
GESELLSCHAFT
FÜR INFORMATIK

Matthias Riebisch,
Marina Tropmann-Frick (Hrsg.)

Modellierung 2022

27. Juni - 01. Juli 2022
Hamburg, Deutschland

Gesellschaft für Informatik e.V. (GI)

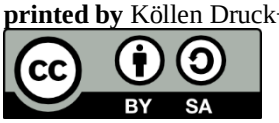
Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-324
ISBN 978-3-88579-718-0
ISSN 1617-5468

Volume Editors
Prof. Dr. Matthias Riebisch
Universität Hamburg, FB Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg
matthias.riebisch@uni-hamburg.de

Prof. Dr. Marina Tropmann-Frick
HAW Hamburg, Department Informatik, Berliner Tor 7, 20099 Hamburg
Marina.Tropmann-Frick@haw-hamburg.de

printed by Köllen Druck+Verlag GmbH, Bonn



Vorwort

Die Fachtagung „Modellierung“ ist eine Plattform zur inhaltlichen Diskussion für eine große Anzahl von Fachgruppen in der Gesellschaft für Informatik (GI), die sich mit unterschiedlichsten Perspektiven des Themas Modellierung beschäftigen. Sie stellt somit ein zentrales Forum für den Erfahrungsaustausch zu akademischen wie auch praxisbezogenen Modellierungsansätzen dar.

Die Themenbereiche der Entwicklung, Nutzung, Kommunikation und Verarbeitung von Modellen sind so vielfältig wie die Informatik mit all ihren Anwendungen. Systeme über mehrere technische Plattformen hinweg sowie die Einbeziehung des organisatorischen und technischen Umfelds in den Entwurf stellen zusätzliche Herausforderungen dar. Digital Twin als Abbildung der Eigenschaften von Systemen ist ein Konzept, das eine solche Unterstützung verspricht. Die Anwendung für Domänen wie Embedded Systems, Automatisierung, Fertigungstechnik, Smart City und Unternehmensorganisation verspricht viele Vorteile.

Als Forum für Grundlagen, Methoden, Techniken, Werkzeuge sowie Domänen und Anwendungen der Modellierung hat sich die Fachtagung „Modellierung“, die vom Querschnittsfachausschuss Modellierung der GI seit 1998 durchgeführt wird, etabliert. Sie zeichnet sich durch aktuelle Themen, lebendige Diskussionen und engagierte Rückmeldungen aus Wissenschaft und Praxis aus.

Im Jahr 2022 wurde die Fachtagung „Modellierung“ mit dem Schwerpunktthema Digital Twin an der HAW Hamburg ausgerichtet. Das Programm der Modellierung 2022 umfasste neben den wissenschaftlichen Fachvorträgen sechs Workshops, ein Tutorial sowie ein Praxisforum zusammen mit Präsentationen von Tools & Demos.

Für das wissenschaftliche Programm der Modellierung 2022 wurden von insgesamt 20 Einreichungen die besten 7 als Langbeitrag ausgewählt. Dies entspricht einer Annahmequote von 38,9%. Weitere vier Einreichungen wurden als Kurzbeiträge akzeptiert. Die Begutachtung aller Einreichungen erfolgte durch ein blindes Beurteilungsverfahren mit jeweils drei Gutachten pro Einreichung. Die akzeptierten Beiträge behandeln aktuelle wissenschaftliche Erkenntnisse zu einer breiten Palette von Themen in den Bereichen Digital Twin, Model-Driven Engineering, Modelle und Semantik, Unternehmens- und Geschäftsprozess-Modellierung.

Im Rahmen der Modellierung 2022 Tagung fanden drei eingeladene Vorträge statt.
Wir danken allen Vortragenden für ihre Beiträge und den Mitgliedern des Programmkomitees für die zeitgerechte Erstellung der hochwertigen Gutachten.

Ein großer Dank geht an die Organisatoren der Workshops, Tools & Demos und Tutorien, welche wesentlich zur Attraktivität und Vielfältigkeit der Konferenz beigetragen haben. Weiterhin bedanken wir uns bei allen Personen, die an der Organisation der Tagung in ganz unterschiedlichen Rollen beteiligt waren.

Hamburg, im Juni 2022

Matthias Riebisch, Marina Tropmann-Frick

Sponsoren

Wir danken den folgenden Institutionen für die Unterstützung der Konferenz.

Hochschule für Angewandte Wissenschaften Hamburg

Universität Hamburg

Gesellschaft für Informatik

Querschnittsfachausschuss Modellierung

Die Plattform der GI zur Diskussion und zum Erfahrungsaustausch über aktuelle und zukünftige Themen der Modellierungsforschung . Beteiligte GI-Gliederungen:

- ARC Architekturen
- ASE Automotive Software Engineering
- EMISA Entwicklungsmethoden für Informationssysteme und deren Anwendung
- FoMSESS Formale Methoden und Software Engineering für Sichere Systeme
- MMB Messung, Modellierung und Bewertung von Rechensystemen
- MobIS Modellierung betrieblicher Informationssysteme
- PN Petrinetze
- RE Requirements Engineering
- ST Softwaretechnik
- WI-VM Vorgehensmodelle für die betriebliche Anwendungsentwicklung
- WM/KI Wissensmanagement

Tagungsleitung

Leitung des Programmkomitees:	Matthias Riebisch, Universität Hamburg
	Marina Tropmann-Frick, HAW Hamburg
Workshops:	Judith Michael, RWTH Aachen
	Andreas Wortmann, Universität Stuttgart
	Jérôme Pfeiffer, Universität Stuttgart
Praxisforum, Tools & Demos:	Dominik Bork, Technische Universität Wien
	Simon Hacks, University of Southern Denmark
Tutorien:	Bernhard Thalheim, Universität Kiel
	Heinrich C. Mayr, Alpen-Adria-Universität Klagenfurt

Programmkomitee

Jörg Desel	Fernuniversität Hagen
Jürgen Ebert	Universität Koblenz-Landau
Gregor Engels	Universität Paderborn
Peter Fettke	DFKI, Universität des Saarlandes
Hans-Georg Fill	University of Fribourg
Ulrich Frank	Universität Duisburg-Essen
Holger Giese	Hasso Plattner Universität Potsdam
Martin Glinz	Universität Zürich
Florian Johannsen	Hochschule Schmalkalden
Gerti Kappel	Technische Universität Wien
Dimitris Karagiannis	Universität Wien
Walter Koch	Schaeffler AG
Agnes Koschmider	Universität Kiel
Thomas Kuehne	Victoria University of Wellington
Florian Matthes	Technische Universität München
Heinrich C. Mayr	Universität Klagenfurt
Jan Mendling	Wirtschaftsuniversität Wien
Judith Michael	RWTH Aachen
Daniel Moldt	Universität Hamburg
Günther Müller-Luschnat	
Friederike Nickl	Swiss Life Deutschland
Markus Nüttgens	Universität Hamburg
Andreas Oberweis	Karlsruher Institut für Technologie
Alixandre Santana	Universidade Federal do Agreste de Pernambuco
Ana Nicolaescu	Mercedes Benz
Andreas Steffens	RWTH Aachen
Antonio Garmendia	JKU Linz
Benjamin Ternes	FernUni Hagen
Jens Gulden	Utrecht University
Jonas Friederich	University of Southern Denmark
Stephan Kühnel	Martin-Luther-Universität Halle-Wittenberg
Syed Juned	Ali TU Wien

Sven Overhage	Universität Bamberg
Henderik Proper	Public Research Centre Henri Tudor
Ulrich Reimer	FH St. Gallen
Wolfgang Reisig	HU Berlin
Anne Remke	Universität Münster
Matthias Riebisch	Universität Hamburg
Stefanie Rinderle-Ma	Universität Wien
Bernhard Rumpe	RWTH Aachen
Kurt Sandkuhl	Univesität Rostock
Klaus Schmid	Universität Hildesheim
Andy Schürr	TU Darmstadt
Christoph Seidl	IT University of Copenhagen
Elmar Sinz	Universität Bamberg
Friedrich Steimann	FU Hagen
Stefan Strecker	FU Hagen
Peter Tabeling	gematik, Berlin
Bernhard Thalheim	Universität Kiel
Marina Tropmann-Frick	HAW Hamburg
Andreas Vogelsang	TU Berlin
Matthias Weske	Hasso-Plattner-Institut
Manuel Wimmer	Johannes Kepler Universität Linz
Robert Winter	Universität St. Gallen
Heinz Züllighoven	Universität Hamburg
Andreas Wortmann	Universität Stuttgart
Dominik Bork	TU Wien

Organisationsteam

Matthias Riebisch	Universität Hamburg
Marina Tropmann-Frick	HAW Hamburg
Stephanie Schulte Hemming	Universität Hamburg
Sabrina Göllner	HAW Hamburg
Tom Weber	Universität Hamburg
Arne Grünhagen	HAW Hamburg

Eingeladener Beitrag

Digital Twin

Model-Driven Engineering

Modelle, Semantik

Unternehmens- und Geschäftsprozess-Modellierung

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 11

Auf dem Wege zur Modellkunde

Bernhard Thalheim [1]

Abstract: Modelle werden genutzt, (1) um sich eine Vorstellung zu machen, (2) um Handlungen jeder Art systematisch zu ordnen und zu begleiten, (3) um Auffassungen, Vorstellungen und Wissen zu verinnerlichen bzw. zu entwickeln und (4) um das soziale und interaktive Leben zu begleiten. Modelle wurden zu allen Zeiten entwickelt und verwendet. Modelle sind kleine Zusammenstellungen, um in der großen und komplexen Welt zurechtzukommen. Sie sind auf den menschlichen Gebrauch ausgerichtet und auch dem menschlichen Auffassungsvermögen angepaßt. Sie werden in der Wissenschaft und im Ingenieurwesen ebenso intensiv genutzt wie im täglichen Leben. Alles kann zum Modell werden, d. h. das Modellsein kann allem zeitweise auf wechselnder Grundlage zugeordnet werden.

Die Modellkunde systematisiert, homogenisiert und verallgemeinert die Erkenntnisse, Herangehensweisen und Nutzungsweisen von Modellen und vom Modellieren zu einer Kunst der Modellentwicklung und -nutzung. Modelle unterstützen in expliziter oder auch impliziter Form beim Wahrnehmen, Beurteilen, Erwarten, Denken, Sozialisieren, Entscheiden und Handeln.

Keywords: Modellkunde; Modelle; Modellieren; Modellsein

1 Modellierung, wo stehen wir heute?

1.1 Eine Bestandsaufnahme

Modelle haben im letzten Jahrtausend bereits zu einer umfangreichen Forschungsliteratur mit weit über 4.000 Publikationen geführt z. B. [Mah21, Mül16, Pag18, TN15a]. Sie müssen nicht als ‚Modell' benannt sein. Es scheint, daß aufgrund der Vielfalt kein Modellbegriff für ‚Etwas' existieren kann[2]. Mit der breiten Benutzung der Bezeichnung ‚Modell' ohne Fixierung einer Definition wird ein Verständnis erschwert, wenn nicht unmöglich. Wir werden im weiteren illustrieren wie dieser Zustand überwunden wird. Der wesentliche Ansatz dabei ist, nicht das ‚Etwas' phänomenologisch zum Modell zu erklären, sondern das Modellsein direkt zu charakterisieren. Kann man beschreiben, warum ‚Etwas' als Modell genutzt wird, dann kann man auch für das ‚Etwas' Eigenschaften heraus destillieren, die das Modellsein des ‚Etwas' als Instrument charakterisieren.

[1] Christian-Albrechts-University Kiel, 24118 Kiel, Germany bernhard.thalheim@email.uni-kiel.de

[2] M.W. Wartofsky konstatiert in [War79]: *„[A]nything can be a model of anything ...". „[I]t is being taken as a model which makes an actual out of a potential model; and every case of being taken as a model involves a restriction with respect to relevant properties."*
B. Mahr hat in einem frühen Vortrag definiert [Mah05]: *„Ein Modell ist die Synthese einer begrifflichen Vorstellung, einer Ausdrucksform und einer Einnahme einer Rolle, durch die es eine Funktion erfüllt."*. Später hat er diesen Standpunkt aufgegeben aufgrund der Vielfalt von Modellen [Mah15b] als Erwiderung zu den Kommentaren zu [Mah15a]: *„Die Frage, was ein Modell ist, habe ich als falsch gestellt abgetan."*

doi:10.18420/modellierung2022-001

Die explizite Bezeichnung ‚Modell' wurde erst mit der Architekturtheorie im alten Griechenland eingeführt. Es kann angenommen werden[3], daß Modelle in den allen Kulturen, Wissenschaften, im täglichen Leben und im Engineering bereits schon eine größere Bedeutung hatten und haben, wobei viele Modelle heute als selbstverständlich akzeptiert und oft nicht bewußt als solche wahrgenommen oder so benannt werden.

1.2 Generationen der Nutzung von Modellen

Modelle sind schon immer betrachtet und benutzt worden. Die Informatik als Technik zur Schaffung des Künstlichen hat eine beginnende Geschichte und zugleich eine spezifische Ausrichtung der Modellierung, der wir fünf Generationen zuordnen können:

1. Modelle als das implizite und oft auch intuitive Verstehen des zu Entwickelnden bzw. des Entwickelten z. B. von Programmen, der Maschine und das maschinenorientierte Verständnis in Kochbuch-Form;
2. Modelle als explizite Vorher-Modelle, die als Anleihe, Inspiration und Blaupause das Verständnis der Kodierung unterstützen in der Models-4-Programming-Form (M4P), d. h. kommunikativ und Ausgangspunkt für das Programmierhandwerk;
3. Modelle als explizite Nachher-Modelle anhand der Vorher-Modelle zur Betrachtung von Systemen, dem Nachweis von Programm-Eigenschaften und dem Übertragen der Erfahrungen auf Folgeaktivitäten;
4. Modelle als Treiber für eine modellbasierte Entwicklung, anfänglich als model-driven architecture, universelle Anwendungen oder conceptual-model programming und im weiteren in der model-2-program-Initiative oder beim modellzentrierten Entwickeln, so daß Modellierung zur Kunst der Programmierung beiträgt; [MMR+17];
5. Modellierung im modelling-as-programming-Zugang oder später im models-as-a-program-Zugang [DPT21], bei denen Modelle transformiert und compiliert werden und der Zusatz zu lauffähigen Programmen bereits mit dem Modell (oder der Modell-Suite) mitgeliefert wird, so daß die Kodierung durch eine Modell-Technologie oder Modell-Kultur ersetzbar wird.

1.3 Drei Zugänge zum Modellbegriff

Ein Modellbegriff scheint schwierig zu sein und wird oft vermieden oder in spezifischer Form verwendet[4]. Es gibt nur wenige tragfähige Definitionsversuche für einen allgemeinen Modellbegriff, obwohl das ‚Modell' in allen Bereichen des Lebens, den Wissenschaften und den Anwendungen breit verwendet wird. Wir können die Vielzahl der existierenden

[3] Im KunstModell-Projekt https://kunstmodell.de/ haben über 10 spezifische Modellkategorien für das alte Äypten ausmachen können.

[4] Siehe z. B. [Mah21, Sta73, Sta92, Tha13, TN15a, Tho05].

Zugänge in der Benutzung von ‚Etwas' als ‚Modell' kategorisieren wie folgt, d. h. daß *Modellsein*:

Modelle als Konstruktion: Alles, was explizit konstruiert wird mit dem Ziel, Modell zu werden, ist ein Modell. Die Konstruktion bestimmt das Modellsein.

Modelle als Phänomen: Es werden allgemeine Eigenschaften von Modellen postuliert. Gegenstände oder auch mentale Vorstellungen sollen diese Eigenschaften in einer gewissen Form erfüllen, um Modell zu sein.

Allgemeine formale Modelldefinition: Der Begriff ‚Modell' kann mathematisch durch Eigenschaften, die zu erfüllen sind, eingeführt werden, so daß Modelle abgegrenzt werden können von Nicht- bzw. Unmodellen. Die Eigenschaften können verfeinernd an die jeweilige Definition angepaßt werden. Das Modellsein ist an die Gültigkeit dieser Eigenschaften gekoppelt. Diese Eigenschaften können schrittweise an spezielle Anwendungen angepaßt werden.

Das Verständnis von Modellen als konstruierte Objekte ist sehr alt und wird vielfach angewandt, wobei der *Konstruktionszugang* ein explizites Modellvorhaben voraussetzt. Dieser Zugang erfordert, sowohl die verfolgten Ziele als auch die Beziehungen zwischen Modell-Übermittler und allen potentiellen Rezipienten im Detail zu kennen sowie die konkrete Konstruktion des Modells für dieses Ziel.

Der oft verwendete *Phänomenologie*-Zugang betrachtet Gegenstände und Gedanken anhand des Auftretens von drei Phänomen: Abbildung, Verkürzung und Pragmatik. Dieser Zugang ist für einfache Modelle in relativ einfachen Zusammenhängen trotz seiner Unschärfe oft ausreichend. Oft wird ein deduktiver Zugang der Prädikatenlogik erster Stufe unterlegt, der jedoch nur einen kleinen Teil des modellbasierten Arbeitens unterstützt [Tha21, Tha22]. Setzt man auf subduktives Arbeiten [Rhe21], dann muß man auch Induktion, Abduktion und andere Formen des Schließens integrieren.

Die dritte Auffassung zum Modellsein orientiert sich an der *Nutzung* von Modellen, d. h. Modelle sind benutzbare, glaubwürdige, hilfreiche *Instrumente*, die einen Mehrwert durch ihre Benutzung versprechen. Inwieweit eine Nutzbarkeit, Glaubwürdigkeit, Nützlichkeit oder Wert vorliegen, unterliegt einem Urteil zum Modellsein. Das Modellsein fußt auf dem Instrumentsein, d. h. auf der Orientierung auf ein zu erreichendes Resultat. Diese Auffassung liegt der *Modellkunde* zugrunde.

1.4 Ausblick auf die Modellkunde

Die Modellkunde kann man entwickeln zu einer übergreifenden Handwerkskunst und -lehre für alle Bereiche des Lebens, der Wissenschaften und der Technologie. Die Modellkunde berücksichtigt die Heterogenität der vielen unterschiedlichen Szenarien, Funktionen, Formen der Nutzung, Befähigungen der Benutzer und Grundlagen durch eine Systematik der Modellkategorien.

Es sind damit u. a. Fragen zu beantworten wie: Wann wird ‚Etwas' zum Modell? Welche Eigenschaften zeichnen ein Modell aus? Welche Qualität wird von einem Modell erwartet? Inwieweit kann man einem Modell vertrauen? Welche Eigenschaften schließen das Modellsein aus? Inwieweit ist ein Modell geeignet bzw. angemessen im gegebenen Anwendungsfall und wann nicht? Welches Potential und welche Leistung ist von einem Modell zu erwarten? Wie wirkt ein Modell durch seine Existenz auf seine Anwendungslandschaft und damit auf sich selbst zurück? Etc.

Die Modellkunde erlaubt eine systematische und begründete Diskussion über Modelle und ihren Eigenschaften, über das Modellieren und über die Modellierung. Aufgrund der Omnipräsenz von Modellen in allen Sphären menschlicher Tätigkeit ist eine Modellkunde allgemein wie eine Wissenschaft und anwendungsorientiert wie eine Technologie und zugleich auf das menschliche Tun ausgerichtet. Zugleich stellt die *Vereinfachungsforderung* für Modelle gegenüber dem Kontext und System- u.a. Welten eine Herausforderung dar.

1.5 Zusätzliche Aufgaben der Informatik-Modellkunde

Da Modelle in der Informatik auch zur Produktentwicklung genutzt werden, müssen Modelle zusätzlich hinterfragt werden: Wie ist die Semantik in allen Facetten erfaßbar? Welche Qualitätskriterien für Modelle – wie z. B. Korrektheit, Berechenbarkeit, Validierbarkeit, Verifizierbarkeit, Vollständigkeit – können auf welche Art gewährleistet werden? Unterstützt die Modellierung auch die Ausprägung und Erstellung eines Systems in allen wichtigen Aspekten? Wie kann eine Modellgesellschaft sowohl technische Aspekte als auch das menschliche Interagieren reflektieren? Wie beherrscht man die Zusammenhänge innerhalb eines Ensembles von Modellen, z. B. für Systemmodelle und deren Reflektion durch deklarative oder informative Modelle? Wie beherrscht man die Fortschreibung, Modifikation, Integration und Evolution von Modellen bei Weiterentwicklung von Systemen? Kann man Modellierung unabhängig von unterlegten Paradigmen der Informatik gestalten? Wie sind Modelle für eingebettete, kollaborierende oder gesteuerte Systeme zu gestalten? Wie kann man Modelle auch für eine Wiederverwendung für andere Szenarien aufbereiten?

Eine Besonderheit der Informatik ist die *Verfeinerungsforderung* für die meisten Modellarten, d.h. aus einem Modell kann man auch ein präzises, hochqualitatives, genaues sowie ggf. formales Modell durch eine Verfeinerung des Ausgangsmodelles erstellen, so daß ein System auch nachgewiesenermaßen den erforderlichen Qualitätskriterien genügt. Die Verfeinerungstheorie [Bör03] ist bislang der erste Schritt dazu.

