

Abbildung 5.1: Startseite mit Datenverwaltung

5.1.1 Blackbox verwalten

Entscheidet sich der Benutzer für das Verwalten einer Blackbox, hat er nun drei Möglichkeiten fortzuschreiten. In der Abbildung 5.2 ist die Blackboxliste leer, da kei-

[1]GUI - Graphical User Interface (dt. Grafische Benutzeroberfläche)

ne Blackboxen vorhanden sind. Abbildung 5.3 zeigt hingegen die Blackboxliste mit einem Eintrag. Der Benutzer hat nun die Auswahl, entweder eine neue Blackbox hinzuzufügen oder die Detailseite einer bereits bestehenden Blackbox aufzurufen.

Abbildung 5.2: Ansicht einer leeren Blackboxliste

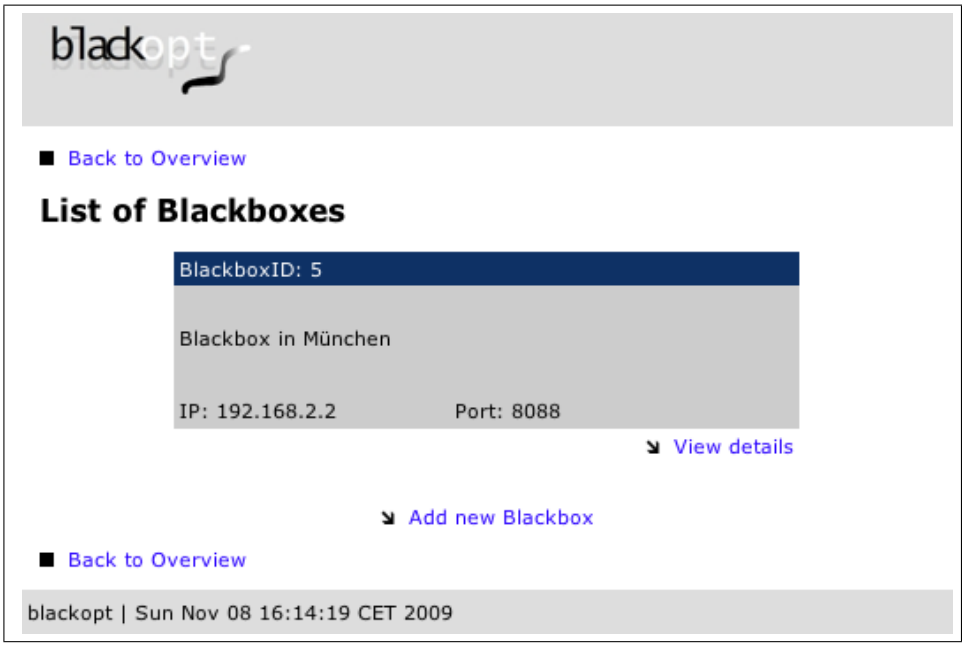

Abbildung 5.3: Blackboxliste mit einem Eintrag

Blackbox hinzufügen

Mit einem Klick auf *Add new Blackbox* erreicht der Benutzer die Eingabemaske zum Erstellen einer neuen Blackbox. Abbildung 5.4 auf Seite 41 zeigt die Eingabemaske mit vordefinierten Eingaben. Folgende Daten müssen für das Erstellen einer Blackbox eingetragen werden:

- Blackbox ID (Die ID wird automatisch erzeugt)
- Beschreibung
- IP-Adresse
- Port
- Input Parameter
- Output Parameter
- Startbefehl des lokalen Aufrufs

