Bachelorthesis Frühlingssemester 2012

Entwicklung eines graphischen Editors zur Modellierung von Systemen mit dynamischer Modellstruktur

Andreas Bachmann
bachman0@students.zhaw.ch

Andreas Butti
buttiand@students.zhaw.ch

Betreuer:
Prof. Dr. Stephan Scheidegger
School of Engineering
Technikumstrasse 9
8400 Winterthur
Telefon: 058 934 74 63
stephan.scheidegger@zhaw.ch

Betreuer:
Dr. Rudolf Marcel Füchslin
School of Engineering
Technikumstrasse 9
8400 Winterthur
Telefon: 058 934 75 92
rudolf.fuechslin@zhaw.ch

Inhaltsverzeichnis

1 Einleitung

fusion->trennen von Meso (nicht implementiert)
FSM->nicht invivo

Es gibt bereits viele Simulationstools, wie z.B. Berekely Madonna um nur ein bekanntes Beispiel zu nennen. Diese Tools unterstützten die Modelierung von Differentialgleichungen als Modell, mit Containern und Flüssen.

Das gleiche Prinzip wird auch von Symlink verwendet, dieses verwendet als Symbole jedoch konsequent die Elektrotechnischen Abbildungen, es gibt Verbindung, Verstärker, Verzögerungen (x[t-1]) etc. Obwohl das Modell damit etwas anders aussieht ist die mathematische Abbildung schlussendlich die gleiche.

Da unsere Simulation keine Elektrotechnischen Hintergrund hat haben wir für die Darstellungen unserer Flussmodelle ebenfalls Container mit Flüssen (Integral) verwendet.

1.1 Bereits existierende Tools

1.2 ~~Vorgabe~~

Bestehende graphische Modelleditoren erlauben eine effiziente Modellierung von kompartimentalen Systemen. Dabei unterstützt die graphische Oberfläche die Strukturierung des Modells bzw. des Systems. Dies kann gerade bei der Erfassung von komplexen Systemen den Zugang zu einer adäquaten Systembeschreibung erleichtern. Gerade aber Modelleditoren wie Berkeley-Madonna sind auf eine kompartimentale Struktur des Systems angewiesen. Räumlich strukturierte bzw. verteilte Systeme lassen sich nur schwer und in vereinfachter Form abbilden. Die Verwendung oder Kopplung verschiedener Simulationswerkzeuge kann für gewisse technische Systeme in Betracht gezogen werden (z.B. elektrische Schaltung mit Komponenten, bei denen die Wärmeabstrahlung und oder Wärmeleitung räumlich modelliert werden). Bei vielen Systemen lässt sich aber durch eine solche Kopplung das System nicht abbilden. Bei biologischen Systemen z.B. können sich Kompartimente bewegen (bei Zellen z.B. Chemo- und Haptotaxis). Zudem zeichnen sich biologische Systeme durch hierarchische Kompartimentstrukturen mit Unterkompartimenten aus. Ein weiterer Aspekt betrifft die Möglichkeit, dass Kompartimente in biologischen Systemen fusionieren oder sich teilen können.

Anforderungen: Das zu entwickelnde Modellierungswerkzeug soll an die intuitive graphische Oberfläche bestehender Modellierungswerkzeuge für kompartimentale Simulationen anknüpfen. Folgene Aspekte sollen konzeptuell untersucht und wenn möglich implemetiert werden:

- Hierarchische Kompartimente
- Räumliche Positionierung von Kompartimenten, welche die Wechselwirkung der Kompartimente auf gleicher Stufe beeinflussen kann und somit Einführung von Koordinaten (erster Schritt 2-Dim.) bzw. orthogonales Grid und beschreibung von Gradienten (z.B. für Änderung der räumlichen Position von Kompartimenten aufgrund von z.B. Gradienten)
- Ausgabe eines Codes in einer Markup Language (z.B. SBML), welcher von einem bestehenden Solver ausgeführt werden kann (z.B. Matlab)