1.6 Die Kieler Modelldefinition

Nach einer Bottom-Up-Analyse in den Wissenschaften und im Ingenieurwesen haben wir für eine allgemeine Definition des Modellseins in [TN15a] einen spezifischen Zugang zur Modellierung gefunden, der auch den Kern der Modellkunde darstellt. Postulate und Prinzipien dieses Zuganges sind die folgenden:

1. Das Modellsein ist eine ***Zueignung*** und basiert auf einem Urteil. Eine Definition des Modellbegriffes setzt auf dem Modellsein auf.
2. Das Modell kann in einer generischen, konfigurierbaren und spezialisierbaren Form [Bie08, TF16] aufgrund das Funktionieren des Modells als Instrument in einem Szenarium oder einem Spektrum von Szenarien dank der Angemessenheit (Adäquatheit und Verläßlichkeit) begriffen werden.
3. Das Modellsein kann *schrittweise und geschichtet* charakterisiert werden durch
   a) die Betrachtung von ‚*Etwas*1' (Origins oder auch eine Konstellation), wobei
   b) vorher das *wozu–warum–Szenario* geklärt ist und
   c) das ‚*Etwas*2' als Instrument genutzt wird, das
   d) mit Intention, Akzeptanz und *Urteil* über das **Modellsein**
   e) durch den Rezipienten *genutzt* werden kann.
4. ‚Etwas2' ist nicht per-se Modell.

Wir nutzen nun den folgenden generischen Modellbegriff für die Modellkunde, wobei die einzelnen Eigenschaften parametrisch gegeben sind und durch Spezialisierung zur Definition der spezifischen Formen von Modellen genutzt werden können:
Ein Modell ist ein wohlgeformtes, angemessenes und verlässliches Instrument, das ‚Etwas'[5] repräsentiert und in Nutzungsszenarien funktioniert.
Dieser Modellbegriff erlaubt eine weiter unen im Detail ausgeführte Anpassung an die vorgesehene Benutzung durch eine Schärfung der Parameter. Alles kann als Instrument in einem Nutzungsszenario genutzt werden, d. h. zum Instrument werden (Instrumentsein). Instrumente werden zur Erreichung bestimmter Ziele in einem Nutzungsszenario genutzt, repräsentieren Origins und werden dadurch zum Modell (Modellsein). Sie funktionieren als Modell in diesem Szenario in einer speziellen Rolle.

1.7 Unsere Agenda zur Einführung in die Modellkunde

Die Modellkunde wird einmal zum eigenstandigen Zweig in der Informatik. Noch ist es keine Modelllehre mit ausgefeilter Didaktik oder eine Modellwissenschaft. In diesem Beitrag besinnen wir uns zuerst auf einige wenige Ideen zur Modellkunde. Im Abschnitt 3 stellen einen allgemeinen Zugang zur Entwicklung von Modellen vor, der erlaubt aus dem allgemeinen Modellbegriff einen auf die Anwendung zugeschnittenen Begriff abzuleiten. Wir beschränken uns hier auf das Modellsein und klammern mit dem Verweis auf [Tha22] die Nutzung von Modellen als Instrument aus. Wir schließen mit der Nennung einiger Forschungsthemen ab.

[5] Origins als Quelle, Ursprung, Ausgangspunkt.

2 Ein kurze Geschichte der Entdeckungen in der Modellkunde

Die Entwicklung der Informatik ist ebenso wie die Entwicklung der Rechenkunst stets von Modellen begleitet worden. Modelle spielten von Anfang an eine wichtige Rolle, unabhängig davon ob man die Geschichte der Informatik beginnt mit der Geschichte der Rechenautomaten vor C. Babbage (z.B. mit dem Modell einer analog-digitalen Rechenmaschine von Leibniz[6]) oder mit der Algorithmik des Mittelalters oder mit den Rechenkünstlern viel früherer Zeiten oder ob man die Geschichte der Informatik beginnt mit den logischen Theorien und Modellen des 20. Jahrhunderts. Wir beschränken uns im Weiteren auf die wesentlichen Schritte hin zur Modellkunde.

2.1 Modelle als Konstruktion

Modelle sind schon sehr früh als physische Modelle zur Darstellung des Funktionierens, einer Komposition, einer generellen Architektur, von theoriebasierten Einsichten oder zur Erklärung von Mechanismen und Konstruktionen eingesetzt worden. Sie veranschaulichen Wissen, Methoden und Techniken. Sie sind auch veranschaulichte und gestaltete Wirklichkeit und unterstützen das visuelle Denken, Verstehen, Erkunden, etc. Sie werden auch genutzt zum Fertigen z. B. mittels Schablonen. Physische Modelle sind Teil der visuellen Kultur in vielen Facetten [CH04]. Sie wirken durch ihre Formen anstatt einer textuellen Beschreibung. Sie stellen eine intentionelle Aktivitatät dar, kombiniert mit Rhetorik-Taktiken, visueller Kommunikation und verstecktem Wissen und Können.

In der Informatik sind 2D/3D-Artefakte oft auch Modelle, die erstellt wurden anhand von Vorstellungen und Erkenntnissen für das entsprechende Einsatzspektrum je nach Kontext, Rezipient und Funktion in Konstruktionsprozessen. Offen ist, inwieweit ein Prototyp oder ein Muster ein Modell eines zu schaffenden oder danach geschaffenen Systems ist.

Die Betrachtung von Modellen als Konstruktion führt direkt zum Kriterium der *Wohlgeformtheit für die Rezipienten* von Modellen. Wir integrieren in die Modellkunde auch die *Prinzipien der visuellen Kommunikation* [Mor20] und das Vorhandensein von *Hintergrundwissen* und von *impliziten* Modellen. Physische Modelle bestehen auch aus einer wohlintegrierten Gesellschaft von Modellen, d. h. aus einer *Modell-Suite*.

2.2 Modelle als Phänomenen

H. Stachowiak [Sta73, Sta92] hat eine logikbasierte Auffassung zum Modellsein auf einer speziellen Form der Adäquatheit anhand von drei Kriterien postuliert: (i) die *Abbildungseigenschaft* von einem Original bzw. Origin auf das Modell; (ii) die *Abstraktionseigenschaft* (auch Verkürzung) vom Original auf dessen essentielle, relevante und bestimmende Eigenschaften; (iii) die *Zweckbestimmung* als Pragmatismus-Eigenschaft eine Unterwerfung des

[6] Der Nachbau [Leh99] und der Nachweis der potentiellen Funktionsfähigkeit basiert auf dem entwickelten aber nicht umgesetzten Modell von Leibniz.

Modells unter einen Zweck und unter seine Verwendung für den jeweiligen Nutzer, seine Ziele, seiner angewandten Techniken und Werkzeuge, seines Zeitraums usw.

Weiterhin stellt er dar: „Ein Modell ist seinem Wesen nach eine in Maßstab, Detailliertheit und/oder Funktionalität verkürzte bzw. abstrahierende Darstellung des originalen Systems.“[Sta73] Das Modellsein kann auch mit einem Quintupel charaktersiert werden: „X ist Modell des Originals Y für den Verwender k in der Zeitspanne t bezüglich der Intention Z.“[Sta92]

Es werden drei Eigenschaften als Phänomene dargestellt, die z.T. zu speziell bzw. zu einschränkend sind. Sie können zu Eigenschaften von Instrumenten ausgebaut werden, die Modelle zumindest erfüllen müssen.

2.3 Die Darstellung des „wofür“ bzw. „wozu“ und „für wen“

Modelle sollen nicht einfach an sich als solche existieren, sondern werden in einer Anwendung in entsprechenden Anwendungsszenarien benutzt, um eine Aufgabe zu bewältigen. Dazu unterscheidet W. Steinmüller Modelle im engeren bzw. weiteren Sinne:
Ein Modell im engeren Sinne ist ein Bild eines Originals, „anhand dessen irgend welche Verhältnisse des Originals untersucht werden können.“ „Ein Modell im weiteren Sinne ist ein Modell (im engeren Sinne) von einem Original für einen Zweck für ein Subjekt ist.“ Kurz „ein Bild von Etwas für einen Zweck von Jemand. Noch kürzer: ‚Modell‘ ist stets ‚Modell-wovon-wozu-für wen‘“. [Ste93]

Modelle haben demzufolge ein Janus-Kopf-Verhalten [Hes06] mit dem Blick auf das Origin und auf das zu erreichende Resultat.

2.4 Das „Urteil“ zum Modellsein

R. Kaschek [Kas03] nutzt die Urteilstheorie A. Pfänders für eine Begründung des Modellseins, insbesondere für konzeptuelle Modelle als soziale Konstrukte. Dies erweitert die Sichtweise von P.C. Lockemann und H.C. Mayr [LM78], nach der ein Modell eine Vorstellung ist, die sich ein Individuum von einem System in seiner Umwelt bildet. Modelle sind nicht per se Modelle, sondern werden erst zu solchen durch entsprechende Bezugnahme in einer CoP. Durch Integration der Theorie des Urteils A. Pfänders [Pfä21] kommt R. Kaschek zur expliziten Berücksichtigung des Modellseins:
„Der Begriff Modell wurde in der vorliegenden Arbeit als Spezialfall des Begriffs Vorstellung erklärt. Vorstellungen wurden als Mengen von aufeinander bezogenen Urteilen charakterisiert.“ [Kas03]

2.5 Das „Origin“ und das „Cargo“

B. Mahr führt die oben genannten Ansätze mit der Modelltheorie Tarskis der Mathematischen Logik fort. Der Ausgangspunkt für die Modellierung ist das Origin als Quelle, Original,

Vorbild, Prototyp oder Bezugssystem, mitunter bei Existenz einer Abbildung auch Urbild. Im Weiteren gehen wir nicht nur von einem Origin aus, sondern von einem Kompositum mit primären und sekundären Elementen.

B. Mahr führt aus, daß „das Modellobjekt durch seine Herstellung oder Wahl mit einem *Cargo aufgeladen* ist, dessen *Herkunft* die Matrix und dessen *Bestimmung* das Applikat ist..." [Mah15a], wobei die Matrix die wesentlichen und relevanten Eigenschaften des Origins darstellt. In [TN15a] umfaßt das Cargo die Mission, die Bedeutung, die Bestimmung und die Identität eines Modells. Modelle transportieren nicht nur das Cargo zum Resultat auf Grundlage des Origins, sondern unterliegen diesem, übernehmen es als Hintergrund, reichern es an und übermitteln auch diesen Teil an das Ergebnis der Modellnutzung. Das Cargo wird damit integriert in ein das Modell begleitendes *informatives Modell*.

2.6 Die Verläßlichkeit: Begründung und ausreichende Qualität

Modellierung in der Informatik beginnt gewöhnlich mit der Übernahme der Ansätze und Erfahrungen in der speziellen Anwendungsart, so daß der Hintergrund und alle Annahmen wie Postulate und Paradigmen unhinterfragt übernommen werden. Vergleicht man innerhalb der Informatik die Herangehensweisen, dann stellt man große Unterschiede fest, so daß mehr als 60 verschiedene Definitionen [Tha18] für ein konzeptuelles Modell nicht verwundern. Analoge Beobachtungen kann man für viele Disziplinen machen. Meist sind die Begründungen für das Modellsein und die Qualitätsanforderungen sowie das Verständnis vom Einsatz von Modellen implizit. Deshalb wurde in [TN15a] die Verläßlichkeit als der zweite bestimmende Teil neben der Adäquatheit eingeführt.

2.7 Die Entdeckung der Nützlichkeit: Modelle als Instrument

Oft wird für Modelle die Allgemeingültigkeit gefordert, obwohl dies nicht notwendig ist. Modelle sollen einem Ziel folgen und anwendbar sein. Sie werden genutzt, d.h. akzeptiert aufgrund ihres potentiellen Nutzens, wobei der Nutzen sich an vielen Funktionen von Modellen in Anwendungsszenarien orientiert. Typische Funktionen orientieren sich z. B. am Verstehen, Kommunizieren, Kooperieren, Dokumentieren, Illustrieren, Erklären, Lernen, Herstellen, Gestalten, Anleiten, Planen, Problemlösen, Erstellen, Herstellen und Testen [Mah15a, TN15a]. In allen Fällen sollen Modelle nützlich sein. Der Nutzen orientiert sich damit an den Szenarien, in denen das Modell eine Funktion hat.

2.8 Gibt es Definition für das „Modell“?

Mit der Modellkunde nutzen wir einen generischen Begriff des Modells. Wir nutzen dazu einen generischen Modellbegriff, der je nach Anwendung, je nach Zweig der Informatik, je nach Modellkategorie schrittweise konfiguriert werden kann, so daß auch *spezifische Modellbegriffe* für eine spezifische Anwendungsart abgeleitet werden können. Dieser Zugang

stellt eine Abkehr vom phänomenologischen Modellbegriff dar. Das volle Begriffsgerüst zum Modellbegriff [TN15a] in Abbildung 1 umfaßt den in Abschnitt 1.6 präsentierten Kieler Modellbegriff in allen seinen Eigenarten.

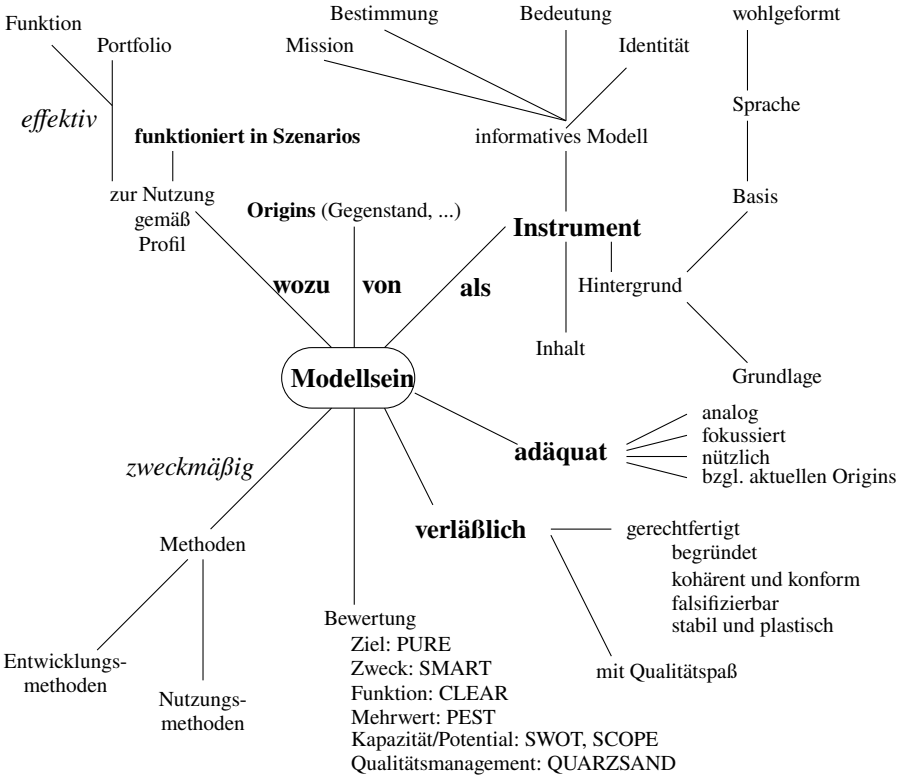


Abb. 1: Das Begriffsgerüst des Modellsein-Begriffes (sprachbasierte Modelle) (modifiziert aus [TN15b])

2.9 Die Modellreise durch Szenarien

Alles kann zum Modell werden, zumindest für eine gewisse Lebensspanne und Gültigkeitsspanne in einem Kontext für Szenarien. Das Modellsein ist dynamisch. Modelle entstehen, werden genutzt und vergehen. Sie kommen ggf. wieder in anderen Szenarien und wirken in einer anderen Funktion. Die Reise des Modells durch Szenarien in Abhängigkeit vom Betrachter kann man formalisieren nach V.J. Propp [Pro72] in Analogie zur Reise des Helden in Erzählungen und in Analogie zur Gestaltung von Websites mit einem Storyboarding [ST19]. Die Modellreise schließt die Entwicklung, die Evolution, die Modifikation und die variable Nutzung des Modells mit ein, sowie auch die Integration in andere Modelle und das Spiel mit der Abstraktion als multiple Verallgemeinerung und als Spezialisierung je

nach Gesichtspunkt (siehe z.B. [Fra22]). Die Modellreise kann auch in Modell-Familien bzw. Modell-Suiten [Pag18, Tha10] stattfinden. Eine Zueignung zum Modell schließt auch eine Rückwirkung auf die gesamte Modellumgebung und damit auch auf das Modellsein selbst im Sinne der Kybernetik zweiter Ordnung mit ein [Ump01].

2.10 Die Modellwerkstatt

Modelle sollen eine Fokussierung auf das Wesen und die Landschaft unterstützen und auch für die Nutzung einen Beistand und entsprechende Hilfsmittel besitzen, sowie auf eine gesicherte Analogie zurückgreifen können. Deshalb wird implizit auch eine Werkstatt vorausgesetzt, die zum einen die Stützung als „way of thinking“ und zum anderen die Ausstattung als „way of making“ (original „way of supporting“ in [DSS03]) einbringt. Die *Stützung* trägt auch zur Verläßlichkeit von Modellen bei. Sie wirkt als Rückhalt, Basis, Träger, Begründung und Bestücker für das Modell selbst und auch für die Nutzung des Modells. Die *Ausstattung* ermöglicht die Formung des Instruments, die Nutzung und damit das Funktionieren des Modells in Szenarien. Sie stellt das erforderliche Instrumentarium und die Mittel zur Entwicklung und Nutzung des Modells bereit. Modelle sind oft sprachbasiert, so daß die Sprachwissenschaft mit Erweiterungen aus der Biosemiotik [Bar07] oder aus kognitiven Wissenschaften breit verwendet wird. Zur Ausstattung kann auch ein formaler Apparat z.B. durch Methodiken der Mathematik heran gezogen werden.

Die Existenz der Modellwerkstatt wird im Triptychon-Zugang zur konzeptuellen Modellierung [MT21] aufgrund der Besonderheiten der konzeptuellen Modellierung zu den beiden Dimensionen *Enzyklopädie-Dimension* und *Sprach-Dimension*. Typische Ausstatter für die sprachbasierte Modellierung sind *semiotische Hilfsmittel* [HM08].

3 Konfiguration und Kanonisierung von Modellen

Jede Disziplin hat ihre Zugänge entwickelt, ihre Pfade ausprobiert, sich auf erfolgreiche Zugänge orientiert, die anderen Zugänge verdrängt und z.T. vergessen, die Ausbildung ausschließlich auf solche Zugänge reduziert und sich auch in der angewandten Forschung nur auf den Erfolgskanon orientiert. Die Kultur im Sinne von G. Hofstede [HHM10] als kollektive erzieherische Ausprägung einer dominierenden Auffassung sollte aber trotzdem alternative Zugänge bewahren und wieder aufnehmen. Auch für die Modellkunde gibt es solche disziplinären Pfade.

3.1 Eine Herangehensweise an die Modellkunde

Im Weiteren konzentrieren wir uns auf eine Herangehensweise der Modellkund, die sich in vielen Projekten auch außerhalb der Informatik bewährt hat z.B. in der Medizin, der Archäologie und der Ägyptologie. Sie ist jedoch eine von vielen. Sie ist jedoch gut begründet in der tiefen Bottom-Up-Analyse des Wirkens mit Modellen in mehr als 20 Disziplinen [TN15a].

1. Der *generische Modellbegriff* ist ein parametrisiertes System, dessen Parameter instantiierbar je nach Profil/Portfolio sind. Generische Zugänge erlauben im Gegensatz zu einem Top-Down-Zugang mit einem allgemeinsten Begriff des Modells eine schrittweise Verfeinerung.

2. Modelle sollen nützlich und hinreichend einfach nutzbar sein. Deshalb steht eine Klärung der *Einsatzziele/-zwecke/-funktionen* für den Einsatz und damit auch die Entwicklung des Instruments Modells vor einer Entwicklung. Wir klären zuerst das *Wesen* eines Modell und schärfen das Verständnis für die *Landschaft* innerhalb von intendierten Anwendungsfällen, deren Szenarien, der community of practice (CoP) und des Kontextes. Darauf können die üblichen Schritte des requirements engineering aufgesetzt werden und die Strategie und Taktik enger gefaßt werden. Mit dieser Meta-Fokussierung des Modells als Instrument erhalten wird eine *Konfiguration*, können auch daraus eine Architektur oder *Anatomie* ableiten und mit einer Beschreibungsform als *Signatur* erfassen.

3. Mit der expliziten Konfiguration für das einzelne Modell kann eine *Kategorisierung* und damit innerhalb der jeweiligen Kategorien eine *Kanonisierung* des Modellseins anhand des Wesens (Natur), der Landschaft, der Strategie und der Taktik für eine systematische erfahrungsintegrierende Entwicklung und Nutzung von Modellen als Instrument vorgenommen werden. Nutzt man eine Unterscheidung nach Szenarien und der primären im Modell erfaßten Aspekte, dann unterteilen sich zuerst die Kategorien in solche (1) zur Reflexion insbesondere von Vorstellungen, (2) zur systematischen Begleitung und zum Ordnen von Handlungen, (3) zum Denken und zur Wissensbehandlung und (4) zur sozialen und kollaborativen Daseinsbewältigung. Diese Kategorien können nach der Ausrichtung innerhalb der CoP weiter unterteilt werden. Mit einer solchen Herangehensweise kann auch eine Kanonisierung der Adäquatheit, der Verläßlichkeit, der Wohlgeformtheit, der Invarianz, des Mehrwertes und der Qualität vorgenommen werden.