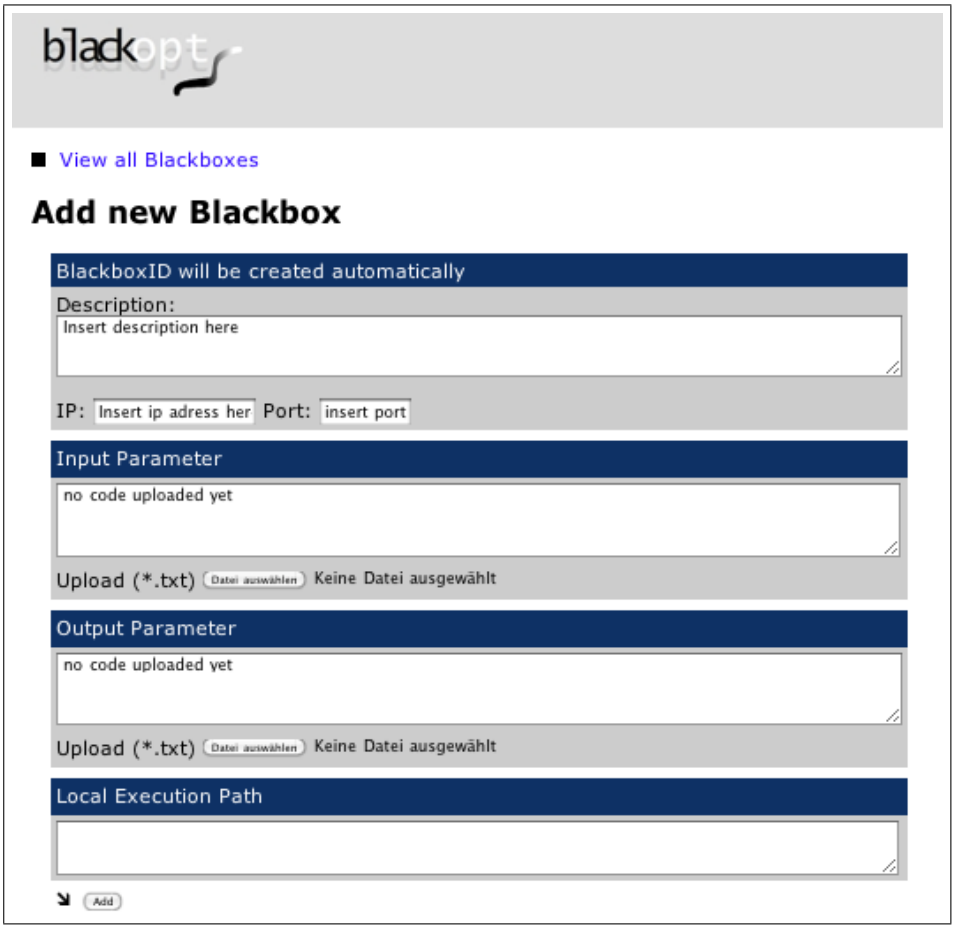

Abbildung 5.4: Eine Blackbox der Datenbank hinzufügen

Die Beschreibung dient zur schnelleren Identifizierung und Verwendung der Blackbox. IP-Adresse und Port müssen dem Benutzer bekannt sein. Unter dieser Adresse wird die Blackbox im späteren Verlauf verfügbar sein. Die In- und Outputparameter haben zu diesem Zeitpunkt noch keine Bedeutung, werden aber in Abschnitt 6.3.1 auf Seite 54 für eine zukünftige Verwendung eingeplant und näher beschrieben. Der Startbefehl des lokalen Aufrufs ist wichtig für den Blackbox Client. Dieser Befehl wird vom Blackbox Client während der Iterationsphase eines Optimierungsproblems ausgeführt. Der

Blackopt Webservice ermittelt aus diesem Befehl den Ordner des lokalen Programms und speichert dort die Eingabedateien ab. Außerdem wird in diesem Ordner nach der Ausgabedatei gesucht. Nachdem man alle Daten eingegeben hat, klickt man auf *add* und die Blackbox wird der Datenbank hinzugefügt.

Blackbox Details aufrufen

Die Blackbox Detail Seite gibt nähere Informationen zur Blackbox. Abbildung 5.3 auf Seite 40 illustriert die Möglichkeit auf die Detailseite aufzurufen. Durch einen Klick auf *View Details* wird die Detailseite der ausgewählten Blackbox angezeigt. Abbildung 5.5 zeigt die Detailseite der Blackbox. Der Benutzer bekommt einen Überblick über die eingetragenen Daten und zusätzlich die Auskunft, ob die Blackbox verfügbar ist. Der Status ist entweder *available*, verfügbar oder *not available*, nicht verfügbar. Falls eine Blackbox nicht verfügbar ist, kann diese Abfrage einen Moment dauern, wodurch sich die Aufbauzeit der Seite auf einige Sekunden herauszögern kann. Im Normalfall, wenn die Blackbox verfügbar ist, wird die Detailsseite sofort geladen. Wichtig für den Benut-

Abbildung 5.5: Detailseite einer Blackbox

zer ist der Download der Konfigurationsdatei und des Blackbox Clients. Der Benutzer lädt den Client herunter und installiert diesen auf der Blackbox. Die Konfigurationsdatei muss im gleichen Ordner mit dem Programm abgelegt werden. Unter dem Punkt *Used by Problems* werden alle Problem ID's angezeigt, die die Blackbox benutzen. Der

Benutzer hat somit zum einen Überblick über die Verwendung der Blackbox und zum anderen wird die Blackbox für das Löschen gesperrt, da sie in Verwendung ist.

Blackbox löschen

Soll eine Blackbox gelöscht werden, muss diese zunächst von allen Problemen gelöst werden. Wird die Blackbox nicht mehr verwendet, kann diese mit einem Klick auf *delete Blackbox* gelöscht werden. Dabei öffnet sich eine weitere Seite, die in Abbildung 5.6 auf Seite 43 gezeigt wird. Mit einer Bestätigung wird die Blackbox aus der Datenbank gelöscht.

Abbildung 5.6: Löschen einer Blackbox

5.1.2 Problem verwalten

Die Verwaltung eines Problems folgt dem gleichen Prinzip wie der zuvor vorgestellten Blackbox. In Kapitel 3 Abschnitt 3.4.1 wird der Anwendungsfall für das Verwalten eines Problems beschrieben. Der Benutzer hat die Wahl zwischen dem Erstellen, Editieren oder Löschen eines Problems. Das Problem besteht aus drei Eigenschaften:

- Beschreibung
- Code
- Blackbox Auswahl

Die Beschreibung dient zur einfachen Erkennung des Problems. Der Code des Problems wird als String eingetragen. Hierbei kann es sich um eine mathematische Modellierspracheh handeln. Der bisherige Stand des Projekts braucht die Probleme jedoch nur für die interne Verwaltung eines Jobs. Der spezifische Code spielt im Moment noch

keine Rolle. Näheres hierzu befindet sich in Kapitel 6 unter 6.3.1. Der letzte Punkt ist die Zuweisung einer Blackbox zu einem Problem. Durch diese Verknüpfung ist es später möglich einen Job zu erstellen, der ein Problem benutzt, das auf die Blackbox zugreift. Das Löschen eines Problems funktioniert auf die selbe Weise, wie das Löschen einer Blackbox in Abschnitt 5.1.1 und wird aus diesem Grund nicht näher beschrieben.

Problem hinzufügen

Mit einem Klick auf *Save Changes* in Abbildung problemcreate werden die Einstellungen gespeichert und ein neues Problem in der Datenbank erstellt.

