Explore

Ein Experimentier-Framework für zeitintensive Simulationen

Ontje Lünsdorf[1] und Michael Sonnenschein[2]

Zusammenfassung

In diesem Paper wird das Explore-Framework vorgestellt, welches bei der explorativen Modellbildung unter anderem häufig wiederkehrende Arbeitsschritte erleichtern und die Auslagerung von Berechnungen ermöglichen soll. Aus einer Analyse des Modellbildungsprozesses werden Anforderungen für diese Zielsetzung abgeleitet und die Umsetzung dieser durch das Framework dargestellt. Anhand eines Beispiels wird die Arbeitsweise mit dem Framework verdeutlicht. Ein abschließender Vergleich ordnet das Framework in die bestehende Softwarelandschaft ein.

1 Einleitung

Bei der Entwicklung neuer Simulationsmodelle werden dieselben Methoden wie im allgemeinen Problemlösungsprozess angewendet. Man bedient sich verschiedener Heuristiken um sich der Lösung eines Problems schrittweise anzunähern. Die einfachsten und universellen Heuristiken, wie z.B. Divide-and-Conquer oder Trial-and-Error, erzeugen Zwischenergebnisse, die anschließend ausgewertet werden müssen. Verbessert sich das Ergebnis durch die Heuristik, verwendet man den modifizierten Lösungsansatz in einem neuen Schritt. Diese Abfolge wird solange wiederholt, bis die optimale Lösung gefunden worden ist.

Der Prozess der Modellbildung folgt diesem allgemeinen Prinzip. In den einzelnen Iterationen des Modellbildungsprozesses müssen stets dieselben Arbeitsschritte (Modifikation, Simulation, Analyse, Kalibration) ausgeführt werden. In diesem Paper

[1] OFFIS Institut für Informatik, Escherweg 2, FuE-Bereich Energie, D-26121 Oldenburg,

Email: ontje.luensdorf@offis.de

[2] Carl-von-Ossietzky-Universität Oldenburg, Uhlhornsweg 84, Umweltinformatik, D-26111 Oldenburg,

Email: michael.sonnenschein@informatik.uni-oldenburg.de

wird ein Softwaretool vorgestellt, das unterstützend für diesen Modellbildungsprozess eingesetzt werden kann.

Die Analyse des Prozesses in Abschnitt 2 identifiziert organisatorische, funktionale und kooperative Aspekte, die die Modellbildung vereinfachen können. Aus den Aspekte werden anschließend Anforderungen an Explore abgeleitet, welche mit der in Abschnitt 3 beschriebenen Architektur erfüllt werden. In Abschnitt 4 wird die Arbeitsweise mit dem Tool anhand eines Beispielexperiments demonstriert. Abschnitt 5 stellt einen funktionalen Vergleich zwischen Explore und bestehenden Tools an. Abschnitt 6 schließt das Paper mit einem Fazit und Ausblick.

2 Problemstellung

Simulationsmodelle sollen einen Ausschnitt der Realität widerspiegeln. Dazu ist die Identifikation von Entitäten und deren Wirkungsbeziehungen mit anschließender Abbildung in ein Modell nötig. Prinzipiell ist der Entwicklungsprozess von Simulationsmodellen analog zu dem von Computerprogrammen. Das Modell bzw. der Programmcode wird schrittweise um Wirkungsbeziehungen bzw. Funktionalitäten ergänzt, bis ein Ziel erreicht ist. Allerdings ist der Entwicklungsprozess bei Simulationsmodellen im Allgemeinen explorativer Natur und verläuft nicht wie die Entwicklung von Computerprogrammen linear ab. Die Simulation eines Modells liefert Ergebnisse, die mit der Realität abgeglichen werden müssen. Ist die Übereinstimmung unzureichend, müssen Adaptionen an dem Modell vorgenommen werden. Im Modellbildungsprozess kommt es daher häufiger zu Refaktorisierungen als dies bei der Programmentwicklung der Fall ist.

Fehler! Verweisquelle konnte nicht gefunden werden. zeigt ein von Jørgensen und Bendoricchio (Jørgenson 2001) empfohlenes Schema des Modellbildungsprozesses. Das Schema läuft iterativ ab, wie durch die linke Feedback-Schleife von „Calibration followed by validation“ zu „Conceptual diagram“ verdeutlicht wird. Innerhalb dieser Schleife findet die Refaktorisierung des Modells statt. Im letzteren Schritt können aufgrund der vorhergehenden Analyse neue Wirkungsbeziehungen integriert bzw. bestehende angepasst oder aus dem Modell entfernt werden. In (Jakeman 2006) wird für eine größere Auswahl an Modellfamilien auf eine ähnliche Art und Weise ein iterativer Modellbildungsprozess vorgeschlagen.

