Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminarausarbeitung

Florian Forsthuber

Entwicklung einer DSL zur Konfiguration eines Multi-Agenten-Simulations-Systems

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Einleitung

Vorliegende Ausarbeitung im Rahmen des Hauptseminars des Informatik Masters beschäftigt sich mit Ideen zur Konzeption und Entwicklung einer domain-spezifischen Sprache (DSL). Der Anwendungsbereich der DSL ist die Konfiguration von Multi-Agenten-Simulationen.
Das Forschungsteam der MARS-Group (Multi-Agent Research & Simulation) entwickelt derzeit ein Framework zur Konstruktion solcher Multiagenten-Systeme, wozu diese Forschungsarbeit beitragen soll.
Ein besonderer Fokus bei der Entwicklungsarbeit der DSL liegt dabei auf der Rolle des End-Users, welcher die Sprache später nutzen soll. Diese sind meist Experten in ihren eigenen Forschungsbereichen, haben aber einen unterschiedlichen Wissenstand über Multi-Agenten-Simulationen. Darüber hinaus können hier nur eingeschränkte Kenntnisse im Bereich der Softwareentwicklung vorausgesetzt werden. Diese Tatsache legt ein besonderes Augenmerk auf die Usability des Produkts während der Designphase der DSL und der Konzeption unterstützenden Tools nahe.
Multi-Agenten-Systeme sind von Natur aus sehr komplexe Programme. Zum einen ist jeder Agent ein autonomes Softwaresystem. Zum anderen wechselwirken diese Systeme untereinander und mit ihrer Umwelt. Je mehr unterschiedliche Agententypen in einer Simulation vorhanden sind, desto mehr Wechselwirkungen sind möglich. Diese Komplexität überträgt sich auch auf den Konfigurationsprozess dieser Systeme. Die Hauptaufgabe beim Entwurf der DSL ist es die Konzepte von Multi-Agenten-Simulationen so zu abstrahieren, dass diese von Nicht-Domain-Experten schnell verstanden und Konfigurationsaufgaben effizient bearbeitet werden können.
Im Folgenden soll die bisher bestehende Architektur des MARS Frameworks beschrieben werden um den Kontext, in dem sich die DSL bewegen soll, abzustecken. Danach wird die Implementation des MARS Frameworks in Bezug auf die technische Umsetzung erläutert. Dies begründet die technischen Vorgaben bei der Entwicklung der DSL. Im nächsten Schritt werden zwei Implementationsansätze für die Sprache betrachtet und auf Vor- und Nachteile überprüft. Zuletzt soll das Prinzip des User-Centered Design erläutert und die Risiken, welche dieses Projekt mit sich bringt, diskutiert werden.

1 Das MARS Framework

Das MARS Framework ist eine Softwarelösung, die die Grundlage zur Entwicklung konkreter Multi-Agenten-Simulationen bildet. Das wichtigste Ziel dabei ist es eine verteilte Programmstruktur zu schaffen, die gleichzeitig hoch skalierbar ist [1]. Dies ist notwendig, da Simulationen dieser Art sehr rechenintensiv sein können und ihre Komplexität sich mit einer zunehmenden Anzahl von Agenten um ein Vielfaches erhöht. Große Mengen an Agenten gleichzeitig zu simulieren ist notwendig um realitätsnahe und damit zuverlässige Daten aus der Simulation zu erhalten, wie das Beispiel der Bewirtschaftung des Abdoulaye Nationalparks zeigt [2].

1.1 Die Architektur des Frameworks

Zuerst soll auf die Architektur des Frameworks eingegangen werden. Dabei werden die Aspekte, welche die Verteilung und Skalierbarkeit des Systems betreffen, nicht besprochen, da sie für den Kontext der DSL keine Relevanz haben. Weitere Informationen zu diesem Aspekt können [1] entnommen werden. Abbildung 1 zeigt eine Übersicht der Framework Komponenten und skizziert den Anwendungsbereich der DSL. Dabei sind die Komponenten in drei Bereiche aufgeteilt.

