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

Abstract

Englische Version von “Zusammenfassung”

Zusammenfassung

Wird erst am Ende der Arbeit geschrieben

Inhaltsverzeichnis

1 Vorwort

In der Forschung aber auch zu Unterrichtszwecken werden viele Theorien vermittelt und Simuliert. Solche Simulationen basieren meistens auf einem Strikt Mathematischen Hintergrund, meinst Integrale erster Ordnung. Forschende Personen im Bereich Physik / Biologie verwenden dabei bevorzugt eine Modellierungssoftware, da es nicht ihr Fachgebiet ist selbst zu programmieren. Andreas Butti hat sich bei der Verwendung von Simulationstools über deren Plattformabhängigkeit und Benutzerunfreundlichkeit gestört. Als Informatiker kommt man da schnell in Versuchung selbst etwas besserer zu schreiben. Nach einem Gespräch mit dem Physikdozenten, Herr Scheidegger, wurde daraus dann diese BA, die jedoch nicht nur ein Benutzerfreundliches Simulationswertzeug sein soll, sondern auch bisher nicht vorhandene Möglichkeiten für die Simulationen von Biologischen Abläufen, wie das innere einer Zelle, darstellen soll. TODO: Danksagung

2 Einleitung

Es gibt bereits viele Simulationstools, wie z.B. Barekely Madonna um nur ein bekanntes Beispiel zu nennen. Diese Tools unterstützten die Modelierung von Differentialgleichungen als Modell, mit Containern und Flüssen. Diese modellart ist schon lange bekannt, und etabliert, daher übernehmen wir dieses Prinzip.

2.1 Bereits existierende Tools

2.2 Individualität unserer BA

Mit unserer Simulation ist es zusätzlich möglich in einem XY Modell mehrere Meso Kompartmente abzubilden, ein Meso Kompartment ist das vorhin genannte Flussmodel. Diese Meso Kompartmente können sich wären der Simulation im XY-Raum bewegen. Es können Dichten angegeben werden, die über den XY Raum verteilt sind, und die Meso Kompartmente können an Ihrer aktuellen Position von der dichte Konsumieren oder dichte Produzieren, somit kann die Umgebung beeinflusst werden. Mit diesen Fähigkeiten ist es möglich das innenleben einer Zelle oder andere Biologische Prozesse einfach, grafisch abzubilden. Die Idee und Vorgabe dieser Simulationsmethode stammt von Herr Scheidegger, und wurde zusammen mit Herr Füchslin und uns ausgearbeitet.

2.3 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

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)