Charakteristisch am explorativen Prozess ist, dass auch Lösungsstrategien verfolgt werden können, die sich trotz ursprünglicher Vermutung als nicht zielführend erweisen. In solch einem Fall muss der bisherige Lösungsweg zurückverfolgt und eine andere Strategie Anwendung finden.

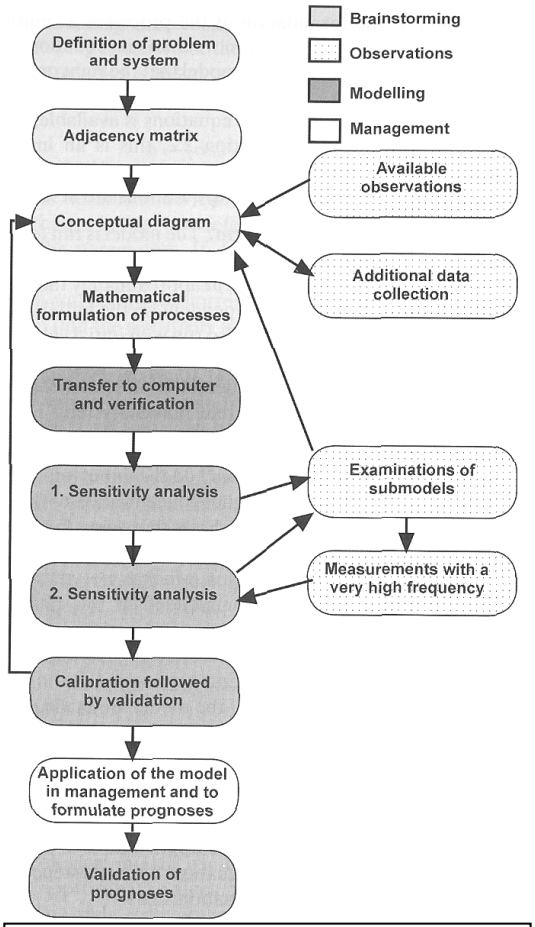

Abbildung 1: Modellbildungsschema (aus Jørgenson 2001)

Übertragen auf den Modellbildungsprozess entspricht dieser Schritt der Wiederherstellung einer früheren Revision des Modells, ab der eine andere Strategie sinnvoll erscheint. Dabei ist zu bemerken, dass auch die Resultate der Modellanalysen wiederherzustellen sind. Denn nur anhand dieser lässt sich die Wahl einer Strategie begründen. Somit ist ein Revisionsmanagement als erste Anforderung an ein Tool festzuhalten.

Aufgrund der explorativen Natur der Modellbildung sollte eine Modellierungsprache verwendet werden, die sich gut zur Anwendung der aus der Softwareentwicklung stammenden agilen Methode anbietet. Die agile Methode zeichnet sich nach (Larman 2003) unter anderem durch eine iterative und evolutionäre Entwicklung und einer sich den ändernden Anforderungen anpassende Planung aus.

In jeder Iteration müssen Sensitivitätsanalysen angestellt werden, welche die Simulation des Modells bedingen. Simulationen müssen ebenfalls in der abschließenden Anwendungsphase durchgeführt werden. Je nach Komplexität des Modells benötigt eine Simulation unterschiedliche Rechenzeiten. Durch die Auslagerung der Simulationen auf größere Kapazitäten kann sich dieser Vorgang beschleunigen lassen. Ein unterstützendes Tool muss diese Auslagerung und auch die damit verbundene Überwachung und Steuerung der Simulation ermöglichen. Um möglichst flexibel eingesetzt werden zu können, sollte das Tool zudem auch auf allen

gängigen Plattformen ausführbar sein.

Die Effizienz, mit der sich ein explorativer Prozess dem Ziel nähert, hängt von der eingebrachten Erfahrung ab. Analog zur Softwareentwicklung profitiert also auch der Modellbildungsprozess von der Kooperation im Team. Alle Arbeitsschritte, die während der Modellbildung anfallen, sollten deshalb parallel bearbeitet werden können. Zu den wichtigsten gehören z.B. die Parametrisierung eines Modells, sowie dessen Analyse und Refaktorisierung. Damit diese Arbeitsschritte parallel im Team bearbeitet werden können, muss der Zugriff und die Steuerung aller beteiligten Komponenten (Modell, Simulation, Daten) von beliebigen Arbeitsplatzrechnern über das Netzwerk erfolgen. Die Parametrisierung des Modells erfordert außerdem eine Benutzeroberfläche, welche auch über Visualisierungsmöglichkeiten zur Aufbereitung der Simulations- als auch der Analysedaten verfügen muss.

