Visuelle Programmiersprachen im Bauwesen

Stand der Technik und aktuelle Entwicklungen

Fabian Ritter[1], Cornelius Preidel[1], Dominic Singer[1] und Stefan Kaufmann[2]

[1]Lehrstuhl für Computergestützte Modellierung und Simulation · Technische Universität München·
Arcisstraße 21 · 80333 München· LEONHARD OBERMEYER CENTER
fabian.ritter@tum.de, cornelius.preidel@tum.de, dominic.singer@tum.de

[2]Lehrstuhl für Architekturinformatik · Technische Universität München·
Arcisstraße 21 · 80333 München· LEONHARD OBERMEYER CENTER
stefan.kaufmann@tum.de

Zusammenfassung

In der heutigen Bauindustrie wird das Entwerfen neuer Bauwerke eine zunehmend komplexe Aufgabe. Um mit dieser Herausforderung umzugehen werden neue Technologien benötigt, die repetitive Arbeiten automatisieren sowie die Erstellung von Varianten und deren Bewertung unterstützen. Parallel zu diesem Trend verbreitet sich der Einsatz von visuellen Programmiersprachen (visual programming language, VPL) im Bauwesen, da sie den Planern und Entscheidungsträgern erlauben Programme auf einfache Weise für spezifische Entwurfsaufgaben anzupassen.

Dieses Paper zeigt den aktuellen Stand der VPLs im Bauwesen auf und gibt einen kurzen Überblick über laufende Forschungsprojekte mit VPL-Einsatz.

Abstract

In today's Architecture, Engineering and Construction (AEC) industry, designing a new building is an increasing complex task. To cope with this issue, new technologies to automate repetitive work, support the creation of variants and the evaluation designs are necessary. Along with this technology, Visual Programming Languages (VPLs) are increasing in use in the AEC industry because they support designers and decision makers with easy to change programs they can modify for their specific design tasks.

This paper introduces the state of the art of VPLs in industry and gives a brief overview on current research projects using a VPL.

1. Einführung

Im letzten Jahrzehnt ist die Nutzung von visuellen Programmiersprachen stark angestiegen. Dies liegt vor allem daran, dass diese Nutzern die Möglichkeit bieten, repetitive Arbeiten oder auch Systeme mit starken Abhängigkeiten einfach zu modellieren, ohne größere Programmierkenntnisse vorauszusetzen.

1.1 Grundlagen

Im Allgemeinen wird eine visuelle Sprache als eine *formale Sprache mit einer visuelle Syntax und Semantik* definiert. Das heißt, es wird ein modulares System aus Zeichen und Regeln durch visuelle anstatt textuelle Elemente repräsentiert, im semantischen wie syntaktischen Bereich. Durch diese Art der Darstellung kann die visuelle Sprache sehr viel schneller und einfacher durch den Menschen interpretiert werden. Oft werden die visuellen Sprachen auch als *flow-based* bezeichnet, da sie komplexe Strukturen als Informationsfluss darstellen (SCHIFFER 1998). Daneben gibt es auch die Definition als *programmierte attribuierte Graph-grammatik (PAGG)* von BENKER ET AL. (1988). Eine Gegenüberstellung von visueller und textueller Beschreibung wird in Abbildung 1 gegeben.

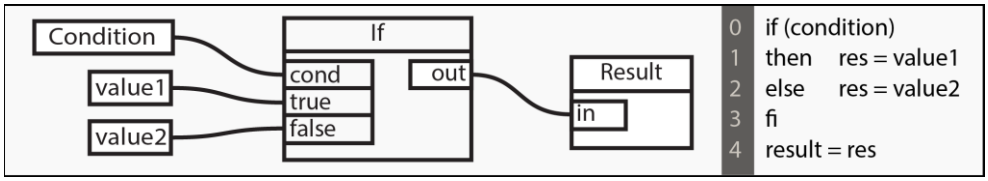

Abb. 1: Graphische Repräsentation und Pseudocode eines if-statements.

Um mit den visuellen Sprachen arbeiten zu können wird in der Regel eine Zeichenfläche (*canvas*) zur Verfügung gestellt Auf ihr können die einzelnen Komponenten (*nodes*) angeordnet und miteinander durch so genannte *wires* verknüpft werden. Des Weiteren werden üblicherweise weitere Funktionen und eine Bibliothek der *nodes* zur Verfügung gestellt, die die Arbeit mit der VPL erleichtern (Abbildung 2).