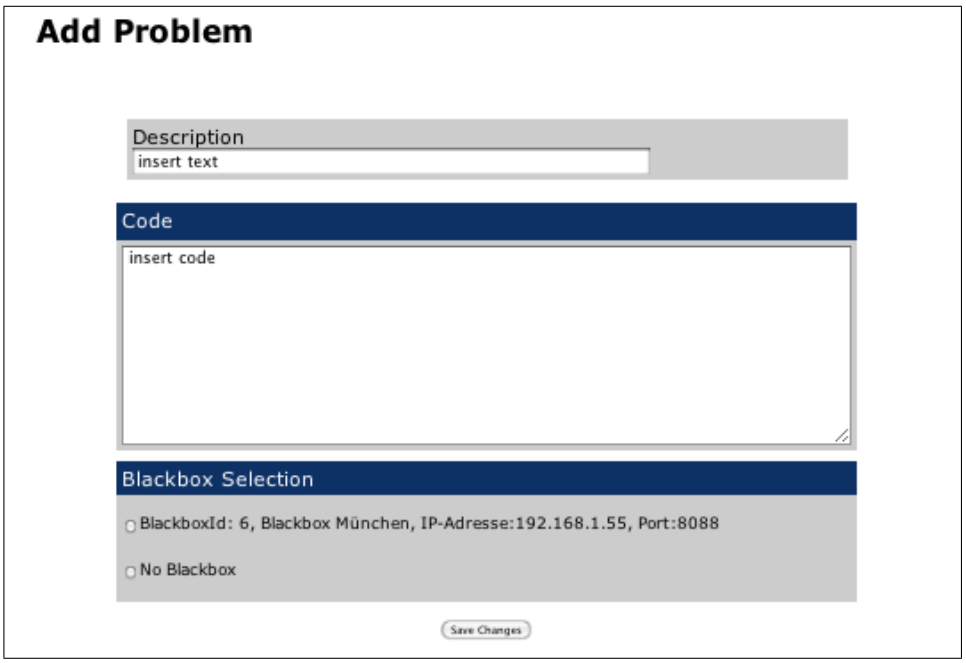

Abbildung 5.7: Erstellen eines Problems

Problem Details aufrufen

Im diesem Abschnitt wird zum Schluss die zum Problem gehörende Detailseite in Abbildung 5.8 gezeigt. *Used by Jobs* gibt an, welche Jobs das Problem verwenden. Die Problem ID wurde wieder automatisch vom System verteilt. Unter dem Punkt *Blackbox Selection* wird die mit dem Problem verknüpfte Blackbox mit einigen Details angezeigt.

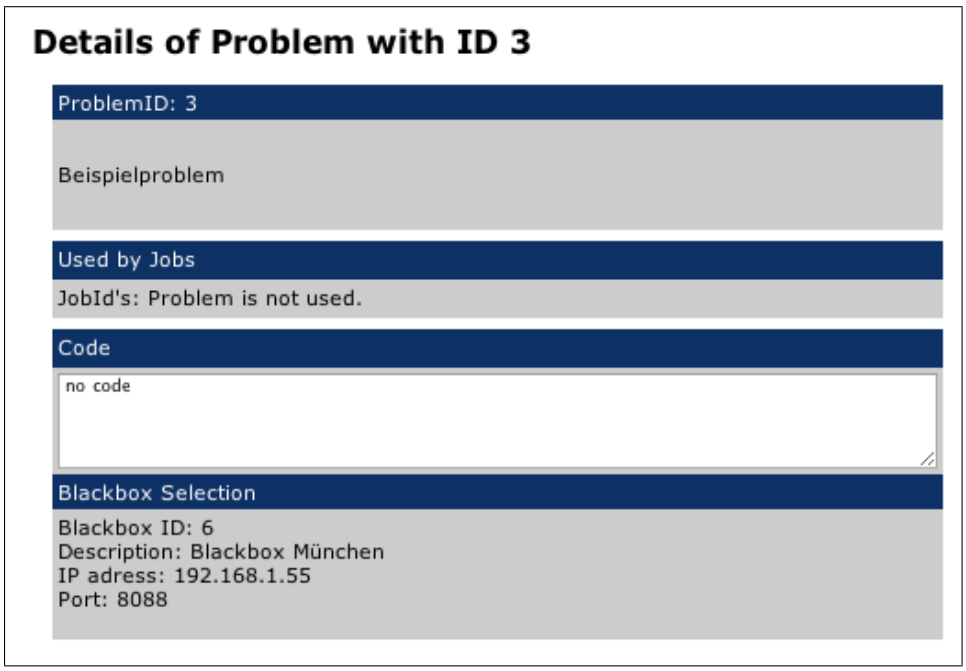

Abbildung 5.8: Detail Seite eines Problems

5.1.3 Job verwalten

Ein Job ist in der Datenstruktur des Blackopt Systems das höchste Element. Ein Job benötigt ein Problem und ein Problem benötigt eine Blackbox. Die Verwaltung eines Jobs verhält sich ebenfalls gleich der Verwaltung der anderen beiden Datenstrukturen. Der Benutzer hat die Möglichkeit, einen Job hinzuzufügen, zu editieren oder zu löschen. Weiterhin kann er die Detailseite eines Jobs aufrufen. Auf den nächsten Seiten wird das Hinzufügen eines Jobs, die Detailseite und die Übersicht über alle Jobs in der Jobliste näher beschrieben.

Hinzufügen eines Jobs

Die Eingabemaske des Job Formulars aus Abbildung 5.9 besteht aus den folgenden Elementen:

- Author
- Beschreibung
- Problem Auswahl

Die Beschreibung des Jobs dient wieder zur leichteren Identifikation des Jobs. Ein Datum und die Job ID werden automatisch vom System vergeben. Der Benutzer setzt das passende Häkchen bei der Problem Auswahl und mit einem Klick auf *Save Changes* werden die Daten gespeichert. Das Problem ist nun an den Job gebunden. Nur Jobs mit Problemen können später auch gestartet werden.

Abbildung 5.9: Einen neuen Job dem System hinzufügen

Jobliste anzeigen

Die Jobliste gibt eine Übersicht über alle im System vorhanden Jobs. Dabei werden die Job ID, der Autor, das Erstelldatum des Jobs, die Beschreibung und der aktuelle Status des Jobs angezeigt. Der Status kann zwei Werte annehmen, *waiting for action*, wenn er darauf wartet, dass der Job gestartet wird und *in progress*, wenn gerade ein Job bearbeitet wird und die Auswertung auf einer Blackbox läuft. Abbildung 5.10 gibt die Liste mit einem Job an, der zurzeit auf darauf wartet, gestartet zu werden.

