Ein deklarativer Ansatz zur domänen-übergreifenden Modellanalyse

Schnittstellen für den Einsatz domänen-spezifischer Sprachen

Ulrich Hartmann

Universität Karlsruhe, Institut für industrielle Bauproduktion, 76128 Karlsruhe, Deutschland, E-mail: Ulrich.Hartmann@ifib.uni-Karlsruhe.de, Telefon: +49(0) 0721/ 608-2166

Kurzfassung

Die Analyse digitaler Modelle erfordert eingehende Kenntnis der Modelsyntax und Semantik sowie der technischen Spezifikation des Modellschemas und seiner internen Repräsentation im Rechner. Die vorbereitenden Arbeiten zur Durchführung von Analysen mithilfe digitaler Modelle können daher leicht umfangreicher werden als die Analyse selbst. Das gleichzeitige Erwerben von Fachwissen zur Bereitstellung der technischen Infrastruktur und Expertenwissen zur Durchführung von Analysen ist daher häufig von Einzelpersonen nicht zu leisten. Dagegen ist ein Produktivitätszuwachs zu erwarten, falls es gelingt den konzeptionellen Teil der Analyse von der technischen Implementierung der digitalen Modellanalyse zu trennen. In dem hier vorgestellten Konzept soll gezeigt werden, wie Experten Zusammenhänge und Elemente einer Analyse in ihrer eigenen domänen-spezifischen Sprache ausdrücken können, während das mapping zwischen konzeptioneller und technischer Sicht auf das System in einer generalisierenden und erweiterbarem Art und Weise gekapselt werden kann. Hierzu werden state-of-the-art Softwarekonzepte eingesetzt.

Stichworte

Domänen-übergreifende Analyse, domain-specific Language, abstrakter Syntaxbaum, Besuchermuster, Fabrikmuster, Interpreter

1 Einleitung

1.1 Motivation

Wie bereits erwähnt, kann der Zugriff auf Daten eines digitalen Modells schnell sehr komplex werden. Obwohl Komplexität -sofern gut dokumentiert- nicht das Hauptproblem darstellt, erfordert der Zugriff auf Strukturen und Werte von Modelinstanzen oft internes Wissen über allgemein verwendete ‚good pratices' über den Einsatz von Datenstrukturen und die Existenz von alternativen Konstrukten. Trotz der öffentlichen Verfügbarkeit des Modellschemas erreicht dieses oft zur effizienten und eindeutigen Arbeit mit Modelldaten nicht aus.

Einige allgemein verwendete Modelltypen (z.B. IFC) haben ihren Fokus in erster Linie auf der Produktbeschreibung, und sind damit für den Datenaustausch gut geeignet. Die Datenanalyse dagegen wird oft durch die Ausprägung der Datenstrukturen erschwert. Zum Beispiel weist IFC Entitäten Eigenschaften über eine Kaskade von mindestens vier Indirektionen zu, bevor der gesuchte Wert schließlich erreicht wird. Während diese lose Kopplung einen sehr eleganten und flexiblen Mechanismus der Zuweisung von Eigenschaften zu Elementen darstellt, ist er für die Datenrecherche unpraktikabel. Dieses ist strukturimmanent und von der eingesetzten Persistenztechnologie unabhängig. Sowohl eine relationale als auch eine objektorientierte Datenmodellierung muss die tiefen Interaktionsbäume auflösen, dieses macht table joins oder äquivalente OO-Mengenoperationen notwendig. Abhängig vom zugrunde liegenden Problem mag dieses entweder durch Konvertierung der logischen Modellstruktur in eine technische Datenbankstruktur gelöst werden, um den Datenzugriff zu optimieren (Performance, Komplexität) oder es führt zu komplexen Anfrageausdrücken. Applikationen, die auf diesen Datenbankstrukturen statt auf nativen Modellstrukturen aufbauen müssen, haben es daher schwerer domänen-spezifische Logik zu implementieren, die per definitionem die nativen domänen-spezifischen Konzepte verwendet. Die zweite Lösung, komplexe Anfragen sind dagegen schwer wartbar und fehleranfällig, mit negativen Folgen auf die Robustheit der Applikation.

1.2 Vision

Die interne digitale Repräsentation eines domänen-spezifischen Modells ist das Resultat der Anwendung von Methoden der Informatik auf die Konzepte der entsprechenden Fachdomäne. Doch obwohl das Ergebnis das Verständnis der Fachdomänen-Konzepte

widerspiegelt, tragen Syntax und Semantik des digitalen Modells häufig das Erscheinungsbild der Informationswissenschaften, nicht der ursprünglichen Fachdomäne. Ein Ansatz der diese sehr wertvollen Computermodelle als Basis verwendet und zugleich das Ausdrücken analytischer Zusammenhänge in einer domänen-spezifischen Fachsprache erlaubt, würde die Fähigkeit von Fachdomänen-Experten stärken Fachdomänenmodelle zu verwenden, ohne dabei gleichzeitig ein Softwarespezialist sein zu müssen. Beispielsweise muss der folgende Indirektionpfad zum Auffinden der Grundfläche aller Büroräume in einem IFC Modell verfolgt werden:

```
IfcSpace→
  IfcRelDefinesByProperty→
    IfcSpaceProgram→
      IfcPropertySingleValue→
        IfcLabel:OccupancyType == “’Büro’”
        …
```

Ein Fachdomänenexperte würde dagegen vorzugsweise den Ausdruck "gesamte Grundfläche aller Büroräume" verwenden, und damit die Summe aller Grundflächen aller Räume vom Typ ’Büro’ im entsprechenden Geschoss des entsprechenden Gebäudes meinen. Statt auszudrücken wie die Werte in der entsprechenden Syntax des digitalen Modells zu beschaffen sind, würde sich ein deklarative Ausdruck nicht um das ‚Wie’ kümmern, stattdessen würde es sich auf das ‚Was’ konzentrieren. Wie kann das erreicht werden? Dieser Artikel versucht einen konzeptionellen und technischen Weg darzustellen, mit dessen Hilfe die Belange von Domänenexperten und technischen Experten gekapselt werden können, mit dem Ziel einer besseren Nutzbarkeit sowie der Wiederverwendbarkeit von Analysekonzepten und technischen Komponenten. An einem Beispiel soll das Prinzip erläutert werden.