2.1 Anforderungen

Zusammengefasst ergeben sich die folgenden Anforderungen an Explore:

- Revisionierung von Modellen und Resultaten
- Auslagerung von Simulationsläufen auf größere Kapazitäten
- Plattformunabhängigkeit
- Unterstützung der agilen Methode
- Kooperationsunterstützung durch:
    - Netzwerktransparenz
    - Parametrisierung
    - Visualisierung

Im anschließenden Abschnitt wird die Architektur beschrieben, mittels der Explore diese Anforderung zu erfüllen versucht.

3 Umsetzung

Explore ist in der objekt-orientierten, dynamischen Programmiersprache Python[3] geschrieben. Dynamische Programmiersprachen bieten sich besonders für die agile Entwicklung an. Gegenüber anderen dynamischen Sprachen zeichnet sich Python durch die zahlreichen Anbindungen an bestehende Bibliotheken aus. Dank dieser Umstände ist Python im wissenschaftlichen Kontext stark vertreten, z.B. durch die Konferenz SciPy[4].

3.1 Begrifflichkeiten

Für das Verständnis der folgenden Übersicht, sind zunächst einige Begrifflichkeiten zu erläutern:

- Experiment: Hierbei handelt es sich um das vom Benutzer in Python geschriebene Simulationsmodell. Ein Experiment kann Schnittstellen, wie z.B. eine grafische Oberfläche, bereitstellen, mittels derer Clients auf das Experiment zugreifen können. Über diese Schnittstellen kann die Parametrisierung des Experiments erfolgen.
- Frontend: Das Frontend bezeichnet die Benutzungsoberfläche mittels derer Benutzer Experimente parametrisieren, überwachen und analysieren können.
- Backend: Das Backend ist ein Server welcher die Laufzeitumgebung für ein Experiment darstellt. Das Backend publiziert alle Schnittstellen des Experiments und ermöglicht den Zugriff von Frontends über das Netzwerk.
- Metaserver: Der Metaserver verwaltet die Revisionen von Experimenten. Dies umfasst auch die von einem Experiment errechneten Daten. Der Metaserver dient als Sammelpunkt für Clients die mit verschiedenen Experimenten oder auch verschiedenen Revisionen eines Experiments arbeiten wollen.

3.2 Module

Die Architektur von Explore ist streng modular gehalten. Einige der Module können

[3] Python: http://www.python.org/

[4] SciPy: http://conference.scipy.org/

daher auch unabhängig vom Framework verwendet werden. In Abbildung 2 sind die Hauptmodule und einige Teilmodule von Explore und die in diesen enthaltenen