1.3 ~~Problemstellung (oder Motivation)~~

Motivation durch Problemstellung!
signalig chains (SimuLink), Biologie. Motivation->Problem->Insiliko

$$\frac{dN}{dt} = f(N, c) \rightarrow \Delta N = f(N, c) \cdot \Delta t$$

$$N(t + \Delta t) = N(t) + \Delta N$$

$$\begin{aligned} a(t) &= \ddot{s}(t) = -g \\ v(t) &= \int a(t)\,dt = \int -g \cdot dt = -gt + v(0) \\ s(t) &= \int v(t) \cdot dt = \int (-gt + v(0)) \cdot dt = -\frac{1}{2}gt^2 + v(0)\,t + s(0) \\ v(t) &= \frac{d}{dt}\left[-\frac{1}{2}gt^2 + v(0)\,t + s(0)\right] = -gt + v(0) \\ a(t) &= \frac{d}{dt}\left[-gt + v(0)\right] = -g \end{aligned} \tag{2.3}$$

scst: kürzen -> warum?

Für die computerunterstützte Berechnung von Simulationen dieser Art können verschiedene Techniken angewandt werden. Dabei gibt es Grundsätzlich zwei verschiedene Vorgehen: symbolische oder numerische. Unsere Simulation übernimmt das numerische Verfahren, wie es auch Berkeley Madonna tut. Das Vorgehen beruht auf einer numerischen Approximation von Gewöhnlichen Differentialgleichungen. Diese Art kann nur Differentialgleichung 1. Ordnung berechnen. Eine Differentialgleichungen 2. Ordnung ist in Gleichung 2.4 zu sehen.

$$\ddot{s}(t) = -g \tag{2.4}$$

Eine Gewöhnliche Differentialgleichung n. Ordnung kann aber in n Differentialgleichungen 1. Ordnung umgeformt werden. In den Gleichungen 2.5 wird die Differentialgleichung 2. Ordnung in zwei Differentialgleichung1. Ordnung umgewandelt.

$$\begin{aligned} v(t) &= -g \\ \dot{s}(t) &= v(t) \end{aligned} \tag{2.5}$$

$\frac{dN}{dt} \rightarrow \frac{\Delta N}{\Delta t}$

2.3 Partielle Differentialgleichungen

Eine Partielle Differentialgleichung (engl. Partial Differential Equation PDE) ist eine mathematische Gleichung, die partielle Ableitungen enthalten, also Ableitungen von Funktionen mehrere Variablen. Eine Dichte, zum Beispiel Nahrung oder Ausscheidungen, überdeckt eine Fläche, zum Beispiel ein See, im $(x, y) = \mathbb{R}^2$ mit der Funktion $f(x\,y,\,t)$. Ein Algenvolk verändert diese Dichte über die Zeit aber auch bei jeder Position unterschiedlich. Bei komplexen Systemen kann es zu analytisch unlösbaren Differentialgleichungen führen. Generell können nur Lineare Partielle Differentialgleichung analytisch gelöst werden. Hier hilft die Numerik weiter, die zwar nur näherungsweise Lösungen ausgibt, doch besser eine Näherung als keine Lösung. Zwei wichtige numerische Verfahren sind die Finite-Differenzen-Methode (FDM) und die Finite-Elemente-Methode (FEM). Die Finite-Differenzen-Methode die für die Simulation benötigt werden sind das Gradienten-Verfahren oder die Diffusiongleichung, die später erläutert werden.