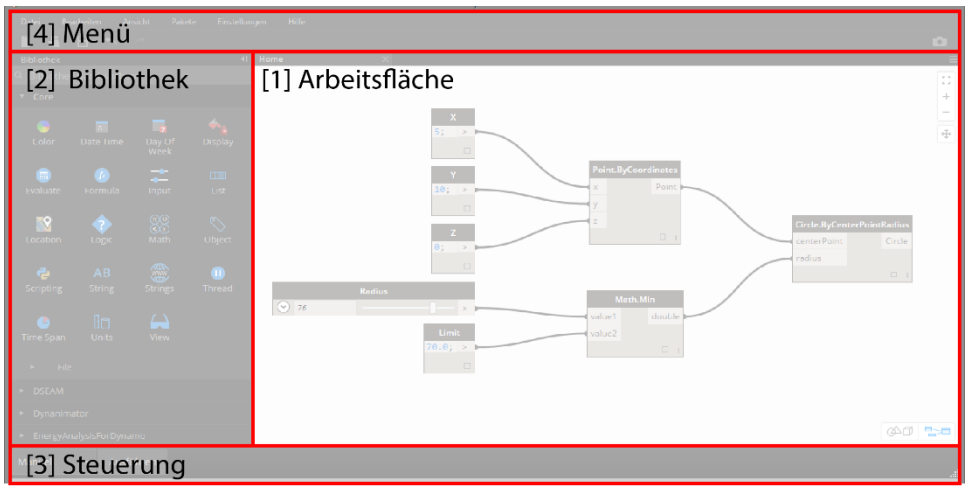

Abb. 2: Umgebung einer visuellen Programmiersprache: [1] Arbeitsfläche, dessen Inhalt ausgeführt wird, [2] Bibliothek, welche die verwendbaren Funktionen enthält, [3] Steuerung, zum Ausführen und stoppen des definierten Skripts und [4] Menü für weitere Funktionen, wie beispielsweise Speichern und Öffnen von Skripten oder undo/redo Funktionalitäten.

1.2 Geschichte der VPL

Bereits in den 1950er Jahren wurden die ersten graphischen Elemente entwickelt die die Grundlage für die Visuelle Programmierung bilden (Tabelle 1, Anhang). In den 60er Jahren kam schließlich mit dem *graphical program editor* das erste Produkt in Umlauf, das als Visuelle Programmiersprache bezeichnet wird. Mit diesem konnten Programme ähnlich einem Schaltplan erstellt werden. Danach folgten weitere Experimente mit den Darstellungsweisen, wie zum Beispiel den Nassi-Schneiderman-Diagrammen, bis sich schließlich die heute übliche Darstellungsweise eines Flussdiagramm ähnlichen Graphen durchsetzte, bei der die Ausgabewerte einer *node* als Eingabewerte für die nachfolgenden *nodes* übergeben werden.

1.3 Unterscheidungsmerkmale

Neben den oben beschriebenen grundlegenden Eigenschaften einer VPL gibt es Unterscheidungsmerkmale. Wichtigste Eigenschaft einer VPL ist deren Granularität. Das bedeutet, wie fein die einzelnen Funktionalitäten aufgelöst werden, d.h. ob Funktionen innerhalb einer *node* gekapselt werden oder jeder Teilschritt als eigene *node* zur Verfügung steht. Dabei steht die einfache Handhabung und Übersicht (gekapselt) der höheren Anpassungsmöglichkeit (Teilschritte in eigenen *nodes*) gegenüber.

Ein weiteres Merkmal ist die Darstellung der *nodes*. Diese können mit beispielsweise mit zugehöriger Klasse und Funktion bezeichnet oder einfach als Piktogramm dargestellt werden. Vorteile der Piktogramme sind der geringere Platzverbrauch und die logische Unterstützung des graphischen Charakters einer VPL. Problem hierbei ist jedoch, dass die Piktogramme oft nicht eindeutig durch den Nutzer interpretiert werden können.