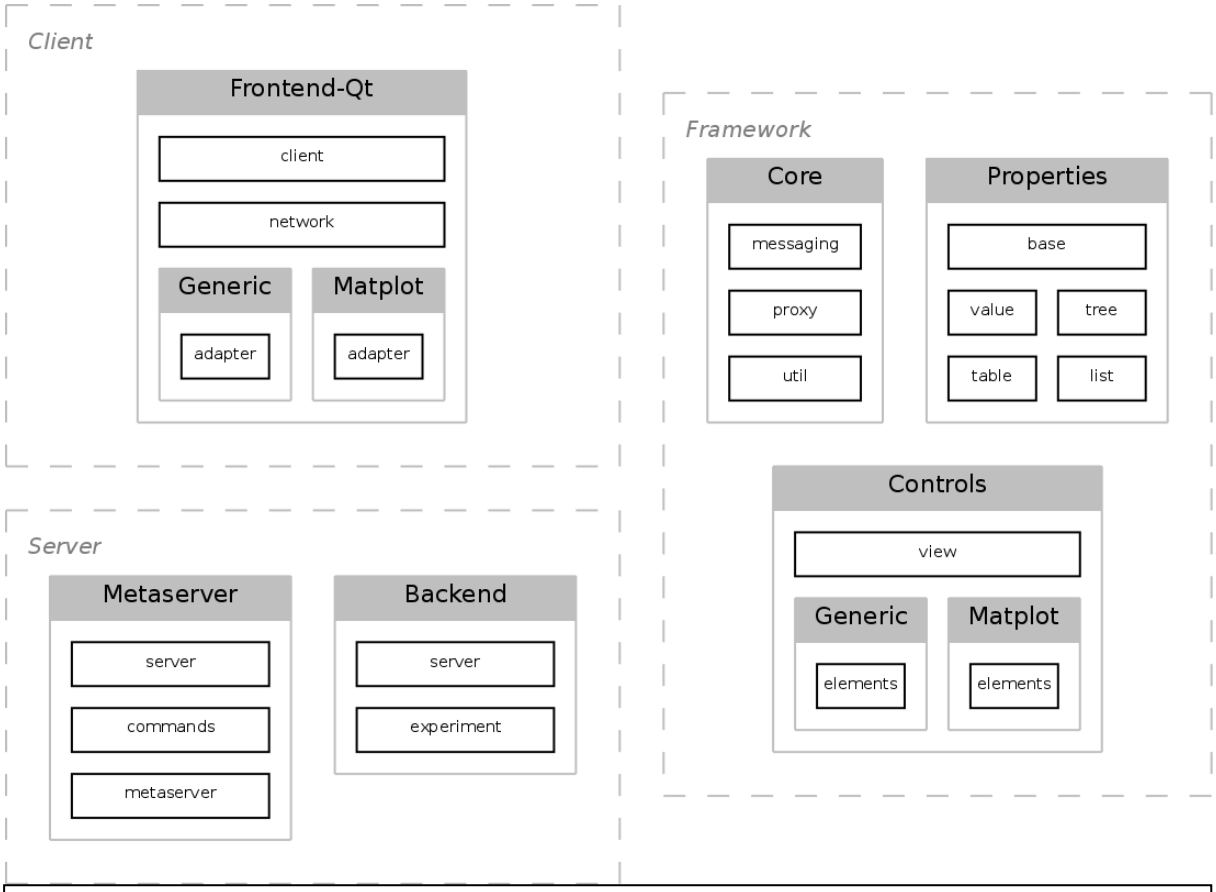

Abbildung 2: Explore Modulaufteilung

Klassen dargestellt. Die wichtigsten Module werden im Folgenden beschrieben.

3.2.1 Core

Das Kernmodul von Explore enthält die Netzwerkfunktionalitäten. Sie regelt den Nachrichtenverkehr zwischen dem Backend und den Frontends. Das Modul bietet dabei Mechanismen an, mittels derer vollkommen Netzwerk-transparent auf die Komponenten des Experiments zugegriffen werden kann. Die Experimente sind damit ohne Eingriff des Benutzers von entfernten Rechnern aus steuerbar.

3.2.2 Control

Das Control-Modul besteht aus einer Sammlung von Elementen für die

Benutzungsoberfläche. Hervorzuheben ist, dass diese Elemente von einer tatsächlichen Realisierung abstrahiert sind und die Benutzungsoberfläche somit nur funktional beschrieben wird. Der Benutzer kann die Control-Elemente in einem Experiment einsetzen, um z.B. dynamische Parametrisierungen und Visualisierungen zu ermöglichen. Das Control-Modul selbst ist ebenfalls modular gestaltet, um eine einfache Erweiterung um Experiment-spezifische Benutzungskomponenten zu gewährleisten. Explore verfügt derzeit über zwei Control-Teilmodule:

- Generic: Dieses Modul umfasst Standardelemente wie Textboxen, Tabellen und Schaltflächen. Das Modul muss von jedem Frontend-Modul realisiert werden.
- Matplot: In diesem Modul sind Elemente zur Visualisierung von Daten enthalten. Mittels der Bibliothek Matplotlib[5] können verschiedene Arten von Graphen erstellt werden.

Da Clients nicht alle Control-Teilmodule implementieren müssen, können Experimente über mehrere sogenannte Views verfügen. Ein View ist eine Benutzungsoberfläche, welche aus Control-Elementen besteht. Bspw. können so komplexe Experimente, die über umfangreiche Visualisierungen verfügen, zwei Views bereitstellen: einen einfachen View, der lediglich Control-Elemente aus dem Generic-Teilmodul verwendet und somit von jedem Client angezeigt werden kann und komplexere Views, welche zusätzliche Control-Teilmodule zur Visualisierung verwenden, deshalb aber nicht von allen Clients darstellbar sind.

3.2.3 Frontend / Backend

Das Frontend-Modul besteht aus dem Client zur Interaktion mit dem Backend-Modul. Das Backend-Modul stellt dabei alle Schnittstellen wie z.B. die oben beschriebenen Views eines Experiments bereit.

Zur Darstellung der Benutzungsoberflächenelemente aus dem Control-Modul verwendet das Frontend-Modul native Bibliotheken. Wie das Control-Modul ist auch das Frontend modular aufgebaut. Einzige Anforderung an ein Frontend, neben der Implementierung eines Clients, ist die Realisation des Generic-Teilmoduls.