3 Übersicht

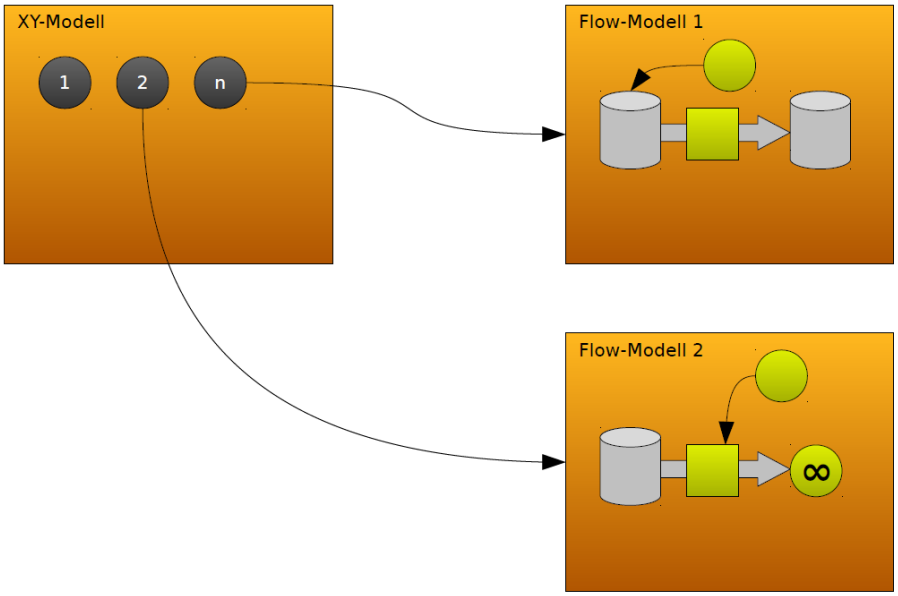

Abbildung 3.1: bli

3.1 Legende

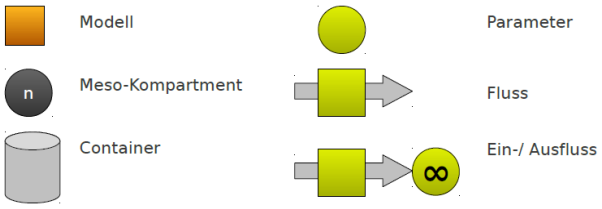

Abbildung 3.2: bla

Es kann entweder ein Herkömmliches «Flow-Modell» erstellt werden, das bereits bekannt ist da es vom Konzept her identisch bereits von vielen Simulationstools angeboten wird. Oder es kann

ein «XY-Modell» erstellt werden, das dann Meso Kopartments beinhaltet, diese befinden sich an einer Position (X / Y) und verweisen auf ein Modell («Flow-Model-X»). Im «XY-Modell» könne sich «Dichten» befinden, das sind Stoffe die eine gewisse Konzentration an einer gewissen Stelle aufweisen. Ein Meso Kompartment kann sich wären der Simulation im «XY-Modell» bewegen, es kann dichten Konsumieren oder Produzieren.

4 Theoretische Grundlagen

4.1 Gewöhnliche Differentialgleichungen

Eine Gewöhnliche Differentialgleichung (engl. Ordinary Differential Equation ODE) ist eine mathematische Gleichung, die Ableitungen, die Funktion selbst sowie die unabhängige Variable enthalten kann. Ableitungen und die Funktion selbst treten nach genau einer unabhängigen Variable auf. Lösung $y\left(t\right)$ einer Differentialgleichung $y^{(n)}$ist eine Funktion, die mit ihren Ableitungen deckungsgleich mit der Differentialgleichung selbst ist.

$$y^{(n)} = f\left(t,\ y\left(t\right),\ y'\left(t\right),\ \ldots,\ y^{(n-1)}\left(t\right)\right) \tag{4.1}$$

Auf analytischem Weg kann eine Differentialgleichung durch Integration erfolgen. Die Fallbeschleinigung in Abbildung soll in Gleichung 4.2 als Beispiel gezeigt werden. Wir kennen die Beschleunigungs-Funktion $a\left(t\right)$, suchen die Weg-Funktion $s\left(t\right)$, finden durch Integration eine Lösung und kommen durch abermaliges Differenzieren wieder auf die Ursprüngliche Beschleunigungs-Funktion zurück.

$$\begin{aligned}
a\left(t\right) &= \ddot{s}\left(t\right) = -g \\
v\left(t\right) &= \int a\left(t\right) \cdot dt = \int -g \cdot dt = -gt + v\left(0\right) \\
s\left(t\right) &= \int v\left(t\right) \cdot dt = \int \left(-gt + v\left(0\right)\right) \cdot dt = -\frac{1}{2}gt^2 + v\left(0\right)t + s\left(0\right) \\
v\left(t\right) &= \frac{d}{dt}\left[-\frac{1}{2}gt^2 + v\left(0\right)t + s\left(0\right)\right] = -gt + v\left(0\right) \\
a\left(t\right) &= \frac{d}{dt}\left[-gt + v\left(0\right)\right] = -g
\end{aligned} \tag{4.2}$$

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 4.3 zu sehen.

$$\ddot{s}\left(t\right) = -g \tag{4.3}$$

Eine Gewöhnliche Differentialgleichung n. Ordnung kann aber in n Differentialgleichungen 1. Ordnung umgeformt werden. In den Gleichungen 4.4 wird die Differentialgleichung 2. Ordnung in zwei Differentialgleichung1. Ordnung umgewandelt.

$$\begin{aligned} \dot{v}(t) &= -g \\ \dot{s}(t) &= v(t) \end{aligned} \tag{4.4}$$

In unserer Simulation haben wir verschiedene numerische Verfahren implementiert, die wir nachfolgen kurz behandeln werden.