4. Diese Herangehensweise erlaubt nun auch Kanonisierung der *Modellsein-Eigenschaften* für ‚Etwas' anhand potentieller Origins und ggf. auch ausgewählter oder aktueller Origins. Wie können damit Modelle von Nicht-Modellen und sogar Un-Modellen unterscheiden. Es entsteht damit ein *spezifischer Modellsein-Begriff* nach der Konfigurierung und der Kanonisierung.

5. Die Kategorisierung und Kanonisierung integriert anhand der Handlungen mit Modell Erfahrungen mit dem Modell. Wir können daraus – ebenso wie in anderen wissenschaftlichen und technischen Disziplinen – *Stereotype*, Muster und Templates für die Erstellung und Benutzung von Modellen ableiten. Diese Stereotype erleichtern nicht nur das Modellieren in analogen Anwendungen sondern erlaubt auch eine Ableitung einer Methodik oder von Methodologien für Anwendungen.

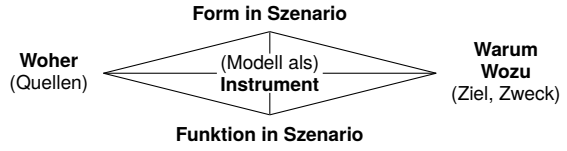
6. Diese schrittweise Verfeinerung ist eine *Ausprägung aller Parameter* des Modellbegriffes, so daß damit sowohl eine Adäquatheit und Verläßlichkeit als auch die

Verwendbarkeit als Modell explizit gegeben ist. Zusätzlich kann ein *Modell des Modelles* abgeleitet werden, das als informatives Modell in Form eines *Modellhauses* analog zur einer Gebrauchsanleitung zum schnellen Verstehen, zu einer Architektur des Modellseins, zum didaktischen Darstellen und auch zur Dokumentation des Instrumentes, seiner Benutzbarkeit und seines Spezifikums benutzt werden kann,

7. Wie in vielen Anwendungen entstehen damit spezielle *Nutzungs,spiel'/-formen* des Modelles für die jeweilige CoP, für die Verfeinerung der Szenarien, für Anleitungen, für Regeln, für Stereotypen etc. Diese Nutzungsformen können für die jeweiligen Origins angepaßt und unproblematisch übernommen werden.

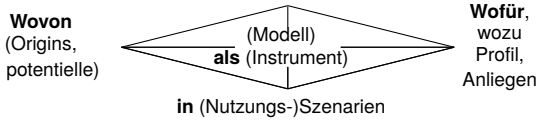
3.2 Die schrittweise Konfiguration von Modellen

Es scheint mit dem weiten Spektrum, das die Modellkunde umfaßt, unmöglich, eine Systematik zu Modellen zu finden. Wir können uns des Tricks bedienen, von der allgemeinen Charakterisierung und der Landschaft, in die das Modell eingebettet ist, zu strategischen und taktischen Entscheidungen, zu der operationalen Umsetzung und letztendlich zur Anwendung von Modellen selbst zu kommen. Diese Schichtung folgt einer Basiert-auf-Beziehung.



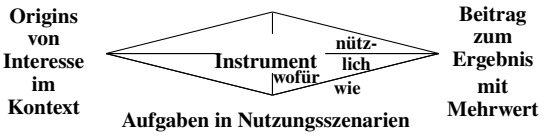
Wir wollen nun zeigen, wie eine Konfiguration und damit eine Kanonisierung des Modellseins erhalten werden kann. Wir nutzen, modifizieren und verallgemeinern dazu die links-rechts-gerichtete Rhombus-Darstellung von B. Mahr [Mah21] zur Visualisierung der unterschiedlichen Elemente des *Modellseins*. Ein Modell reflektiert Origins und Kontext, wird in einer bestimmten Form für Funktionen genutzt und funktioniert als Instrument entsprechend Zielen und Zwecken.

Das Wesen: Mit dem Instrumentsein von Ideen und Objekten ist auch die *Natur* bzw. das *Wesen* des Modells anhand des Anliegens bzw. der zentralen Fragestellung allgemein erklärbar.



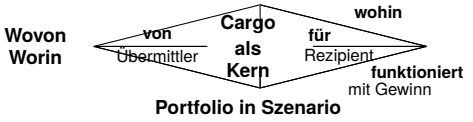
Ein Modell ist eine Reflexion oder Repräsentation von Origins, die eine Reihe von gemeinsamen Eigenschaften als *Invarianten* haben, die auch im Modell vorhanden sein sollen und auch voll an das Resultat weitergegeben werden sollen. Damit folgt das Modellsein einem **Wovon-Als-Wofür-In**-Muster, d. h. wir nutzen Modell **wovon** Origins **als** Instrument **wofür** das Erreichen eines Resultats im Rahmen des Profils **in** bestimmten Nutzungsszenarien. Das Wesen wird am allgemeinen Profil von Modellen orientiert, d. h. dem Ziel, Zweck und der Funktion des Modells in einer Anwendung.

Die Landschaft: Die Mission beschreibt den intendierten Gebrauch von ‚Etwas' als Modell in einem Kontext und einer Situation.



Mit der Anwendungsorientierung von Modellen, dem Modellieren und der Modellierung kommt die zweite Perspektive zum Tragen, die wir als ‚Landschaft' des Modells bezeichnen: die Anwendung und Nutzung von Modellen in einem (oder auch mehreren) Anwendungsszenario, in dem das Modell entsprechend seinem Profil funktioniert und den Invarianten genügt. Im Anwendungsszenario stehen Aufgaben, die gelöst werden sollen mit dem Modell. Das Modell wird genutzt, um eine Lösung zu gewinnen. Damit hat das Modell auch einen Mehrwert durch seine Nutzung.

Die Landschaft: Die Marke beschreibt die Akzeptanz des Modellsein. Das Instrument transportiert das Cargo als *Kern* des Modells auf seine Weise (inwiefern) aus der Origin-Welt (wovon, worin) zm Ergebnis und dies gewinnbringend in angemessener (adäquat, verläßlich) Form durch den Modellierer für den Rezipienten in der Modellreise (wohon, worin).

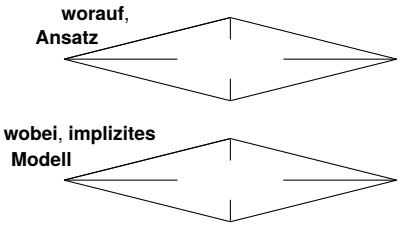


Zweites Element der Anwendungslandschaft ist die Marke, nach der das Modell konstituiert wurde. Mit der Marke wird auch der Kontext – z. B. der disziplinäre – und insbesondere die Nutzergemeinschaft determiniert. Ein Übermittler (wer) – bzw. im Idealfall der Modellierer – ‚übergibt' das Modell (was) in diesem Kontext (worin) einem Rezipienten (wem) für eine bestimmte Aktivität (Aktion) in den Grenzen der Invarianten. Die Marke ist demzufolge charakterisiert durch ein Muster der Form
von$^{\text{Origins, Kontext}}$ 2 für$^{\text{funktioniert gemäß Portfolio}}$.

Die Strategie: Die Matrix konditioniert das Modellsein und damit das Instrument als Modell in zwei Ausprägungen, zum einem durch einen Ansatz bzw. eine Methodik und zum anderen durch Übernahme von bereits erprobten Modellen. Die *disziplinäre Matrix* verallgemeinert die Ansätze von T. Kuhn und L. Fleck [Fle11, Kuh70][7]: (1) strategische Bestandteile, (2) Überzeugungen und Werte innerhalb der CoP, (3) Erfahrungen anhand von Beispielen, (4) eine Leitfrage als Hauptanliegen und (5) Methodiken bzw. Ansätze. Methodiken schließen auch Schablonen, Direktiven, Leitlinien und Methodologien mit ein. Eine Matrix ist etwas (z. B. eine Umhüllung) *„in oder aus dem etwas anderes entsteht, sich entwickelt, oder entnommen wird."* [BBW15] Eine Modellgemeinschaft teilt eine ‚Philosophie' und auch Techniken der Modellentwicklung und -nutzung. Wir können die

[7] Die ersten drei Teile der Matrix wurden von L. Fleck eingeführt und von T. Kuhn übernommen.

Matrix als eine strategische Grundlage für das Modell verstehen.

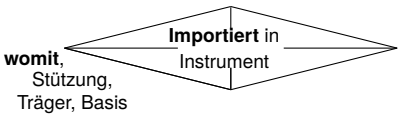


Modelle basieren meist auf früheren Erfahrungen, Erkenntnissen und Erfolgen. Sie sind eingebettet in diesen impliziten Hintergrund, der auch typisch für andere Modelle sein kann. Die Matrix besteht zum ersten aus einem Ansatz und zum zweiten aus impliziten Modellen.

Innerhalb des durch das Wesen und die Landschaft gegebenen Rahmens stellt der *Ansatz* ein Muster oder eine *Schablone des Vorgehens* sowohl für die Entwicklung und als auch für die Nutzung und Verwendung von Modellen bereit, der an den konkreten Anwendungsfall noch anpaßbar ist. Mit dem Ansatz wird auch eine Konfiguration der Adäquatheit und der Verläßlichkeit für Modelle übernommen, so daß sowohl das Spiel von Analogie und Fokus als auch für die Verläßlichkeit die Begründung und die Ausprägung der Qualitätscharakteristiken geprägt ist.

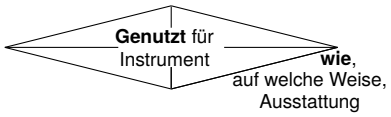
Unterlegte implizite Modelle sind zum einem intrinsische, immanente, innere und eingeschlossene Modelle, die als unverrückbarer Hintergrund einem Modell unterlegt werden, und zum anderen in der Denkfamilie gut bekannte Spendermodelle für ein effektives und vereinfachendes Benutzen des Gesamtmodells. Oft sind implizite Modelle bereits mit einem generischen Modell gegeben.

Die Taktik: Die Stützung von ‚Etwas' als Modell erfolgt durch einen Import bzw. eine modifizierte Übernahme von Spendern, Trägern, Stützpunkten, Weltwissen, allgemeinen persönlichen Auffassungen und Vorstellungen, verwendeten Grundlagen, etc.



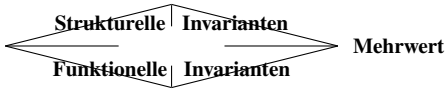
Modelle besitzen eine Umgebung zur Unterstützung und zur Befähigung, ähnlich zu einer Werkstatt. Sie benötigen einen Träger, eine Unterlegung bzw. eine Unterstützung und eine Fundierung. Dabei integrieren sie sowohl Spendermodelle als auch das Wissen aus dem Anwendungsgebiet wie z. B. Begriffswelten. Der Modell-Denkraum wird für Modelle als taktische Umgebung in einer Fokussierung vorausgesetzt.

Die Taktik: Die Ausstattung eines ‚Etwas', das als Instrument bzw. Modell genutzt wird, umfaßt alle genutzten Hilfsmittel, die man mit dem Bild eines Instrumentariums einer Werkstatt beschreiben kann.



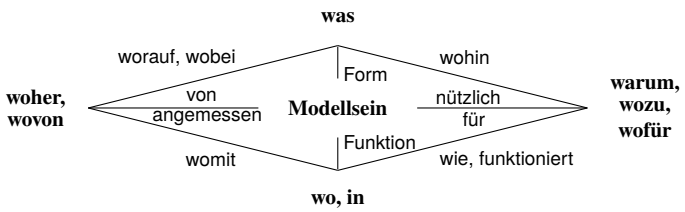
Modelle besitzen eine Umgebung zur Unterstützung, Ausstaffierung und zur Befähigung, ähnlich dem Instrumentarium einer Werkstatt. Das Modellsein wird ermöglicht durch Hilfsmittel zu ihrer Formung und Nutzung und bereitgestellte Methoden. Sie werden ausgestattet mit Techniken als auch einer Methodologie aus dem Anwendungsgebiet. Ein oft verwendetes Hilfsmittel sind an den Anwendungsbereich angepaßte Sprachen, z. B. zur Kommunikation und für Dialoge. Andere Werkzeuge unterstützen Handlungen mit Modellen. Die Werkzeug-Ausstattung des Modell-Denkraumes wird für Modelle als taktische Umgebung vorausgesetzt.

Die Invarianz überspannt ebenso wie der Mehrwert alle Teile des Arbeitsblattes. Sowohl strukturelle als auch funktionelle Invarianten werden von den potentiellen Origins über das Modell zum Resultat der Modellanwendung übertragen.



Wesentliche Aspekte der erfaßten Welt, insbesondere der Origins, sollen sich im Modell und auch im Resultat der Anwendung wiederfinden. Diese Aspekte sind gleichartig für alle potentiellen Origins und bilden einen stabilen Kern für die Betrachtung. Sie werden als Invariante zusamenngefaßt. Wir unterscheiden hierbei primäre, sekundäre und optionale Invarianten. Es wird hierbei auch eine Kompromißbetrachtung erforderlich, um Modelle nicht zu überladen. Invarianten sind sowohl in die Landschaft, die Strategie und die Taktik eingebettet zum einem durch Angabe, zum anderen durch eine Einbettung und des weiteren durch eine Erzwingung.

Die Konfiguration des Modellseins faßt das Wesen, die Landschaft, die Strategie und die Taktik als eine Anpassung des Modellseins an die Modellsituation zusammen. Sie wird oft als gegeben vorausgesetzt.



Die Konfiguration stellt ‚Rohfassung' dar, die genutzt werden kann, um sich auf die speziellen Origins anhand der gegebenen Funktion in einem Anwendungsszenario zu konzentrieren, d. h. ein Modell zu konfigurieren sowohl einerseits für den Übermittler als auch andererseits für den Rezipienten. Damit wird anhand dieser Konfiguration ein Modell vorgefertigt.
Diese Konfiguration mit einem ‚Rohling' als Werkstück wird genutzt, um systematisch eine

Modellkomposition vorzunehmen und einen Aufbau und eine Struktur sowohl deklarativ als auch funktional zu entwickeln, d. h. eine *Anatomie* eines Modells, z. B. als Modell-Suite zu konzipieren.

Die Konfiguration führt direkt auf einen abgeleiteten *disziplinären Modellbegriff* [Tha22]: „*Gegeben sei eine Konfiguration für die Modellierung. Ein* disziplinäres Modell *ist ein Modell mit vorkonfigurierter Adäquatheit und Verlässlichkeit, das gemäß den Anforderungen der Konfiguration in vorgesehenen Szenarien funktioniert.*“

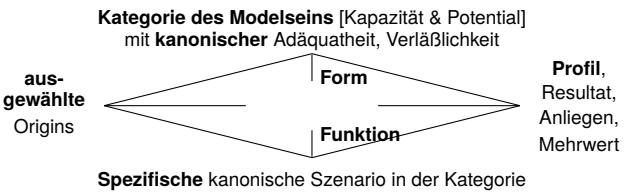
Der disziplinäre Modellbegriff richtet sich an den *Kernpunkten* (für Modelle am Wesen und der Landschaft), an den *üblichen* Ansätzen und *Herangehensweisen* sowie den impliziten *Erfahrungen* (für Modelle die impliziten Modelle) und an den *vorhandenen Mitteln* (für Modelle an der Stützung und Ausstattung) der Disziplinen aus.

3.3 Die Kategorisierung von Modellen

Die Vielfalt der Verwendung von Modellen scheint eine Modellkunde zu verhindern. Der präsentierte Zugang zur Konfiguration erlaubt eine Ableitung von spezifischen Modellbegriffen aus dem generischen Modellbegriff je nach Anwendungsszenario.

Ein Kategorisierung des Modellseins: Modelle werden primär (1) zur Präsentation von Vorstellungen, (2) zur Begleitung von Handlungen, (3) als Denkhilfe für Auffassungen, Vorstellungen bzw. Wissen oder (4) zur Daseinsbewältigung genutzt. Damit erhalten wir bereits ein grobe Kategorisierung. Nutzen wir weiterhin die Marke und die Mission, dann kann man eine feinere kanonische Kategorisierung erhalten.

Eine kanonische Parametrisierung für das Modellsein anhand der Kanonisierung der Modell-Eigenschaften und der Szenarien basiert auf typischen Herangehensweisen in den Anwendungen.



In vielen Anwendungen, die Modelle verwenden, finden wir zum einem eine solche Kanonisierung der Adäquatheit und Verläßlichkeit und zum anderen eine Kanonisierung der Szenarien, in den das Modell als Instrument funktioniert, vor. Diese Kanonisierung führt auch direkt auf eine Kategorisierung von Modellen. Es werden dann die Kanons für die jeweilige Kategorie vorausgesetzt und übernommen.

Vom generischen Modellbegriff zum disziplinären Kategorien-Modellbegriff: Die Kategorisierung, Kanonisierung und Konfiguration führt direkt auf einen spezifischen Modellbegriff, der ein Kategorie-Modellbegriff ist. So kann dann mit [MT21], ein **konzeptuelles Modell**, das eine weit genutzte Kategorie ist, definiert werden als:
„*Ein konzeptuelles Modell stellt eine präzise und zielgerichtete Konsolidierung einer Reihe von abstrakten Begriffen dar, die mit Hilfe von Termen in einem einem vordefinierten sprachlichen Format dargestellt werden. Als solches schafft es eine Sicht auf einen bestimmten enzyklopädischen Hintergrundraum abstrakter Begriffe.*“
Ein alternative explizite Ausformulierung der Konfiguration, die das Modellsein voraussetzt, führt zu [Tha21]:
Ein konzeptuelles Modell ist als ein Modell [8]

- *ein Präsentationsmodell,*
- *das unterstützt wird von abstrakten Begriffen aus einem enzyklopädischen Hintergrundraum,*
- *das formuliert wird durch referenzierende Terme in einer Sprache, die gut strukturierte Formulierungen ermöglicht,*
- *das basiert auf mentalen/Wahrnehmungs-/Situationsmodellen der Origins mit ihren eingebetteten abstrakten Begriff(en), wodurch es seine Verlässlichkeit und andere Kanons von Origins und den Anwendungen erbt, und*
- *das sich orientiert an einer allgemein akzeptierten strategischen Matrix.*

Die gesamte Vielfalt der Begriffe von einem konzeptuellen Modell in [Tha18] kann mit einer jeweiligen speziellen Anpassung aufgrund der Ausrichtung des jeweiligen Begriffes abgeleitet werden.

Vom Kategorien-Modellbegriff zum konkreten Anwendungsfall: Das Kategorien-Modellsein kann nun einfach an den konkreten Anwendungsfall angepaßt werden. Z. B. für die konzeptuelle Datenbank(struktur)-Modellierung mit einer erweiterten Entity-Relationship-Modellierungssprache übernehmen wir das Wesen, die Landschaft, Strategie und Taktik der konzeptuellen Kategorie und verfeinern diese zum einem um *Paradigmen* der Datenbank-Welt (Kompositionalität, System-Konstruktion als Kommunikationsorientierung, statisch dominiert dynamisch, Syntax-Semantik-Separation, DB-IS-Kultur, Kontextarmut, etc.), zum anderen um *Postulate* der objekt-relationalen DBMS (Global-As-Design, Prädikatenlogik als Fundierung, objekt-relationale Strukturierung, Homomorphie (oder der komplexere Infomorphismus) als Analogie, Salami-slice, OR-Normalisierung, Maschinenorientierung, Datenqualität, ER-Denkschule und -Gemeinschaft, Daten-Abstraktion, Visualisierung zuerst, etc.) sowie ergänzen des weiteren diese durch die sieben *Prinzipien* der Konzeptualisierung [Gri09] und andere Commonsense-Prinzipien (OR-Normalisierung, Generalisierung und Spezialisierung, strenge Separierung von Gesichtspunkten, Redundanzarmut, etc.) neben den Prinzipien der Modellierung (Substitution, KISS, Form folgt Funktion,

[8] adäquat, verläßlich, wohlgeformt, in Szenarien funktionierend

etc.). Diese Verfeinerung wird noch angereichert werden durch Plattform-spezifische *Annahmen* (z. B. Datentyp-Systeme, Programme als Hilfsmittel).

3.4 Das Arbeitsblatt

Das Arbeitsblatt präsentiert ein intendiertes Modell (model-what-is, model-as-should-be, what-how-why-when-where, model-used) und erlaubt eine Eingrenzung für die Identität eines Modells. Die Situation und das Szenario, in dem ‚Etwas' als Modell angesehen wird, werden durch das Arbeitsblatt untersetzt durch Antworten auf Fragen wie die folgenden, die analog für die Systemtheorie gestellt werden, z. B. [Alt13, Gre06]: Welches Problem und welche Möglichkeiten stehen primär und ggf. sekundär im Mittelpunkt? Welche Modelle und Modellkategorien erlauben es, diese Probleme anzugehen? Welche Faktoren charakterisieren das Modell? Worin liegen die Grenzen einer solchen Modellkategorie? Welche Funktionen werden in den gegebenen Szenarien durch welche Herangehensweisen durch das Instrument unterstützt? Welche Verläßlichkeit ist damit zu erwarten? Auf welche Art wird das Spiel von hinreichend genauer Analogie und sparsamem Fokus aufgelöst? Welche CoP agiert in welcher Rolle, welchem Bestreben und welcher Ausführung? Welche Stützung und Ausstattung sollte das Modell als Instrument haben und welche hat es aktuell? Ist die gewählte Orientierung auf den gegebenen Ansatz und das implizite Modell ausreichend? Welchen Beitrag kann ‚Etwas' als Modell in der gegebenen Situation haben? Inwieweit kann das Modell bei Veränderungen der Situation weiter verwendet werden? Welches Potential und welche Kapazität hat ein solches Modell und welche fehlen? Welche Form und Ausprägung sollte das Modell als wohlgeformtes und nutzbares Instrument haben?