2.4 Einschrittverfahren

Euler

Das Euler-Verfahren, von **Leonard Euler** 1768 in seinem Buch *Institutiones Calculi Integralis* präsentiert, ist ein einfaches numerisches Verfahren. Von einer Schrittweite h multipliziert mit der Ableitung $y' = f$ zählt man den Anfangswert y_0 dazu und bekommt y_1.

$$k_i = f\left(t_n + hc_i,\ y_n + h\sum_{j=1}^{s} a_{ij}k_j\right),\ i = 1, ..., s \tag{2.10}$$

Für den ersten Parameter der Funktion $f\,(t\,y\,(t))$ benötigen wir einen Koeffizienten c_i, der die unabhängige Variable t_n variert. Jedes $c_i,\ i = 1, ..., s$ bildet eine Summe aller Elemente eine Zeile der Matrix $A = (a_{ij})$.

$$c_i = \sum_{j=1}^{s} a_{ij} \tag{2.11}$$

Klassisches Runge-Kutta

Um eine bessere Genauigkeit zu erhalten haben **Carl Runge** und **Martin Wilhelm Kutta** 1900, 60 Jahre vor der Präsentation des *Buchter Tableaus*, ein leistungsfähigeres Verfahren als Euler in Gleichung 2.6 entwickelt, die Differentialgleichungen numerisch zu lösen. Das heisst, sie kannten die Möglichkeit noch nicht, die Nachfolgen soll jedoch der Ansatz des des Buchter Tableau verwendet werden. Dabei rechnet das Klassische vier Zwischenschritte oder Stützstellen, arbeitet also mit Dimension $s = 4$. Die Tabelle und die dazugehörigen Gleichungen sind in Gleichung 2.12 zu finden.

was ist die Bedeutung ?

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

(2.12)

$$\begin{aligned}
k_1 &= f(t_n,\ y_n) \\
k_2 &= f(t_n + hc_2,\ y_n + h \cdot [a_{21}k_1]) \\
k_3 &= f(t_n + hc_3,\ y_n + h \cdot [a_{31}k_1 + a_{32}k_2]) \\
k_4 &= f(t_n + hc_4,\ y_n + h \cdot [a_{41}k_1 + a_{42}k_2 + a_{43}k_3])
\end{aligned}$$

hier fehlt definitiv Text !

$$\begin{aligned}
k_1 &= f(t_n,\ y_n) \\
k_2 &= f(t_n + h \cdot \frac{1}{2},\ y_n + h \cdot \frac{1}{2} \cdot k_1) \\
k_3 &= f(t_n + h \cdot \frac{1}{2},\ y_n + h \cdot \frac{1}{2} \cdot k_2) \\
k_4 &= f(t_n + h \cdot 1,\ y_n + h \cdot 1 \cdot k_3)
\end{aligned}$$

$$y_{n+1} = y_n + h \cdot \left(\frac{1}{6} \cdot k_1 + \frac{1}{3} \cdot k_2 + \frac{1}{3} \cdot k_3 + \frac{1}{6} \cdot k_4\right)$$

Dormand-Prince fR

[HB06]

(2.13)

Problem beschreiben: herausfinden was Problem ist / wo Problem ist und Lösungen finden

4.1 Programmiersprachen

Als Programmiersprache kam Java zum Einsatz. Java ist Plattformunabhängig ~~und sehr angenehm zum Programmieren~~. Es existieren gute Bibliotheken für grafische Darstellungen, und Java war bereits beiden Studierenden bekannt.

4.2 Markup Language

Was ist eine Markup Language?
Warum dieser Weg?

Als Markup Language für die Simulation kam der Matlab Syntax zum Einsatz, der ebenfalls vom Open Source Tool «octave» verarbeitet werden kann.

4.3 Entwicklungsumgebung

statt "Andreas Ba und Andreas Bu" besser "es wurde" -> nicht personifizieren

Bei Java ist man nicht fest an eine Entwicklungsumgebung gebunden. Das Projekt kann mit Hilfe der Ant-Buildfiles automatisch kompiliert werden, somit war es kein Problem das Andreas Bachmann IntelliJ und Andreas Butti Eclipse als Entwicklungsumgebung verwendet hat.