Da alle diese Merkmale ihre Vor- und Nachteile haben, beherrschen einige VPL Umgebungen Möglichkeiten die Merkmale zu kombinieren, sei es durch Gruppieren von sub-graphen in eine eigene *node*, um Teilschritte innerhalb der VPL zu kapseln oder durch umschalten zwischen der Darstellung der *nodes* durch Bezeichnung oder eines Piktogramms.

1.4 Vor und Nachteile

Der Nutzen einer visuellen Programmiersprache wird kontrovers diskutiert. Als Nachteil wird zumeist angeführt, dass die mit einer VPL erstellten Programme meist nicht den hohen Anforderungen an eine Programmierumgebung genügen. Außerdem können komplexere Sachverhalte, wie beispielsweise die Rekursion, oft nicht umgesetzt werden.

Dem gegenüber steht die Nutzerfreundlichkeit einer VPL. Sie ist aufgrund ihrer abstrakten Repräsentation für Personen ohne Programmierkenntnisse leichter zu verstehen und wird daher auch schneller eingesetzt. Dies ist dadurch zu begründen, dass Bilder Dinge einfacher kommunizieren und knapper darstellen können, das Verstehen und Erinnern unterstützen sowie keine Sprachbarrieren aufweisen und somit von Personen jeder Sprache verstanden werden können (SHU 1988): Einen Beweis hierfür liefern einerseits die Verbreitung der VPL wie eine Studie von CATARCI UND SANTUCCI (1995), die die Nutzerfreundlichkeit einer Visuellen Anfragesprache mit der Nutzung von SQL verglichen haben.

2. Einsatzgebiete

Nach den Grundlagen werden in den folgenden Abschnitten verschiedene Einsatz- und Anwendungsbereiche für eine VPL aufgezeigt. Allerdings verlaufen die Grenzen zwischen den einzelnen Gebieten sehr fließend und eine eindeutige Klassifizierung ist nicht möglich. Eine erste grobe Einteilung ist, dass VPLs im Bauwesen zum einen für generative Zwecke (generativer Charakter) eingesetzt werden, um Beispielsweise Modelle zu erzeugen, oder für das Überprüfen bzw. Abfragen von Informationen bestehender Modelle verwendet werden (prüfender Charakter). In diese zwei Kategorien können alle unten vorgestellten Ansätze gegliedert werden.

2.1 Anfragesprachen

Der größte Einsatzbereich der VPLS befindet sich im Bereich der Anfragesprachen. Sie werden für Anfragen in Datenbanken eingesetzt (TJAN ET AL. 1993; KWAK AND MOON 1993) oder zur direkten Visualisierung von wissenschaftlichen Daten in einer grafischen Oberfläche (HILS 1993). Auch in kommerziellen Produkten wie Microsoft ACCESS haben sich grafische Anfragen durchgesetzt.

Im Bauwesen wird diese Funktionalität zunehmend für die Anfrage von Bauwerksmodellen genutzt. Beispiele hierfür sind das Tool BIMcraft (WÜLFING ET AL. 2014) im Projekt *eWork-Bau* für den Einsatz von Handwerkern auf der Baustelle, um BIM Modelle nach relevanten Bauteilen ohne Kenntnisse von Anfragesprachen zu bekommen und die Sprache QL4BIM, die parallel zur klassischen Anfrage im Textformat eine Anfrage per VPL anbietet. Damit lassen sich 4D-Bauwerksmodelle im IFC-Standard (DAUM UND BORRMANN 2015) und in Kombination mit CityGML filtern (DAUM ET AL. 2015).

2.2 Geometrische Modellierung

Die bekannteste und älteste Sprache im Bereich der geometrischen Modellierung ist das seit September 2007 erhältliche Grasshopper für Rhinoceros 3D. Mithilfe dieser Sprache lassen sich Freiformgeometrien erstellen und modifizieren. Durch den langjährigen Einsatz hat Grasshopper die derzeit größte Community auf diesem Gebiet und daher auch die meisten Erweiterungen, darunter auch Berechnungs-, Vernetzungs- und Optimierungsalgorithmen.