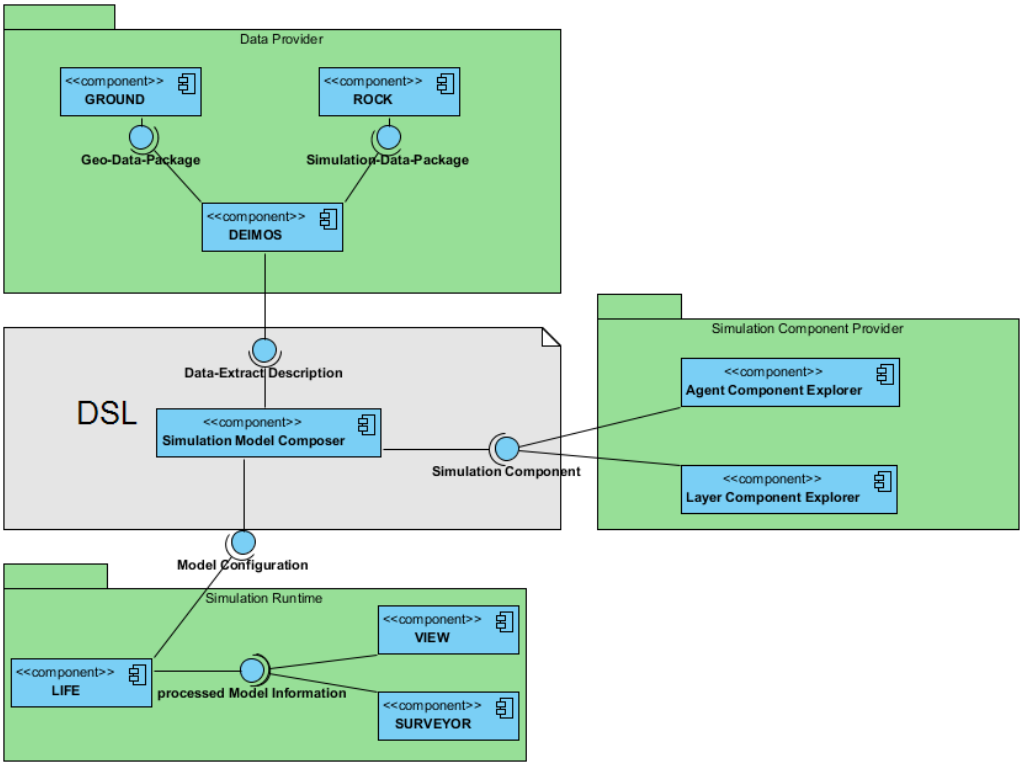

Abbildung 1: Übersicht der Framework Komponenten

1.1.1 Der Bereich Data Provider

Der Bereich Data Provider beinhaltet die Systembestandteile, welche die notwendigen Daten zur Initialisierung eines Simulationsmodells bereitstellen. Dabei ist die Komponente GROUND für die Interaktion mit einem Geoserver verantwortlich. Der Geoserver ist eine Datenbank, von der sich Geoinformation wie Temperatur, Niederschlagsmengen oder topografische Daten abrufen lassen. ROCK ist das Interface zu einem Datawarehouse, welches Daten zur Modellinitialisierung liefert, die keine expliziten Geodaten sind. Darunter fallen beispielsweise Informationen, wie Bewegungsprofile oder Bevölkerungswachstum. Die Komponente DEIMOS bietet dem Simulationsmodelldesigner eine GUI, mit deren Hilfe sich die von GROUND und ROCK zur Verfügung gestellten Daten eingrenzen, zusammenfassen und skalieren lassen. Hat der User alle für die Simulation relevanten Daten zusammengestellt, wird eine Beschreibungsdatei der zusammengestellten Daten erzeugt, die als Grundlage zur Initialisierung der

Simulationskomponenten verwendet wird. Eine Aufgabe der DSL ist es die Konfiguration dieses Initialisierungsprozesses abzubilden.