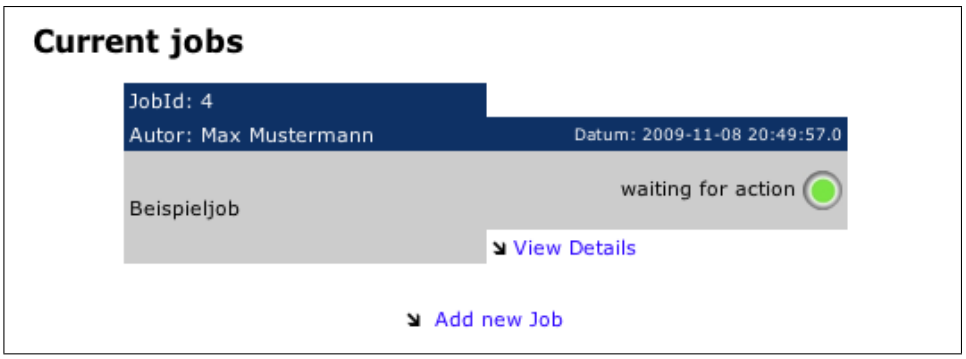

Abbildung 5.10: Überblick über die im System vorhandenen Jobs

Detailseite eines Jobs aufrufen

Die Detailseite eines Jobs beinhaltet alle Informationen, die auch in der Jobliste angezeigt werden und zusätzlich die Informationen über das ausgewählte Problem. Abbildung 5.11 zeigt einen Job, der an ein Problem gebunden ist und zurzeit darauf wartet, gestartet zu werden.

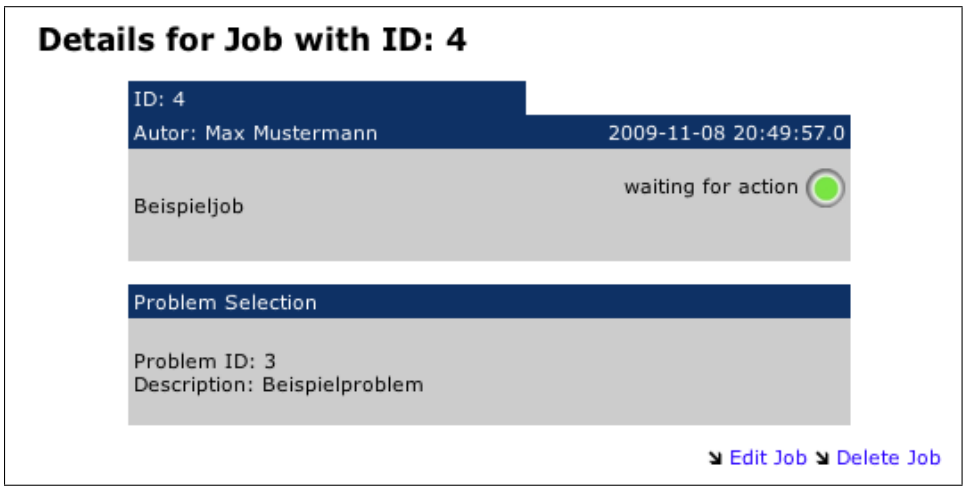

Abbildung 5.11: Detailseite eines Jobs

5.2 Installation des Blackbox Clients

Der Blackbox Client wird in Kapitel 4 in Abschnitt 4.3 auf Seite 30 ausführlich diskutiert. Der Download des Clients ist auf der Detailseite [2] der jeweiligen Blackbox vorhanden. Der Download besteht aus zwei Dateien,

- dem Blackbox Client
- und der Konfigurationsdatei.

Wichtig bei der Installation der Dateien auf der Blackbox ist, dass beide Dateien im gleichen Ordner liegen, da der Blackbox Client nach einer Konfigurationsdatei im gleichen Ordner sucht. Die Konfigurationsdatei ist eine XML-Datei mit den nötigen Informationen zur Blackbox. Enthalten sind alle Daten, die im Webinterface in der Blackbox Detailseite angegeben sind. An dieser Stelle ist vor allem der Ausführungsbefehl sehr wichtig, da dieser den Aufruf des lokalen Programms auf der Blackbox enthält sowie den Ordner des lokalen Programms vorgibt. Die genauen Eigenschaften einer Blackbox wurden in Abschnitt 5.1.1 auf Seite 40 vorgestellt.

Abbildung 5.12 auf Seite 48 zeigt eine beispielhafte Konfigurationsdatei der Blackbox im XML Format.

Im folgenden werden die XML Tags der Konfigurationsdatei von Abbildung 5.12 erläutert. Die *ReceiverAdress* ist die IP Adresse der Blackbox. Unter dieser Adresse wird der lokale Service auf der Blackbox gestartet. Der *Path* ist der Pfad unter dem das lokale Programm liegt, das durch den Service gestartet wird. *ShellCommand* beinhaltet den Ausführungsbefehl, der im Webinterface bei der Konfiguration der Blackbox eingegeben wurde. Die Blackbox ID *Bbid* identifiziert die Blackbox eindeutig im System und der Datenname *DataName* wurde ebenfalls im Webinterface definiert. Dieser wird benötigt um dem lokalen Programm die Eingabedaten zur Verfügung zu stellen,

[2] Detailseite einer Blackbox - siehe Abschnitt 5.1.1 auf Seite 42

```
<?xml version='1.0' encoding='utf-8'?>
<Blackbox>
        <ReceiverAdress> address of blackbox </ReceiverAdress>
        <Path> path of local software </Path>
        <ShellCommand> execution command </ShellCommand>
        <Bbid> blackbox id </Bbid>
        <DataName> dataname </DataName>
</Blackbox>
```

Abbildung 5.12: Konfigurationsdatei einer Blackbox

sowie zum Auslesen der Ausgabedatei des lokalen Programms.

5.3 Durchführung eines Jobs

Das Blackopt System ist in diesem exemplarischen Einsatz auf Matlab ausgelegt. Um einen Job zu starten, muss der Benutzer seinen Matlab Code um die Funktionen des Blackopt Systems erweitern. Abbildung 5.13 zeigt die Interaktion des Benutzers und die Auswirkungen im Blackopt System. Das Anbinden der Java Methoden in Matlab wird in Abschnitt 5.3.1 beschrieben. Der Job wird in Matlab gestartet, in dem der Benutzer die Eingabedaten an den Blackopt Webservice sendet. In der Zukunft ist es vorgesehen, den Job aus dem Blackopt Webinterface heraus zu starten. Dieses Feature wird in den Erweiterungsmöglichkeiten beschrieben.