Angeregt durch den Erfolg von Grasshopper entwickeln derzeit weitere Softwarehersteller eigene VPLs. Das wohl vielversprechendste Produkt ist Dynamo für Revit von Autodesk. Es hebt sich von Grasshopper dadurch ab, dass es eine direkte Modifikation von in Revit erstellten Komponenten zulässt und Semantik versteht. Außerdem ist Dynamo open-source, kann an weitere Anwendungen angeknüpft werden und ist daher nicht auf eine Applikation beschränkt. Eine weitere, derzeit noch nicht verfügbare Alternative wird gerade von Vectorworks mit der Sprache Marionette entwickelt.

Daneben finden sich in Anwendungen für die geometrische Modellierung bisweilen auch an visuelle Programmiersprachen angelehnte Materialeditoren, vor allem in auf das rendern spezialisierten Programmen wie Autodesk 3dsMax und Maya, Maxon Cinema 4D oder Blender von blender.org.

2.3 Wissensbasierte Konstruktion

Als Spezialgebiet der geometrischen Modellierung kann die Erzeugung von Geometrie durch wissensbasierte Systeme gesehen werden. Ziel der wissensbasierten Konstruktion ist es, das dem Entwurf von Ingenieurbauwerken zugrunde liegende Konstruktionswissen (Regeln bzw. Prozeduren) so abzubilden, dass ein teilautomatisierter Entwurf bzw. eine automatisierte Anpassung des Modells an veränderte Randbedingungen möglich ist. Derzeit werden zur Modellierung des Datenmodells sowie zur Definition von Design Regeln innerhalb von gängigen wissensbasierten Systemen schwer verständliche deklarative domänenspezifische Sprachen eingesetzt. Die geometrische Modellierung des Produkts findet meist direkt in den verknüpften CAD System statt. Ersteres ist für Anwender oder Experten ohne Programmierkenntnisse nur schwer verständlich, zweites ist aufgrund der Abhängigkeit der Wissensdatenbank vom eingesetzten CAD System negativ zu bewerten.

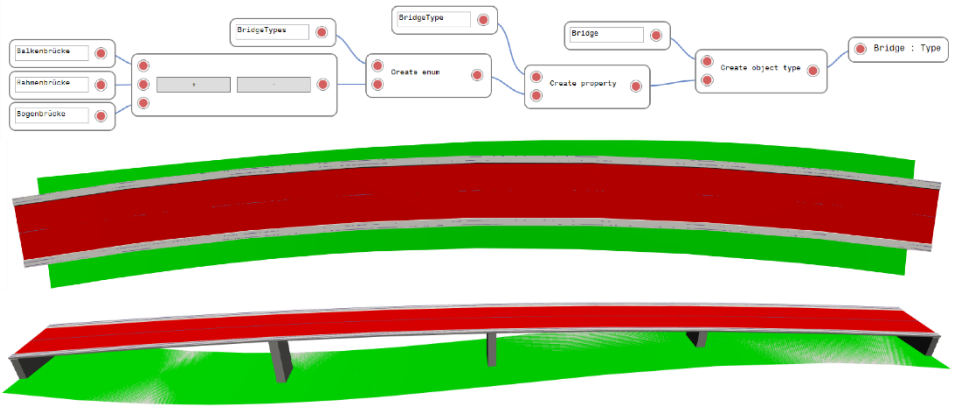

Abb. 3: Einsatz einer VPL in einem wissensbasierten System für den Infrastrukturbau.

Einen Einsatz von VPLs für alle genannten Bereiche, nämlich der Modellierung des Datenmodells, der Definition von Design Regel und der geometrischen Modellierung der digitalen Bauwerksmodelle, zeigt SINGER (2015) in seinem Forschungsansatz eines wissensbasierten Systems für den Infrastrukturbau. Der Einsatz einer VPL ist sowohl für die Wissensmodellierung durch Experten als auch für die Nachvollziehbarkeit des regelbasierten Konstruktionsablaufs durch Nicht-Experten als vorteilhaft anzusehen. Die Verwaltung, Erweiterung und Verbesserung der Wissensdatenbank ist mit Hilfe einer VPL auch ohne klassische Programmierkenntnisse möglich. Ferner entfällt die Erklärungskomponente, die den Anwender üblicherweise mit Informationen zum Ablauf einzelner Konstruktionsschritte versorgt, da in einer VPL die weitergegebenen Daten an allen Ports verfolgt werden können.