Es kam JDK 1.6 (Java 6) zum Einsatz, obwohl unterdessen Java 7 erschienen ist sind wir bei 6 geblieben, denn es ist immer noch sehr verbreitet, und Java 7 würde uns keinen Vorteil bringen.

4.4 Plugin Handling

Die Applikation hat einige Punkte, die sinvollerweise durch Plugins erweitert werden können. Es gibt bereits fertige Frameworks die Pluginhandlings ermöglichen, eines der bekannten ist Eclipse mit dem OSGi. Dieses basiert auf Extension Points und Extensions, welche wider Extensio Pointes bereitstellen können. Die komplette Applikation wird dann als Plugin aufgebaut, Abhängigkeiten können spezifiziert werden und Seiteneffekte können vermieden werden durch die Abschottung einzelner Plugins gegeneinander. Sogar das mehrfache Laden einer Library in verschiedenen Versionen ist ohne Probleme möglich.

Für unsere Applikation ist soviel Flexibilität aber nicht nötige, zudem würde dies auch einen hohen mehraufwand bei der Implementation bedeutet.

Unsere Pluginimplementation funktioniert daher viel einfacher:

1. Es wird ein Interface für ein Plugin definiert, dies geschieht in der Hauptapplikation
2. Ein Plugin implementiert diese Schnittstelle. Zudem wird ein XML File erstellt, indem steht
   a) Der Name des Plugins
   b) Eine Beschreibung
   c) Der Autor
   d) Die Klasse die geladen werden soll beim Start des Plugins
3. Der Javacode und das XML File werden in ein .jar gepackt und in einem vordefinierten Ordner abgelegt (meist konfigurierbar in config.properties)

Version ist ein Java Property File, mit zwei Einträgen:

version=1

compatible=1

«version» ist ein Ganzzahlwert, die Version die Datei, «compatible» ist ebenfalls eine Ganzzahl, die angibt mit welcher Version die Datei immer noch eingelesen werden kann, ohne das Probleme auftreten. Kleinere Unstimmigkeiten werden in Kauf genommen, aber das Modell kann immer noch eingelesen werden, somit ist die Rückwärtskompatibilität genau geregelt.

Wird eine Inkompatible Änderung am Dateiformat vorgenommen, so müssen beide Versionsnummern um eins hochgezählt werden, wird eine kompatible Erweiterung am Dateisystem vorgenommen so wird lediglich «version» hochgezählt.

mimetype ist ein Textfile mit einem einzigen String, «application/zhaw.simulation.project». Diese Datei wird oft verwendet um den Typ eines auf Zipbasierenden Dateiformats zu erkennen. Zum Beispiel verwendet .odt als mimetype «application/vnd.oasis.opendocument.text», dank solchen Kennungen kann Zipfile eindeutig als gültiges Simulationsfile erkennen.

configuration.xml enthält die Konfiguration der Plugins, z.B.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <simconfig>
3     <double name="simulation.dt" value="0.1"/>
4     <double name="simulation.end" value="5.0"/>
5     <double name="simulation.start" value="0.0"/>
6 </simconfig>
```

Diese Einstellungen können von den Plugins beliebig festgelegt werden, hier sind nur Einstellungen gespiechert, und keine Business Daten.

simulation.xml Enthält das Modell, oder bei einem XY-Model auch mehrere Modelle. Diese Datei sieht folgendermassen aus (gekürzt, kommentiert):

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <simulation>
3     <model grid="20" height="400" type="xy" width="400" zerox="200"
          zeroy="200">
4         <meso derivative="FIRST_DERIVATIVE" directionx="grad(&amp;quot;
             d1&amp;quot;, &amp;quot;x&amp;quot;)" directiony="1" name="m0" value
             ="model3" x="151" y="89"/>
5         <meso derivative="FIRST_DERIVATIVE" directionx="sin(1/2)"
             directiony="cos(1/2)" name="m1" value="model3" x="10" y=
             "88"/>
6         <meso derivative="FIRST_DERIVATIVE" directionx="1"
             directiony="1" name="m5" value="model2" x="256" y="281"/
             >
7         <global name="g0" value="5" x="252" y="37"/>
8
9         <density name="d1" text="" value="10*sin(x/20)+10*cos(y/20)
             "/>
10        <density name="d2" text="" value="x"/>
11
```