[5] Matplotlib: http://matplotlib.sourceforge.net/

Aufgrund der Trennung zwischen Front- und Backend muss auf Arbeitsplatzrechnern nur das Frontend mit seinen Abhängigkeiten installiert sein. Alle Bibliotheken, die vom Backend (also dem Experiment) benötigt werden, müssen nur auf dem Rechner installiert sein, der das Backend ausführt. Dank dieser Trennung kann von jedem Arbeitsplatzrechner aus auf ein beliebiges Experiment zugegriffen werden, ohne dass die Installation dessen Bibliotheken erforderlich ist. Analog benötigt die Ausführung des Backends keine Visualisierungs- oder Benutzungsoberflächenbibliotheken.

Explore verfügt zur Zeit über ein Frontend, das Qt[6] als native Bibliothek verwendet. Qt ist eine Softwarebibliothek, mit welcher plattformunabhängige Benutzungsoberflächen realisierbar sind. Damit ist dieses Frontend auf Windows, Mac und Unix Betriebssystemen einsetzbar.

3.2.4 Metaserver

Der Metaserver verwaltet, wie eingangs erwähnt, mehrere Revisionen eines Experiments. Dabei bündelt er sowohl den Experiment-Quelltext als auch berechnete Resultate. Experimente werden vom Metaserver ausgeführt, wobei hervorzuheben ist, dass der Metaserver für jedes Experiment eine exklusiven Umgebung erstellt. In dieser Umgebung werden alle Abhängigkeiten zu dritten Bibliotheken installiert. Dieser Schritt ist notwendig, um die Reproduzierbarkeit von Resultaten zu gewährleisten. Die Reproduktion eines Resultats hängt nicht nur von dem Experiment-Quelltext ab, sondern auch von den verwendeten Bibliotheken. Würde eine gemeinsame Umgebung verwendet werden, ist nicht mehr zu garantieren, dass bei der Reproduktion eines Resultats dieselben Bibliotheken wie bei der ursprünglichen Berechnung verwendet werden.

Der Quelltext eines Experiments lässt sich über einen Befehl zu einem Metaserver hochladen. Das Experiment kann nun vom Benutzer instanziiert werden. Während der Ausführung können Frontends auf die Views des Experiments zugreifen und Parametrisierungen vornehmen. Beendet sich das Experiment, erzeugt der Metaserver eine neue Revision bestehend aus Umgebung, Quelltext und Resultaten. Mit diesen Informationen kann die Revision, zu Zwecken der Reproduktion jederzeit

[6] Qt: http://www.qtsoftware.com/products/products/appdev

erneut gestartet werden.

3.2.5 Experiment

Das Experiment ist das zentrale Modul aus der Perspektive des Anwenders. Ein Experiment umfasst das Modell, dessen Parameter und die Simulation. Dem Anwender steht dabei frei, welche Modellart und Simulationsmethode verwendet wird.

Auf Implementationsebene ist das Experiment eine Klasse mit drei zu implementierenden Methoden:

- start: Startet das Experiment.
- stop: Beendet das Experiment.
- reset: Setzt alle bisher berechneten Ergebnisse auf den Ursprungswert zurück.

Zusätzlich muss das Experiment über einen Konstruktor verfügen, der alle notwendigen Initialisierungen durchführt (z.B. Views erzeugen).

Über die start, stop und reset Methoden kann der Zustand eines Experiments verändert werden. Nach der Konstruktion eines Experiments befindet es sich im Ausgangszustand und ist nicht aktiv. In diesem Zustand können Änderungen an den Parameterwerten vorgenommen werden. Dazu kann in einem Experiment ein View erzeugt werden, welcher Control-Elemente zur Parametrisierung enthält. Zur Parametrisierung werden sogenannte Properties angeboten, mittels derer ein Parameterwert durch ein Control-Element modifiziert werden kann. Nach erfolgter Parametrisierung kann das Experiment über die start-Methode in den aktiven Zustand versetzt werden. Gemäß der Parametrisierung führt das Experiment seine Berechnungen durch. Sobald alle Berechnungen abgeschlossen sind, versetzt sich das Experiment wieder in den inaktiven Zustand. Sind die berechneten Resultate zufriedenstellend, kann das Experiment über die stop-Methode beendet werden. Andernfalls können die Resultate und Parameterwerte über die reset-Methode zurückgesetzt werden und eine andere Parametrisierung vorgenommen werden.

4 Anwendung

In diesem Abschnitt soll nun ein Einblick in die Arbeitsweise mit Explore gegeben werden. Dazu soll zunächst ein Workflow vorgeschlagen werden, der sich am explorativen Modellbildungsprozess orientiert. Die Vorstellung eines Beispielexperiments vermittelt anschließend einen Eindruck der Benutzungsoberfläche von Explore.