Der Rhombus in Abbildung 2 faßt die Konfiguration für ein Modellsein zusammen. Daraus ist ggf. eine Kanonisierung und eine Kategorisierung ableitbar. Die Kategorisierung schließt intern auch das Wesen, die Landschaft, die Strategie und die Taktik mit ein, so daß man damit auch das spezifische Modell in der konkreten Anwendung als „normales" Modell ausarbeiten kann.

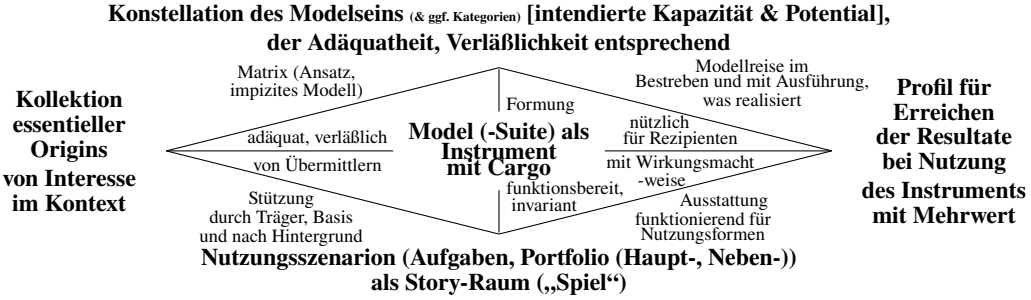


Abb. 2: Das Arbeitsblatt für das Modellsein zur Erstellung eines Modells nach Vorgaben eines Bestellers und zur Nutzung durch einen Rezipienten

Ein Arbeitsblatt kann durch Fallstudien unterlegt werden. Es beschreibt nur die Modellart oder Modell-Kategorie und kann durch eine Modell-Signatur auch systematisch geordnet dargestellt werden. Methodologien dienen zur Ergänzung, Ordnung und Gliederung der Arbeitsblätter und enthalten auch normative und informative Empfehlungen und Hilfen sowie Arbeitsabfolgen.

4 Ein kleiner Ausblick auf Forschungsthemen

Die Modellkunde ist eine junge Disziplin, die sowohl in der Praxis als auch der Forschung und des weiteren in der Lehre als eine geordnete Faktensammlung zusammengestellter Kenntnisse und Erfahrungen z. T. funktioniert. Sie kann systematisiert und verallgemeinert werden zur ‚technischen Kunst' der Beherrschung von Handlungen – insbesondere Herstellungshandlungen – hin zu einer theoretisch bewußten und gesicherten und in der Praxis erfolgreich erprobten *Kunde* für eine Erschließung von Kenntnissen, für die Nutzung von Erkenntnissen und Erklärungen, für die Vermittlung von Anschauungen und Vorstellungen und für die verstehende und begründete Praxis.

In der Informatik erhalten Modelle mit der Schaffung, der Gestaltung, der Erschließung und der Nutznießung des Künstlichen (technē) einschließlich formaler Verfahrensweisen eine zusätzliche Aufgabe neben dem für Modelle üblichen Begreifen, Orientieren und Darstellen (logos, $\lambda\acute{o}\gamma o\varsigma$ mit doxa und noesis) von Gewißheiten (Meinung, Überzeugung, Praxis), von Kulturen und eines Konsenses. Mit verschiedenen Wesen und Landschaften von Modellen wird es eine spezifische pluralistische Modellkunde in der jeweiligen Teildisziplin der Informatik geben (Proliferationsprinzip), die wir mit einer angepaßten Konfiguration des Modellseins begleiten können.

Demzufolge ranken sich Forschungsaufgaben um allgemein gesicherte Grundlagen und um eine begründbare Handlungsorientierung für Modelle und Instrumente unter Einschluß von bildungssystematischen und praxisorientierten Schemata für Modellentwicklung, -fortschreibung und -nutzung bei gleichzeitig theoretisch und praktisch orientierten Tätigkeiten und als dies im zeitlich wechselndem und meist ungesichertem Verständnis der beobachtbaren Realität der Informatik-Systeme und der Informatik-Nutzer.

Anmerkung: Ich danke den acht MMM-Diskussionsforen (Model–to_Model–Modelling) der letzten Jahre (Informatik (deutsch, englisch, konzeptuell), Medizin, Ägyptologie, Archäologie, MIS, Pädagogik), die regelmäßig Aspekte der Modellkunde vertiefen.

Literaturverzeichnis

[Alt13] Steven Alter. Work system theory: overview of core concepts, extensions, and challenges for the future. *Journal of the Association for Information Systems*, page 72, 2013.

[Bar07] M. Barbieri. *Introduction to biosemiotics: The new biological synthesis*. Springer Science & Business Media, 2007.

[BBW15] S. Bosco, L. Braucher, and M. Wiechec. *Encyclopedia Britannica, Ultimate Reference Suite*. Merriam-Webster, 2015.

[Bie08] A. Bienemann. *A generative approach to functionality of interactive information systems*. PhD thesis, CAU Kiel, Dept. of Computer Science, 2008.

[Bör03] E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 15:237–257, 2003.

[CH04] S. Chadarevian and N. Hopwood, editors. *Models - The third dimension of science*. Stanford University Press, Stanford, California, 2004.

[DPT21] A. Dahanayake, O. Pastor, and B. Thalheim. *Modelling to Program: Second International Workshop, M2P 2020, Lappeenranta, Finland, March 10–12, 2020, Revised Selected Papers*, volume 1401 of *CCIS*. Springer Nature, 2021.

[DSS03] A. N. W. Dahanayake, H. G. Sol, and Z. Stojanovic. Methodology evaluation framework for component-based system development. In *Advanced Topics in Database Research, Vol. 2*, pages 213–246. Idea Group, 2003.

[Fle11] L. Fleck. *Denkstile und Tatsachen, edited by S. Werner and C. Zittel*. Surkamp, 2011.

[Fra22] U. Frank. Multi-level modeling: cornerstones of a rationale. *Softw. Syst. Model.*, 21(2):451–480, 2022.

[Gre06] S. Gregor. The nature of theory in information systems. *MIS quarterly*, pages 611–642, 2006.

[Gri09] J. J. Van Griethuysen. The Orange report ISO TR9007 (1982 - 1987) Grandparent of the business rules approach and sbvr part 2 - The seven very fundamental principles. *Business Rules Journal*, 10(8), August 2009. http://www.brcommunity.com/a2009/b495.html.

[Hes06] W. Hesse. Modelle - Janusköpfe der Software-Entwicklung - oder: Mit Janus von der A-zur S-Klasse. In *Modellierung 2006*, volume 82 of *LNI*, pages 99–113. GI, 2006.

[HHM10] G. Hofstede, G.J. Hofstede, and M. Minkow. *Cultures and Organizations: Software of the Mind: Intercultural Cooperation and Its Importance for Survival*. McGraw-Hill, New York, 2010.

[HM08] W. Hesse and H. C. Mayr. Modellierung in der Softwaretechnik: eine Bestandsaufnahme. *Informatik Spektrum*, 31(5):377–393, 2008.

[Kas03] R. Kaschek. *Konzeptionelle Modellierung*. PhD thesis, University Klagenfurt, 2003. Habilitationsschrift.

[Kuh70] T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, Illinois, 2nd, enlarged, with postscript edition, 1970.

[Leh99] N.J. Lehmann. Leibniz als Erfinder und Konstrukteur von Rechenmaschinen. *Wissenschaft und Weltgestaltung*, pages 255–267, 1999.

[LM78] P.C. Lockemann and H.C. Mayr. *Rechnergestützte Informationssysteme*. Springer, 1978.

[Mah05] B. Mahr. Was ist ein Modell. Der Modellbegriff in Natur- und Ingenieurwissenschaften. TU Berlin Summer term 2005, 2005. Accessed on August 3rd, 2020.

[Mah15a] B. Mahr. Modelle und ihre Befragbarkeit – Grundlagen einer allgemeinen Modelltheorie. *Erwägen-Wissen-Ethik (EWE)*, Vol. 26, Issue 3:329–342, 2015.

[Mah15b] B. Mahr. Replik. *Erwägen-Wissen-Ethik (EWE)*, Vol. 26, Issue 3:425–433, 2015.

[Mah21] B. Mahr. *Schriften zur Modellforschung*. Herausgegeben von K. Robering. Brill Mentis, Michaelisbund, 2021.

[MMR+17] H. C. Mayr, J. Michael, S. Ranasinghe, V. A. Shekhovtsov, and C. Steinberger. Model centered architecture. In *Conceptual Modeling Perspectives.*, pages 85–104. Springer, 2017.

[Mor20] T. Moritz. *Szenografie digital – Die integrative Inszenierung raumbildender Prozesse*. Springer Vieweg, Wiesbaden, 2020.

[MT21] H.C. Mayr and B. Thalheim. The triptych of conceptual modeling – a framework for a better understanding of conceptual modeling. *Softw. Syst. Model.*, 20(1):7–24, 2021.

[Mül16] R. Müller. Geschichte des Systemdenkens und des Systembegriffs. http://www.muellerscience.com/SPEZIALITAETEN/System/systemgesch.htm, 2016. Assessed Oct. 29,2017.

[Pag18] S. E. Page. *The model thinker: What you need to know to make data work for you*. Basic Books, 2018.

[Pfä21] A. Pfänder. *Logik*. Verlag von Max Niemeyer, Halle a. d. Saale, 1921.

[Pro72] V.J. Propp. *Morphologie des Märchens*. Carl Hanser Verlag, München, 1972.

[Rhe21] H.-J. Rheinberger. *Spalt und Fuge: eine Phänomenologie des Experiments*. Suhrkamp, 2021.

[ST19] K.-D. Schewe and B. Thalheim. *Design and development of web information systems*. Springer, Chur, 2019.

[Sta73] H. Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.

[Sta92] H. Stachowiak. Modell. In H. Seiffert and G. Radnitzky, editors, *Handlexikon zur Wissenschaftstheorie*, pages 219–222. Deutscher Taschenbuch Verlag GmbH & Co. KG, München, 1992.

[Ste93] W. Steinmüller. *Informationstechnologie und Gesellschaft: Einführung in die Angewandte Informatik*. Wissenschaftliche Buchgesellschaft, Darmstadt, 1993.

[TF16] M. Tropmann-Frick. *Genericity in Process-Aware Information Systems*. PhD thesis, Christian-Albrechts University of Kiel, Technical Faculty, Kiel, 2016.

[Tha10] B. Thalheim. *The Conceptual Framework to Multi-Layered Database Modelling based on Model Suites*, volume 206 of *Frontiers in Artificial Intelligence and Applications*, pages 116–134. IOS Press, 2010.

[Tha13] B. Thalheim. The conception of the model. In *BIS*, volume 157 of *Lecture Notes in Business Information Processing*, pages 113–124. Springer, 2013.

[Tha18] B. Thalheim. Conceptual model notions - a matter of controversy; conceptual modelling and its lacunas. *EMISA International Journal on Conceptual Modeling*, pages 9–27, February 2018.

[Tha21] B. Thalheim. Models, to model, and modelling. *collections of papers.* https://www.researchgate.net (search keyphrase "Towards a theory of models, especially conceptual models and modelling"), 2009-2021.

[Tha22] B. Thalheim. Models: The fourth dimension of computer science – towards studies of models and modelling. *Software & Systems Modeling*, 21:9–18, 2022.

[Tho05] O. Thomas. Das Modellverständnis in der Wirtschaftsinformatik: Historie, Literaturanalyse und Begriffsexplikation. Technical Report 184, Institut für Wirtschaftsinformatik, DFKI, Saarbrücken, Mai 2005.

[TN15a] B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.

[TN15b] B. Thalheim and I. Nissen. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*, chapter Ein neuer Modellbegriff, pages 491–548. De Gryuter, Boston, 2015.

[Ump01] S. A. Umpleby. What comes after second order cybernetics? *Cybern. Hum. Knowing*, 8(3):87–89, 2001.

[War79] M. W. Wartofsky. The model muddle: Proposals for an immodest realism. In *Models: Representation and the Scientific Understanding*, pages 1–11. Springer, 1979.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 33

Generating Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines

Judith Michael[1], Imke Nachmann[1], Lukas Netz[1], Bernhard Rumpe[1], Sebastian Stüber[1]

Abstract: The complexity of wind energy systems combined with an increased trend towards mass customization require the collaboration of many experts to achieve high quality products. Currently, a major issue arises from the lack of data integration among the different tools used during the engineering process which may cause system failures eventually. Existing tools largely do not support automatic detection and indication of erroneous or contradictory parameter values between artifacts of different tools. Employing a model-driven and functional engineering approach enables to establish an integrated toolchain for the management and visualization of engineering artifacts that consume and produce the data. Within this paper, we present an automatic approach to derive an engineering digital twin for the cooperative development and management of engineering artifacts from functional models of the system under development. We evaluate our approach on the example of a hydraulic pump within the cooling system of a wind turbine. The prototype can be coupled with an existing engineering tool ecosystem. The approach enables to exchange the data produced by engineering artifacts according to a functional system model which facilitates the cooperation between different stakeholders throughout the development process.

Keywords: Parameter Management; Functional Modeling; Model-Driven Systems Engineering; Digital Twin Cockpit; Wind Turbine

1 Introduction

The use of renewable energy is facing a massive growing. Better engineering processes, cost reductions and a strong political will constituting in the Paris Agreement of 2016 [LJW18] are the driving factors for this growth. The complexity of wind energy systems combined with an increased trend towards mass customization [Ve19] require the collaboration of many experts to achieve high quality products. Such wind energy systems have to be customized for each site and the requirements of wind park owners. Their engineering needs various iterations, information exchange between different stakeholders and the simulation of properties of a large number of individual components within different engineering and simulation tools.

Mechanical engineering employs heterogeneous tools to represent the system's geometry, behavior, etc. These artifacts describe different aspects and perspectives of the system using a common set of system parameters. Yet, in many engineering projects there is

[1] Software Engineering, RWTH Aachen University, Germany, www.se-rwth.de
{michael,nachmann,netz,rumpe,stueber}@se-rwth.de

doi:10.18420/modellierung2022-012

no standardized form of data storage, naming and exchange, making the set of common parameters and their values implicit and only known to experienced users. Instead, data is exchanged via E-Mail attachments, or as links to a temporary cloud storage. A central data management system increases productivity, especially when combined with analysis functionalities [Fe17].

Existing Product Data Management (PDM) systems are designed for large companies, where a dedicated team configures and maintains these systems. For small or medium-sized companies, setting up a commercial PDM can be very costly and time intensive. Agile modifications of the data structure is often not feasible and data sharing often relies on the manual exchange of parameters in diverse artifacts.

Research Question. Within this paper, we further investigate how to enhance the manual engineering process by using modeling, data management and an integrated toolchain.

Contribution. We employ Model-Driven Engineering (MDE) to generate a Digital Twin (DT) cockpit from models which provides the graphical interface to visualize its data and the interaction with services of the digital twin. Within this stage, it is an "as-designed" digital twin, which exists during design including technical design and simulation. The DT cockpit allows for artifact exchange, versioning and visualization from architectural models of the system under development. The DT cockpit continuously evolves towards a digital twin of the real physical object called digital twins "as-manufactured" during construction and "as-operated" during runtime. Our approach relies on software and systems engineering methods using code synthesis from model artifacts. The generated DT cockpit allows to modify data via a web-interface or using a python Application Programming Interface (API). The API enables to integrate existing development tools, such as simulation programs like MATLAB[2] as digital twin services. We demonstrate the results on a concrete use case from the engineering of wind turbines.

The paper is structured as follows: Sect. 2 introduces preliminaries and Sect. 3 presents the vision for an MDE process. Sect. 4 described the use case. Afterwards, Sect. 5 shows the implemented tools. Sect. 6 discusses their features and shortcomings in comparison to related work and Sect. 7 concludes.

2 Preliminaries

In this section, we explain the basic principles of functional systems models and our technology stack to generate a digital twin cockpit.

[2] https://www.mathworks.com/products/matlab.html

2.1 Functional Modeling of Systems

Modern systems realize complex functionalities confronting engineers with highly challenging engineering tasks. Part of the complexity is caused by the fact that finding solutions to implement these functionalities requires the collaboration of experts from heterogeneous domains, i. e. software engineering, electrical engineering, and mechanical engineering [Dr20]. The challenge arises from a conceptual gap, because requirements are stated in natural language at a very high level of abstraction, which most often concerns the function of the system from which engineers derive technical product architectures directly. The artifacts that describe the geometry and behavior of the geometric components in these architectures are highly detailed down to the molecular level. In systems engineering, MDE therefore aims at exploiting the ideas introduced in mechanical design theory [KK98, Pa07], that a system realizes a function by transforming energy, material, and data. Functional models capture this function which can be derived from the requirements more directly and decomposed into functional components iteratively [Pa07]. Design catalogs, e. g. [KK98], provide a list of so called elementary functions which represent functions and link information on possible effects, geometries and materials to engineer implementations to these functions. We refer to these implementations as principle solutions [Pa07]. The modeling technique to represent the functional system architecture in Systems Modeling Language (SysML) proposed in [Dr20] allows to link system attributes along functional interaction lines to support testing [Ze21] and optimal dimensioning [Ho21] across the geometric boundaries of the product architecture.

2.2 Model-Driven Engineering of Digital Twins

Digital twins are developed for various application domains such as healthcare [Li19], laser cutting [Li21], injection molding [Bi20], or automotive [Ku18]. In our understanding, "a **digital twin** of a system consists of a set of *models* of the system, a set of *digital shadows*, and provides a set of *services* to use the data and models purposefully with respect to the original system [Da20]." In this work we focus on the development of a specific part of the digital twin: the digital twin cockpit, that we specify as follows:

A **digital twin cockpit** is "the user interaction part (UI/GUI) of a digital twin. It provides the graphical user interface for visualizations of its data organized in digital shadows and models, and the interaction with services of the digital twin" [Ba22]. This enables humans to access, adapt, and add information, and we enable them to monitor and partially control the physical system.

Considered heterogeneous *models* from various disciplines help to understand the system in focus, which includes its' structure, behavior, functions, and physical, geometrical or mathematical relations. Model-driven approaches for digital twin engineering [Bo20] aim to use this models during runtime as well as for the generation of a digital twin. A *digital*

shadow is a passive set of data which includes "a set of contextual data-traces and/or their aggregation and abstraction collected concerning a system for a specific purpose with respect to the original system" [Be21]. Different *services* provide additional functionalities for a digital twin such as analyzing, prediction or simulation of behavior or to control the physical object.

2.3 The MontiGem Generator Framework

MontiGem [Ad20] is a generator framework for the engineering of web-based information systems and DTs [Da20]. The full version of MontiGem uses a set of models as input and synthesizes software code for the Java-based backend and Typescript/HTML-based frontend within the Angular framework. These models include UML Class Diagrams (CDs) for defining domain concepts, data models to define the transportation of data in command objects between the backend and frontend of the application, models defining the graphical user interface and Object Constraint Language (OCL) models to define data validation within the frontend and backend of the derived application.

The MontiGem version used within this paper includes an extension called CD2GUI [Ge20b] before code synthesis: The generation process takes only one class diagram as input and generates data and GUI models which are further processed by the MontiGem generator framework into a running application.

MontiGem is used since several years in real-world projects for financial management [Ge20a], for energy management systems and for creating digital twin cockpits [Da20]. Moreover, we use it in research projects on privacy-preserving information systems [Mi19] and goal modeling in assistive systems [MRV20].

3 The Vision Towards Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines

In many engineering projects there is no standardized form of data storage and exchange. Instead, engineers exchange data via E-Mails with attached informal documents or via instant messages with links to a temporary cloud storage. This form of data management has several downsides: 1) Changes in the data are not propagated fast enough, hence outdated data might be used, 2) data of past projects is not easily accessible, leading to a slower development process and repetition of previous errors, and 3) different tools cannot interoperate due to different data formats.

Vision. Thus, we propose an agile, model-driven engineering process for mechanical systems which are supported by an engineering digital twin and its' according cockpit. Functional models contain the behavior of each subsystem and the interaction of the subsystems. A functional model contains parameters, which are used in the behavior description.

For example, the functional module ‘Counter Weight’ would have the parameter ‘mass’ and a description of the mechanical-behavior using this parameter. The concrete value of the mass is not part of the functional module. This enables to reuse sub-components across different systems. Two ‘Counter Weights’ with different masses still use the same functional module, only the parameters change.

From the functional models and models of the used data, we generate digital twin cockpit. Parameters and all other data are stored in a central database. When users need the concrete values for parameters, e.g., for a simulation, the values are queried. Moreover, the results of simulations are also stored in the central database. To achieve reproducibility, the result of a simulation is linked with the input parameters. Hence it is possible to later inspect the parameters or run the simulation with the same parameters again. Logs of changes, an archive of old versions of data and access control are also provided. These aspects are especially important if contributors are spatially distributed.