1.1.2 Bereich Simulation Component Provider

Der Bereich Simulation Component Provider bildet das Gegenstück zum Bereich Data Provider. Hier werden die Komponenten, aus denen ein Simulationsmodell zusammengestellt werden kann, verwaltet. Diese werden dann mit den über DEIMOS extrahierten Daten initialisiert. Im MARS Framework setzt sich ein Simulationsmodell aus Agenten und Layern zusammen, weswegen der Component Provider dementsprechend unterteilt ist. Das Konzept des Layers erlaubt es unterschiedliche Aspekte eines Modells in in sich geschlossene Bereiche aufzuteilen. Dadurch erreicht man eine thematische Trennung von Informationen und es ist möglich nur bestimmte Teilbetrachtungen einer Simulation darzustellen und die restlichen Aspekte auszublenden [3]. Ein Layer kann nicht nur einen speziellen Typ von Umweltdaten wie Temperatur oder Niederschlag beinhalten, sondern auch jeweils einen Typ von Agenten.

1.2 Der Komponenten Ansatz

Im MARS Framework setzen sich Layer und Agenten aus in sich geschlossenen, vorkompilierten Komponenten zusammen. Eine Übersicht über die im MARS Framework verwendeten Komponententypen stellt Abbildung 2 dar. Komponenten können mit Hilfe des MARS LDK (LIFE Development Kit) realisiert werden. Das LDK stellt dazu eine API zur Verfügung, die die Rahmenbedingungen zur Entwicklung von Simulationskomponenten vorgibt und somit das Zusammenspiel aller entwickelten Elemente im Gesamtsystem gewährleistet. Für die Entwicklung von Komponenten mittels LDK werden fortgeschrittene Kenntnisse im Bereich der Softwareentwicklung benötigt, da der Quellcode direkt in der Programmiersprache des MARS Frameworks abgebildet wird. Dies fällt daher nicht in den Aufgabenbereich der End-User. Je nach Art der Komponente müssen diese gewisse Schnittstellen realisieren. Das soll dem DSL-Nutzer ermöglichen Simulationskomponenten nach dem „Baukastenprinzip“ zusammenzufügen. Mittels dieses Ansatzes könnten Elemente wie spezifische Sensoren oder Actions schnell ausgetauscht oder ergänzt werden. Dadurch entstehen neue Agenten ohne Programmieraufwand. Das Userinterface zum Zusammenstellen und Austausch dieser Komponenten liefert die DSL. Im Folgenden wird die Komponentenübersicht aus Abbildung 2 kurz erläutert.