1.3 Nutzen

Die Trennung von Fachkonzepten und technischen Konzepten fördert nicht nur die Nutzbarkeit digitaler Modelle durch Domänenexperten, die Möglichkeit der Definition problemzentrierter Namensräume kann darüber hinaus eine domänen-übergreifende Zusammenarbeit von Analyseexperten aus verschiedensten Fachbereichen fördern. Eine Modellsicht, die vertraute Strukturen und Konzepte wiedergibt, kann auch dann zur Verfügung gestellt werden, wenn die Analyse mehr als nur eine Fachdomäne betrifft. Die Verknüpfungsebene zwischen Expertensicht und technischer Sicht muss dabei an die wechselnden Anforderungen aus der Expertenebene anpassbar sein. Geeignete

Schnittstellen reduzieren dabei die Komplexität digitaler Modelle und fördern die Erstellung robuster und wieder verwendbarer Applikations-Komponenten.

2 Beispielszenario

Die Stadtverwaltung von Stadt x plant eine neue Buslinie, um den öffentlichen Nahverkehr zu verbessern. Die Route soll Geschäftsbezirke und Wohngebiete verbinden und so eine Alternative zur Benutzung des Autos bieten. Der Berufsverkehr soll dadurch deutlich verringert werden.

Datenmaterial

Das kommunale Katasterssystem, häufig im Kern ein GIS-System, beinhaltet den Stadtplan mit Liegenschafts- und Straßeninformationen. Jeder Liegenschaftseintrag hat einen Verweis auf ein elektronisches Gebäudedokument, das ein IFC-basiertes Modell des Gebäudes enthält[1]. Zur Darstellung des Konzepts ist die technische Spezifikation jedoch nicht relevant.

Analysealgorithmus

Zur Routenoptimierung müssen unterschiedliche Routenführungen verglichen werden. Das Einzugsgebiet der optimalen Route erfasst die meisten Pendler am Morgen und bringt sie so nahe wie möglich an ihren Arbeitsplatz und sinngemäß umgekehrt am Abend. Zwar wäre es theoretisch möglich den Arbeits- bzw. Wohnort jede Einzelperson zu berücksichtigen, es erscheint zur Optimierung der Linienführung jedoch statistisch tragfähig genug, die jeweiligen Einzugsgebiete der verbundenen Zusteige- bzw. Aussteigezonen zu berücksichtigen. Die Grundflächen aller Gebäude im Einzugsgebiet, aufgeteilt nach Nutzungsart wie „Büro", „privat“, werden aufsummiert. In diesem einfachen Beispiel lassen wir weitere Überlegungen wie Routenlänge, Fahrzeit, Umsteigevorgänge usw. außer Betracht. Hier soll kein komplexes Analysemodell für die Optimierung urbanen Verkehrsflusses vorgestellt werden Der Fokus liegt auf der domänen-übergreifenden Analyse. Es sollen die Vorzüge aufgezeigt werden, die eine Abstraktionsebene zwischen einer software-zentrischen und einer fachdomänen-spezifischen Sicht bieten kann. Nahezu dieselben Algorithmen können auch zur

[1] zugegeben eine bis heute optimistische Annahme, die es ggfs. notwendig macht, ein alternatives Instrument zur Datenbeschaffung einzusetzen

optimalen Positionierung von Blockheizkraftwerken, Bibliotheken oder Verkaufsfilialen im Stadtgebiet eingesetzt werden. Gleichermaßen wichtig ist die Aufteilung der Belange in zwei unabhängige physikalische Ebenen. Diese unterstützt nicht nur die Wiederverwendung von Konzepten, sondern auch von Applikationskomponenten.

Grundannahmen

- Die Berechnung des Einzugsgebiets erfolgt unter der Annahme einer maximal zumutbaren Gehstrecke. Jenseits dieser Gehstrecke ist die Benutzung der Buslinie nicht mehr attraktiv.
- Die Schätzung des Fahrgastaufkommens wird auf Grundlage der ermittelten Grundflächen von Wohn- bzw. Büroräumen vorgenommen.

Austausch von Algorithmen

Als erste grobe Annahme könnte die Berechnung des Einzugsgebietes mittels der kürzesten Verbindung zwischen Gebäude und Buslinie (Luftlinie) ermittelt werden. Gebäude innerhalb dieses Maximalabstandes um die Route gehören zum Einzugsgebiet.

In einer zutreffenderen (aber auch zeitintensiveren) Berechnung könnte die exakte Gehstrecke zwischen Gebäuden und Bus-Route durch Nutzung externer Navigationsdienste (z.B. Google maps) ermittelt werden. Durch Laden entsprechender Komponenten zur Laufzeit der Analyse-Applikation kann Fachlogik dynamisch eingebunden werden, dadurch können unterschiedliche Algorithmen deklarativ hinzugezogen bzw. gewechselt werden.

3 Deklarative Analyse-Umgebung

3.1 Entwurfsprinzipe und –anforderungen

Eine Systemlösung muss folgende Anforderungen erfüllen

- Trennung von konzeptioneller und technischer Ebene
- Erweiterbarkeit beider Ebenen
- Konfigurierbare Unterstützung domänen-spezifischer Sprachen (DSL[2])

[2] DSL – Domain-specific Language

- Unabhängigkeit vom zugrunde liegenden Fachdomänen-Modell
- Unabhängigkeit vom zugrunde liegenden Daten-Model
- Unterstützung des objekt-orientierten und relationalen Paradigmas
- Nutzung von Mainstream Software-Technologie
- Gleichzeitige Unterstützung unterschiedlicher Modelltypen

3.2 Systemarchitektur

Zur Trennung der domänen-spezifischen Sprachebene (DSL layer) von der Daten-Management-Ebene ist eine Abstraktionsschicht zwischen beiden Ebenen erforderlich. Die in der DSL-Ebene verwendete Sprache kann als eine Zwischenstufe angesehen werden, die weiterer Interpretation bedarf. Der erste Schritt zum Bau eines Interpreters ist die Definition der zu interpretierenden Sprache. Diese Definition wird häufig in der EBNF-Notation (Erweiterte Backus-Naur Form[3]) –einer Sprache zur Beschreibung von Sprachen- vorgenommen. Diese Arbeit sollte von Fachdomänenexperten und Informatikern gemeinsam durchgeführt werden. Dabei werden Schlüsselworte, Operatoren und Funktionen definiert, die in der domänen-spezifischen Sprache verwendet werden, z.B.