This vision requires to provide multiple ways to create, read, or update data: A graphical user interface and an programming interface. The graphical user interface, the digital twin cockpit, can be used by developers to quickly modify parameters or interpret the results of simulations. Tools can be integrated with the programming interface. The standardized way of data-storage enables the integration of heterogeneous tools.

Long-term goal. We consider the early phase where a physical object is planned and designed as starting point for a digital twin, i.e., an *engineering digital twin*, which continuously evolves over time to a fully-functional digital twin representing a physical product. Suitable *models* in the early phase are engineering models and models to describe behavior-to-be, that can be simulated and analyzed. Relevant *data* are simulation inputs and outputs, and result from different versions of the planned physical objects. *Services* include simulation and analysis of the physical system-to-be. The latter fully-functional digital twin allows for bi-directional synchronization of a digital twins and the physical system. However, this evolvement is not further discussed in the paper, as we focus on the creation of engineering digital twins and especially their digital twin cockpits.

4 Use Case

Our running example is a use case for the engineering of a cooling system of a wind turbine to which we have applied our approach. The functional architecture [Dr20] of this system serves as a basis to derive a digital twin cockpit using a generation process.

The main function of a wind turbine is to transform wind into electrical energy. Generally, wind is the movement of air in the earth’s (or another planet’s) atmosphere when it flows from areas of lower pressure to areas of higher pressure [Br16]. These pressure differences arise because energy is transferred into the air in the form of heat generated by the sun. Therefore, wind can be considered a stream of energy that is transferred from the atmosphere

to the air molecules [FR76]. Speaking in terms of [FR76], the energy being transferred appears as motion (of the air molecules), which is bound to the physical quantities velocity and force. Thus, a wind turbine transforms motion energy to electrical energy. The general structure of a wind turbine [Br16] comprises a rotor with several blades, which transforms the motion energy to rotational energy, characterized by torque and angular velocity. The rotor generates very high torque and low angular velocity. The main shaft transmits the rotational energy to a gearbox which decreases the torque to increase the angular velocity. Finally, a generator transforms the so transformed rotational energy into electrical energy, i. e. voltage and current. These components make up the wind turbine's drive train.

In this setting, the generator and the gearbox generate heat due to energetic losses during the transformation [To10]. Further, wind turbines are usually installed outside of urban areas and are often exposed to extreme climate. To assure proper functioning, these systems need sophisticated thermal management systems, that keep the temperatures of the wind turbine's components within an operating range while assuring a high efficiency of the power generation [To10].

The wind turbine's components, and, in particular the generator, produce large amounts of heat during operation, which makes efficient cooling essential for proper and efficient operation the system. From a functional perspective, the wind turbine's cooling system assimilates that of an automotive engine (see [Dr20, Ze21, Ho21]). The SysML Block Definition Diagram (BDD) in Figure 1 shows the components of the cooling system with their functional inputs and outputs together with a set of SysML value properties, while the SysML Internal Block Diagram (IBD) in Figure 2 shows the interaction of these components.

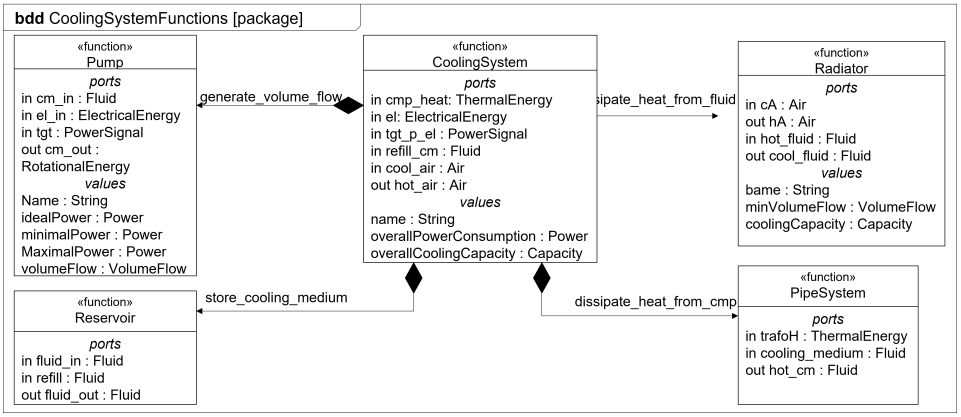


Fig. 1: BDD that models the functions of the cooling system.

The value properties model the characteristics of the component they belong to, often, these are parameters of simulations or CAD-models that could be linked to the functional structure [Ho21]. The cooling system in Figure 1 consists of a Pump, a Radiator, a Reservoir and a Pipe System. The reservoir stores a cooling medium, which is a fluid that is particularly

suitable for dissipating or absorbing heat. The pump causes a pressure difference that causes the fluid to flow out of the reservoir and into the Pipe System which surrounds the component from which heat has to be absorbed (see Figure 2). In our case, the thermal energy that is input to the Pipe System's function stems from the wind turbine's generator. The flowing motion of the cooling medium facilitates the absorption of the heat from the generator by the cooling medium. The cooling medium's temperature therefore rises. The radiator is responsible for cooling the cooling medium, i. e. for dissipating the heat absorbed by the cooling medium into the surrounding air. The incoming flow of cool air passes the cooling medium and thereby heats up by absorbing heat from the cooling medium. The temperature of the cooling medium thereby drops and it is released into the reservoir.

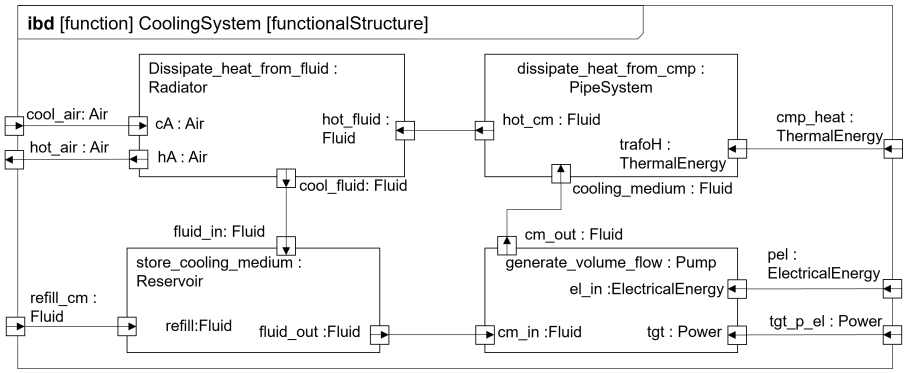


Fig. 2: IBD that shows the internal structure of the cooling system.

Therein, the pump for example is characterized by five parameters, i. e. a name, an ideal, maximal, and minimal power and a volume flow the pump generates during operation, which the BDD in Figure 1 shows. These values parametrize a possible simulation of the pump's physical behavior which facilitates, e. g. virtual testing [Ho21]. The approach presented in the upcoming sections allows to establish a systematic data management behind these parameters which facilitates validation and reuse of the functional models.

5 Realization of the Approach

We demonstrate how we have applied our approach to the use case. First, we model the data-structure for the cooling system based on the functional model presented in Sect. 4 as a UML CD. From the CD a DT cockpit is automatically generated. We show the Python API to send and read data from the server. Finally, we extend the generated DT cockpit for the analysis of simulation data.
The tool was developed in cooperation with an industry partner. The use case is a simplification of the CD from the industry partner, which contains more than 100 classes.

The Extended MontiGem Framework (Figure 3) consists of three major components: A preliminary model-to-model transformation, a data structure (DS) generator and a user

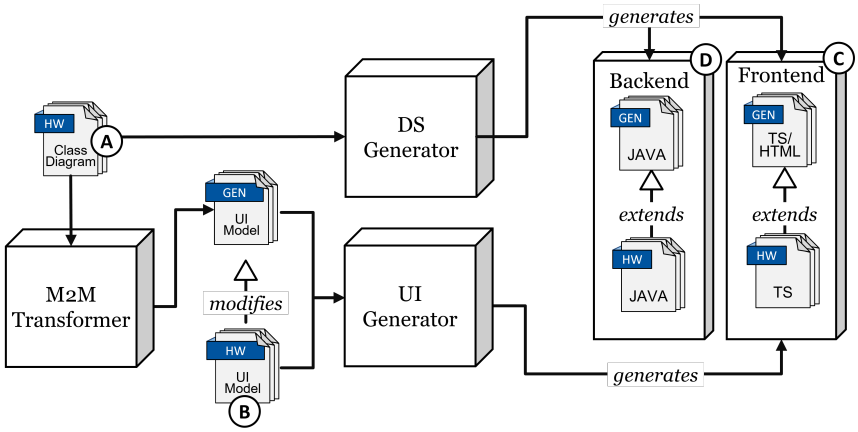


Fig. 3: Overview of the MontiGem Generation Process

interface (UI) generator. This framework only requires a data structure model as input, but can be supplemented with additional models to refine the target application. In order to use MontiGem, a domain model (A) has to be provided in the form of a CD. The model is passed on to the DS-Generator, that will create a corresponding database and infrastructure to access it. The model is also passed to a model-to-model transformation that will create a set of three UI-Models for each class (Navigation, Overview, Editing). These models define user interfaces enabling the end user to inspect data and perform basic CRUD operations on the database. The models can be extended with handwritten ones (B) by the developer, in case further user interfaces or modifications to the existing ones are needed. Both generated and handwritten models are used as input for the UI-Generator. In combination both DS- and UI-Generator produce the the application front end (C) and back end (D).

```
1  class Radiator {
2    String name;
3    Double minimalVolumeFlow;
4    Double coolingCapacity;
5  }
6  class Pump {
7    String name;
8    Double idealPower;
9    Double minimalPower;
10   Double maximalPower;
11   Double VolumeFlow;
12 }
```

```
13 class CoolingSystem {                    CD
14   String name;
15   Double /overallPowerConsumption;
16   Double /overallCoolingCapacity;
17 }          derived attribute
18 <<ordered>>
19 association CoolingSystem -> Radiator [*];
20
21 <<ordered>>   order of items is kept
22 association CoolingSystem -> Pump [*];
23
24
```

Fig. 4: Class Diagram for Case Study

For the Use Case in Sect. 4 we have extracted a CD describing the data schema. Fig. 4 shows an extract this CD. Each class has a human-readable `name`, which describes the model or configuration. The classes on the left represent the two types of components in our cooling system: Radiators (l.1-5) and Pumps (l.6-12). On the right side is the composed

system (l.13-17). The composed systems has links to its subcomponents. The attributes `overallPowerConsumption` and `overallCoolingCapacity` are derived from the subcomponents. For example, the `overallCoolingCapacity` is the sum of the `coolingCapacity` of each radiator (see List. 1).

```
1 @Override
2 public Double getOverallCoolingCapacity() {
3  return this.getRadiators().stream()
4    .collect(Collectors.summingDouble(x->x.getCoolingCapacity()));
5 }
```

List. 1: Computing a derived value

From the CD, MontiGem automatically generates a DT cockpit as described in Figure 3. On this application we can create new entries, search existing entries, update values and delete entries. Fig. 5 displays two generated elements of the DT cockpit. The left side is an overview of all available `Pump` instances currently stored in the database. In this list only the most important attributes are displayed, however the table can be customized to hide or display any of the attributes of the pump class. To make the table more readable less relevant attributes (e.g. `minimalPower`) are hidden. Via the "settings button" (German "Einstellungen"), the user can configure which attributes are displayed.

The user can inspect an object by clicking on the "eye" icon (see right side of Fig. 5) which opens a separate page where the object can be modified. Pressing the "edit" button allows to modify the attributes.

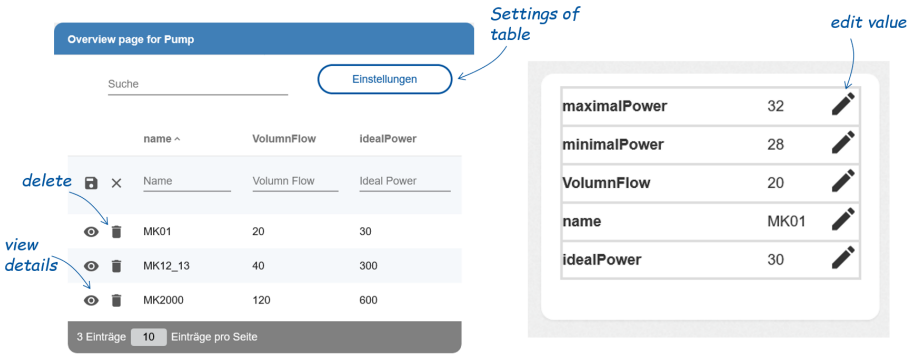


Fig. 5: Generated user interfaces for `Pump` class. Left: Overview showing all Pump-Objects currently stored in DB. Objects can be created and deleted in this view.

A `CoolingSystem` has links to multiple `Pump` objects. This is displayed in a separate table, shown in Fig. 6. Visually this table is similar to the overview of all pump objects, but only linked objects are listed. Furthermore, the "delete" button only removes the link between the objects, and not the object itself.

When a user adds a `Pump` object to a `CoolingSystem`, the derived attribute

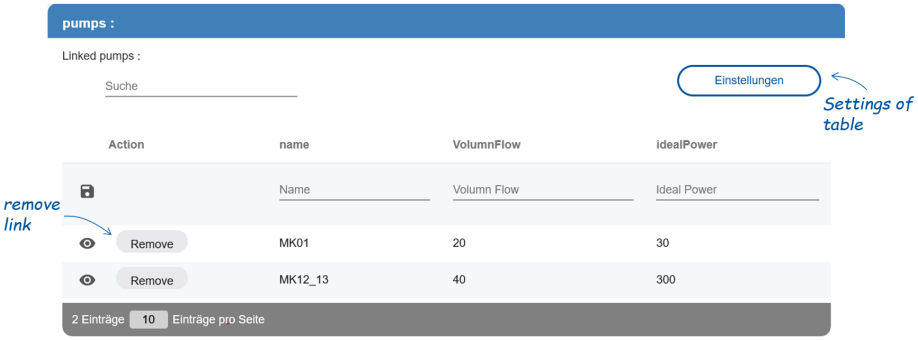


Fig. 6: UI showing associated `Pumps` of `CoolingSystem`. Links to objects can be edited.

`overallCoolingCapacity` is automatically updated. Directly modifying derived values is not possible (see Fig. 7).

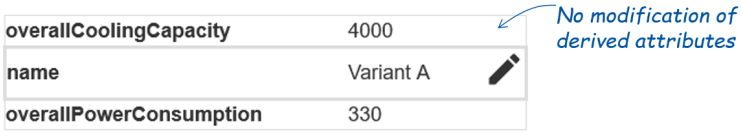


Fig. 7: Attributes of `CoolingSystem`. Only attributes stored in the database can be edited, aggregated or otherwise derived attributes are immutable.

Python Extensions. The data in MontiGem should be automatically usable with existing mechanical engineering tools. Many tools (e.g. Rhino, Ansys Workbench, SpaceClaim, Mechanical) already come with python support, so we used python for the integration. To make the API as usable as possible in practice, the developed code runs both in IronPython2 and CPython3.

The CRUD-operations are fully supported and navigation of associations is handled transparently. For performance optimization, associated objects are lazy-loaded and only retrieved from the MoniGem server on read-access.

```
obj = Factory.new(type_name="Radiator",
        data={"name": "DIA_AB",
              "minimalVolumnFlow": 300,
              "coolingCapacity": 4000})
obj.push()
```

List. 2: Create new `Radiator` via PythonAPI

Archive and Metadata. The MontiGem application also stores a history of all changes made to the data. This acts as an archive and ensures that no information is lost. For the

future usability of the data, meta-information can be linked to various places. An example is the “release status” which tracks the lifecycle state of an object. Customized phases can be configured, e. g. `draft` > `review` > `released` > `outdated`. Before switching to the `released` status, MontiGem searches for associated objects that are not yet released and lists them to the user. The user can inspect them individually or release all associated objects. This way it is ensured, that `released` objects only link to other `released` objects.

Display Differences. When manually analyzing the data, for example before releasing an object, the developer can be overwhelmed by the large amount of information. We have implemented a difference operation, which only displays the changed values. This helps the developer to focus on modified values.

Visualization. The automatically generated DT cockpit in its current version can only display values in tables. While this is enough for many applications, some require a customized visualization to be usable. For example, simulation results, which typically consist of large lists. These are hard to analyze in the default table layout.
For this case, we have manually developed an extension to the generated layout. In a pivot-table, the user can configure the ideal visualization. The attributes on the x- and y-axis can be selected using drag and drop, there are multiple aggregation functions (e.g. sum, average, count, maximum) and multiple different visualization methods are available. Examples for the visualization methods are shown in Fig. 8.

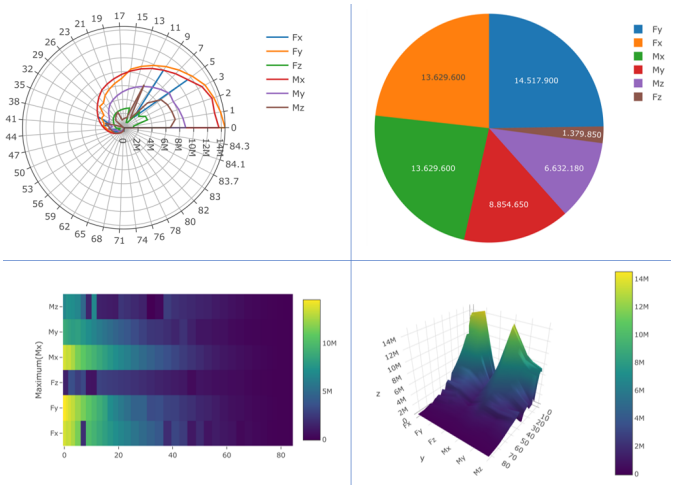


Fig. 8: Four exemplary visualizations of simulation results

6 Discussion and Related Work

Discussion. Our model-driven approach allows to change the data-schema definition, and automatically generate a new customized DT cockpit. For example, adding a new class works straightforward.

Generality of Approach. While the presented case study is specific to wind turbines, our approach is independent of the application domain. Other domains might use a different programming language than python in their specific tools. A wider language-support for the API would ease the integration.

Digital Twin Functionality. Our generated DT cockpit focuses on the data management and visualization services. The functional part of the digital twin was out of focus of this project. The generated DT cockpit is intended to use in combination with other services, which allow for predictive control [Ja14], model the behavior of physical systems, and can influence the physical system. Especially a combination with other model-driven approaches as in [Ho21] or [Br21] is promising.

Integrate External Tools. The ability to bind external tools with a database enables interesting quality control and optimization methods. Geometric requirements like 'Component X has to fit into Y' are very common and can be checked fast and automatically. Similar to a continuous integration pipeline, this can be checked after each modification.

Lifecycle. While the integration of the "release status" enables a basic lifecycle tracking, this falls behind a fully fledged workflow-tool. Furthermore, a notification system which reminds an engineer of outstanding tasks would increase the usability.

Generate GUI. In the context of this project, it was the first time the extension to automatically create GUI-models for the provided CD was applied to a real world scenario. There are improvements needed within the UI as well as increasing the performance for data access. Currently, there is only one option to perform a database operation. Experiences from other software projects have shown that having multiple options is required to work efficiently and pleasantly with the system. This could be combined with more options to customize already generated models in order to have a more use case specific GUI.

Data Migration. Changes to an existing data-schema can lead to inconsistency and require a manual data-migration. Guiding the administrator and providing semi-automatic upgrade scripts generated using model-comparison approaches is a possible future extension.

Related Work. Existing approaches for data management in engineering processes evolved from PDM to PLM [AG00] and more recent system lifecycle management systems [Ei21]. While our system offers a standardized way to read and write data, many CAD-Tools only provide proprietary storage formats and require data-conversion. Hislop, Lacroix and Moeller [HLM04] discuss this and notes requirements for a data management system.

While most approaches consider a digital twin from the moment a physical object exists [MML19, Go21] over its lifetime, models and data from the planning and engineering phase of this physical object are neglected. There is not a lot of research done in the automatic derivation of GUI-models, however *low-code development platforms* are emerging which enable a developer to define a (web) application by configuration in the editor or by defining a set of models [Wa19, BGS20, Da22]. The current field of low-code applications is well documented by Gartner [Vi19] and recent studies [BF21].

Model-based systems engineering has recently become a broad research topic. Therein, modeling the system under development using dedicated modeling languages to obtain a single-source of truth is a major concern, to facilitate the cooperation among experts of very different backgrounds, i. e. software, mechanics and electronics. The modeling technique applied in this paper, relies on [Dr20] which allows to link the functional architecture of a system to implementations of the functional components. Such models have also been used for automating virtual tests and dimensioning in [Ho21]. However, the latter approach does not rely on a database and generative methods to systematically manage the data that arise during the development. Other modeling techniques that do not necessarily focus on the functional perspective of the system's architecture or do not provide the necessary link between a system's function and its solutions are proposed, e. g. in [GDN10, EKM17, WS09]. Using informal sketches as in [Mo15] does not suffice for generating data schemes and hinders efficient and systematic data management in the engineering process. To the best of our knowledge, existing approaches to automate activities of the systems engineering process do not rely on functional models of the system to generate data management infrastructure or DTs.