Im Rahmen der automatisierten Erzeugung von BIM Modellen ist eine VPL in Entwicklung, die sowohl deklarative als auch prozedurale Elemente zur Beschreibung des Entwurfswissens vorsieht. Dies wird durch die Implementierung abstrakter Geometrieobjekte so genannte *High-Level Primitives* (HLP) und Prozessmodulen so genannten *Capability Moduls* (CM) direkt in der VPL realisiert. Für die Verwendung einer VPL innerhalb eines wissensbasierten Systems ist außerdem die Wahl eines brauchbaren Abstraktionsniveaus für den Endanwender

notwendig. Durch die Einführung von *Abstraction Layern* innerhalb der VPL können verschiedene Detaillierungsstufen für die verschiedenen Beteiligten wie Anwender oder Experte definiert und so das Arbeiten auf einer Abstraktionsschicht je nach Rolle beschränkt oder erlaubt werden.

2.4 Design Decision Support

Eine Erweiterung des Einsatzes zur Erzeugung von Geometrie bietet der *design decision support* (DDS). Hier werden durch eine VPL zusätzliche Analysen an das geometrische Modell gekoppelt, um zusätzliche Informationen zum Modell zu erzeugen und somit die Entscheidungsfindung im Entwurf zu unterstützen. Beispiele hierfür sind der *urban strategy playground* (USP), die Informationen zu Stadtquartieren wie Bebauungsdichte, Abstandsflächen, Mindestbauhöhen analysiert und visualisiert um Nachverdichtungsstrategien entwickeln zu können (SEIFERT UND MÜHLHAUS 2014, 2015), oder die *design space exploration assistance method* (DSEAM), die zu einem Gebäudemodell simulationsbasierte Informationen zum späteren Energiebedarf und Materialverbrauch erzeugt, um Rückschlüsse auf Auswirkungen des geometrischen Entwurfs auf diese immer wichtiger werdenden Eigenschaften zuzulassen (RITTER 2015).

2.5 Code Checking

Ein weiterer nachgelagerter Schritt ist die Überprüfung einer Gebäudeplanung hinsichtlich ihrer Konformität mit geltenden Richtlinien. Diese wird zumeist in einem immer wiederkehrenden, manuellen Kontrollprozess in der Planungsphase eines Bauwerks durchgeführt. Dies führt aufgrund der enormen Anzahl unterschiedlicher regionaler bzw. nationaler Regelungen und der Vielzahl von Fachdisziplinen im Bauwesen zu hoher Fehleranfälligkeit, Arbeitsaufwand und Kosten.

Bei der Automatisierung dieses Prozesses auf Basis eines digitalen Gebäudemodells stellen sich zwei Kernanforderungen: die Aufbereitung der Informationen des Modells und die Übersetzung der Inhalte eines Regelwerks in eine maschinen-interpretierbare Sprache (EASTMAN 2009). Mit der Entwicklung der *visual code checking language* (VCCL) soll die Übersetzung der Regelwerke mit Hilfe einer visuellen Sprache ermöglicht werden und gleichzeitig die Transparenz des gesamten Prozesses gewahrt bleiben (PREIDEL 2015). Auf der semantischen Ebene der VCCL können mit Hilfe von Objekt- und Operatorknoten sämtliche Entitäten, Informationen und Verarbeitungsprozesse sowohl von Gebäudedatenmodell und Regelwerk dargestellt werden, so dass alle Objekte einer Überprüfung ganzheitlich abgebildet werden können. Um diese zu verknüpfen und somit schließlich den Überprüfungsprozess in Form eines Informationsflusses zu formulieren, besitzt die VCCL auf syntaktischer Ebene die klassischen Instrumente einer visuellen Sprache: gerichtete Kanten und Schnittstellen, welche den Informationsfluss innerhalb des Graphen eindeutig und konsistent abbilden. Auf diese Weise soll es auch Anwendern ohne fundierte Programmierkenntnisse möglich sein, ihr Fachwissen und Erfahrung in den Überprüfungsprozess einzubringen und den gesamten Prozess transparent und übersichtlich zu gestalten, so dass der Anwender schließlich auch die Verantwortung für die erhaltenen Ergebnisse übernehmen kann.