4.1 Workflow

Der von Explore vorgesehene Workflow besteht aus den folgenden Arbeitsschritten:

1. Erstellen des Experiments. Neben dem Quelltext des Experiments gehört dazu auch eine Auflistung der benötigten Bibliotheken.
2. Hochladen des Experiments auf einen Metaserver.
3. Ausführung des Experiments durch den Metaserver. Dabei erzeugt dieser wie oben beschrieben eine eigene Laufzeit-Umgebung welche die Reproduzierbarkeit der Resultate gewährleistet.
4. Analyse der Resultate. Dazu können neben externen Tools bereits die in Explore integrierten Control-Elemente zur Visualisierung eingesetzt werden.
5. Bei zufriedenstellenden Resultaten ist das Experiment abgeschlossen. Andernfalls erfolgt eine Refaktorisierung bzw. Ergänzung des Quelltextes und eine neue Iteration ab Schritt 2.

4.2 Beispielexperiment

Dem Beispielexperiment liegt die Frage zugrunde, ab welcher Anzahl zufällig parametrisierter Kühlschränke sich ein stabiler Gesamtstromverbrauch einstellt. An eine solche Gruppe von Kühlschränken können Steuersignale gesendet werden und der Verbrauch ohne Kenntnis der Einzelzustände vorhergesagt werden. Ein Planungssystem würde deshalb nur einen unidirektionalen Kommunikationskanal zur Steuerung der Kühlschränke benötigen. Bei diesem Experiment wurde das Kühlschrankmodell aus (Stadler et al. 2007) verwendet und mittels der Bibliothek

SimPy[7] implementiert.

Ein Kühlschrank verfügt über Parameter wie z.B. Außentemperatur, erlaubter Innentemperaturbereich, Stromverbrauch, Kühlungseffizienz, usw. Das Experiment erlaubt es für jeden Parameter einen Wertebereich anzugeben, aus dem für jeden Kühlschrank zufällig ein Wert ausgewählt wird. Anschließend wird eine ebenfalls parametrisierbare Anzahl an Kühlschränken erzeugt und deren Stromverbrauch simuliert. Der Verbrauch kann während der Simulation an einem Graphen abgelesen