entstanden wäre

2. **BJavalibs:** Generelle Java GUI Libraries von Andreas Butti, dieser Code ist grösstenteils vor der BA entstanden
3. **Editor:** Abstrakter Editor für XY- und Flow Modelle. Beinhaltet alle basis Klassen, Globale Parameter, Clipboard, Undo / Redo, Toolbar / Menubar und den Formeleditor
4. **Editor.Flow:** Der Editor für Flussdiagramme, basierend auf «Editor»
5. **Editor.XY:** Der Editor für XY-Diagramme, basierend auf «Editor»
6. **ExternLibraries:** Dies ist nur ein Pseudoprojekt, und beinhaltet kein Code. Hier sind unsere externen Libraries untergebracht.
   a) JFreeChart: Diagramm Library
   b) JCommon: Wird von JFreeChart benötigt
   c) JXLayer: Wird verwendet um beim Simulieren den Editor Unscharf darzustellen und ein Statusdialog direkt im Editor darzustellen
   d) SwingX: Erweiterte Swing GUI-Komponenten, werden an diversen Stellen benötigt
7. **ImageExport:** Der «Speichern als Bild» Dialog
8. **ImportFilter:** Definition für Importplugins
9. **ImportFilter.Dynasys:** Dynsys Import Plugin
10. **ImportFilter.Madonna:** Berkeley Madonna Import Plugin
11. **Model:** Das «Domänenmodell», dies ist die interne Abbildung aller Daten die modelliert werden können.
12. **NetbeansDirchooser:** Dieser Code entstammt dem Netbeans Projekt, es ermöglicht die komfortable auswahl eines Ordners, die mit Swing Boardmittel wesentlich weniger konfortabel wäre: Es können die Ordner wie gewohnt als Baum ausgeklappt werden.
13. **OnscreenKeyboard:** Hierbei handelt es sich um die Tastatur für die Spezialzeichen, die bei Feldern im Diagramm zur Verfügung steht.
14. **Plugin:** Dies ist die Implementation um Plugins zu laden. Diese 285 Zeilen Code stammen aus einem vorgängigen Projekt von Andreas Butti
15. **Simulation:** Dies ist der Einsprungpunkt der Applikation. Hier befinden sich auch die globalen Komponenten
    a) About Dialog
    b) Einstellungen
    c) Speichern / Laden
16. **SimulationBuild:** Dies ist ein Pseudoprojekt das nur für den Build der Applikation benötigt wird, es enthält kein Code

Design Pattern

Die Applikation wurde nach MVC (Model View Control) bzw. Domänenmodell aufgebaut. Das Bedeutet das die Daten, die Logik und der View nur lose gekoppelt sind, und somit auch der View ersetzt werden könnte, ohne das der Rest angepasst werden müsste. Diese beiden Designpattern beschreiben ein ziemlich ähnliches vorgehen, wobei MVC sich hauptsächlich auf einen GUI Komponenten beziehen lässt, wie z.B. ein TreeView, während das Domänenmodell sich auf eine komplette Applikation anwenden lässt. Beides sind Vorgehensmuster, und keine exakten Vorgaben.

Das (Domänen)Modell befindet sich im Projekt «Model» und ist entsprechend von allem anderen entkoppelt. Die Simulation ist nur vom Model abhängig, und es besteht keine direkte Verbindung zur GUI. Die GUI selbst ist ein relativ grosser Teil der Applikation, und ist auch nicht überall klar abgegrenzt.