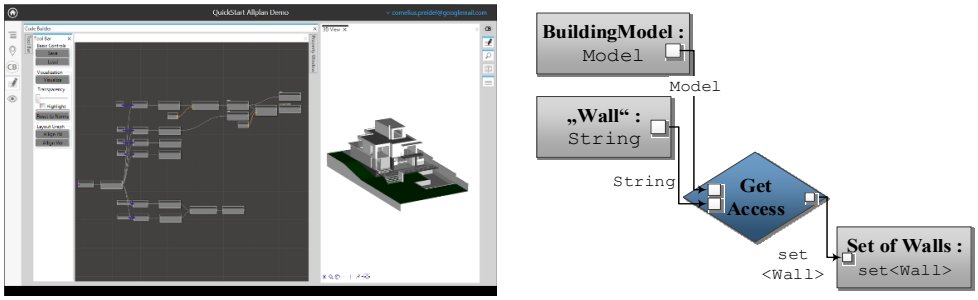

Abb. 4: links: Nutzeroberfläche für die Formulierung von Überprüfungen mit Hilfe der VCCL; rechts: Exemplarische Anfrage aller Wände in einem Gebäudemodell mit Hilfe der VCCL

In enger Kooperation mit der Nemetschek Group wurde ein erster Prototyp auf Basis der BIM-Plattform bim+ entwickelt und erfolgreich für die Anwendung und Überprüfung Deutschen Brandschutznorm DIN 18232-2:2007 getestet (DIN 18232-2:2007, PREIDEL 2015).

2.6 Modellierung von Systemen

Ein weiteres bekanntes Einsatzgebiet im Bauwesen ist die Modellierung von technischen Gebäudesystemen. Diese dienen als Grundlage für die Simulation von Energiebedarf sowie der Dimensionierung der Systeme und hat sich dort schon etabliert. Bekannte Beispiele hierfür sind das Simulation Studio für TRNSYS von TransSolar sowie Dymola (Catia Systems Engineering), SimulationX (ITI GmbH) und SystemModeler (Wolfram) für die Simulationsbibliothek (Open)Modelica.

3. Zusammenfassung

Visuelle Programmiersprachen (VPL) rücken nach der anfänglichen Euphorie Ende der 80er, Anfang der 90er Jahre wieder stark in den Fokus aktueller Entwicklung für die Bedienung und Anpassung von Software. Dieses Paper gibt nach einer kurzen Einführung und Einordnung einen Überblick über die bekannten und aufkommenden Einsatzgebiete der VPL im Bauwesen. Dabei wurden generative Ansätze zur geometrischen Modellierung, der wissensbasierten Konstruktion und der Modellierung von Systemen sowie prüfende Ansätze der Anfragesprachen, Design Decision Support und Code Checking vorgestellt.

Literatur

BENKER, B., GÖTTLER, H. UND NIESKENS, G. (1988). Ein Entwicklungssystem zur Unterstützung des 'Visual Programming': Drei Beiträge zum Einsatz programmierter attributierter Graphgrammatiken. Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik) 21(3).

CATARCI, T., AND SANTUCCI, G. (1995). Are Visual Query Languages Easier to Use than traditional Ones? An Experimental Proof. In: People and computers X: Proceedings of HCI '95, Hudders-field, August 1995, edited by Kirby, M.A.R, Dix, A., Finlay, J.E., Cambridge programme on human-computer interaction. Cambridge University Press.

DAUM, S., BORRMANN, A. (2015), Simplifying the Analysis of Building Information Models Using tQL4BIM and vQL4BIM. In: Proceedings of the EG-ICE 2015, Eindhoven.

DAUM, S., BORRMANN, A., KOLBE, T. (2015), A spatio-semantic query language for the integrated analysis of city models and building information models. In: Proceedings of the 3D Geoinfo 2015, Kuala Lumpur.

DIN 18232-2 (2007), Rauch- und Wärmefreihaltung – Teil 2: Natürliche Rauchabzugsanlagen (NRA); Bemessung, Anforderungen und Einbau.

EASTMAN, C.; LEE, J.; JEONG, Y.; LEE, J.-K. (2009), Automatic rule-based checking of building designs. Automation in Construction. 18, pp 1011–1033.