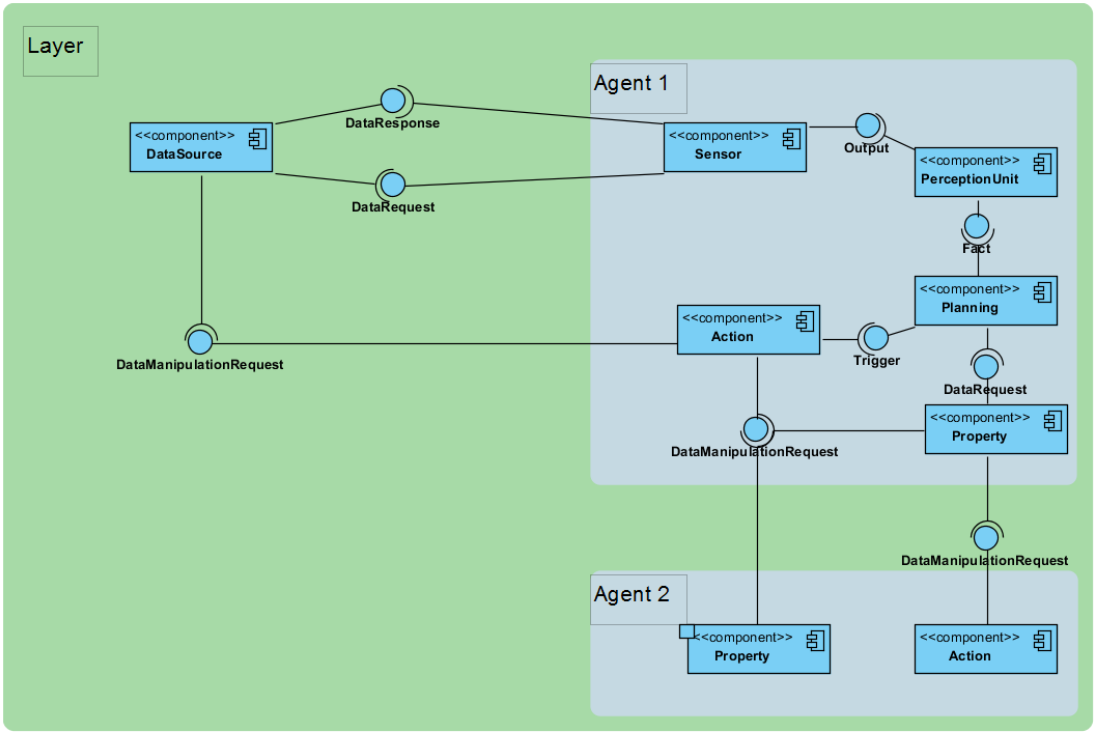

Abbildung 2: Übersicht der Simulationskomponenten

Wie in Abschnitt 1.1.2 erwähnt ist ein Layer unter anderem ein Container für Umweltdaten, welche ein Simulationsmodell benötigt. Quelle dieser Daten ist der Bereich Data Provider, wie in Abbildung 1 zu erkennen ist. Um diese zu kapseln und mit einem Interface zur Abrufung während der Simulation zu versehen, existiert die DataSource Komponente. Über diese Schnittstelle ist es Agenten möglich für sie relevante Informationen während eines Simulationsdurchlaufs mittels eines DataRequest Objekts abzurufen.
Um die Informationen wahrzunehmen die eine DataSource bereitstellt sind Agenten mit Sensor Komponenten ausgestattet. Sie regeln das Abfragen der Daten von DataSources. Die von Sensoren aufgenommen Daten werden in einer PerceptionUnit gefiltert und zusammengeführt. Die daraus resultierenden Fakten werden an eine Planning Komponente weitergegeben, welche die Fakten im Hinblick auf die Ziele des jeweiligen Agenten bewertet und einen Handlungsplan erstellt. Je nach Priorität der einzelnen Handlungsschritte triggert die Planning Komponente die entsprechenden Action Komponenten. Diese üben ihrerseits

Einfluss auf die Umwelt des Agenten, auf andere Agenten oder den agierenden Agenten selbst aus. Dabei manipulieren sie den Inhalt der DataSources von Layern oder Property Komponenten von Agenten. Property Komponenten enthalten Agenten spezifische Daten und Eigenschaften, die den Zustand des Agenten während der Simulation wiedergeben. Diese Parameter, wie beispielsweise Hunger oder Stresslevel haben ebenfalls Einfluss auf die Planungskomponente des Agenten.

1.3 Konfiguration der Komponenten

Neben dem bloßen Zusammenfügen von Simulationskomponenten soll es mittels DSL auch möglich sein einzelne Komponenten selbst zu konfigurieren. Beispielsweise kann die Wahrnehmungsstrategie der PerceptionUnit Komponente justiert und somit bestimmt werden, welcher Sensor-Input wie stark gewertet werden soll. Somit könnten unterschiedlich starke Reize dargestellt werden. Innerhalb der Planning Komponente könnten die Ziele welche ein Agent verfolgen kann mittels DSL priorisiert oder völlig neue Strategien eingestellt werden. Sicherlich würde mit diesen Möglichkeiten auch die Komplexität DSL zunehmen. Dies gäbe zwar den End-Usern mehr Mittel an die Hand das Simulationsmodell zu gestalten, ohne dass zusätzlicher Programmieraufwand auf der Seite des MARS LDK nötig wäre. Zum anderen könnte sich die zusätzliche Komplexität negativ auf die Usability der DSL auswirken.
Nachdem nun auf die Anwendungsbereiche der DSL eingegangen wurde, soll im Folgenden das technische Umfeld für die eigentliche Konzeption und Implementierung der Sprache erläutert werden.