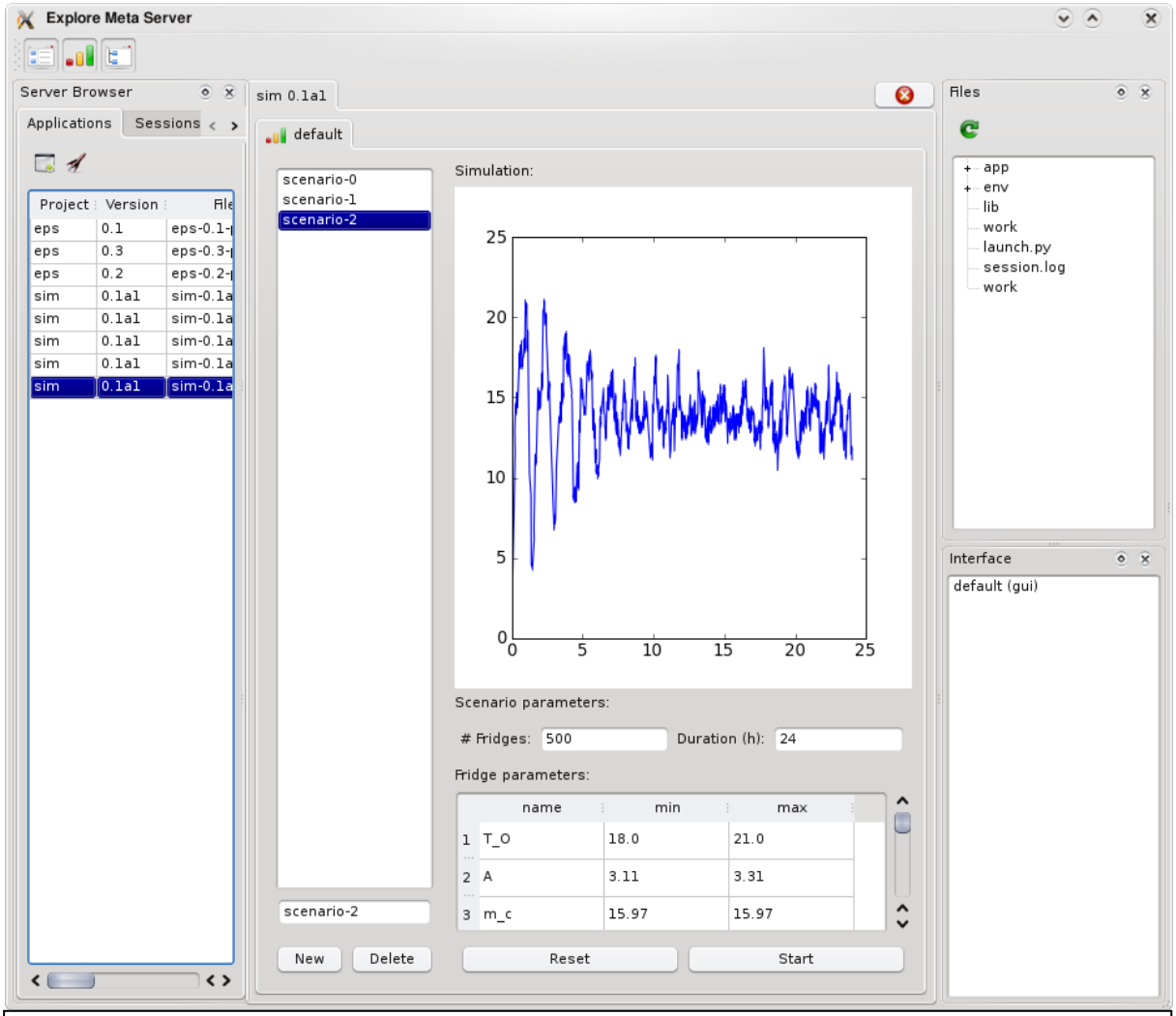

Abbildung 3: Screenshot des Qt-Frontends

werden.

In Abbildung 3 ist der Client des Qt Frontends abgebildet. Im linken Bereich „Server Browser“ ist die Liste mit Experiment Revisionen zu sehen. Die älteren Revisionen

[7] SimPy: http://simpy.sourceforge.net/

stammen aus früheren Schritten des Workflows und können jederzeit erneut ausgeführt werden, sollte ein Rückschritt erforderlich sein. Die neueste Revision des Experiments ist in diesem Beispiel aktiv.

Im rechten Bereich sind die Dateien der Experimentumgebung inklusive berechneter Daten unter „Files“ zu sehen, während unter „Interfaces“ alle vom Experiment exposierten Views aufgelistet sind. Von diesen ist der „default“-View im mittleren Bereich dargestellt. Über diesen View können die Kühlschränke parametrisiert und die Simulation des Stromverbrauchs, welcher im oberen Graphen visualisiert wird, angestoßen werden.

5 Einordnung

Das Explore-Framework ist für die Erstellung und Evaluierung von Simulationsmodellen konzipiert. In diesem Abschnitt soll ein Vergleich zu den in diesem Kontext vertretenen Softwaretools auf Grundlage der folgenden Kriterien angestellt werden:

- Komplexität. Welche Modellkomplexität ist mit dem Tool abbildbar?
- Plattformunabhängigkeit. Kann das Tool auf allen Plattformen ausgeführt werden?
- Auslagerung auf entfernte Rechner. Sind Berechnungen auf entfernte Rechner zwecks Kapazitätenauslastung auslagerbar?
- Integrierte Revisionierung. Unterstützt das Tool den iterativen Entwicklungsprozess der Modellbildung?

Die Kategorisierung und Auswahl von Softwaretools ist an die Übersicht aus (Wainwright 2004) angelehnt.

5.1 Tabellenkalkulation (OpenOffice Calc, Excel)

Tabellenkalkulationen bieten sich in diesem Kontext vorrangig zur Visualisierung von Datensätzen an. Komplexere Simulationsmodelle sind nur umständlich oder gar nicht in einer Tabellenkalkulation zu implementieren.

Der Einarbeitungs- und Anwendungsaufwand ist bei Tabellenkalkulationen

hinsichtlich der Visualisierungen geringer als bei Explore. Aufgrund des modularen Aufbaus von Explore können allerdings zusätzliche Visualisierungen bspw. durch Einbindung weiterer Bibliotheken hinzugefügt werden. Gleiches gilt für die Simulationsbibliotheken. Darüberhinaus ist die Programmierung eines Simulationsmodells in einer Programmiersprache besser strukturier- und lesbar als in einer Tabellenkalkulation.

Tabellenkalkulationen fehlen Funktionalitäten zum Revisionsmanagement und zur Auslagerung von Berechnungen auf entfernte Rechner.

5.2 Graphische Modellierungstools (SIMILE, STELLA, VENSIM)

Diese Tools stellen das Modell als Graphen dar. Knoten in diesem Graphen repräsentieren bspw. Variablen, Konstanten usw., während die Kanten durch verschiedene Annotationen Wirkungsbeziehungen ausdrücken. Modellgraphen sind intuitiv zu verstehen und vermitteln einen guten Überblick über das Modell. Bei zunehmender Komplexität besteht allerdings die Gefahr, dass der Graph aufgrund der vielen Komponenten unübersichtlich wird.