7 Conclusion

We have shown that the approach enables the exchange of data produced by engineering artifacts according to a functional system model which facilitates the cooperation between different stakeholders throughout the development process. The generated engineering DT cockpit provides a graphical user interface for searching, inspecting and editing values. Furthermore, existing mechanical engineering tools like Ansys Workbench or SpaceClaim can be integrated through a python programming interface. We have successfully demonstrated the approach on a system provided by an industry partner and on the cooling system from a wind turbine presented in this paper. However, for long term usage in practice, further additions such as adding digital twin services for optimization, maintenance prediction, or self-adaption of the wind turbine are needed.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612

Bibliography

[Ad20] Adam, Kai; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In: 40 Years EMISA: Digital Ecosystems of the Future: Methodology, Techniques and Applications (EMISA'19). volume P-304 of LNI. GI, pp. 59–66, 2020.

[AG00] Abramovici, Michael; Gerhard, Detlef: A flexible web-based PDM approach to support virtual engineering cooperation. In: 33rd Annual Hawaii Int. Conf. on System Sciences. IEEE, pp. 10–pp, 2000.

[Ba22] Bano, Dorina; Michael, Judith; Rumpe, Bernhard; Varga, Simon; Weske, Matthias: Process-Aware Digital Twin Cockpit Synthesis from Event Logs. Journal of Computer Languages (COLA), 2022.

[Be21] Becker, Fabian; Bibow, Pascal; Dalibor, Manuela; Gannouni, Aymen; Hahn, Viviane; Hopmann, Christian; Jarke, Matthias; Koren, Istvan; Kröger, Moritz; Lipp, Johannes; Maibaum, Judith; Michael, Judith; Rumpe, Bernhard; Sapel, Patrick; Schäfer, Niklas; Schmitz, Georg J.; Schuh, Günther; Wortmann, Andreas: A Conceptual Model for Digital Shadows in Industry and its Application. In: Conceptual Modeling, ER 2021. Springer, pp. 271–281, October 2021.

[BF21] Bock, Alexander C.; Frank, Ulrich: In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms. In: 2021 ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C). 2021.

[BGS20] Bexiga, Mariana; Garbatov, Stoyan; Seco, João Costa: Closing the Gap between Designers and Developers in a Low Code Ecosystem. In: 23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Companion. MODELS '20. ACM, 2020.

[Bi20] Bibow, Pascal; Dalibor, Manuela; Hopmann, Christian; Mainz, Ben; Rumpe, Bernhard; Schmalzing, David; Schmitz, Mauritius; Wortmann, Andreas: Model-Driven Development of a Digital Twin for Injection Molding. In: Int. Conf. on Advanced Information Systems Engineering (CAiSE'20). volume 12127 of LNCS. Springer, pp. 85–100, 2020.

[Bo20] Bordeleau, Francis; Combemale, Benoit; Eramo, Romina; van den Brand, Mark; Wimmer, Manuel: Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges. In: Systems Modelling and Management. Springer, pp. 43–54, 2020.

[Br16] Breeze, Paul: Chapter 2 - The Wind Energy Resource. In: Wind Power Generation. Lehmanns, 2016.

[Br21] Brockhoff, Tobias; Heithoff, Malte; Koren, István; Michael, Judith; Pfeiffer, Jérôme; Rumpe, Bernhard; Uysal, Merih Seran; van der Aalst, Wil M. P.; Wortmann, Andreas: Process Prediction with Digital Twins. In: Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C). ACM/IEEE, 2021.

[Da20] Dalibor, Manuela; Michael, Judith; Rumpe, Bernhard; Varga, Simon; Wortmann, Andreas: Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits. In: Conceptual Modeling. Springer, pp. 377–387, October 2020.

[Da22] Dalibor, Manuela; Heithoff, Malte; Michael, Judith; Netz, Lukas; Pfeiffer, Jérôme; Rumpe, Bernhard; Varga, Simon; Wortmann, Andreas: Generating Customized Low-Code Development Platforms for Digital Twins. Journal of Comp. Lang. (COLA), 70, 2022.

[Dr20] Drave, Imke; Rumpe, Bernhard; Wortmann, Andreas; Berroth, Joerg; Hoepfner, Gregor; Jacobs, Georg; Spuetz, Kathrin; Zerwas, Thilo; Guist, Christian; Kohl, Jens: Modeling Mechanical Functional Architectures in SysML. In: 23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems. ACM, pp. 79–89, October 2020.

[Ei21] Eigner, Martin: Forty Years of Product Data Management from PDM via PLM to SysLM. In: System Lifecycle Management: Engineering Digitalization (Engineering 4.0). Springer Fachmedien Wiesbaden, Wiesbaden, pp. 5–25, 2021.

[EKM17] Eigner, Martin; Koch, Walter; Muggeo, Christian, eds. Modellbasierter Entwicklungsprozess cybertronischer Systeme. Springer, 2017.

[Fe17] Fernández-Miranda, S Suárez; Marcos, M; Peralta, María Estela; Aguayo, F: The challenge of integrating Industry 4.0 in the degree of Mechanical Engineering. Procedia manufacturing, 13:1229–1236, 2017.

[FR76] Falk, Gottfried; Ruppel, Wolfgang: Energie und Entropie: Die Physik des Naturwissenschaftlers. Eine Einführung in die Thermodynamik. Springer, 1976.

[GDN10] Gausemeier, Jürgen; Dorociak, Rafal; Nyßen, Alexander: The mechatronic modeler: A Software Tool for Computer-Aided Modeling of the Principle Solution of an Advanced Mechatronic System. In: 11th Int. WS on Research and Education in Mechatronics. 2010.

[Ge20a] Gerasimov, Arkadii; Heuser, Patricia; Ketteniß, Holger; Letmathe, Peter; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Generated Enterprise Information Systems: MDSE for Maintainable Co-Development of Frontend and Backend. In: Comp. Proc. of Modellierung 2020 Short, Workshop and Tools & Demo Papers. CEUR, pp. 22–30, 2020.

[Ge20b] Gerasimov, Arkadii; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Continuous Transition from Model-Driven Prototype to Full-Size Real-World Enterprise Information Systems. In: 25th Americas Conference on Information Systems (AMCIS 2020). AIS, pp. 1–10, August 2020.

[Go21] Govindasamy, Hari Shankar; Jayaraman, Ramya; Taspinar, Burcu; Lehner, Daniel; Wimmer, Manuel: Air Quality Management: An Exemplar for Model-Driven Digital Twin Engineering. In: International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 229–232, 2021.

[HLM04] Hislop, Dave; Lacroix, Zoé; Moeller, Gerald: Issues in mechanical engineering design management. ACM SIGMOD Record, 33(2):135–138, 2004.

[Ho21] Hoepfner, Gregor; Jacobs, Georg; Zerwas, Thilo; Drave, Imke; Berroth, Joerg; Guist, Christian; Rumpe, Bernhard; Kohl, Jens: Model-Based Design Workflows for Cyber-Physical Systems Applied to an Electric-Mechanical Coolant Pump. In: IOP Conference Series: Materials Science and Engineering. volume 1097:012004. IOP Publishing, 2021.

[Ja14] Jassmann, U; Berroth, J; Matzke, D; Schelenz, R; Reiter, M; Jacobs, G; Abel, D: Model predictive control of a wind turbine modelled in Simpack. Journal of Physics: Conference Series, 524:012047, jun 2014.

[KK98] Koller, Rudolf; Kastrup, Norbert: Prinziplösungen zur Konstruktion technischer Produkte. Springer, Berlin, Heidelberg, 1998.

[Ku18] Kumar, Sathish; Madhumathi, R.; Chelliah, Pethuru Raj; Tao, Lei; Wang, Shangguang: A novel digital twin-centric approach for driver intention prediction and traffic congestion avoidance. Journal of Reliable Intelligent Environments, 4, 10 2018.

[Li19] Liu, Y.; Zhang, L.; Yang, Y.; Zhou, L.; Ren, L.; Wang, F.; Liu, R.; Pang, Z.; Deen, M. J.: A Novel Cloud-Based Framework for the Elderly Healthcare Services Using Digital Twin. IEEE Access, 7:49088–49101, 2019.

[Li21] Lipp, Johannes; Sakik, Siyabend; Kröger, Moritz; Decker, Stefan: LISSU: Integrating Semantic Web Concepts into SOA Frameworks. In: 23rd Int. Conf. on Enterprise Information Systems - Vol 1: ICEIS. INSTICC, SciTePress, pp. 855–865, 2021.

[LJW18] Lacal Arantegui, Roberto; Jäger-Waldau, Arnulf: Photovoltaics and wind status in the European Union after the Paris Agreement. Renewable and Sustainable Energy Reviews, 81:2460–2471, 2018.

[Mi19] Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Towards Privacy-Preserving IoT Systems Using Model Driven Engineering. In: MODELS 2019. Workshop MDE4IoT. CEUR Workshop Proceedings, pp. 595–614, 2019.

[MML19] Madni, Azad M.; Madni, Carla C.; Lucero, Scott D.: Leveraging Digital Twin Technology in Model-Based Systems Engineering. Systems, 7(1), 2019.

[Mo15] Moeser, Georg; Kramer, Christoph; Grundel, Martin; Neubert, Michael; Kümpel, Stephan; Scheithauer, Axel; Kleiner, Sven; Albers, Albert: Fortschrittsbericht zur modellbasierten Unterstützung der Konstrukteurstätigkeit durch FAS4M. In: Tag des Systems Engineering, pp. 69–78. Carl Hanser Verlag GmbH & Co. KG, 2015.

[MRV20] Michael, Judith; Rumpe, Bernhard; Varga, Simon: Human Behavior, Goals and Model-Driven Software Engineering for Assistive Systems. In: Enterprise Modeling and Information Systems Architectures (EMSIA 2020). volume 2628. CEUR Workshop Proceedings, pp. 11–18, 2020.

[Pa07] Pahl, Gerhard; Beitz, W.; Feldhusen, Jörg; Grote, Karl-Heinrich: Engineering Design - A Systematic Approach. Springer, London, 3 edition, 2007.

[To10] Tong, Wei: Wind Power Generation and Wind Power Design. WIT Press, 2010.

[Ve19] Veers, Paul; Dykes, Katherine; Lantz, Eric; et al.: Grand challenges in the science of wind energy. Science, 366(6464), 2019.

[Vi19] Vincent, Paul; Iijima, Kimihiko; Driver, Mark; Wong, Jason; Natis, Yefim: Magic quadrant for enterprise low-code application platforms. Gartner report, 2019.

[Wa19] Waszkowski, Robert: Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine, 52(10):376–381, 2019. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019.

[WS09] Wölkl, Stefan; Shea, Kristina: A Computational Product Model for Conceptual Design Using SysML. In: Int. Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2009.

[Ze21] Zerwas, Thilo; Jacobs, Georg; Spuetz, Kathrin; Hoepfner, Gregor; Drave, Imke; Berroth, Joerg; Guist, Christian; Konrad, Christian; Rumpe, Bernhard; Kohl, Jens: Mechanical Concept Development Using Principle Solution Models. In: IOP Conference Series: Materials Science and Engineering. volume 1097:012001. IOP Publishing, 2021.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 49

Decision-Making About Federated Digital Twins – How to Distribute Information Storage and Computing

Leif Bonorden[1], Marc Frerichs[2], Matthias Riebisch[1], Stephanie von Riegen[3], Florian Hartke[4], Rainer Herzog[3], Lothar Hotz[3], Dennis Jürgensen[2], Markus Kiele-Dunsche[5], Seeko Schottler[4], Rafael Schroeder[6]

Abstract: Digital Twins are commonly used as virtual representations of physical objects in manufacturing industries. Information for Digital Twins may be collected from multiple sources and stored in a distributed manner, leading to a Federated Digital Twin. Since decisions about such a federation are crucial for the system and its architecture, they should be guided by reliable and well-evaluated methods. However, current research is focused on distributed data sources but is missing decisions about the distribution of the digital twin itself. We present an approach to partition Federated Digital Twins by classifying information types, computing resources, and concerns of data suppliers. Furthermore, we show how decisions are made based on the Decision Model and Notation standard and evaluate the approach using an industrial case study.

1 Introduction

The idea of a digital twin (DT) addresses the need to digitally reflect the complete life cycle of physical objects, from planning to construction, operation and discontinuation. Different kinds of data, from design drawings and current runtime data to configurations and order data, can be accommodated in corresponding submodules [DI16, Pl18] of the DT. These data are then easily accessible for engineering, for example, to achieve more efficient utilization or optimize parameterization. The ADAM project[7] in the field of factory automation is concerned with the automated adaptation of production plants using a DT. In the project, concerns have arisen about the visibility and sharing of critical data such as configuration, production and order data. In addition, production plants produce a tremendous amount of data, the processing of which is intended as part of the digital transformation but is not easily possible with the resources of plants with some legacy elements. A monolithic DT is not sufficient to resolve these concerns, thus, federation of the DT is necessary.

[1] Universität Hamburg, Department of Informatics, {leif.bonorden, matthias.riebisch}@uni-hamburg.de
[2] Universität Hamburg, Hamburg Research Center for Information Systems, {marc.frerichs, dennis.juergensen}@uni-hamburg.de
[3] HITeC e.V., {stephanie.von.riegen, rainer.herzog, lothar.hotz}@hitec-hamburg.de
[4] encoway GmbH, DOCK ONE, {florian.hartke, seeko.schottler}@encoway.de
[5] Lenze SE, markus.kiele-dunsche@lenze.com
[6] Friedrich Remmert GmbH, rsc@remmert.de
[7] `https://uhh.de/adam-projekt`

doi:10.18420/modellierung2022-006

Digital twins are usually defined as virtual representations of physical objects that are continuously updated with changes in properties or behavior. In manufacturing industries, DTs are commonly used to monitor and optimize production processes, identify and mitigate unexpected events, and test and simulate conditions [Gr14, Se21, BCF19]. In particular, DTs tightly link virtual and physical counterparts by representing structures (e.g., subcomponents, physical properties), a current configuration (e.g., settings, firmware), and a current state (e.g., voltage, speed). In ADAM, a DT is available for demonstration purposes in a prototype of some components of the plant as well as for the whole plant. This DT contains, among other things, product information, dimensions, configuration information (e.g., settings, firmware), and even design data in some corresponding submodels.

However, this DT is built as a monolithic one, leading to three main negative concerns:

C1 data ownership, access rights, and intellectual property of all parties involved need to be preserved,

C2 computing power and storage capacity are limited in most units,

C3 the volume of communication data might exceed data transmission capabilities between units.

Therefore, a distribution into a federated DT (FDT) has to be performed. Even if this task is primarily concerned with the distribution of data, we are aware that this also leads to a distribution of data processing.

To generalize the decision-making process, this paper's contribution is a decision model (DM) for dividing a DT and distributing the FDT to computing units. The DM considers multiple goals and constraints, such as data ownership concerns, resource capacity constraints, and availability goals. The DM is formalized via the standard Decision Model and Notation (DMN). We organize our work along with the following research questions:

RQ1 Which criteria are relevant to split a DT?

RQ2 How can the DT parts be assigned to system parts?

RQ3 What is a general (formalized) decision process to distribute a DT?

The resulting DM has to fulfill the goals of clarity, understandability, mastering complexity, appropriateness, adaptability, and generalizability and transferability to other domains. The notation has to be flexible, understandable, and appropriate. Many of these goals demand an evaluation in a real-world context. We want to stress the point that there are two notions of the term *Model* in this paper: firstly, the DT represents a model of a system and its parts, and secondly, a DM is developed and discussed throughout this paper.

The remainder of this paper is structured as follows: After relating our approach to other works in Sect. 2, we introduce the case study in Sect. 3. In Sect. 4, we present the research

methodology Design Science Research and apply it to develop a decision model using the case study as an illustrative example. Subsequently, we evaluate the result in Sect. 5, and summarize the paper in Sect. 6.

2 Related Work

Eramo et al. consider a *digital twin* as a "virtual representation of an actual system" [Er22]. Thus, DTs are models themselves or contain models among data and possibly other information, e.g., historic information and predicted behavior [WD20, Er22]. Furthermore, five significant properties of DTs are commonly considered [Se21]: A DT mirrors the actual system's entire life cycle. A DT obtains and includes real-time data. A DT is synchronized with the existing system. A DT includes behavioral information about the existing system. A DT provides means for interaction.

The construction of a DT from multiple models and various data sources has been discussed in research, e.g. [Ta19, KMM20, Sc17], but usually these approaches aim to construct a single, monolithic DT. To the best of our knowledge, no attempts to split the DT into an FDT have been conducted.

Digital Shadows are a related concept for describing, abstracting, aggregating, and connecting data from Digital Twins [Be21], modeling a one-way data flow with the state of an existing physical object. However, since ADAM is supposed to influence the physical object automatically, this concept is not applicable here.

The integration of models from multiple engineering disciplines precedes the concept of DTs, e.g., with AutomationML, which is now commonly used in the implementation of DTs [Sc16, Se21].

To model our approach to FDTs, we use the *Decision Model and Notation* (DMN) standard [Ob21] because of its openness and wide acceptance. An alternative would be *The Decision Model* [vHG09]. However, it is owned by a specific vendor and subject to restricted distribution. All expressions are limited to using simple values or defining their syntax, unlike FEEL of DMN. Another alternative is given by the *IBM Operational Decision Manager* [IB22], but it requires a license and bears the risk of discontinued support by the vendor.

3 Case Study

We develop and evaluate the decision model in the factory automation domain, focusing on the construction and operation of a production system where different physical components are used, together with a DT. A significant use case consists of adaptations of a production system due to changes in products or plans, which demand changes in production systems,

each consisting of different machines, components, or configurations. These changes are triggered by changing requirements or sensor data, and they are driven by engineering recipes out of a solution library of a component provider or a plant manufacturer.

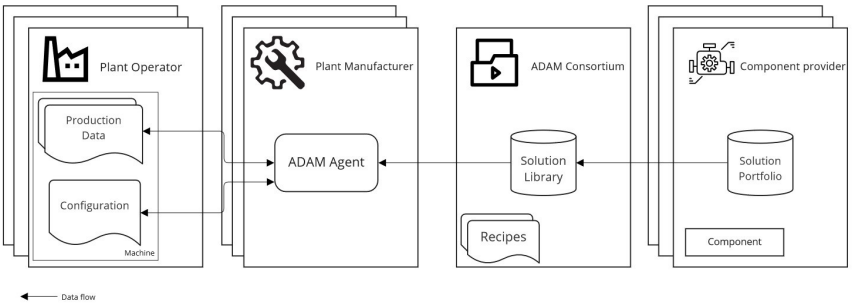


Fig. 1: ADAM stakeholders and structure

First, we introduce a typical use case, and then we give more details on stakeholders and structure, with Fig. 1 illustrating. As a use case for adaptation, we consider a production system in operation. Its configuration is set up for "sheet metal separation" consisting of components of different types like the transport units, e.g., conveyor belts, drive units, and sensors. After a new order has been accepted, the processed material changes to heavier metal sheets. This change demands higher force and load on some of the production system's components, e.g., some drive units must provide higher torque. Therefore, an adaption of the production system is needed to increase the plant's capabilities.

To perform this adaptation, the plant operator needs support from the plant manufacturer. However, the required higher torque demands a change of the drive provided by a component provider. This component provider gets the needed parameters and suggests a new drive solution consisting of an electrical drive with a parameter set. The suggestion by the component provider is developed using recipes. With a recipe, engineering expertise is made available for integrating the new solution into the production system. The ADAM agent (see Fig. 1) monitors sensor data. In a case of unsatisfied needs, and performs a lookup for suitable engineering solutions from plant manufacturers and component providers (solution libraries). For selecting appropriate recipes, configuration data of the production plant, the machines, and their components have to be assessed.

The plant and its machines produce a tremendous amount of sensor data, which leads to a high update frequency of data, which could cause limitations in transmission and processing resources in the different parts of the system, which refers to concerns C2 and C3 (see Sect. 1). These data could be filtered, not hampering their usage as a trigger for changes or adaptations. Moreover, the control computers' storage capacity and computing power in currently existing plants are mostly too small for extensive computing, e.g., by a DT, which would increase these concerns.

Further data is needed to perform the adaptation task as an ADAM result. For parts of them, some of the stakeholders have concerns regarding privacy, which refers to concern C1 (see Sect. 1). Information about the current drive units is held within the configuration model of the production system in the DT. Data about torque and speed is covered by the DT as well. The configuration model might be stored in the production system, but in our case, the plant manufacturer is concerned about their release as intellectual property and critical for competition. Order information such as scope and execution shall not be released to other orderers for competition and privacy reasons. Plant operators do not want to disclose their capacity utilization because it might affect negotiations about future contracts and pricing. Therefore, historical data must be considered particularly critically. Additionally, access to the production plant, the machines, and the plant control station from the outside is not desired to avoid disruptions to the production process. Recipes for drive application and parameterization are considered as engineering expertise, and therefore, as business secrets of the drive component provider.

4 Approach

The initial situation in the ADAM research project (see Sect. 3) is applied as a starting point to explain the decision-making and to elaborate the approach.

Initially, there is a prototype that visualizes and simulates components within parts of machines. A DT is used as the model driving visualization and simulation. Some components are even integrated as real hardware elements. The prototype is used for demonstration, e.g. in an exhibition. Some of the DT data are imported from engineering tools – e.g., for electrical drive configuration.