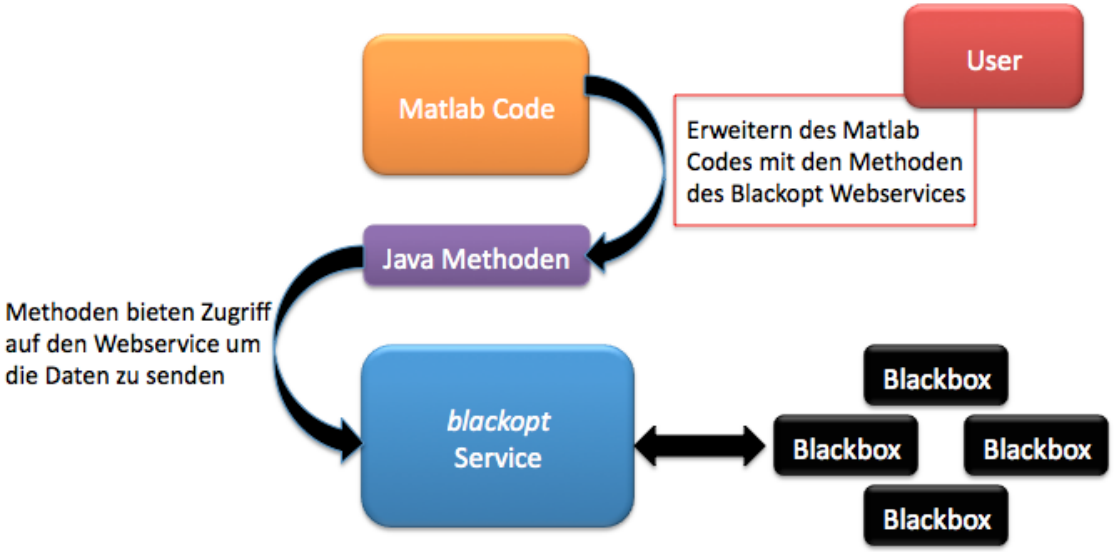

Abbildung 5.13: Aktivität des Blackopt Systems

5.3.1 Erweitern des Matlab Codes

Abbildung 5.14 auf Seite 49 beschreibt einen entfernten Aufruf innerhalb des Matlab Codes. Mit dem Befehl *javaaddpath* werden die Java Klassen Matlab bekannt gemacht. Vorraussetzung ist, dass der Benutzer die benötigten Klassen zuvor in einem Verzeichnis seiner Wahl abgelegt hat. Näheres dazu wurde in Kapitel 4 im Abschnitt 4.4 auf Seite 36 beschrieben. Mit Hilfe von *javaObject* wird ein Java Instanz der ConnectionTool Klasse erstellt. Nun kann der Benutzer Methoden der ConnectionTool Klasse aufrufen. In diesem simplen Beispiel wird die Methode *sendToBlackopt* mit den Parametern JobID und einem String aufgerufen. *isReady* gibt 1 zurück (in Matlab entspricht die "1" dem "true"), wenn der Job bearbeitet wurde und ein Ergebnis vorliegt. Um dieses Ergebnis abzurufen wird die Methode *getResult* mit der JobID als Parameter aufgerufen. Die Variable *result* enthält nun einen String mit der Ausgabe, der weiterverarbeitet werden kann.

```
1 % Java Klassen bekannt machen
2 javaaddpath('C:\BlackoptMatlab');
3
4 % Java Objekt instanziieren
5 connection = javaObject('ConnectionTool');
6
7 % sendToBlackopt(int JobID, String input)
8 % senden an den Blackopt Webservice
9 connection.sendToBlackopt(javaObject('java.lang.Integer','1'),
10 javaObject('java.lang.String','Hello World'));
11
12 % Testabfrage für den Benutzer, ob der Job fertig ist
13 ready = connection.isReady('java.lang.Integer','1');
14
15 % Wenn ready 'true' ist (in Matlab gleich 1)
16 % nun kann man die Ausgabedatei abholen
17 result = connection.getResult('java.lang.Integer','1');
```

Abbildung 5.14: Java Methoden in den Matlab Code integrieren

5.3.2 Job starten und beenden

Das Starten des Jobs wird aktuell durch den Benutzer so ausgeführt, wie in Abschnitt 5.3.1 beschrieben. Aus Matlab heraus wird eine Java Instanz aufgerufen, die die Eingabedaten an den Blackopt Webservice schickt. Abschnitt 6.3 in Kapitel 6 stellt die Erweiterungsmöglichkeiten zur Verwaltung eines Jobs vor. Die Gründe einen Job vorzeitig zu beenden, könnten die Folgenden sein, Blackbox-Auswertung dauert zu lange,

Auswertung wird nicht mehr benötigt oder die Blackbox ist nicht mehr verfügbar. Um einen Job vorzeitig zu beenden, ruft der Benutzer das Webinterface auf und öffnet die Detailsseite des ausgewählten Jobs. Abbildung 5.15 auf Seite 50 zeigt die Detailseite eines Jobs während einer Blackbox-Auswertung. Mit einem Klick auf *Kill Job* kann der Job beendet werden und die Blackbox wird freigegeben für andere Anfragen.

Abbildung 5.15: Beenden eines Jobs

6 Diskussion und Ausblick

Die Ergebnisse aus dem exemplarischen Einsatz, Kapitel 5, sind die Grundlagen dafür das Blackopt System der in Kapitel 2 definierten Anforderungen gegenüberzustellen. Dabei sollen die möglichen Schwachpunkte des Systems identifiziert und hervorgehoben werden. Weiterhin sollen die Auswertungen der Ergebnisse mit den erhofften Zielen verglichen werden. Daraus ergeben sich die sinnvollen Erweiterungen sowie die nächsten Schritte.