Die Visualisierungsmöglichkeiten dieser Werkzeuge ist analog zu Tabellenkalkulationen auf einen festen Satz beschränkt. Auch bieten diese Werkzeuge in der Regel keine Unterstützung für das Revisionsmanagement oder die Auslagerung von Berechnungen.

5.3 Textuelle Modellierungstools (MATLAB, MAPLE)

Das Modell wird in diesen Werkzeugen in Textform beschrieben. Dabei wird in der Regel, wie z.B. bei MATLAB, eine eigene domänenspezifische Sprache verwendet. Die prominentesten dieser Werkzeuge sind proprietäre Softwarepakete. Sie verfügen über einen großen Funktionsumfang, der in Form von Komponenten kostenpflichtig erweitert werden kann. Zusätzliche Visualisierungsmöglichkeiten und Funktionalitäten zur Auslagerung von Berechnungen sind ebenfalls in der Regel durch Komponenten erhältlich.

Allerdings bieten auch diese Tools kein integriertes Revisionsmanagement an.

5.4 Eigenentwicklungen

In eigenen Programmen können externe Bibliotheken wie bei Explore direkt eingebunden werden. Die Flexibilität eigener Programme fällt höher als bei Explore aus, da auch die verwendete Programmiersprache frei wählbar ist, wodurch sich die Auswahl verfügbarer Bibliotheken verbreitert.

Allerdings müssten Auslagerungs- oder Revisionsmanagementfunktionalitäten für den eigenen Programmcode implementiert werden, was einen erheblichen Zusatzaufwand darstellt. Je nach Wahl der Programmiersprache ist die Plattformunabhängigkeit mit erheblichem Aufwand verbunden. Zwar existieren viele Bibliotheken für die Auslagerung von Berechnungen auf entfernte Rechner, aber die Einbindung dieser Funktionalitäten erfordert sogenannten Boilerplate-Code. Das Explore-Framework hingegen stellt diese Funktionalitäten mit sehr geringem Aufwand auf Seiten des Benutzers bereit.

6 Fazit und Ausblick

Das Explore-Framework erfüllt alle gestellten Anforderungen in ausreichendem Umfang. Gegenüber existierenden Tools zeichnet sich Explore über die Integration der Modellrevisionierung und der strikten Trennung von Darstellung und Berechnungen aus.

Das Navigieren durch den bisher beschrittenen Lösungsweg erlaubt es dem Benutzer, an beliebigen Stellen neue Strategien anzusetzen. Die Auslagerung der Experimente auf einen beliebigen Rechner ermöglicht die Ausnutzung größerer Rechnerkapazitäten. Langwierige Berechnungen können dadurch je nach verfügbarer Kapazität schneller ausgeführt werden. Aufgrund der Netzwerktransparenz ist es Benutzern möglich, jederzeit eine Verbindung zu einem laufenden Experiment herzustellen, wodurch Simulationen überwacht oder deren Resultate im Team analysiert werden können.

Für Explore befindet sich derzeit ein weiteres Frontend im Rahmen einer Diplomarbeit in Entwicklung, das den Zugriff auf ein Experiment über das Web ermöglichen wird.

Darüberhinaus ist geplant, die Berechnung durch die Ausnutzung von

Parallelisierung (z.B. mittels ParallelPython) beschleunigen.

Eine weitere mögliche Erweiterung ist ein Mechanismus, der mit minimalen Aufwand automatisch Anbindungen zu einem C/C++ oder Matlab-Code erstellt. Mit dieser Funktionalität könnten bereits bestehende Simulationsmodelle oder Programme einfacher in das Framework eingebunden werden.

7 Literatur

Jakeman, A. J.; Letcher R.A.; Norton J.P. (2006): Ten iterative steps in development and evaluation of environmental models. In: Environmental Modelling & Software 21, 602-614.

Jørgensen, S. E.; Bendoricchio, G. (2001): Fundamentals of ecological modelling. Dritte Auflage. Elsevier, Oxford.

Larman, C. (2003): Agile and Iterative Development: A Manager's Guide, Addison-Wesley.

Stadler, M. et al. (2007): The Adaptive Fridge – Comparing different control schemes for enhancing load shifting of electricity demand. In: Proc. 21st Conference on Informatics for Environmental Protection – Enviroinfo Warsaw 2007, Shaker Verlag, Deutschland, 199–206.

Wainwright, J.; Mulligan, M. (Hrsg) (2004): Environmental modelling. Wiley, West Sussex.