2 Technisches Umfeld

Die Programmiersprache, in der das MARS Framework realisiert wird, ist C#. Hierauf basieren die Laufzeitkomponenten, die für Verteilung, Skalierung und Kommunikation der laufenden Simulation verantwortlich sind. Auch die Klassen und Interfaces, aus denen das MARS LDK besteht, basieren auf der Sprache. Die Simulationskomponenten, welche aus der Entwicklung mittels LDK hervorgehen, werden als Plugins in Form von DLLs kompiliert.

Das Userinterface der MARS Applikation ist als Rich Internet Application (RIA) ausgelegt und läuft als Webanwendung in einem Webbrowser. Sowohl das Frontend als auch das Backend der RIA basiert auf JavaScript. Im Backend kommt dabei das Webframework Express[4], welches auf der NodeJS Technologie basiert[4], zum Einsatz. Auf NodeJS basieren auch zahlreiche Plugins, die dem Entwickler Aufgaben wie Userverwaltung, Routing oder Datenbankanbindung bereits abnehmen. Ein essentielles Plugin ist dabei EdgeJS [5]. Es fungiert als Schnittstelle zwischen den in C# entwickelten Systemteilen und dem in JavaScript entwickelten Backend des webbasierten Userinterfaces. Damit ist es möglich in JavaScript C# Code auszuführen. Dabei kann sowohl direkt C# Quelltext in Javascript eingebettet, als auch vorkompilierte DLL-Dateien geladen und in der Javascript-Umgebung ausgeführt werden.
Das Frontend basiert auf dem Framework AngularJS [6]. In AngularJS lassen sich ebenso wie in NodeJS zahlreiche JavaScript Plugins einbinden. Diese bilden die Basis für die unterstützenden Tools der DSL. Die umfangreiche Web IDE AceIO [7] kann beispielsweise als Grundlage für einen Feature reichen DSL-Editor verwendet werden. Das Karten Tool LeafletJS [8] bietet zahlreiche Möglichkeiten Geodaten und Kartenmaterial darzustellen und mit user-generiertem Content anzureichern. Dieses Plugin könnte als Grundlage für ein unterstützendes Tool verwendet werden, welches den End-Usern das Verteilen von Agenten auf dem Simulationsraum erleichtert.

3 Implementationsmöglichkeiten der DSL

Im Folgenden soll auf zwei unterschiedliche Möglichkeiten der Implementation der DSL eingegangen werden, die auf Grund der in 2 erwähnten technischen Voraussetzungen in Frage kommen.

3.1 Parser Generatoren

Parser Generatoren sind Programme, die aus einer Sprachenbeschreibung einen Parser für die zu entwerfende DSL erzeugen. Die Sprachbeschreibung erfolgt meist