6.1 Ergebnisse

Der Bezug zum aktuellen Stand wird unter den Aspekten der Performanz und der Bedienung des Blackopt Systems gemacht, um eine qualitativ gute Bestimmung des Ansatzes zu gewährleisten. Die zentrale Problemstellung der Anbindung von Blackboxen an bestehende Optimierungsinfrastrukturen wurde im Laufe dieser Bachelorarbeit analysiert, in ein Konzept gebracht, implementiert und anhand eines exemplarischen Einsatzes evaluiert.

Der Abschnitt 5.1 auf Seite 39 demonstriert die Lösung für das interne Management der Datenstrukturen des Blackopt Systems. Mit Hilfe dieses Verwaltungstools wurde in Abschnitt 5.2 auf Seite 47 gezeigt, wie eine Blackbox an das System angebunden wird. Zum Schluss demonstriert der Abschnitt 5.3 auf Seite 48 das Durchführen eines Jobs. Diese drei Abschnitte decken die Evaluation der Problemstellung komplett ab. Im Inneren des Systems wurden Komponenten entwickelt, die nach aussen hin nicht direkt sichtbar sind. Die Funktion dieser Komponenten wurde in Kapitel 4 Umsetzung detailliert beschrieben.

Die Ergebnisse sind vielversprechend und liefern eine Umsetzung der formulierten Anforderungen bzgl. der gestellten Problemstellung, bieten jedoch auch ein breites Feld der Weiterentwicklung des Systems.

6.2 Kritik

Mit Sicherheit sind auch in der dieser Bachelorarbeit Schwachpunkte zu finden. Dieser Abschnitt soll diese identifizieren und kurz darauf gehen. So kann man auch absehen, welche Aspekte mögliche Verbesserungen benötigen.

6.2.1 Anbindung weiterer Modellierungssprachen

Der Anwender hat die Möglichkeit Optimierungsprobleme zu formulieren, die in der Programmiersprache Matlab geschrieben werden müssen. Da es aber viele weitere mathematische Modelliersprachen auf dem Markt gibt, deckt Matlab nur einen sehr geringen Teil ab. Im Moment kann demnach lediglich Matlab auf Seiten des Optimierers eingesetzt werden. Um dies zu erweitern, müssen zusätzliche Tools zur Anbindung weiterer Modelliersprachen implementiert werden.

6.2.2 Datenstruktur

Das Blackopt System besitzt die drei Datenstrukturen Job, Problem und Blackbox. Der folgende Abschnitt erläutert Kritikpunkte an den Datenstrukturen *Problem* und *Blackbox*.

Problemmodellierung

Die Datenstruktur aus Abbildung 6.1 eines Problems wurde in Kapitel 4 in Abschnitt 4.5.1 auf Seite 37 erläutert. Probleme sind die in Kapitel 2 Abschnitt 2.2 eingeführten Optimierungsprobleme mit der Funktion $f(\boldsymbol{x}, \mathbf{y}(\boldsymbol{x}))$. Blackopt berücksichtigt bisher nur das $\mathbf{y}(\boldsymbol{x})$, das die Blackbox darstellt und reicht diese zwischen den Komponenten durch. Die Datenstruktur enthält neben ProblemID und einer Beschreibung auch den Code des mathematischen Optimierungsproblems und ein Feld für eine multiple Anzahl an Blackboxen, die mit dem Problem verknüpft werden können. Das Blackopt

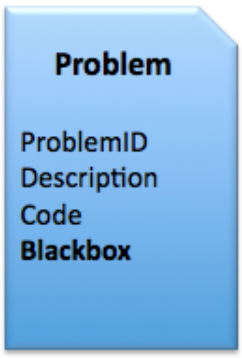

Abbildung 6.1: Datenmodell eines Problems

System nutzt das Problem im Moment jedoch nur von der theoretischen Seite, denn jeder Job beinhaltet ein Problem und jedes Problem eine Blackbox. Die Jobs werden aus dem Matlab Code heraus direkt gestartet, indem die Eingabedaten an das Blackopt System gesendet werden. Das Problem Datenmodell sieht vor die Jobs später aus dem

Webinterface zu starten, in dem das Problem an den Optimierer gesendet wird. Das Blackopt System würde somit die Verwaltung von Blackboxen und den Optimierungsproblemen zusammenführen.

Blackboxmodellierung

Die zwei Elemente Input und Output des Blackbox Modells aus Abbildung 6.2 sind im Moment direkt als String abgespeichert.Im Element Input werden die Eingabedaten für die Blackbox gespeichert und das Element Output beinhaltet die Ergebnisse der Blackbox Auswertung. An dieser Stelle sollen nur die für die Optimierung benötigten Daten übertragen werden, jedoch nicht wie zurzeit, die kompletten Eingabedateien für Blackbox und Ausgabedateien für den Optimierer. Im Moment muss der Optimierer, nachdem er ein Ergebnis in Form einer Textdatei erhalten hat, diese Auslesen und nach den geforderten Werten filtern. Dieser Ansatz wird auch in den kommen Erweiterungsmöglichkeiten diskutiert.

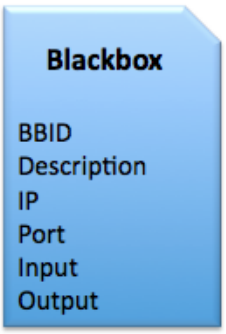

Abbildung 6.2: Datenmodell einer Blackbox

6.2.3 Statistiken über die Auswertung der Blackboxen

Das Statistikmodell zur Auswahl der Blackbox, an die eine Anfrage gesendet wird basiert zurzeit nur auf Erfahrungswerten der Roundtriptime [1]. Weitere Statistiken wie die Leistung und Ressourcen des Blackbox Rechners, die Verfügbarkeit könnten einbezogen werden. Man könnte auch ein anderes Warteschlangenmodell auf Seiten des Blackopt Systems integrieren. Zurzeit wird "First in First out" verwendet, welches aber durchaus mit einem Modell, welches auch Prioritäten zulässt, ersetzt werden kann.