Komponeten wie Undo / Redo sind bei uns komplett in der GUI untergebracht, da diese auch komplett von dieser abhängig sind. Designtechnisch ist die Einordnung solcher Komponenten nicht eindeutig, denn z.B. Undo / Redo enthält ein Modell, nämlich die letzten Änderungen. Es enthält zudem Logik, es kann Änderungen Rückgängig machen / Wiederherstellen. Es ist aber ein GUI Komponent, denn es ist 100% von der GUI abhängig. Wenn wir jedoch definieren das es sich bei den Undo / Redo Daten nicht um Business Daten, sondern um Metadaten handelt sollte es auch nach den Pattern kein Problem sein wenn wir es Innerhalb der GUI unterbringen. Zudem war dies die einzig technisch sinvolle Möglichkeit.

5.2 Technische Beschreibung Softwarekomponenten

Model

Das Datenmodell ist wie folgendermassen aufgebaut: Der Einsprungpunkt ist das «SimulationDocument», welches entweder ein «SimulationFlowModel» oder ein «SimulationXYModel» beinhaltet. Sowohl das Flow als auch das XY Modell erben von , «AbstractSimulationModel», dieses beinhaltet «AbstractSimulationData», welches ebenfalls eine Abstrakte Klasse für alle Simulationskomponenten darstellt, wie

- Container
- Parameter
- Global
- Meso Kompartment.

«SimulationFlowModel» enthält zusätzlich noch Verbindungen, welche entweder ein Fluss oder eine Parameterverbidung darstellen. Abgebildet werden diese als «AbstractConnectorData<?>». Wobei der Fluss «FlowConnectorData» und der «ParameterConnectorData» die beiden möglichen Implementationen darstellen.

«SimulationXYModel» enthält zusätzlich zur Basisklasse noch Dichten, welche von der Klasse «DensityData» sind, hier besteht keine Abstraktion, es gibt nur eine Implementation.

Zudem kann ein XY Model beliebig viele Flow Modelle enthalten, welche über «SubModelList» gehandelt werden.

Die Modelle enthalten jede Menge an Methoden zur Manipulation der Daten, welche hier nicht aufgeführt werden, diese können der Codedokumentation oder dem Code entnommen werden.

unüblich ist und daher von Swing so nicht unterstützt wird.

Auch die Selektion wird nach dem Zeichen aller Komponenten einfach überzeichnet. Es gibt zudem die Möglichkeit das Komponenten einen Schatten zeichnen, dies wird angewendet um Abhängigkeiten von / zu Globalen darzustellen. Wir eine Globale angewählt werden alle abhängigen Elemente mit einem Schatten versehen.

Als Layoutmanager kommt «SimulationLayout» zum Einsatz, welches alle «AbstractDataView<?>» korrekt platziert. Alle anderen Komponenten werden nicht umplatziert, dies bedeutet wenn etwas anderes angezeigt wird muss dieses manuell mit «setBounds» platziert werden, was auch verwendet wird z.B. bei der Pfeil-erstell-UI (TODO Name?).

Die Clipboard implementation ist Architekturbedingt in mehrere Klassen aufgeteilt, unterstützt unter anderem das Kopieren als Rastergrafik. Das Exportieren als Vektorgrafik konnte nicht erfolgreich implementiert werden, da dies je nach Betriebsystem anders gehandhabt wird, und damit es z.B. unter Windows zuverlässig funktioniert muss eine Vektorgrafik (EMF / WMF) eingebettet in einem RTF exportiert werden, während unter Linux eher ein SVG direkt in die Zwischenablage kopiert wird. Alle Simulationsinternen Datenstrukturen werden in die Datenstruktur «TransferData» abgefüllt. Zudem werden alle Objekte mit IDs versehen, damit nach dem einfügen die Pfeile wider verbunden werden können.