HILS, D.D. (1993). A Visual Programming Language for Visualization of Scientific Data. UIUCDS-R-93-1809.

KWAK, J.C., MOON, S. (1993). "Object Query Diagram: An Extended Query Graph for Object-Oriented Databases." In: Proceedings of the IEEE Symposium on Visual Languages, 1993, 44–48.

PREIDEL, C.; BORRMANN, A. (2015), Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling. In: Proc. of the 32nd ISARC 2015, Oulu, Finland.

RITTER, F., GEYER, P., BORRMANN, A. (2015). Simulation-based Decision-making in Early Design Stages. In: Proceedings of the 32rd international CIB W78 conference, Eindhoven, Netherlands.

SCHIFFER S. (1998), Visuelle Programmierung - Grundlagen und Einsatzmöglichkeiten. Bonn: Addison-Wesley.

SHU, N.C. (1988). *Visual programming.* New York: Van Nostrand Reinhold.

SEIFERT, N.; MÜHLHAUS, M.; SCHUBERT, G.; FINK, D.; PETZOLD, F. (2014), Decision support for inner-city development – An interactive customizable environment for decision-making processes in urban planning. In: Proceedings of eCAADe 2014, Newcastle, UK, S. 43-52

SEIFERT, N., MÜHLHAUS, M. (2013). Entscheidungsunterstützende Werkzeuge für die Nachverdichtung. In: 25. Forum Bauinformatik 2013. 1., Aufl. Herzogenrath: Shaker (Berichte aus der Bauinformatik), S. 107–119.

SINGER, D. (2015), Einsatz von Knowledge-based Engineering Methoden in frühen Phasen des Brückenentwurfs. In: Proceedings des 27. Forum Bauinformatik, Aachen.

TJAN, B.S., BRESLOW, L., DOGRU, S., RAJAN, V., RIECK, K., SLAGLE, J.R., POLIAC, M.O. (1993). A Data-Flow Graphical User Interface for Querying a Scientific Database. *Visual Languages, 1993. Proceedings 1993 IEEE Symposium on*, 49–54.

WÜLFING, A., WINDISCH, R., SCHERER, R. J. (2014), A visual BIM query language. In *10th European Conference on Product & Process Modelling*, edited by A. Mahdavi, Bob Martens, and R. J. Scherer, 157–64: CRC Press.

Anhang

Tabelle 1: Entstehungsgeschichte der Visuellen Programmierung (nach SCHIFFER 1998)

Jahr	Sprache (Autor)	Neuheit/Besonderheit
Ende der 50er	- (Haibt)	Automatische Generierung von Flussdiagrammen
1963	Sketchpad (Sutherland)	Erstes Zeichenprogramm mit Mensch-Maschine Interaktion
1966	Graphical Program Editor (Sutherland)	Erstes VP, repräsentiert Programme ähnlich Hardware-Schaltplänen
1969	Grail (Ellis et al)	Maschinenlesbare Flussdiagramme
1971	AMBIT/G und /L (Christensen)	Erzeugung durch Grafentransformation
1975	Pygmalion (Smith)	Arbeiten mit Piktogrammen auf einem *executable electronic blckboard*
1978	Programming Support System (Frei et al)	Eingabe durch Nassi-Shneiderman-Diagramme (Struktogramme)
1978	QBE: Query-by-Example (Zloff)	Datenbanksprache
1984	Pict (Glienert und Tanimoto)	Eingabe rein mit Maus und Joystick
1985	Prograph (Matwin und Pietrzykowski)	Grundlage für die erste kommerzielle VPL

Tabelle 2: Bekannte VPL Bibliotheken.

Name	Anwendung	Sprachen	Verfügbar unter	Open Source
Google Blockly	Web-based		https://developers.google.com/blockly/ https://github.com/google/blockly	■
Dynamo	Autodesk Produkte	C# IronPython	https://github.com/DynamoDS/Dynamo	■
VPL	Microsoft		https://msdn.microsoft.com/en-us/library/bb964572.aspx	□
Grasshopper	Rhinoceros3D	C#, VisualBasic, Python	http://www.grasshopper3d.com/page/scripting-and-code-tutorials	□
TUM.CMS. VPLControl	Allgemein	.NET	https://github.com/tumcms/TUM.CMS.VPLControl	■