While the initial DT is a monolithic one, its transformation into a real plant is not an option because of the concerns introduced in Sect. 1. Therefore it has to be split and distributed, answering the three research questions (see Sect. 1).

For illustration purposes, Tab. 1 shows a separation into different operational units with respect to computing and aggregating data. The units correspond to the case study's architecture. The distinctions between the units that lead to this example separation are:
A – B: Memory limitation of the control computer.
B – C: Visibility of order data of different orderers to others.
C – D: Insufficient computing capacities for data analytics to compute adaptations.
D – E: Non-disclosure of business secrets of component providers, therefore filtering configuration data according to actual component installation at a plant.

4.1 Basic Methodology

Design Science Research (DSR) aims to develop new results (so-called artifacts) through a practice-oriented research approach and make them usable [He04]. The research presented

Tab. 1: Example separation between units

Unit and Location	Data	Computing	Part of DT
A machine	with process events, measured values for heat, torque, max rotation, speed of drives, error logs	preprocessing of data	production data, structure information, runtime data
B production system	history of process events, current order data	data aggregation	aggregated machine and current order data
C plant control station	history of order data	planning, monitoring, filtering, aggregation	scheduling of orders, capacity and utilization, error messages from production, maintenance requests, requests for change
D ADAM Agent	list of observed Trigger data sets, candidate solutions for alternative drive units	decision-making on adaptation with computing solutions	decision-making on adaptation, selection from a set of proposed adaption solutions
E drive (solution) provider	sample cases for application of drive units	filtering	set of proposed solutions for a variety of applications and configurations

follows the DSR methodology. Österle et al. [Ö11] suggest a framework that divides DSR into analysis, design, evaluation, and diffusion phases.

For the design of the proposed artifacts, the authors primarily used the design principles of induction [KGM12], and prototyping [NJ82]. In DSR, design principles (DPs) are "generic, high-level representations of the class of solutions addressing a class of problems" [Ko16], and they are derived from meta-requirements (MRs). The DSR methodology covers three essential steps of design principles induction: (1) deduce (meta-)requirements, (2) distill adequate DPs, and (3) formulate design decisions (DDs) as instantiations of DPs. Consequently, this leads to a one-to-many deduction from DPs to DDs [KGM12].

We use the Decision Model and Notation (DMN) [Ob21] standard to depict the decisions adequately. Published by the Object Management Group (OMG), this widely accepted and well-documented open standard creates the possibility of documenting decision rules. Furthermore, the adoption of this standard might increase applicability in other domains. Sect. 4.4 describes the adoption of DMN in more detail.

4.2 Eliciting Requirements for Decision-Making

We discuss the requirements according to the concerns **C1–C3** as introduced in Section 1.

C1 There are stakeholders with different roles such as plant operators, component providers, and plant manufacturers involved, each with their own concerns for their realm — especially about sharing relevant but critical data (process data, production data). The stakeholders are aware that they not only have to protect the data but also to release them partially to benefit from their processing. Therefore, this is a criterion for splitting the DT into partitions (RQ1). According to RQ2, the DT partitions have to be assigned to the realms of each stakeholder.

C2 There are computing devices with limited resources, namely network, storage capacity and computing power. This is especially urgent for legacy devices. The available resources of the components, machines, or system parts have to suffice the needs to process the amount of data. According to RQ1, the DT has to be split into partitions. These partitions are then assigned in a way that the needed resources does not exceed the available ones (RQ2).

C3 Network connections between parts differ according to speed, capacity and even robustness. For periodic updates there is a certain need for data transmission. The DT must not be split into partitions in a way that the needed transmission between them would exceed the available capacities (RQ1).

As the first step of design principles induction, meta-requirements have been collected based on the concerns mentioned before (C1–C3). Aiming at a DM for the distribution of a DT (RQ3), these meta-requirements need to be transferred into DPs, thus, representing another step of our DSR approach. The meta-requirements, together with the corresponding DPs, are given in Tab. 2.

Regarding meta-requirement MR1, derived from concern C1, it can be concluded that this requirement has a high priority because the respective owner only shares data if privacy is assured and/or if need and benefit are accepted. This is always a trade-off between risk and need for the data owner. To fulfill this requirement and keep the hurdle for sharing data low, sensitive data should remain locally with the owner. External access should only be possible via appropriate interfaces and following the granted access rights (DP1). Tab. 2 shows the DPs and the underlying MRs identified based on the concerns described above.

Meta-requirement MR2 addresses concern C2. We propose splitting the DT into subsystems when computing needs or storage capacity require it (DP2). To resolve conflicts of needs vs. resources, a proper assignment of tasks to the existing systems must also be achieved.

To fulfill meta-requirement MR3, which is based on concern C3, an appropriate computing and distribution setup is proposed: The update rate and data aggregation should be defined following the given resources (DP3). Furthermore, the amount of data requested should also be reduced according to the resources.

Tab. 2: Design principles and meta-requirements

	Design Principle	Meta-requirement
DP1	Make sure that the *sensitive data is located locally on the owner's system* — access only via *defined interfaces* and according to *granted permissions*.	*MR1* Data ownership, privacy, and intellectual property must be guaranteed at all times.
DP2	Use *subsystems* if the need for computing power or storage capacity requires it and *assign the tasks properly*.	*MR2* The computing power must be sufficient for the chosen purpose. The needs must be satisfied by the resources.
DP3	Use *appropriate computing and distribution setup*, *set update rates*, and *aggregate data* according to the given resources.	*MR3* Computing and distribution must be carried out even on (and for) systems with limited resources.

4.3 From Design Principles to Design Decisions

According to the DSR procedure, the next step is to convert the DPs into concrete DDs that form the basis for the FDT. Tab. 3 summarizes the DDs and their interrelation to the DPs from which they are derived.

Tab. 3: Design decisions derived from design principles

	Design Decision	DP(s)
DD1	Local data service accesses parts of DT	DP1
DD2	Interfaces with certified/trusted components	DP1
DD3	Data transmission in aggregated or anonymized form	DP1
DD4	DT parts access and accessed by distributed services/subsystems	DP1, DP2
DD5	Federated computing & storage	DP1, DP2, DP3
DD6	Proper assignment of tasks to subsystems	DP2, DP3
DD7	Caching & Queuing	DP3
DD8	Data aggregation & Sampling rate reduction	DP3

Based on DP1, we conclude that data services providing sensitive data should be designed to access local parts of the DT (DD1). Since the data exchange corresponds to an upstream-downstream relationship, it could—depending on the use case—be designed using the customer-supplier or conformist pattern [Ev04]. In addition, we propose the use of well-defined and purpose-driven interfaces that only grant specific permissions for data access to certified or trusted components (DD2). Thus, data ownership, privacy, and intellectual property protection are considered. Where applicable, the data transmission should occur in aggregated or anonymized form instead of raw data, thus, related to the principles of data economy and data adequacy (DD3). Machine operators and manufacturers have to perform a risk/benefit analysis which data should be shared.

The second meta-requirement, on which DP2 is based, addresses the different computing requirements and resources of the various system levels. DP2 proposes the use of subsystems and a proper assignment of tasks to the respective systems related to both RQ1 and RQ2. Thus, we conclude that the DT should consist of distributed services/subsystems (DD4) with an appropriate needs and resources balance (DD6). The requirement for federated computing and storage (DD5) is derived following DPs 1–3.

Caching and queuing (DD7) is proposed as a design decision related to DP3. These techniques, e.g., control load on the respective systems. Data aggregation and sampling rate reduction (DD8) are also means of adapting the amount of data to the resources of the respective systems and the actual data requirements. Message queues or producer-consumer patterns are proposed to accomplish distributed computing and storage, queuing, or aggregation since they can process, transform and distribute data streams.

Fig. 2 gives an exemplary overview of the application of design principles induction, starting with the three concerns up to derived design decisions.

	C1: Data Ownership & Data Privacy	**C2: Computing Power & Storage Capacity**	**C3: Volume of Periodic Updates**
Scenario	Solution providers want to remain data owners and only share specific data via agreed interfaces.	Processing of machine data requires high computing power, which is not always available on the machine itself.	The high amount of generated machine data must be processed, aggregated and distributed.
Meta-Requirement	Data ownership and privacy must be guaranteed at all times.	The computing power must be sufficient for the chosen purpose.	Data processing and distribution must be carried out even when there is a high volume of data.
Design Principle	Data that resides outside of the DT and is worth protecting remains encapsulated - access only via defined interfaces.	Adequately equipped subsystems based on computing power or storage capacity needs.	Use a scalable data processing and data transmission setup.
	Federated Digital Twin		
Design Decision	Local data service that is connected to the DT using the Customer-Supplier pattern.	DT parts access and are accessed by distributed services/subsystems.	Message Queuing / Producer-Consumer framework for data processing and transmission.

Fig. 2: Overview of applied design principles induction

4.4 Formalizing Decisions Using the Decision Model and Notation (DMN)

By formalizing the decision-making using the DMN standard [Ob21], we address RQ3: Based on the identified design decisions, a Decision Requirements Diagram (DRD) is elaborated that shows the relationships between the decisions to be taken into account when developing an FDT. Starting with the originally conceived concerns, the connections, influences, and interdependencies between decisions are visualized in the DRD. Decision tables represent details about these decisions.

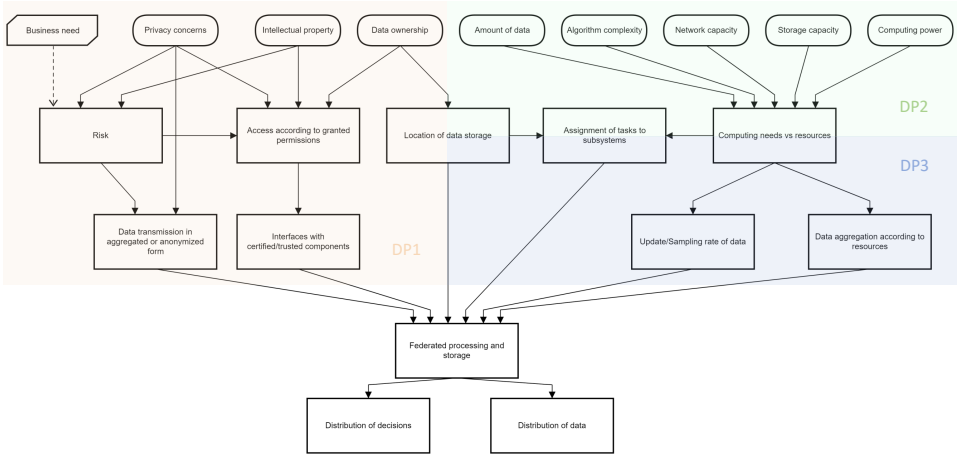


Fig. 3: Decision model represented as Decision Requirements Diagram (DRD)

For the notation, we chose DMN since it is an open standard for decision rules providing extensive specification and having wide acceptance, thus, offering the potential for generalization and transfer to other domains. One of the strengths of the DMN is the ability to show dependencies between decisions. DMN reduces complexity by decoupling decision and control flow logic [Fi18], i.e., decisions do not have to be depicted using process models. With the originally conceived goals and constraints (see Sect. 1), denoted as *DMN Input Data*, and the derived design decisions (shown as *DMN Decisions*), we created an exemplary DRD. Thus, we show the connection between decisions regarding the development of an FDT, and we give an example of how to use DMN for decision-making purposes for FDTs. According to the previous statements, *privacy concerns*, *intellectual property*, and *data ownership* (as input) can be assigned to DP1, thus influencing decisions concerning, e.g., data exposure (access, interfaces, ...). *Amount of data*, *algorithm complexity*, *network or storage capacity*, and *computing power* are inputs for decisions related to the trade-off between computing needs and computing resources. Both DP2 and DP3 derive from such considerations. Fig. 3 shows the resulting DRD.

Ultimately, all decisions affect *federated computing and storage* and potentially lead to the distribution of decisions and data. The DRD shows the interdependencies of the decision-making process very well. For example, if we consider *data ownership* as a goal that influences the decision regarding the *location of data storage*, which itself references the decision about the *assignment of tasks to subsystems* in the DT. At the same time, the latter could also be influenced by a decision about *computing needs vs. resources*.

When focusing on specific goals, we propose using *DMN Decision Services*. Even if human

experts perform our decision-making, Decision Services are suitable for encapsulating decisions. For example, let us consider the case of data allocation (i.e., data access and data distribution). It could be designed in the way as shown in Fig. 4: there are two decisions, of which the first is about the data access policy. The inputs for this decision are the data owner's *privacy concerns* denoted as DMN input. Furthermore, there are several stakeholders in competition with each other. The decision about data access directly influences the decision about data separation. When deciding about data separation, we could also consider existing knowledge about the design of distributed systems. This knowledge is presented as tactics by [BCK21], represented as *DMN Business Knowledge*). The use of decision services can be applied for each goal, e.g., related to ownership/privacy, resources (computing power, storage space, network capacity), or availability (e.g., redundancy).

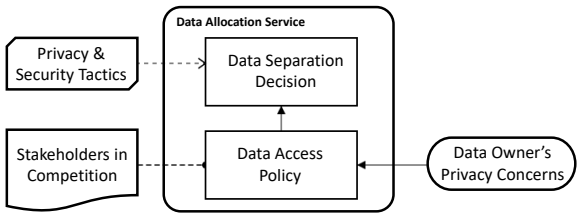


Fig. 4: Privacy decision as DRD

For representing decisions in detail, decision tables proved to be an appropriate way of formalization. As DMN also supports them, we propose decision tables to denote specific decisions in case of conflicts and competing goals. As an example, we examine the case of a plant operator (PO) considering providing data access for ADAM (see Fig. 5). The decision table represents the negotiation of a PO with the ADAM consortium. The input values (blue) are the goals of the negotiating partner, while the output values (red) contain options for action.

PO privacy risk	acceptable delay for ADAM	data history needed by ADAM	data access granted by PO	measures due to PO's privacy concerns	measures for data accuracy in ADAM	arguments in negotiation, comments on decision
high	none - required for immediate response	no	Push after demand by ADAM or triggered by PO	only for immediate evaluation by triggers or rules, no storage within ADAM	none	PO needs ADAM's urgent response
high	some delay accepted - no decision without access	no	Push as cyclic transmission, max delay = cycle time	no history is kept	Caching of most recent value	PO gets better results or decisions when new data is provided
medium	helpful for better decision - in case of missing access calculation with recent data	yes	Push as cyclic transmission	none	Caching of most recent value	
low	don't care	don't care	Pull	not relevant	not defined	Pull access is preferred by ADAM if granted by PO

Fig. 5: Data access decision table: ownership concerns vs. ADAM's data needs

According to DMN, decisions, as well as business knowledge, can be expressed using decision tables. As described above, such knowledge could be, e.g., existing knowledge about the decomposition and distribution of systems. For example, we consider a decision about computing at the current location (on-site). In this case, we could rely on (business) knowledge of resource-driven decomposition, i.e., the decision if a computation is executed on-site is dependent on the resources available; see Fig. 6.

Required data are completely available on site	Required data can be received in time (network capacity)	Storage capacity available	Processing power available	Computing and decision-making on site	Data aggregation on site
y	don't care	y	y	y	y
y	don't care	y	n	n	y
n	y	y	y	y	y
n	y	y	n	n	y
n	n	don't care	don't care	n	n
don't care	don't care	n	don't care	n	n

Fig. 6: Decision table: business knowledge on resource-driven decomposition

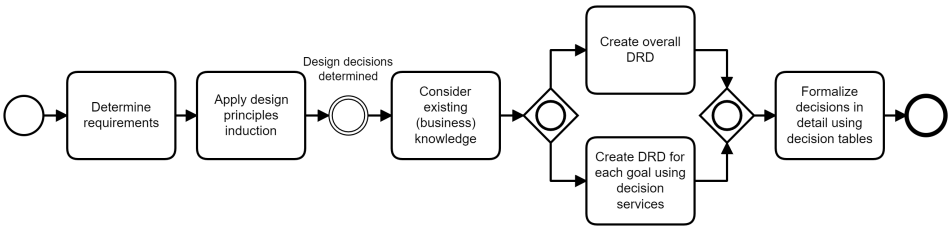


Fig. 7: Decision-making process

In summary, the application of the proposed approach for decision-making about FDTs, especially for other projects and use cases, can be realized as follows (also see Fig. 7):

1. Determine requirements and goals to be considered
2. Apply design principles induction to determine the resulting decisions
3. Consider existing knowledge (to not reinvent the wheel)
4. Create DRD as
   a) Overall DRD based on design principles (see Fig. 3) or
   b) DRD for each decision, use decision services if applicable (see Fig. 4)
5. Formalize decisions and business knowledge in detail using decision tables (see Fig. 5 and Fig. 6)

5 Results and Evaluation

Evaluation of Design Science artifacts is an essential part of DSR [Pe12, GH13]. As mentioned in Sect. 4.1, an evaluation in a real-world context is required to assess our approach, i.e., an evaluation in a real environment is mandatory—also known as *naturalistic evaluation* [VPHB12]. The practitioners involved in the ADAM project formed a *focus group* to examine the perception of experts in the field [Bu10]. To evaluate our approach, we applied it to the already existing implementation in the ADAM project. In the following, we describe how the design decisions toward an FDT, based on the previously presented approach, affected the implementation of the ADAM project. The overall architecture resulting from the application of the decision process comprises the units given in Tab. 1: the machine and production system, the plant control station, the ADAM agent, and solution libraries (DD4).

The individual **machines** provide their configuration and runtime data from the DT via the MQTT protocol. To accumulate runtime data over a period of time, a so-called data aggregator for the **production system** has been implemented in an integrated time-series database InfluxDB. The aggregator provides average data over a timeframe but withholds more privacy-critical event data. In the **plant control station**, these data are accepted and checked against triggers for adaptation (DD1) by a rule engine. It uses rules to determine if the described circumstances occur (DD7). If rules apply, this is passed to the **ADAM agent**, thus withholding the actual machine data (DD3). The plant operator continuously controls the machine data and only provides abstract and summarized data to the ADAM agent (DD8). As a result, the amount and frequency of data transfer are reduced, and data computing and storage are distributed (DD5).

In the **ADAM agent**, an adaption, as well as a monitoring component, is defined as subcomponents that communicate via Apache Kafka. The latter was introduced because of the amount and frequency of data (DD6). In the adaptation component, possible adaptations to the trigger are compiled. A detailed description of the adaptation and adjustments—a 'recipe'—is used for this purpose.

Furthermore, verified **solution libraries** from different drive (solution) providers can be addressed to retrieve updated component information suitable for the specific production system (DD2). These libraries are located at the drive (solution) providers because they contain sensitive data that must be protected and considered the providers' intellectual property. Possible simulation and a knowledge-based monitoring component in the agent can warn against unnecessary adaptations. Proposed adaptations are presented to the plant operator, who might select the most suitable one. Because of the risk of interventions in the production process, downtime, and costs, decisions have to be approved by a responsible person. After the adaptation is approved and implemented, the machine configuration is updated in the DT.

The resulting prototype implementation has successfully identified adaptation scenarios

using information from multiple parts of the FDT. It has determined suitable solutions (e.g., changes in software configuration, hardware replacement) using the solution libraries. Additionally, the experts have evaluated the approach and the solution: The decision process was comprehensible and the result feasible. Furthermore, applying a rigorous decision process leads to increased trust in the result and better acceptance of connecting the systems and sharing sensible data.

We conclude that the DDs applied for splitting and distributing the DT are suitable and thus answer **RQ1** (Which criteria are relevant to split a DT?) and **RQ2** (How can the DT parts be assigned to system parts?). Moreover, we presented the decision process using these criteria in response to **RQ3** (What is a general (formalized) decision process to distribute a DT?).

Finally, we relate the results to the overall goals from Sect. 1: The application in a real-world case study with a high level of complexity and the evaluation with experts have shown the approach to be appropriate for the given context, to provide *clarity* and *understandability*, and to *master complexity*. Furthermore, by providing a general DM and using a standard notation, the approach offers variability allowing *adaptation* in other use cases as well as *transferability* to other domains. In particular, it may be applied with different structures and domains, changed priorities, and design principles.

6 Conclusion and Future Work

In this paper, we presented a DM for splitting and distributing the DT. We evaluated the approach in the ADAM project, which is relevant for cases when a DT cannot be built as one single element in one place, e.g., due to insufficient resources or missing data availability. Therefore, the DM supports splitting a DT into an FDT and assigning FDT parts to the different computing units. Furthermore, the DM considers the issues and constraints regarding distribution, such as resources, privacy, availability, and similar. The DM has been developed following the DSR methodology and uses the standard DMN as notation. The DM and its application have been illustrated with a case study on factory automation. Practitioners evaluated it in its application in an industrial project. Data ownership issues and intellectual property and privacy concerns turned out to be of particular relevance in this project, and they required great attention in elicitation and decision-making.

In future work, we plan to research how extensions of a distributed DT can be performed that lead to meta-model changes, i.e., considering added parameters for a component. Synchronization of partly overlapping decision models constitutes another critical issue. Furthermore, research is necessary regarding optimal decisions that demand global optimization. Especially the optimization in the case of distributed data is an unsolved question. Moreover, reusable solutions within the solution portfolio need to be improved and optimized by learning from results, which demands verification of the improved solutions.

Acknowledgements

This research was partly funded by the German Federal Ministry of Education and Research (BMBF) within the project *ADAM – Autonomously aDApting Machines* (program *IKT 2020*, reference number *01IS18077A*).