in einer Form, welche an BNF (Backus-Naur-Form) angelehnt ist. Sie ist damit eine Meta-Sprache zur DSL und wird als Grammatik der Sprache bezeichnet. Eine Grammatik besteht aus Produktionsregeln, anhand derer später die eingegebene DSL validiert wird. Der Generator kompiliert die Grammatik und erzeugt Quellcode für einen DSL-Parser einer unterstützten Zielsprache, wie Java oder C#. Dieser kann dann in einem beliebigen Projekt verwendet werden. Ein Parser kann die eingegebene DSL nicht nur validieren, sondern auch Aktionen ausführen. Wenn ein DSL-Statement eine Produktionsregel erfüllt, wird Programmcode, welcher in der Zielsprache (oder Hostsprache) definiert ist, ausgeführt. Dieser wird entweder direkt in die Grammatik eingebettet oder die durch den Generator erzeugten Parserklassen werden an spezifischen Stellen erweitert[9]. Damit ist die DSL unabhängig von Syntax und Semantik der Hostsprache und diese Aspekte können individuell nach den Vorgaben der Domain gestaltet werden.
Parser Generatoren, die sich für die MARS Architektur eignen sind beispielsweise Coco/R[10] oder ANTLR[11]. Beide unterstützen die Hostsprache C#. Die erzeugten Parserklassen können in eine DLL kompiliert werden. Der DSL Code kann über ein Textfeld auf der in Abschnitt 2 beschriebenen Weboberfläche eingegeben und mittels EdgeJS an die Parser-DLL weitergereicht werden. Diese validiert die Eingabe und gibt auf dem selben Weg Rückmeldung über eventuelle Eingabefehler.
Ein Nachteil dieses Ansatzes ist die Tatsache, dass die erzeugte DSL auf einer Grammatik beruht. Bestehende Grammatiken sind schwierig zu erweitern und Fehler beim Bearbeiten solcher Dateien können auch bereits funktionierende Teile der DSL beschädigen. Auch wenn eine Grammatik erfolgreich erweitert wurde, müssen aus ihr wiederum neue Parserklassen erzeugt und die alten im bestehenden Projekt ausgetauscht werden. Danach muss der entsprechende Programmteil neu kompiliert werden. Diese Tatsachen wirken sich negativ auf die Erweiterbarkeit der DSL aus. Auch kann der Vorteil, welcher sich aus der Möglichkeit der freien Sprachgestaltung ergibt, ein Nachteil sein. Eine funktionierende Sprache mit kohärenter Syntax und aussagekräftiger Semantik von Grund auf zu designen ist eine schwierige und zeitaufwendige Aufgabe, die viel Erfahrung in diesem Bereich voraussetzt[9].

3.2 Embedding Ansatz

Beim Embedding Ansatz wird eine DSL implementiert, indem die Hostsprache um neue Konstrukte erweitert wird. Dabei kann ein domain-spezifisches Problem durch neue Datentypen und Operatoren ausgedrückt. In Hostsprachen, wie C++, lassen sich auch Operatoren überladen um ihnen einen domain-spezifischen Kontext zu

geben. Auch wenn eine Sprache kein Operator Overloading bietet, reicht es schon aus eine Klassenbibliothek zu konstruieren um eine Problem Domain abzubilden. In Verbindung mit einer API kann jede GPL als DSL fungieren. Die API drückt domain-spezifisches Vokabular in Form von Klassen, Methoden und Eigenschaften aus, deren Namensgebung und Signaturen sich stark an der Ontologie der Domain orientieren[9]. Über Objekterzeugung und Methodenaufrufe, lässt sich das domain-spezifische Vokabular dann anwenden. Mit diesem Ansatz kann der Compiler oder Parser der Hostsprache verwendet werden um die DSL zu verarbeiten. Ein weiterer Vorteil dieses Ansatzes ist es, dass für eine Erweiterung des Funktionsumfangs der DSL nur die API ausgebaut werden muss. Die Gefahr bereits bestehende funktionierende Teile der DSL durch Erweiterung zu beschädigen ist hierbei viel geringer als bei den in Abschnitt 3.1 diskutierten Parser Generatoren. Außerdem muss eine Erweiterung nicht neu kompiliert werden, falls eine Skriptsprache als Host dient. Obwohl dieser Ansatz einfacher zu implementieren ist und viele Vorteile bietet, müssen einige Dinge beachtet werden, damit DSL Funktionalität erreicht werden kann.
Für die DSL User ist der volle Funktionsumfang einer GPL überfordernd. Komplizierte Programmier Konstrukte sollten in der API vermieden werden. Jedoch lassen sich für die DSL ungewollte Hostsprachen Konstrukte nur mit hohem Aufwand herausfiltern und für den DSL Nutzer besteht immer die Möglichkeit von der für ihn vorgesehenen API abzuweichen. Dies ist ein Nachteil, der mit dem Parsergenerator Ansatz vermieden wird.
Im Beispiel der MARS DSL würde diese beim Embedding Ansatz im Webserver Backend auf Basis von JavaScript entwickelt werden. Mit jedem API Aufruf würden dann C# Prozeduren über EdgeJS angestoßen werden.