Räume→Nutzungsart(„Büro“)→Grundfläche

oder alternativ

Grundfläche VON_ALLEN Räume MIT Nutzungsart „Büro“

Scheinbar (virtuell) wird dem Anwender hier ein Raumelement mit den Attributen Nutzungsart und Grundfläche angeboten, das es so im zugrunde liegenden digitalen Modell garnicht gibt. Die Wahl der Schlüsselworte obliegt allein den Domänenexperten. In ersten Beispiel oben wird der Pfeil ‚→‘ als De-Referenzierungsoperator verwendet. Intern würde er in eine Recherche nach Raumelementen mit dem Attributwert Nutzungsgart gleich „Büro“ umgesetzt. Das zweite Beispiel nutzt eine andere Syntax mit identischer Bedeutung. Unabhängig vom Entwurf der domänen-spezifischen Sprache liefern beide Ausdrücke nach ihrer Interpretierung die gleich Liste von Werten der Grundflächen aller zutreffenden Räume. In diesem einfachen Beispiel enthält die

[3] ISO Standard 14977

domänen-spezifische Sprache die Operatoren VON_ALLEN und MIT, die Schlüsselworte Grundfläche, Räume und Nutzungsart und den Parameter „Büro“ als passenden Attributwert für Nutzungsart. Die Freiheit bei der Namenswahl von Operatoren und Schlüsselworten beim Entwurf der domänen-spezifischen Sprache wird hier deutlich.

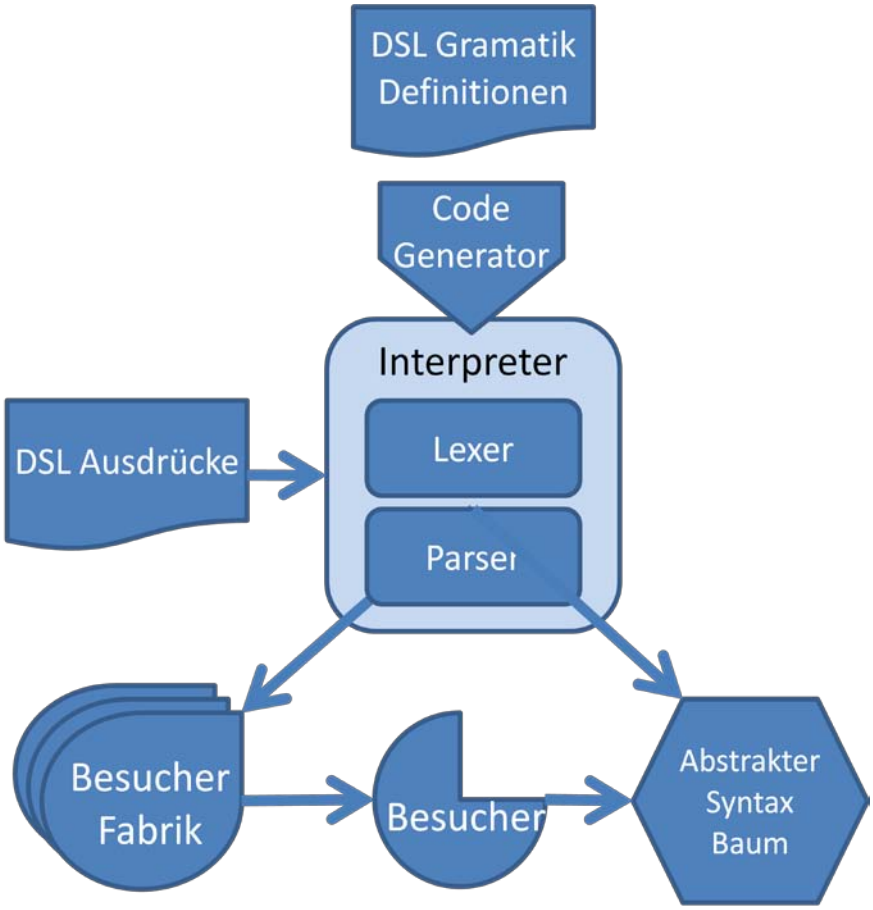

Abb. 1: DSL Interpreter

DSLs müssen durch einen Interpreter ausgewertet werden[4]. Ein Interpreter ist ein Programm, das in der Lage ist dynamisch Kommandos in einer vordefinierten Sprache auszuführen, in diesem Fall einer DSL. Ein mathematischer Formelauswerter ermöglicht beispielsweise die dynamische Eingabe und Auswertung von Formeln, während ein Model-Analyse-Interpreter in der Lage sein sollte, Ausdrücke auszuwerten, die die

[4] oder durch einen Compiler in Maschinecode übersetzt werden

Konzepte und Eigenschaften der jeweiligen Fachdomäne wiedergeben. Er muss in der Lage sein, spezifizierte Objektmengen temporär zur weiteren Bearbeitung abzulegen. Die Eindeutigkeit einer DSL muss gewährleistet sein, DSLs sind jedoch bewusst nicht als Turing-vollständige Sprachen ausgelegt, wie dieses bei universellen Sprachen (GPL[5]) der Fall ist.

Die Erstellung eines Interpreters kann weitgehend durch Tools unterstützt werden. Bekanntlich wird unter lexikalischer Analyse der Prozess der Konvertierung einer Folge von Zeichen in eine Folge von Merkmalen (tokens) verstanden. Programme, die eine lexikalische Analyse durchführen werden lexikalischer Scanner oder kurz Lexer genannt. Tokens besitzen eine durch Grammatik gegebene Bedeutung (Datentyp, Schlüsselwort, Funktion, etc.). Ein Parser kann anschließend einen abstrakten Syntaxbaum (AST[6]) aufstellen, der die Hierarchie zwischen den Tokens wiedergibt. Der abstrakte Syntaxbaum spiegelt die Syntax wieder, mit der die DSL definiert wurde.