[1] Roundtriptime - Zeit eines Jobs vom Senden zur Blackbox bis zum Erhalt der Ergebnisse

6.3 Erweiterungsmöglichkeiten

In diesem Abschnitt sollen einige Erweiterungsmöglichkeiten vorgestellt werden, die zum Teil Lösungen für die in Abschnitt 6.2 auf Seite 51 diskutierten Probleme darstellen. Die Erweiterungen beziehen sich zum einen auf die interne Datenstruktur des Blackopt Systems und zum anderen auf das Webinterface um den Komfort für den Benutzer zu steigern. Die beiden folgenden Abschnitte beschreiben die Erweiterung der Datenstruktur des Blackopt Systems. Der Entwickler muss hierzu auch die MySQL Datenbanken anpassen.

6.3.1 Integration von Problemen

Im vorigen Abschnitt wurde die Kritik bzgl. der Problem-Datenstruktur diskutiert. Um das Problem vollständig dem Blackopt System zu übergeben, muss das System um eine effiziente Anbindung an Optimierer erweitert werden. Ähnlich wie der Neos Server[2] kann das Blackopt System eine Schnittstelle bieten, so dass Optimierungsprobleme direkt ins Webinterface geladen werden und von dort aus auch gestartet werden können. Das Blackopt System würde an dieser Stelle mit dem Optimierer verschmelzen, was für den Benutzer den Vorteil hat, dass er sein Problem direkt hochladen kann und sich nicht um die Anbindung des Optimierers kümmern muss.

6.3.2 Erweitern der Blackbox Datenstruktur

Die in Abschnitt 6.2.2 auf Seite 53 beschriebene Kritik am Blackboxmodell soll nun umgesetzt werden. Das Erweitern der Blackbox Datenstruktur hängt stark mit der Entwicklung eines Editors zum Verwalten der Ein- und Ausgabedateien zusammen, der in Abschnitt 6.3.3 diskutiert wird. Der Blackbox Client (siehe 4.3 auf Seite 30) und der Blackopt Webservice (siehe Seite 4.2 auf Seite 26) müssen beide angepasst werden auf die Input und Output Dateien der Blackbox. Das aktuelle System sieht nur das Senden von Strings zwischen den Komponenten vor. Durch die Integration eines Editors zum Verwalten dieser Dateien muss an dieser Stelle ein neues Konzept der Blackbox Datenstruktur erstellt werden.

6.3.3 Ausbau des Webinterfaces

Interne Erweiterungen verlangen zwangsläufig auch die Anpassung des Webinterfaces auf die neuen Funktionen. Mit dem Ausbau kann ein noch besserer Komfort für den Benutzer gewährleistet werden.

[2]Neos Server - siehe Abschnitt 2.4 auf Seite 9

Editor zum Verwalten der Ein- und Ausgabedateien

in Abschnitt 6.3.1 wurde bereits die Erweiterung der Blackbox Datenstruktur beschrieben. Abbildung 6.3 zeigt eine Skizze eines Editors. Der linke Teil beinhaltet den Code der Ein- oder Ausgabedatei der Blackbox. Diese kann mittels *Datei laden* geladen werden. Auf der rechten Seite werden die Codestellen, eingetragen, die im Code links markiert werden. Die Markierungen der Eingabedatei der Blackbox dienen dazu, die Werte zu identifizieren, die die Blackbox benötigt. Die Markierungen der Ausgabedatei beinhalten die Stellen, an denen der Blackopt Webservice die Rückgabewerte findet. Das Hauptnutzen des Benutzer in Bezug auf den Editor, ist die einfachere Anbindung weiterer Blackboxen und Optimierern. Das aktuelle System unterstützt nur komplette Ein- und Ausgabedateien, die an die Komponenten weitergeleitet werden. Blackbox und Blackopt Webservice sind selbst dafür zuständig, wie sie diese Dateien interpretieren.

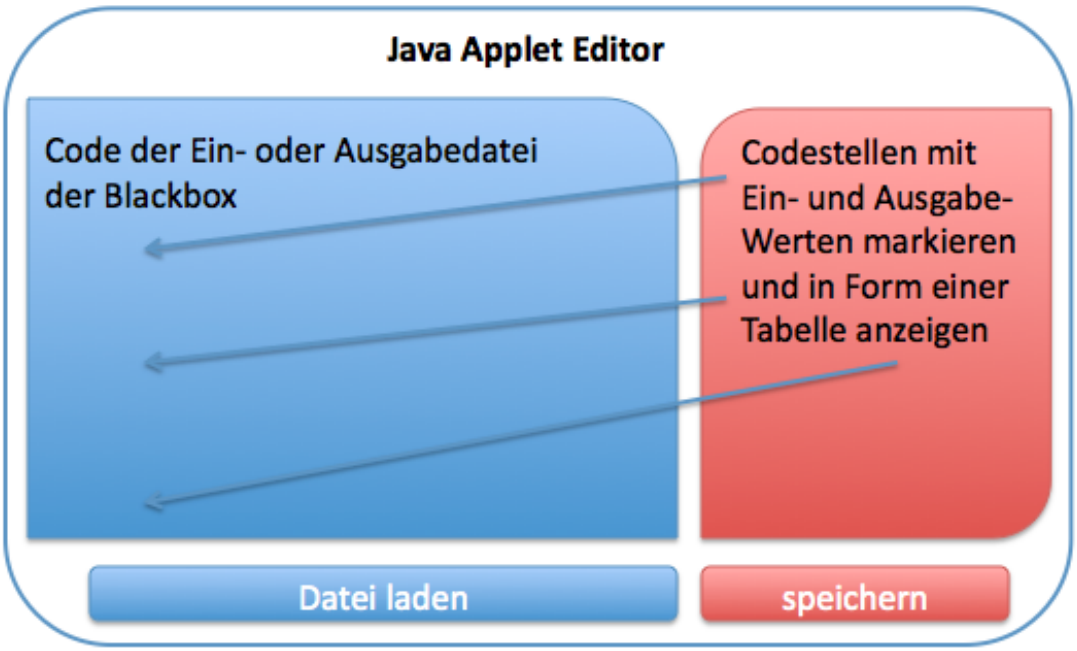

Abbildung 6.3: Java Applet des Editors zum Verwalten der Ein- und Ausgabedateien der Blackbox