4 Der Entwicklungsansatz User-Centered Design

Das wichtigste Ziel beim Design einer DSL ist es, dass die End-User mit der Sprache produktiv arbeiten können. Es muss ihnen möglich sein eine Problemstellung ihres Forschungsbereichs möglichst intuitiv über die Sprache abzubilden. Dabei ist

Usability ein Schlüsselkriterium zur der Evaluierung der Qualität eines Userinterfaces. [12]
Usability wird nach ISO 9241-11 folgendermaßen definiert:

"*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*"[13]

Dabei bestimmt der Begriff 'effectiveness' den Grad der Genauigkeit und der Vollständigkeit der DSL. Genauigkeit sagt dabei aus, wie nah ein DSL Statement am jeweiligen Konzept der Domain ist, die es repräsentiert. Vollständigkeit trifft dabei eine Aussage über den Grad der Abdeckung, die die repräsentierte Domain durch die DSL erfährt.
Unter dem Begriff 'efficiency' versteht man das Verhältnis von Aufwand, welcher betrieben werden muss um eine Problemstellung mit Hilfe der DSL abzubilden,
zum Grad der Zielerreichung bei der Formulierung des Problems.
Der Begriff 'satisfaction' zeigt dabei an, in wie weit der End-User subjektiv den Gebrauch der DSL und der zughörigen Tools als positiv und frei von Hindernissen erachtet.
Zusätzlich zu den Kriterien, welche ISO 9241-11 spezifiziert, führt [12] den Aspekt der 'accessibility' an. Damit soll ausgedrückt werden, wie viel Aufwand vom End-User betrieben werden muss um den Umgang mit der DSL und seinen Tools zu erlernen und im Gedächtnis zu behalten. Dabei spielt der Grad der Intuitivität beim Arbeiten mit der DSL eine erhebliche Rolle.
Ziel des User-Centered Design ist es in den Bereichen 'effectiveness', 'efficiency', 'satisfaction' und 'accessibility' gute Ergebnisse zu erzielen, in dem der Nutzer in jede Phase des Entwicklungsprozesses der DSL mit einbezogen wird.
Zunächst ist es wichtig den späteren Anwendungsbereich der DSL zu verstehen und zu spezifizieren. Dies geschieht einerseits durch Analyse von Domain-Artefakten wie bestehende Anwendungen, Dokumentationen oder anderen schriftlichen Quellen. Andererseits ist es wichtig die Domain aus Sicht der End-User und ihrer täglichen Arbeit zu betrachten. Durch regelmäßigen Austausch muss in Erfahrung gebracht werden, mit welchen Tools dabei gearbeitet wird und die Problemstellungen abgebildet werden. Dabei wird geklärt, über welche Stärken und Schwächen diese Ansätze verfügen. Dies erfolgt sowohl aus Sicht der Anwender als auch aus Sicht der DSL Designer. Mit den End-Usern wird ein DSL-Konzept entworfen, welches die identifizierten Stärken der bisherigen Lösungen beibehält und gleichermaßen versucht die Schwächen zu beheben. Auf dieser Basis kann dann ein erster Prototyp entstehen.

Der Prototyp wird dann in einem Zyklus von Evaluierung durch die End-User und Verbesserung der Sprache und Tools zu einem einsatzfähigen Produkt weiterentwickelt. Wie genau dabei der Prozess der Evaluierung vonstattengeht, muss noch recherchiert werden.