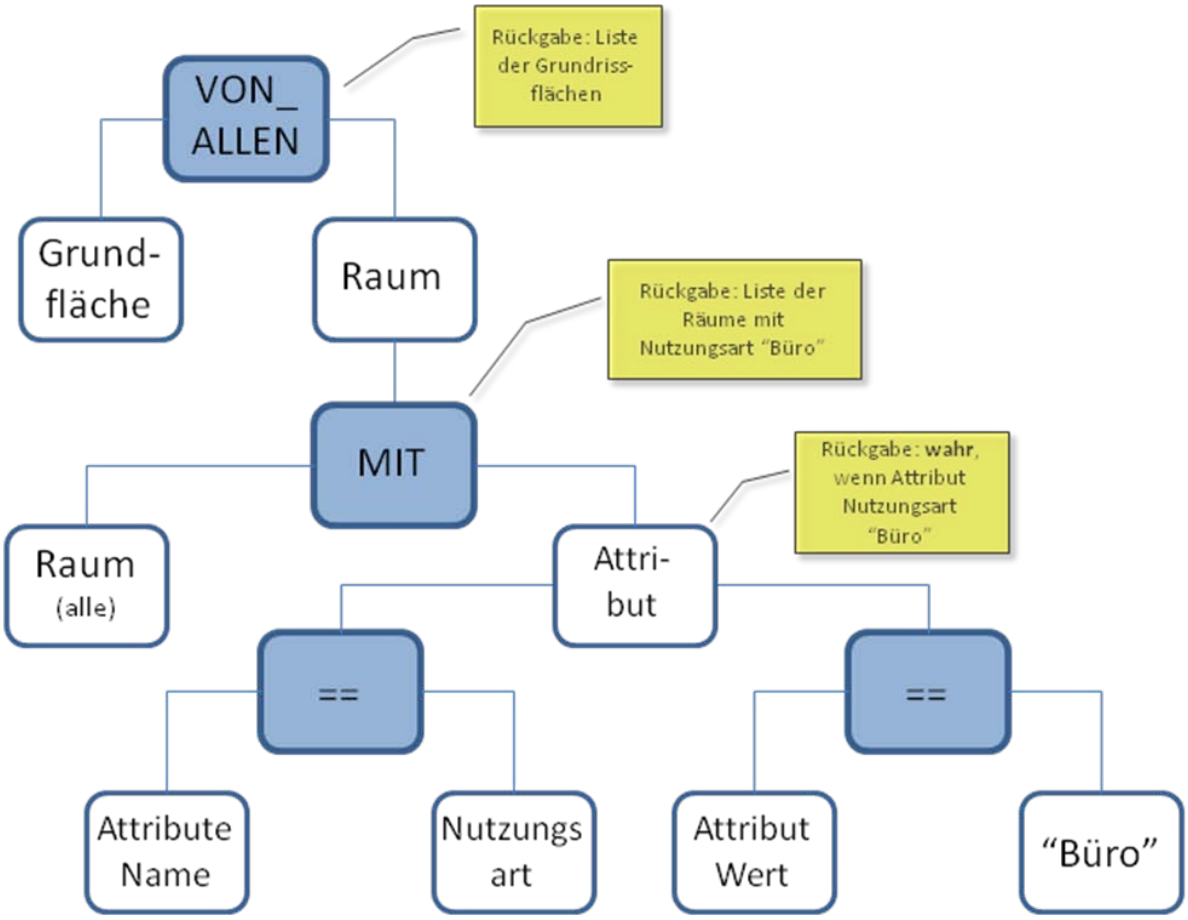

Abb. 2: AST Instanz des Beispielausdrucks

[5] GPL – general purpose language
[6] AST – abstract syntax tree

Um die Auswertung des Syntaxbaums von der Model-Logik zu trennen, nutzen wir das Entwurfsmuster des Visitor Pattern (Besuchermuster) nach [4].

Besucherobjekte können separat in modularer Weise entwickelt werden. Sie stellen die Inhalte zum Hinzufügen domänen-spezifischer Fachlogik durch Konfiguration bereit, dadurch kann Fachlogik zur Analyse nach Bedarf hinzugefügt werden. Es existiert eine Vielzahl von Tools zur halbautomatischen Erzeugung von Lexern und Parsern. Sie setzen EBNF Notationen in Programm Code um. Auf das genaue Verfahren zur Erzeugung eines Interpreters kann im Rahmen dieses Aufsatzes jedoch nicht näher eingegangen werden [2, 11].

Verbinden von Syntax und Semantik

Eine Hauptmotivation zur Nutzung von DSLs liegt darin, die technische Komplexität zu kapseln und dem Domänenexperten die Nutzung seiner eigenen Sprache zu ermöglichen. Der Interpreter kann die analytischen Ausdrucke einer DSL traversieren und „verstehen“. Die Semantik ist dabei innerhalb des Interpreters implementiert. Trifft der Interpreter einen Knoten im abstrakten Syntaxbaum an, ruft er die assoziierte Funktion auf und versorgt sie ggfs. mit den relevanten Daten als Parameter. Wichtig ist die Existenz einer wohldefinierten Relation zwischen den Namenskonventionen im abstrakten Syntaxbaum und im Besucherobjekt. Dieses wird durch die Verwendung derselben EBNF Notation für den Lexer und die Implementierung des Besucherobjekts sichergestellt.