Ausgabe der Eingabe- und Ausgabewerte auf der Blackbox Detailseite

Dieser Abschnitt gibt eine Erweiterung der in Abschnitt 5.1.1 beschriebenen Blackbox Detailseite an. Durch die Umgestaltung der Datenstrukturen aus Abschnitt 6.3.1 und 6.3.2 können die Eingabe und Ausgabewerte der Blackbox in Form einer Tabelle auf der Blackbox Detailseite dargestellt werden. Die nötigen SQL-Abfragen ergeben sich

aus der Anpassung der Datenstrukturen. Der Benutzer kann dadurch einen Eindruck über die Blackbox gewinnen und so vielleicht schneller an die gewünschten Ergebnisse kommen.

6.4 Fazit

Die Auswertung der Ergebnisse hat gezeigt, dass es nicht einfach ist Blackboxen in die Umgebungen von Optimierungsservern einzubinden. Durch die Vielzahl an Modellierungssprachen und Optimierern muss die Software an diese angepasst werden. In dieser Bachelorarbeit wurde der Optimierungsserver mit Matlab an das Blackopt System angebunden. Die gewünschten Anforderungen an das System wurden bis auf einzelne Eigenschaften, die in Abschnitt 6.2 auf Seite 51 beschrieben wurden, erfolgreich umgesetzt. Der Blackbox Client kann auf allen Betriebssystemen installiert werden und ist somit universell einsetzbar. Durch die Erweiterung des Matlab Codes mit die Java Methoden, hat der Entwickler alle Mittel zur Verfügung, um weitere Modellierungssprachen anzubieten. Die nächsten Schritte wären nun das System um die Funktionen, die in den Erweiterungsmöglichkeiten beschrieben sind, zu erweitern, um ein effizientes, in der entfernten Optimierung einsetzbares System, zu gewährleisten.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Wikipedia. *Lineare Optimierung*. Verfügbar unter: `http://de.wikipedia.org/wiki/Lineare_Optimierung`, Abruf: 15.Oktober 2009

[2] Buecher.de. *Optimierungsprobleme und Modellbildung*. Verfügbar unter: `http://bilder.buecher.de/zusatz/12/12072/12072244_lese_1.pdf`, Abruf: 15.Oktober 2009

[3] NEOS Wiki. *Non Linear Programming*. Verfügbar unter: `http://wiki.mcs.anl.gov/NEOS/index.php/Nonlinear_Programming_FAQ`, Abruf: 15.Oktober 2009

[4] Wikipedia. *Extensible Markup Language*. Verfügbar unter: `http://bilder.buecher.de/zusatz/12/12072/12072244_lese_1.pdf`, Abruf: 15.Oktober 2009

[5] GOR - Gesellschaft für Operations Research e.V.. *Modellierungssprachen in der Mathematischen Optimierung*. Verfügbar unter: `https://gor.uni-paderborn.de/Members/AG06/WorkshopModellierungssprachen`, Abruf: 25.Oktober 2009

[6] Marco Colombo. *A Structure-Conveying Modelling Language for Mathematical and Stochastic Programming*. Verfügbar unter: `http://www.optimization-online.org/DB_FILE/2009/03/2262.pdf`, Abruf: 15.Oktober 2009

[7] Civil and Environmental Engineering, University of California, Berkeley. *Introduction to AMPL A Tutorial*. Verfügbar unter: `http://www.ce.berkeley.edu/~bayen/ce191www/labs/lab3/ampl2.pdf`, Abruf: 10.September 2009

[8] Kevin Kofler, Uni Wien. *Black Box Optimization with Data Analysis*. Verfügbar unter: `http://tigen.ti-fr.com/kevin.kofler/bbowda/bbowda-thesis.pdf`, Abruf: 25.Oktober 2009

[9] Elizabeth D. Dolan, Robert Fourer. *The NEOS Server for Optimization*. Verfügbar unter: `http://www.optimization-online.org/DB_FILE/2002/05/474.pdf`, Abruf: 25.Oktober 2009

[10] Elizabeth D. Dolan. *Kestrel: An Interface from Optimization Modeling Systems to the NEOS Server*. Verfügbar unter: `http://www.optimization-online.org/DB_FILE/2007/01/1559.pdf`, Abruf: 25.Oktober 2009

[11] Martin Stöcker. *Untersuchung von Optimierungsverfahren für rechenzeitaufwändige technische Anwendungen in der Motorenentwicklung*. Verfügbar unter: `http://archiv.tu-chemnitz.de/pub/2007/0161/data/diplomarbeit.pdf`, Abruf: 26.Oktober 2009

[12] COmputational INfrastructure for Operations Research. *COIN-OR*. Verfügbar unter: `http://www.coin-or.org/`, Abruf: 26.Oktober 2009

[13] Acro. *A Common Repository for Optimizers*. Verfügbar unter: `https://software.sandia.gov/trac/acro`, Abruf: 26.Oktober 2009

[14] NEOS Server. *NEOS Server for Optimization*. Verfügbar unter: `http://www-neos.mcs.anl.gov/`, Abruf: 26.Oktober 2009

[15] Wikipedia. *Extensible Markup Language*. Verfügbar unter: `http://de.wikipedia.org/wiki/ExtensibleMarkupLanguage`, Abruf: 26.Oktober 2009

[16] Wikipedia. *Document Object Model*. Verfügbar unter: `http://de.wikipedia.org/wi-ki/DocumentObjectModel`, Abruf: 19.Oktober 2009

[17] W3C *Web Services Description Language - WSDL*. Verfügbar unter: `http://www.w3.org/TR/wsdl.html`, Abruf: 19.Oktober 2009

[18] Wikipedia. *Simple Object Access Protocol - SOAP*. Verfügbar unter: `http://de.wikipedia.org/wiki/SOAP`, Abruf: 19.Oktober 2009