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.

$$\begin{aligned} y' &= f(t,\ y(t)) \\ y_{n+1} &= y_n + h \cdot f(t_n,\ y_n) \end{aligned} \tag{4.5}$$

In unserem Beispiel mit der Fallbeschleunigung müssen wir das Verfahren zwei Mal anwenden pro Zeitschritt, da wir eine Differentialgleichung 2. Ordnung lösen möchten. Dabei müsen die Anfangswerte v_0 und s_0 sowie die Schrittweite h bekannt sein.

$$\begin{aligned} h &= 0.1 \\ v_0 &= 15 \\ s_0 &= 0 \\ v_1 &= 15 + 0.1 \cdot (-9.81) = 14.019 \\ s_1 &= 0 + 0.1 \cdot [15 + 0.1 \cdot (-9.81)] = 1.4019 \end{aligned} \tag{4.6}$$

Bei Differentialgleichungen 1. Ordnung und kleiner Schrittweite h erhält man ausreichende Genauigkeit. Unser Beispiel mit 2. Ordnung büst bei jedem Schritt an Genauigkeit ein. Es gibt bessere Verfahrung, die auch Ordnungen höheren Grades mit weniger Genauigkeitsverlust zulassen.

Butcher Tableau

Um weitere Verfahren besser zu Verstehen, wird zuerst eine Tabelle eingeführt. Die Tabelle wurde in den 1960er Jahren von **John Charles Butcher** entwickelt für den besseren Umgang mit dem nachfolgenden Runge-Kutta-Verfahren und nennt sich *Butcher tableau* in Gleichung 4.7. Die Tabelle beinhaltet einen Vektor c_i, eine Matrix $A = (a_{ij})$ und einen Vektor b_i wobei $i, = 1, ..., s,\ j = 1, ..., s$ die Zeilen- und Spalten-Indizes und s die Dimension ist.

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} \tag{4.7}$$

Wie beim Euler-Verfahren in Gleichung 4.5 ergibt sich ein neuer Wert y_{n+1} aus dem alten Wert y_n addiert mit einer festen Schrittweite h multiplitiert mit der Ableitung. Doch beim Runge-Kutta-Verfahren werden statt einer Ableitung verschieden gewichtete Ableitungen $b_i k_i$ aufsummiert, wobei das Gewicht b_i und die Ableitung k_i ist.

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \tag{4.8}$$

Eine Ableitung, auch Zwischenschritt genannt, ist in Gleichung 4.9 erläutert. Je grösser die Dimension s ist, desto mehr Zwischenschritte werden berechnet. Zu Beginn jedes Schrittes wird der Vektor k_i zurückgesetzt.

$$k_i = f\left(t_n + hc_i,\ y_n + h \sum_{j=1}^{s} a_{ij} k_j\right),\ i = 1, ..., s \tag{4.9}$$

Für den ersten Parameter der Funktion $f\left(t\, y\left(t\right)\right)$ 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 $A = (a_{ij})$.

$$c_i = \sum_{j=1}^{s} a_{ij} \tag{4.10}$$

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 entwickelt Differentialgleichungen numerisch zu lösen.

Dormand-Prince f

(4.11)

4.2 Gradienten-Verfahren

5 Werkzeuge und Hilfsmittel

5.1 Programmiersprachen

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

5.2 Markup Language

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

5.3 Entwicklungsumgebung

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.

5.4 Testing

Die Zeit hat leider nicht gereicht eine komplette Testumgebung einzurichten. Das Automatische Testen einer GUI-Applikation ist auch nicht ganz trivial. Trotzdem wurden einzelne TestCases für JUnit geschrieben, jedoch nicht für die GUI. Auf nachfrage war auch den Unterrichtenden Java Dozenten an der ZHAW keine Lösung bekannt die für GUI Applikationen allgemein funktionieren würde. Da sowieso zuwenig Zeit vorhanden war wurde das automatische Testing der GUI dann Ersatzlos gestrichen.

6 Vorgehen

6.1 MVC

6.2 Tests und Validierung

Test der Matheengine, Vergleich mit Berkeley-Madonna

7 Resultate

Beschreibung, Screenshots, Beispielsimulationen mit Ergebniss-Diagramm

8 Diskussion und Ausblick

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

9 Verzeichnisse

9.1 Literaturverzeichnis

9.2 Glossar

Meso-Kompartment: Ein Teil des XY-Simulationsmodells