Im Ausdruck „a + b“ hat jedermann eine unmittelbare Vorstellung des plus-Zeichens. Bei näherem Hinsehen stellt sich diese Sicherheit jedoch als unbegründet heraus. Offensichtlich hängt es vom Datentyp von ‚a‘ und ‚b‘ ab, welche Operation ausgeführt werden muss. Beispielsweise unterscheidet sich eine Matrixoperation „a + b“ deutlich von einer einfach Ganzzahladdition. Einige Programmiersprachen bieten einen Mechanismus zur Überdefinition existierender Operatoren an (z.B. operator overloading in C++ und C#), um Operatoren in unterschiedlichen Kontexten wieder zu verwenden. In unserem Beispiel wird ein Besucherobjekt für einen spezifischen Kontext entwickelt. Dieser Kontext ist durch die Fachdomäne gegeben und wird repräsentiert durch das zugrundeliegende Modellschema, beispielsweise IFC[7]. Deshalb wird der DSL-Ausdruck „Räume MIT Nutzungsart ‚Büro‘“ zwar immer in denselben Syntaxbaum transferiert, aber die Implementation des auswertenden Besucherobjekts hängt vom Schema des

[7] IFC – Industry Foundation Classes, ISO Standard 16739 für Produktmodelle im Bauwesen

verwendeten Modeltyps ab. So wird beispielsweise in einem cityGML-Modell[8] der Standort eines Gebäudes formal anders ausgedrückt, als in einem IFC-Modell. Deshalb sucht der Algorithmus zur Berechnung der Distanz zwischen Gebäude und einer gegebenen Route (Polylinie) nach unterschiedlichen Entitäten im Modell und löst Verweise unterschiedlich auf, je nach den Charakteristika des jeweiligen Modelltypen.

In unserem Fall repräsentiert der DSL-Interpreter die Sprache einer bestimmten Analyse, während die Besucherobjekt-Implementation die Zugriffsmethoden auf das Domänen-Modell kapselt. Ein DSL-Interpreter kann daher mit unterschiedlichen Besucherobjekttypen zusammenarbeiten. Dabei repräsentiert jeder dieser Typen die Semantik einer konkreten DSL.

Dynamische Einbindung von Methoden

Bevor das Beispiel weiter verfolgt werden kann, sind noch zwei Probleme zu lösen:

- Wie soll eine domänen-spezifische Syntax (AST) dynamisch mit einer domänenmodell-spezifischen Semantik assoziiert werden?
- Wie können Besucherobjekte dynamisch zur Laufzeit hinzugefügt werden?

Ein anderes Entwurfsmuster kann hier vorteilhaft eingesetzt werden: das Muster der ‚Abstrakten Fabrik' [4]. In unserem Fall ist es eine Fabrik für Besucherobjekte.

Alles was über ein konkretes Besucherobjekt (Instanz) wissen müssen ist die Basisklasse von der es abgeleitet wurde. Der konkrete Typ ist (und bleibt) für die aufrufende Applikation unbekannt.

Nach Gamma [4] verbirgt eine Fabrik den konkreten Typ eines erzeugten Objekts gegenüber der Anwendung. Die Anwendung erzeugt (indirekt über die Fabrik) das Objekt ohne seinen Typ zu kennen. Unser Ziel hier ist, Funktionalität per Konfiguration hinzufügen zu können. Daher muss das Fabrikobjekt selbst einen dynamischen Mechanismus besitzen, um neue Typen von Besucherobjekten instanziieren zu können. Dazu muss die Fabrik wissen, wo der ausführbare Code zu finden ist (z.B. in welcher .Net-Assembly), wie die Klasse des neuen Besucherobjekts benannt wurde und wie der

[8] cityGML – ISO Standard zur Repräsentation urbaner 3D-Objekte

interne Name der Besucherklasse lautet. Der interne Name wird zur Führung einer internen Liste verfügbarer Typen benötigt.