Der Bilderexport gestaltet sich relativ einfach, dank der verwendeten «freehep» Library wird einfach mit dem Graphics von «freehep» «paint» aufgerufen, den eigentlichen Export übernimmt die Library.

Undo / Redo war und ist einer der umständlichen teile der Implementation, denn es muss für jede Aktion die der User vornimmt manuell eine Implementation vorgenommen werden, die diese Aktion auch wider Rückgängig machen kann. Dies ist auch beim erweitern der Applikation zu bedenken, denn wenn dies vergessen geht ist die Konsistenz nicht mehr gewährleistet, was zu Fehlern führt. Für das Debugging des Undo / Redo Mechanismus musste daher eine kleine Hilfe implementiert werden. Wir die Applikation mit dem Parameter "–debug-undo" aufgerufen erscheint ein Dialog der alle Aktionen des aktuellen Undo / Redo Handlers live auflistete, und je nach Status einfärbt. Zu bedenken ist, dass nach dem erstellen einer neuen Datei ein neuer Undo / Redo Handler erstellt wird, und der Dialog daher nicht mehr funktioniert.

Errorhandling / Logging

Bei Serverapplikationen ist es üblich das alle Aktionen geloggt werden, bei GUI Applikationen ist es weniger üblich, daher haben wir uns entschieden nur die Fehler zu loggen. Andere Ausgaben werden direkt auf STDOUT geschrieben, und sind somit nur lesbar wenn dieser entweder umgeleitet oder (AB)² Simulation direkt auf einer Konsole gestartet wird. Dies ist jedoch nur bei der Entwicklung sinnvoll, alle Meldungen die für den Benutzer relevant sind werden als Popups ausgegeben.

Ein wichtiger Punkt, der oft vergessen geht, ist, das die Exceptions im Eventloop auch abgefangen werden müssen. Dies geschieht mit "Errorhandler.registerAwtErrorHandler();" somit wird unserer eigener Errohandler als Eventloop errorhandler registriert, und kann somit alle Fehler die nicht gecatcht sind und im Eventloop auftreten abfange, dem User anzeigen und loggen.

Fehler die zu erwarten sind werden in unserer Applikation abgefangen, und dem Benutzer eine Entsprechende Meldung angezeigt. Zu erwartende Fehler sind z.B. Ungültige Benutzereingaben, nicht genügend Schreibrechte oder Fehler bei Formeln wie log(-1). Unerwartete Fehler sind z.B. IOException aufgrund eines kaputten Datenträgers, manipulierte Simulationsdateien etc.

Alle zu erwartenden Fehler werden den Benutzer mit «Messagebox.showError» dargestellt, während unerwartete Fehler mit «Errorhandler.showError(e, <optionale Beschreibung>);» dar-

- settingsPath=config/: Der Pfad für die Einstellungen (ggf. relativ zur, je nach «portable»)
- importPluginFolder=plugin/import/: Immer relativ zur App, wo die .jar's oder .xml's gesucht werden um die Plugins zu laden, hier die Importplugins für Fremdformate.
- simulationPluginFolder=plugin/simulation/: Siehe oben; Hier die Plugins zur Simulation
- predefinedDash: Die Liniendefinitionen für das Diagram. Getrennt mit Semikolon (;). Durchgezogene Linie: "0"; 5 Punkte gestrichelte Linie: "5 5"
- defaultsFonts: Die Fonts die verwendet werden für das Diagram, die erste Schrift, die installiert ist, der mit Semikolon (;) separierter Liste wird verwendet
- keyboard...: Definition des Sonderzeichen On-Screen-Keyboard, wird verwendet um bei den Diagrammen in den Namen Sonderzeichen einzugeben. Parameter definieren das Aussehen der Tastatur und werden hier nicht alle Im Detail erläutert. Die Zeichen die Angezeigt werden sind hier definiert, man beachte die Unicodeschreibweise (wird z.B. von Eclipse automatisch escaped): keyboard.keys=\u03B1;\u03B2;\u03B3;...