5 Risiken

Das größte Risiko, welches die Aufgabe der Erstellung einer DSL für den Bereich Multi-Agenten-Simulation birgt, ist, dass eine Erhöhung der Produktivität der End-User nicht erreicht wird. Ein Grund dafür könnte eine zu hohe Komplexität und zu geringe Domainnähe der entwickelten DSL sein. Weiterhin sind die unterstützenden DSL-Tools, die DSL-Code generieren oder DSL-Templates füllen, von entscheidender Bedeutung. Auch diese sollen intuitiv und gut bedienbar sein. Schließlich muss sich die Lösung am gesamten Workflow bei der Konfiguration von Multi-Agenten-Simulationen messen lassen.
Ein weiterer Risikofaktor für die Umsetzung des Projekts ist die begrenze Entwicklungszeit. Eine DSL-Suite soweit zu entwickeln, dass eine alltagstaugliche Lösung entsteht, ist ein ambitioniertes Ziel. Hilfreich dabei ist die große Anzahl
JavaScript basierten Plugins, die in die webbasierte DSL-Anwendung eingebunden werden können und dem DSL-Entwickler viel Arbeit abnehmen.
Jedoch kommt im zeitlichen Rahmen der Arbeit das Projekt vielleicht nicht
über eine prototypische Lösung hinaus. Daher sind gute Dokumentation, Modularität und Erweiterbarkeit bei der Umsetzung der DSL wichtig, damit das Projekt mit möglichst wenig Einarbeitungszeit von einem Nachfolger fortgeführt werden kann. Die erhöhte Sorgfalt, die diesen Aspekten zu Gute kommen muss, erfordert zusätzlichen zeitlichen Aufwand.

Quellen

[1] Hüning, Christian; Wilmans, Jason; Feyerabend, Nils; Thiel-Clemen, Thomas *MARS - A next-gen multi-agent simulation framework,* Wittmann; Müller, (Ed.): Simulation in Umwelt- und Geowissenschaften, Workshop Leipzig, GI Shaker, 2014

[2] Pereki H., Wala K., & Thiel-Clemen T, *Woody species diversity and important value indices in dense dry forests in Abdoulaye Wildlife Reserve (Togo, West Africa)*. International Journal, 5 (June), 358–366, 2013

[3] Hünning C., *Konzeption und Entwurf einer Architektur zum Einsatz von Multi-Agenten-Simulationen in der ökologischen Systemmodellierung, Masterthesis*, HAW-Hamburg, 2013

[4] Yaapa H., Express Web Application Development, Packt Publishing, Birmingham (UK), 2013

[5] Janczuk T., edge.js, *http:// tjanczuk.github.io/edge*, Website. Abruf am 15.10.2014 20:16 Uhr

[6] Ford B., Ruebbelke L., *AngularJS in Action*, Manning Publications Co., Stamford(USA), 2014

[7] Ace - The High Performance Code Editor for the Web, *http://www.ace.c9.io*, Website. Abruf am 25.01.2015 20:16 Uhr

[8] Leaflet - a JavaScript library for mobile-friendly maps, *http://www. http://leafletjs.com/*, Website. Abruf am 25.01.2015 21:16 Uhr

[9] Mernik M., Heering J., Sloane A. M. *When and how to develop domain-specific languages*. ACM Comput. Surv. 37, 4 2005, 316-344

[10] Mössenböck H., Wöß A., Löberbauer M., *Der Compilergenerator Coco/R*, Johannes Kepler Universität Linz, Institut für Praktische Informatik, 2003

[11] Paar T., Fisher K. S., *LL(*): The Foundation of the ANTLR Parser Generator*, In Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation , New York, 2011

[12] Barišić A., Amaral V., Goulão M., Barroca B. *How to reach a usable DSL? Moving toward a Systematic Evaluation*, Proceedings of the 5th International Workshop on Multi-Paradigm Modeling (MPM'2011), Wellingnton, New Zealand, 2011

[13] Bevan, N., Macleod, M. *Usability measurement in context*, Behaviour & Information Technology., 1994