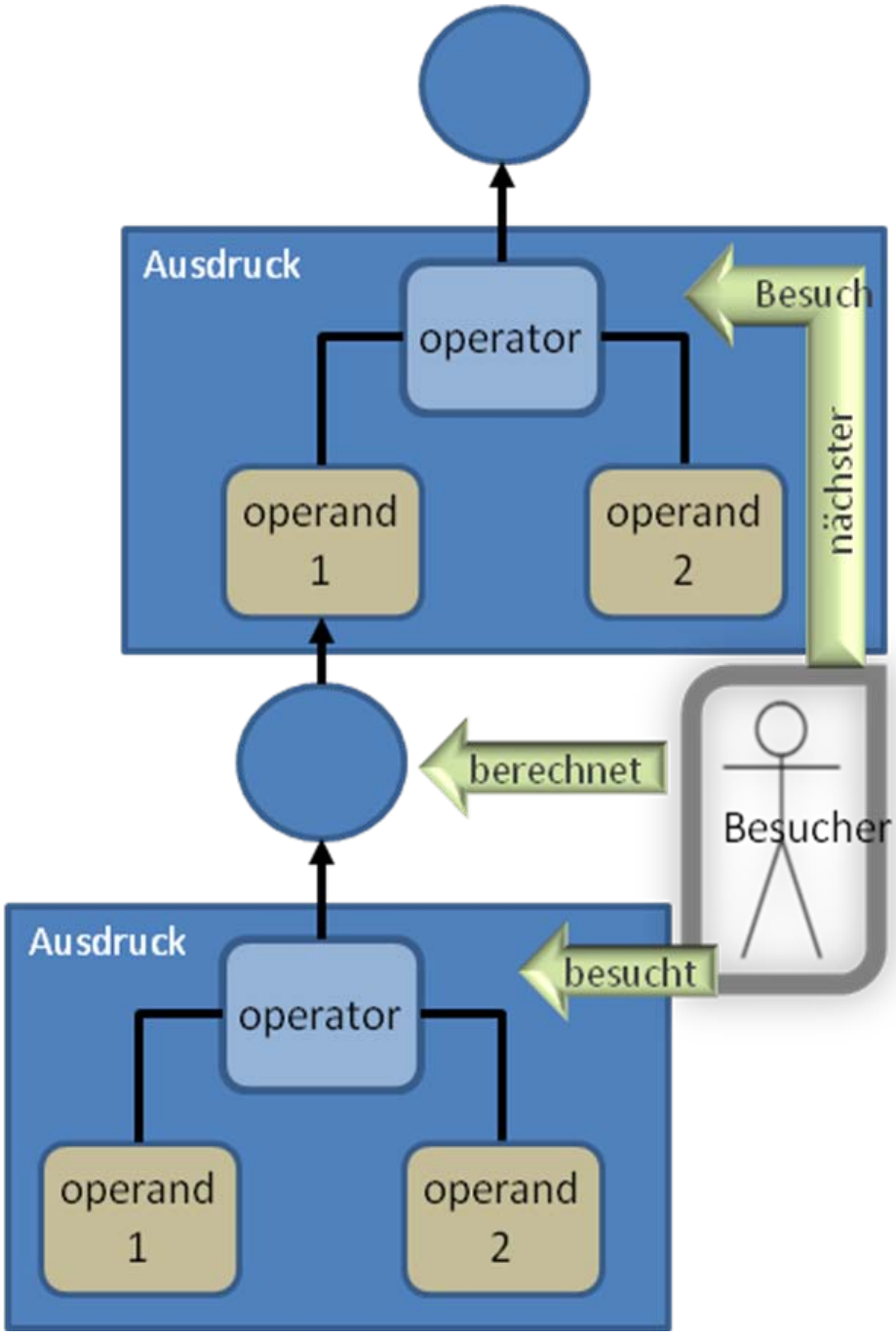

Abb. 3: Besucherobjekt traversiert einen AST

All diese Informationen können in einer externen Konfigurationsdatei gehalten werden, die zur Laufzeit vom Fabrikobjekt gelesen wird. Das Fabrikobjekt kann daraufhin alle referenzierten Code-Assemblies laden und nach Bedarf die gewünschten Besucherobjekte instanziieren (siehe C# Code Ausschnitt unten).

```
public static object CreateFactory(string virtualName)
{       string path = "../../VisitorFactory.config";
        XmlDocument doc = new XmlDocument();
        doc.Load(path);
        XmlNode factoriesNode = doc.SelectSingleNode("//Factories");
        foreach (XmlNode factoryNode in factoriesNode.ChildNodes)
        {
                if (virtualName.Equals(
                factoryNode.Attributes["virtualName"].Value))
                {
                        String assemblyAndTypeName =
                                factoryNode.Attributes["type"].Value;
                        return CreateInstance(assemblyAndTypeName);
                }
        …
```

Die interne Liste besitzt einen Eintrag zur Domänenzugehörigkeit eines bestimmten Besucherobjekts. Dieses versetzt den Interpreter in die Lage, die für die jeweilige Domäne zuständige Fabrik aufzurufen, um das geforderte Besucherobjekt zu instanziieren.

Zur bequemen Verarbeitung wird als Format für die Konfigurationsdatei XML gewählt (siehe XML Beispiel unten).

```
<configuration>
    <Factories>
        <Factory virtualName="OccupancyAnalysis"
            type="Factories.OccupancyFactory,AssemblyName1"
                domaineName=”IFC” subDomaine=”3.0”/>
        <Factory virtualName="BusLineAnalysis" type="Factories.
            BusLineFactory2, AssemblyName2" domaineName=”CityGML”
            subDomaine=”1.0”/>
    </Factories>
</configuration>
```

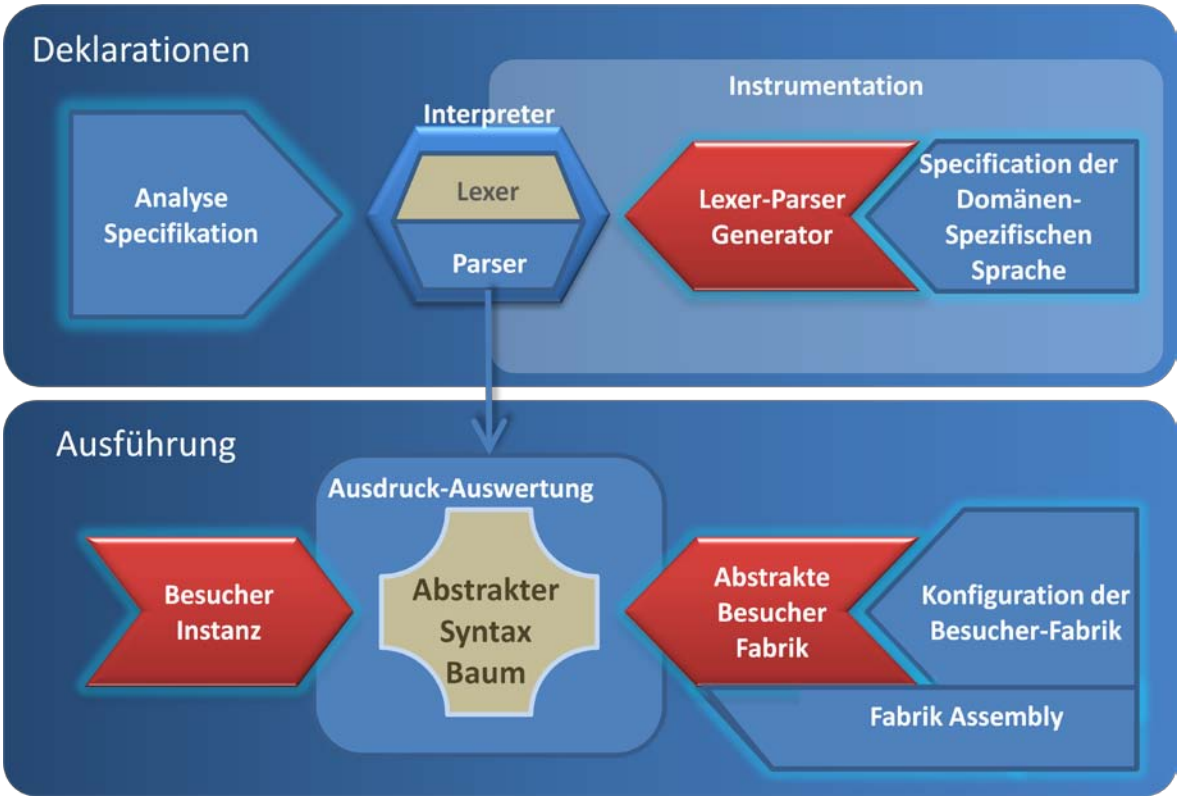

Bild 4: Analyse-Deklarationen, Instrumentierung und Ausführung

Zur Laufzeit kann der Interpreter nun die korrekte Visitorobjektfabrik aufrufen wenn er den voll qualifizierten Funktionsnamen (z.B. Domänenname + ‚.‘ + FunktionsName) im AST antrifft. Die Fabrik erzeugt nun das Besucherobjekt, das zur Auswertung des betreffenden Knotens im AST herangezogen wird.