Im Settingsordner werden mehre Dateien abgelegt, alle Einstellungen die von der Applikation und den Plugins definiert werden sind in der Datei "settings.ini" abgelegt. Es ist nicht vorgesehen das die Datei von Hand editiert wird. Sollte Probleme mit den Einstellungen auftreten kann die Datei einfach gelöscht werden, es werden dann die Defaulteinstellungen verwendet.

Die Dateien endend auf ".windowPos" Speichern die Position der Fenster, um nach dem nächsten Start der Applikation alle Fenster wider an der gleichen Position anzuzeigen.

Dies kann zu probleme führen, z.B. wenn die Bildschirmkonfiguration verändert wurde / Beamer an / abgesteckt wurde. Um solche Probleme zu lösen, können einfach alle Dateien endend auf ".windowPos" gelöscht werden, dann werden beim nächsten Start alle Fenster auf dem Hauptbildschirm zentriert angezeigt.

5.3 Tests und Validierung

Test der Matheengine, Vergleich mit Berkeley-Madonna

6 Resultate

Beschreibung, Screenshots, Beispielsimulationen mit Ergebniss-Diagramm

7 Diskussion und Ausblick

Markup Language

Wir haben uns die Matlab Markup Language entschieden, da wir auf viele vordefinierten Funktionen ("Toolbox") zurückgreifen konnten, und uns somit den Mathematischen Teil teilweise vereinfachen.

Es haben sich jedoch immer wider Probleme mit den in Matlab vordefinierten Funktionen ergeben, grundsätzlich erfüllen die Funktionen unsere Anforderungen. Dies haben wir auch vorgängig abgeklärt, jedoch sind bei der Verwendung Probleme aufgetreten, die die Verwendung der Toolboxfunktionen verunmöglichten.

einer Zeile zugeordnet werden, es ist jedoch für den Anwender nicht offensichtlich wo der Ursprung des Fehlers liegt.

- Die Simulation kann einfach abgebrochen werden: Wenn die externe Simulation abgebrochen werden muss ist dies wesentlich komplizierter, und funktioniert ggf. nicht unter allen Umständen.

- Parallelisierung: Heutzutage besitzt jeder moderne PC mehr als einen Rechenkern. Mit Matlab / Octave ist es jedoch nicht möglich dies zu nutzen. Bei einer Internen Simulation wäre dies möglich, wenn auch aufwendig. Somit könnte die Simulation grösserer Modelle um Faktoren beschleunigt werden. (Voraussetzung: mehr als ein Core, und das Modell muss entsprechend unabhängige Teile aufweisen, ein Aufteilen eines einzelnen Integrals ist theoretisch zwar möglich, praktisch ist es jedoch langsamer als die Simulation auf einem CPU, da die Synchronisation sehr viel Leistung benötigt) => MPC

TODO

Was Fehlt noch, was muss noch gemacht werden? Interpretation und Validierung der Resultate Rückblick auf Aufgabenstellung, erreicht bzw. nicht erreicht Legt dar, wie an die Resultate (konkret vom Industriepartner oder weiteren Forschungsarbeiten; allgemein) angeschlossen werden kann; legt dar, welche Chancen die Resultate bieten 18

8 Verzeichnisse

8.1 Literaturverzeichnis

[HB06] HANKE-BOURGEOIS, Martin: *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens.* Bd. 2. Auflage. Teubner, 2006. – 551–628 S.

[Kos94] KOST, Arnulf: *Numerische Methoden in Der Berechnung Elektromagnetischer Felder.* Springer-Lehrbuch, 1994. – 20–38 S.

[Sch11] SCHEIDEGGER, Stephan: *Modelle in der medizinischen Biophysik.* 2011

8.2 Glossar

Anhänge

Meso-Kompartment: Ein Teil des XY-Simulationsmodells
Swing: Java GUI Framework