3.3 Technologien und Hilfsmittel

ANTLR, ANother Tool for Language Recognition, ist ein Sprachen-Werkzeug, zur Erzeugung von Interpretern, Compilern und Übersetzern aus grammatikalischen Beschreibungen in einer Vielzahl von Programmiersprachen. Der generierte Code für Lexer, Parser und ähnliches kann von Anwendungen verwendet werden. ANTLR erwartet EBNF Notationen als Input zur Code-Generierung. Es stellt ausgezeichnete Unterstützung zur Erzeugung von Bäumen, Traversierung, Übersetzung und Fehlerbehandlung bereit [2].

„Oslo“ ist der Code-Name für Microsofts nächste Generation von Entwicklungsplattformen. „Oslo“ nutzt domänen-spezifische Modelle, Sprachen und Werkzeuge. Es fördert den Gedanken der modell-getriebenen Methodik und stellt eine Werkzeugpalette zur Erzeugung von DSLs bereit. Es wurden zwei Modellierungssprachen, MGrammar und MSchema, sowie das Datenformat MGraph entwickelt. Der erweiterbare Editor Intellipad unterstützt die Entwicklung von Schemas und Grammatiken [11].

Die Programmiersprache C#[9] besitzt eine in die Sprache integrierte Unterstützung für die Durchführung von Abfragen mit derselben formalen Abfragesprache, unabhängig vom Typ der Datenquelle[10]. Nach [5] ist die seit C# Version 3.0 zur Verfügung stehende Funktionalität (LINQ) annähernd identisch mit der von OCL[11]. Über seine Plattformunabhängigkeit hinaus bestehen die Hauptvorteile von OCL gegenüber traditionellen Programmiersprachen in der deklarativen Natur der Sprache, seiner mächtigen Navigationsfunktionalität durch die Verwendung von Iteratoren und die Verfügbarkeit von Tupeln. LINQ / OCL ist ein mächtiges Instrument zur Spezifikation einer Objektmenge in deklarativer Art und Weise. Die Transformation zwischen deklarativen Ausdrücken einer DSL und der entsprechenden Auswertung auf der semantischen Ebene kann daher Hand in Hand ohne Paradigmenwechsel durchgeführt werden.

Reflektions-orientierte Programmierung[12] erweitert das objekt-orientierte Paradigma um die Fähigkeit zur Selbstuntersuchung, Selbst-Modifikation und Selbst-Replikation. Das Hauptgewicht der reflektions-orientierten Programmierung liegt jedoch auf der dynamischen Programmmodifikation, die zur Laufzeit ermittelt und durchgeführt werden

[9] Standard ECMA-334 and ISO/IEC 23270
[10] LINQ – Language INtegrated Query
[11] OMG Standard
[12] Reflektions-getriebenes Paradigma, auch als reflective programming bezeichnet

kann. Reflektion wird in der Besucherfabrik zur Typvalidierung geladener Besucherobjekte und zur Ableitung des Datentyps durch den Interpreter verwendet.

4 Anwendung

Die Analyseplattform ist nun mit allen wesentlichen Elementen bestückt, wir können nun mit unserem Beispiel " Streckenführung einer neuen Buslinie“ fortfahren. Der Interpreter ist nun in der Lage das folgende Skript auszuführen:

modelKA = CityGML.LOAD_MODEL “Karlsruhe”

Im Interpreter wird nun das Besucherobjekt für die Domäne CityGML instanziiert, die mit dem Schlüsselwort ‚LOAD_MODEL’ assoziierte Methode wird mit dem Parameter " Karlsruhe" aufgerufen, das entsprechende Modell wird instanziiert. Schließlich wird die Modellinstanz der Variablen modelKA zugewiesenen, aus dem Rückgabewert des Funktionsaufrufs wird der Datentyp CityGML abgeleitet. Diese Implementierungsdetails bleiben bewusst vor dem Domänenexperten verborgen, damit er sich auf seine domänen-zentrische Sicht voll konzentrieren kann. Anschließend wird der nächste Ausdruck

route = LOAD_ROUTE “BUS_LINE_ALT_1”

verarbeitet. Hierdurch wird die geometrische Beschreibung der Streckenführung geladen und in der Variablen ‚route’ gespeichert.

Im nächsten Schritt werden die Objekte für den anschließenden Analyseschritt erzeugt. Dieses könnte auch vollständig innerhalb eines Besucherobjekts ablaufen, z.B. durch eine verknüpfte LINQ-Abfrage (join) innerhalb des Besucherobjekts, zur Verdeutlichung wird es jedoch hier im Detail gezeigt, um die Abfrage über die zwei Datenquellen der beteiligten Domänen hinweg darzustellen. In einer Schleife über alle Gebäude hinweg werden die Gebäudestandorte aus der CityGML-Domäne und die Grundflächen (‚Büro’ und ’privat’) aus der IFC-Domäne abgefragt. Innerhalb der Schleife werden IFC-Modelle instanziiert und die Werte der entsprechenden Grundflächen abgefragt.

ForEach (cityBuilding in modelKA)

Bemerkung: ‚cityBuilding' stammt aus der Domäne cityGML (abgeleitet durch den Interpreter)

ifcBuilding = IFC.LOAD_MODEL cityBuilding

instanziiert das IFC-Gebäudemodell, das im cityGML-Modell referenziert wurde

officeFloor += ifcBuilding.OfficeFloor

verwendet ein Besucherobjekt der IFC-Domäne, um OfficeFloor zu de-referenzieren. Aufeinander folgende Werte werden in der Variablen officeFloor aufsummiert

privateFloor += ifcBuilding.PrivateFloor

verwendet ein Besucherobjekt der IFC-Domäne, um PrivateFloor zu de-referenzieren. Aufeinander folgende Werte werden in der Variablen PrivateFloor aufsummiert

location = cityBuilding.Location

nutzt das Besucherobjekt der cityGML-Domäne, um den Gebäudestandort im cityGML-Modell zu de-referenzieren.

buildingList.Add(cityBuilding.ID, location, officeFloor, privateFloor)

Speichert die für die Analyse relevanten Attribute in einer Liste, ein Eintrag pro Gebäude

EndForEach

profile = EMBARKATION_PROFILE(buildingList)

berechnet die zu Zusteige-Profile

STORE_PROFILE(profile)

speichert das berechnete Profil

SHOW_PROFILE(profile)
visualisiert das Profil

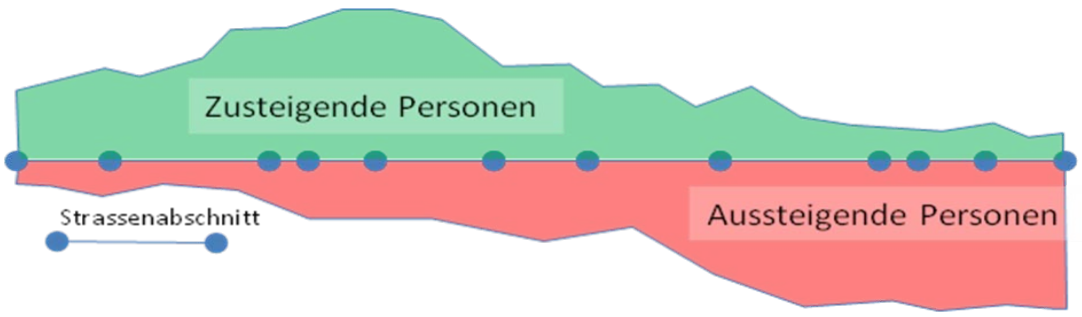

Abb. 5: Profil-Visualisierung (fiktives Beispiel)

Es wird deutlich, dass alternative Streckenführungen auf einfache Weise verglichen werden können, indem das gleiche Skript mit unterschiedlichen Routendaten ausgeführt wird.

Eine Verfeinerung der Ergebnisse durch Austausch der Algorithmen zur Grobabschätzung des Einzugsgebietes könnte jetzt ‚behind the scences'durch Austausch der Verweise auf die Besucherobjekte in der XML-Konfigurationsdatei stattfinden. Anschließend könnte das Skript mit den in die engere Auswahl kommenden Streckenführungen erneut ausgeführt werden.

5 Ergebnis und Ausblick

Leider bleibt bis heute die interne Logik vieler digitaler Modelle in Anwenderhandbüchern oder auch nur in den Köpfen einiger Spezialisten verborgen. Der Nutzen digitaler Modelle steigt mit deren Fähigkeit zur Interaktion mit Domänenexperten in einer effizienten und weniger zeitaufwändigen Art und Weise. Die Kapselung technischer Komplexität durch die Verwendung domänen-spezifischer Sprachen als Abstraktionsebene bietet sich zur effizienten Handhabung von Modellen an.

Der Grad der Komplexität den eine domänen-spezifische Sprache aufweist kann leicht skaliert werden, je nach der spezifischen Granularität, die für die jeweilige Analyse erforderlich ist. Wiederverwendbare Komponenten können dabei die Modellsemantik und die Modellzugriffe implementieren. Sie dienen dabei der semantischen Ebene als Grundlage, um unterschiedliche DSL-Ebenen aufzusetzen.

Heute beschränken sich die meisten Anwendungen auf die Speicherung und den Austausch digitaler Modelle. Durch die beschriebene Schnittstelle können Applikationen Nutzen ziehen, indem sie vorverarbeitete Daten je nach Anforderung der Domäne vom Modellerhalten können. In diesem Aufsatz sollte gezeigt werden, dass komplexe domänen-übergreifende Analysen deklarativ auf Domänen-Expertenebene gehandhabt werden können, wenn geeignete Attraktionen und Hilfsmittel zur Verfügung stehen. Standardtechnologien haben sich in den letzten Jahren außerordentlich weit entwickelt um domänen-spezifische Anforderungen abzudecken.

Literaturverzeichnis

[1] Starke, G.: Effektive Software Architekturen - Ein praktischer Leitfaden. Hanser, 2008

[2] [ANTLR] ANother Tool for Language Recognition, framework for constructing interpreters, compilers, etc. from grammatical descriptions. http://www.antlr.org/

[3] Denton, T. & Jones, E. & Srinivasan, S. &Owens, K. &Buskens, R. 2008. NAOMI-An Experimental Platform for Multi-modeling. In Proceedings of MODELS, pages 143–157, Toulouse, France, October 2008.

[4] Gamma, E. & Helm; Johnson, R. E. 1995. Design Patterns. **Elements of Reusable Object-Oriented Software.** Addison Wesley

[5] Akehurst, D.H. & Howells, W.G. & Scheidgen, M. & Mcdonald-Maier, K. D. 2007. Proceedings of the Workshop Ocl4All: Modeling Systems with OCL at MoDELS 2007, C# 3.0 makes OCL redundant!

[6] Hessellund, A. & Lochmann, H. 2008. An Integrated View on Modeling with Multiple Domain-Specific Languages. IASTED submission

[7] Hessellund, A. 2009. Domain-Specific Multimodeling, Ph.D. Dissertation (preprint), Supervisors: Peter Sestoft and Kasper Østerbye

[8] Mernik, M. & Heering, J. & Sloane. A. M. 2003 When and how to develop domain-specific languages. Submitted to ACM Computing Surveys

[9] Mellor, S. & Kendall, S. & Uhl, A. & Weise, D. 2004. MDA Distilled – Principles of the Model-Driven Architecture. Addison-Wesley

[10] [OMG_MDA] Object Management Group: Model Driven Architecture. www.omg.org/mda

[11] [oslo] Microsoft model-driven development platform pre-release. http://msdn.microsoft.com/oslo

[12] Stahl, T. & Völter, M. 2007. Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management. 2. Auflage, dpunkt,

[13] White, J.; Hill, J. H.; Tambe, S.; Gokhale, A., Schmidt, D. C.; Gray, J. 2009. Improving Domain-specific Language Reuse through Software Product-line Configuration Techniques. Submitted to IEEE Software Special Issue: Domain-Specific Languages and Modeling, Jul/Aug 2009. (http://www.dre.vanderbilt.edu/~jules/white-dslreuse.pdf)