Bibliography

[BCF19] Barricelli, Barbara Rita; Casiraghi, Elena; Fogli, Daniela: A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications. IEEE Access, 7:167653–167671, 2019.

[BCK21] Bass, Len; Clements, Paul C.; Kazman, Rick: Software Architecture in Practice. Addison-Wesley Professional, 2021. Fourth Edition.

[Be21] Bergs, Thomas; Gierlings, Sascha; Auerbach, Thomas; Klink, Andreas; Schraknepper, Daniel; Augspurger, Thorsten: The Concept of Digital Twin and Digital Shadow in Manufacturing. Procedia CIRP, 101:81–84, 2021. 9th CIRP Conference on High Performance Cutting.

[Bu10] Burgess, Stephen: The Use of Focus Groups in Information Systems Research. International Journal of Interdisciplinary Social Sciences, 5:57–68, 01 2010.

[DI16] DIN SPEC 91345: Reference Architecture Model Industrie 4.0 (RAMI4.0). Technical report, DIN Deutsches Institut für Normung e. V., Berlin, Germany, 2016.

[Er22] Eramo, Romina; Bordeleau, Francis; Combemale, Benoit; Brand, Mark van den; Wimmer, Manuel; Wortmann, Andreas: Conceptualizing Digital Twins. IEEE Software, 39(2):39–46, 2022.

[Ev04] Evans, Eric: Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, 2004.

[Fi18] Figl, Kathrin; Mendling, Jan; Tokdemir, Gul; Vanthienen, Jan: What we know and what we do not know about DMN. Enterprise Modelling and Information Systems Architectures, 13(2):1–16, 2018.

[GH13] Gregor, Shirley; Hevner, Alan R.: Positioning and Presenting Design Science Research for Maximum Impact. Management Information Systems Quarterly, 37(2):337–356, 2013.

[Gr14] Grieves, Michael: , Digital Twin: Manufacturing Excellence through Virtual Factory Replication. White paper, 2014.

[He04] Hevner, Alan R.; March, Salvatore T.; Park, Jinsoo; Ram, Sudha: Design Science in Information Systems Research. Management Information Systems Quarterly, 28(1):75–105, 2004.

[IB22] IBM: , Operational Decision Manager. online, 2022.

[KGM12] Koppenhagen, Norbert; Gaß, Oliver; Müller, Benjamin: Design Science Research in Action – Anatomy of Success Critical Activities for Rigor and Relevance. In: 20th European Conference on Information Systems. 2012.

[KMM20] Kaur, Maninder Jeet; Mishra, Ved P.; Maheshwari, Piyush: The Convergence of Digital Twin, IoT, and Machine Learning: Transforming Data into Action. In (Farsi, Maryam; Daneshkhah, Alireza; Hosseinian-Far, Amin; Jahankhani, Hamid, eds): Digital Twin Technologies and Smart Cities. Springer International Publishing, Cham, pp. 3–17, 2020.

[Ko16] Koppenhagen, Norbert; Mueller, Benjamin; Maedche, Alexander; Li, Ye; Hiller, Stephanie: Designing a supply network artifact for data, process, and people integration. Information Systems and e-Business Management, 14:613–636, 2016.

[NJ82] Naumann, Justus D.; Jenkins, A. Milton: Prototyping: The New Paradigm for Systems Development. Management Information Systems Quarterly, 6(3):29–44, 1982.

[Ob21] Object Management Group: , Decision Model and Notation (DMN), 2021. Version 1.3.

[Pe12] Peffers, Ken; Rothenberger, Marcus; Tuunanen, Tuure; Vaezi, Reza: Design Science Research Evaluation. In (Peffers, Ken; Rothenberger, Marcus; Kuechler, Bill, eds): Design Science Research in Information Systems. Advances in Theory and Practice. Springer, pp. 398–410, 2012.

[Pl18] Plattform Industrie 4.0: . Reference Architectural Model Industrie 4.0 (RAMI4.0) – An Introduction, 2018.

[Sc16] Schroeder, Greyce N.; Steinmetz, Charles; Pereira, Carlos E.; Espindola, Danubia B.: Digital Twin Data Modeling with AutomationML and a Communication Methodology for Data Exchange. IFAC-PapersOnLine, 49(30):12–17, 2016. 4th IFAC Symposium on Telematics Applications TA 2016.

[Sc17] Schleich, Benjamin; Anwer, Nabil; Mathieu, Luc; Wartzack, Sandro: Shaping the digital twin for design and production engineering. CIRP Annals, 66(1):141–144, 2017.

[Se21] Semeraro, Concetta; Lezoche, Mario; Panetto, Hervé; Dassisti, Michele: Digital twin paradigm: A systematic literature review. Computers in Industry, 130, 2021.

[Ta19] Tao, Fei; Zhang, He; Liu, Ang; Nee, A. Y. C.: Digital Twin in Industry: State-of-the-Art. IEEE Transactions on Industrial Informatics, 15(4):2405–2415, 2019.

[vHG09] von Halle, Barbara; Goldberg, Larry: The Decision Model – A Business Logic Framework Linking Business and Technology. Auerbach Publications, 2009.

[VPHB12] Venable, John; Pries-Heje, Jan; Baskerville, Richard: A Comprehensive Framework for Evaluation in Design Science Research. In (Peffers, Ken; Rothenberger, Marcus; Kuechler, Bill, eds): Design Science Research in Information Systems. Advances in Theory and Practice. Springer, pp. 423–438, 2012.

[WD20] Wright, Louise; Davidson, Stuart: How to tell the difference between a model and a digital twin. Advanced Modeling and Simulation in Engineering Sciences, 7, 2020.

[Ö11] Österle, Hubert; Becker, Jörg; Frank, Ulrich; Hess, Thomas; Karagiannis, Dimitris; Krcmar, Helmut; Loos, Peter; Mertens, Peter; Oberweis, Andreas; Sinz, Elmar J.: Memorandum on design-oriented information systems research. European Journal of Information Systems, 20(1):7–10, 2011.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 65

Modeling the Enterprise Digital Twin: Towards an Open Platform for Analytics & Compliance Operations

Marc Frerichs[1] Markus Nüttgens[1]

Abstract: With its origins in the manufacturing and production sector, the concept of the Digital Twin has now become firmly established. Companies are increasingly relying on the potential advantages of operating a digital representation of their corporate structure with the Digital Twin of an Organization. This is done by incorporating relevant dynamic and operational data on the slower-changing structural data of traditional business. This paper deals with the question of how a digital twin can be designed and actually implemented, particularly taking into account data and processes extracted from business information systems. A real, implemented use case is used to show how operational decision-making can be supported by a digital twin and what role the concept of Hybrid Intelligence plays in this context. The findings of this work are the basis for an open platform for business analytics and resulting (semi-)automated actions, e.g., for implementation of compliance operations such as double payment analysis.

Keywords: Enterprise Digital Twin; Digital Twin of an Organization; Business Analytics

1 Introduction

According to Gartner, the concept of the Digital Twin (DT) is one of the most emerging innovations ("innovation profiles") [Da21]. DT is the most widely used innovation profile, showing the need for organizations to consider the impacts of their innovations. While DTs have been firmly established in the domain of manufacturing, companies are increasingly relying on the potential advantages of operating a digital representation of their corporate structure with the Digital Twin of an Organization (DTO). However, the creation of a DTO is a grand challenge [AHW21; Ca20], e.g., because the boundaries of an organization are not clear. In this paper, we propose the notion of the so-called Enterprise Digital Twin (EDT)—an approach for a generic/reference architecture to support the creation of DTOs in an enterprise context, i.e., especially aiming for organizations using business information systems such as Enterprise Resource Planning (ERP) systems. Furthermore, we conduct a case study to show feasibility of the approach by developing and using a software prototype. In this work, we address the following research questions:

RQ1: How can an EDT architecture be designed? Using Design Science Research (DSR), especially design principle induction, we derive a generic system architecture representing fundamental parts relevant to the realization of a DTO.

[1] Universität Hamburg, Hamburg Research Center for Information Systems (HARCIS), {marc.frerichs, markus.nuettgens}@uni-hamburg.de

doi:10.18420/modellierung2022-008

RQ2: How can a DTO based on the EDT architecture be implemented? By conducting a case study, including the development of a software prototype, we show how a DTO can be implemented on the basis of the EDT architecture.

The paper is structured as follows: First, the background and related work is discussed briefly. Afterwards, the methodology is described. The next section deals with the development of the EDT architecture. This is followed by a case study in the context of double payment analysis. The paper closes with a conclusion and outlook.

2 Background

DTs [Gr14] have emerged as a popular research field and also seem to raise high expectations in practice. A high level of relevance can be observed above all in the area of manufacturing, but also in aviation and health care [BCF19]. A DTO [KK21] aims to represent the elements and connections of an organizational system in virtual models in order to continuously simulate and analyze the behavior of the organization [Ca20]. *Process Mining* [Aa16] can help to facilitate such a DTO by providing methods to discover the control-flow model [PA21]. Van der Aalst et al. propose *Hybrid Intelligence* (HI)—"the ability to achieve complex goals by combining human and artificial intelligence" [De19b]—to make DTOs more resilient to unprecedented situations, and thus coined the term *Resilient Digital Twin* [AHW21]. To the best of our knowledge, existing approaches in the context of generic or reusable DT architectures primarily focus on industrial cases like Cyber-Physical Production Systems or Industry 4.0 [Bo21].

3 Methodology

The research presented follows the research approach of Design Science Research (DSR), which aims to develop artifacts that contribute to the solution of an existing problem [He04]. Österle et al. [Ös11] suggest a framework that divides DSR into the phases analysis, design, evaluation, and diffusion. Accordingly, our paper is structured corresponding to these phases (Fig. 1). For the design of the proposed artifacts, we used design principles induction [KGM12] to determine design principles (DPs) as the basis for an EDT architecture, and prototyping to develop a software artifact that implements the intended core functionality [NJ82]. DPs are generic, high-level representations derived from (meta-)requirements (MRs) [Ko16]. After analyzing existing knowledge regarding the realization of DTs in an enterprise context, we distill challenges and requirements. From these requirements, we derive DPs, which are the basis for a generic, high-level EDT architecture. For the implementation of a prototype, design decisions (DDs), i.e., instantiable and tangible features, are deduced. Several scholars highlight the importance of rigorous evaluation in design science-oriented research [RSB09; VPB12]. We therefore evaluate the proposed notion of an EDT by implementing a working prototype and validate it against a real-world scenario in a case study.

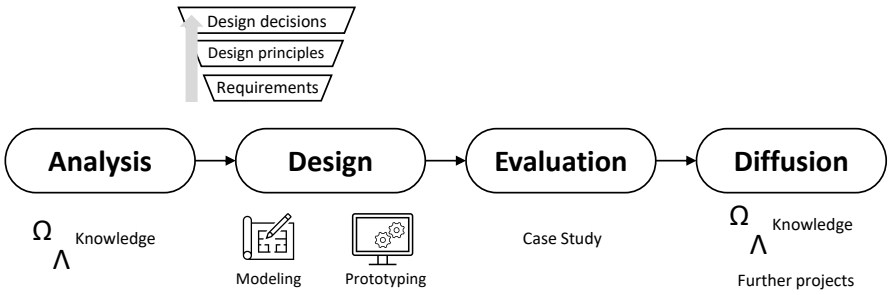


Fig. 1: DSR phases applied in this paper

4 Enterprise Digital Twin

Approaching a characterisation of DTs in general, different definitions, challenges, requirements, or properties can be named [Jo20; Se21]. Arguably the lowest common denominator in defining the concept of DT is that it is a digital representation of a real-world entity or system with bi-directional data connections [Gr14]. One area of application that has gained importance in recent years is the representation of an organization using DTs, also called DTO. Creating a DTO is one of the grand challenges in information systems [AHW21; Ca20]. With our work we want to narrow down the concept of a DTO a little and establish the notion of the EDT. The EDT provides a basic framework for realization of a DTO based on data extracted from enterprise information systems and intends to provide orientation when developing such DTOs. The approach is independent of a specific application scenario and instead provides a reference for the implementation of a concrete DTO. In this section, we first examine challenges and requirements, after which meta-requirements are derived. We then induce generic DPs and propose an architecture that corresponds to these principles. The EDT architecture and respective DPs are the basis for the deduction of concrete DDs.

4.1 Challenges and Requirements

Based on the definition by Caporuscio et al. [Ca20], essential aspects of a DTO can be inferred: DTOs should be able to *represent all elements and connections of an organization* in virtual models, which can be perpetually simulated and analyzed to achieve *continuous assessment* and *optimization of the organization*. Van der Aalst [Aa21a] derives an abstract definition of a DT by describing the differences between a digital model, a digital shadow, a DT, and a resilient DT. By introducing resilient DTs, Van der Aalst et al. [AHW21] complement the three definitions with a type of DT in which HI [De19a] comes into play. The complementary strengths of HI are presented in two scenarios [Aa21b; De19b]: (1) AI in the loop of human intelligence—support in decision-making by humans using AI, and (2) human intelligence in the loop of AI—humans continuously train machine learning models

to improve the results of the AI. Becker and Pentland [BP22] state that organizations involve agents, typically human, with the ability to make choices and learn from experience. The bi-directional data connection between DT and reality is also referred to as twinning [Jo20]. To ensure twinning, continuous data updates are necessary. Furthermore, both historical and current data is needed as a basis of a DT [AHW21]. Thus, for the operation of an EDT, both initial (for historical data) and continuous data transfer (for current/real-time data) are required. With regard to data extraction, especially in terms of process mining, but in data-driven projects in general, scholars state that about 80% of the time is spent with ETL-related activities such as data access or data preparation [Sa21]. We therefore deduce that this problem must also be dominant in the context of DTs. When it comes to relevant data for an EDT, it must be specified how the state and behavior of information systems are represented in an EDT. According to Polyvyanyy and Van der Werf [PW20], data and processes in information systems go hand in hand. Becker and Pentland [BP22] state that organizations include multiple, interdependent processes. We deduce that both data—in business information systems one often speaks of documents, business objects or events—as well as different types of processes must be mapped in an EDT. As stated above, the EDT approach is "use-case agnostic". For example, Park and Van der Aalst [PA21] propose triggers based on constraint checking in their approach based on action-oriented process mining. While this is practical for this particular use case, it may not be for another purpose. We therefore expect an EDT to be open to (highly specialized) tools that are most suitable for the respective use case [FLN21]. Based on the previous statements, we derive key challenges and requirements to be considered when designing the EDT (Tab. 1).

Tab. 1: Key challenges and requirements

	Key challenge/requirement	References
(a)	Represent all elements and connections of an organization	[Ca20]
(b)	Continuous assessment and optimization of the organization	[Aa21a; AHW21; Ca20]
(c)	Twinning / bi-directional data connection	[Aa21a; Gr14; Jo20]
(d)	Data and processes as basic representation of the organization or underlying information system	[BP22; PW20]
(e)	Historical and current data	[AHW21; Ca20]
(f)	Combination of human and machine intelligence	[Aa21b; AHW21; BP22; De19a; De19b]
(g)	Tool-agnostic EDT	[FLN21; Ri05]

4.2 Design of the EDT

According to the methodology described in Sect. 3, the next step is to derive meta-requirements that aggregate the essential requirements of an EDT. Afterwards, design principles are induced, which are used for the design of an EDT architecture.

Meta-Requirements Aggregation. We cluster key challenges (a) and (d) as first meta-requirement (MR1) since they both address the representation of the organization in the EDT. (b), (c) and (e) affect (continuous) data transfer between "real" (in this case one or more information system(s)) and virtual entity (EDT), considering both historical and current data, thus leading to meta-requirement MR2. (f) addresses the involvement of humans in the EDT, significantly inspired by the concept of HI and resilient DTs, resulting in meta-requirement MR3. The idea of a (method- and) tool-agnostic EDT (g) is a result of the generic notion of the EDT, and forms the last meta-requirement (MR4).

Tab. 2: Design principles, meta-requirements and key challenges

	Design principle	Meta-requirement	Addresses key challenge(s)
DP1	Consider data and processes on different abstraction levels	*MR1* Adequate representation of the organization with data and processes	a, d
DP2	Use data exchange interfaces supporting batch and continuous procedure calls	*MR2* (Continuous) data transmission, considering both historical and current data	b, c, e
DP3	Enable 1) fully manual, 2) fully automated, and 3) hybrid decision-making	*MR3* Decision-making including Hybrid Intelligence	f
DP4	Provide adequate generic interfaces for tools to access business and process data	*MR4* Method- and tool-agnostic EDT	g

Design Principles Induction. In the following, the induction of DPs is described. They provide the basis for the design of the EDT architecture. Since the design of the EDT should consciously not be dependent on a specific use case, we propose a representation of the processes on different levels of abstraction, i.e. instance and model level [KK02; We12]; addressing MR1. On the one hand, one would want to investigate single executions of a case, e.g., for purposes of compliance checking [We17], while on the other hand an aggregated ("managerial") level of abstraction, e.g., for calculation of performance measures, is needed [DA05]. Consequently, this leads to design principle DP1, which aims for consideration of data and processes on different abstraction levels. To fulfill meta-requirement MR2, we propose an EDT with interfaces to the respective information systems (in and out), capable of both batch (for historical data) and continuous data transmission (DP2). Meta-requirement MR3 deals with HI to involve humans, either as part of decision-making (AI in the loop of human intelligence) or to train AI (human in the loop of AI). However, since a DT must still be able to make autonomous decisions [Aa21a], there should be several options for decision-making or "intervention in reality". We therefore propose 1) fully automated decision-making, 2) fully manual decision-making by humans, and 3) decision-making using HI, thus defining DP3. Next to meta-requirement MR2, the idea of a method- and

tool-agnostic EDT, as described in meta-requirement MR4, is still more of a challenge than a requirement. A method- and tool-agnostic EDT should at least provide reasonable generic interfaces so the respective tools can benefit from the data and processes available in the EDT (DP4). However, the continuous development of a number of special interfaces tailored to the respective tools would be better. For example, this would be conceivable in an open source setting. Tab. 2 shows the derived DPs, the underlying meta-requirements, and key challenges.

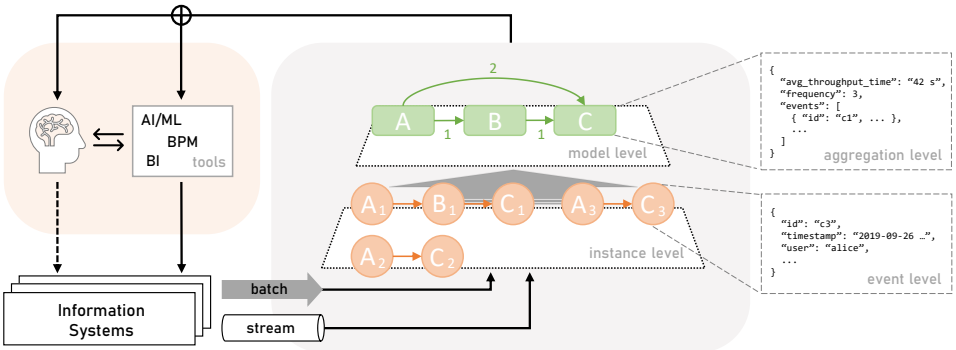


Fig. 2: Enterprise Digital Twin

Modeling the EDT. Based on the DPs, we create a generic, high-level architecture of the EDT. It essentially consists of three parts: (1) the EDT itself, (2) a human-in-the-loop and tool part, and (3) the information system(s) representing the organization. As DP1 prescribes, the EDT should contain different abstraction levels so that different needs are addressed. Process instances are stored on an instance level, with events containing detailed information, e.g., a document containing a document ID or name of the responsible user. These process instances can be aggregated (model level), i.e., a process model is created, with activities containing aggregated data such as frequency or throughput time. The EDT is fed with data from the information system(s). This is possible via batch ("one-time shot" with potentially high data volume) and/or event stream (data is produced and transferred continually). In Fig. 2, this is indicated by two parallel processing pipelines (DP2). The "way back" to the information systems is accomplished via tool integration (DP3). DP3 and DP4 are realized via what we called human-in-the-loop and tool part. As described in Sect. 4.1, decision-making (and the like) should not be integrated into EDT, but outsourced to tools. This prevents us from reinventing the wheel (just for the sake of integrating it into the EDT) and leaves this job to tools that are more suitable. Since humans can also be involved in decision-making, the human-in-the-loop was integrated in the architecture. In this case, the human interacts with the tool(s) such as an AI or a business intelligence tool. Furthermore, the architecture also allows only the human (i.e., fully manual; e.g., by taking an action via the user interface of an information system) and only the machine/tool (i.e., fully automated; e.g., via a system interface such as BAPIs[2] in SAP) to make decisions.

[2] BAPI: Business Application Programming Interface; a standardized API for SAP business objects

Tab. 3: Design decisions derived from design principles

		DP(s)
DD1	Property graph as target data structure for process instance graphs using a graph database	DP1
DD2	Instance aggregation algorithm for construction of process models	DP1
DD3	Data extraction component allowing batch and continuous extraction of accounting data from an ERP system	DP2
DD4	Bulk importer for initial batch imports; message queue for processing data streams	DP2
DD5	REST interface for accessing business and process data	DP3, DP4

5 Evaluation

The goal of the case study is the concrete application of the design principles and the resulting reference architecture for EDTs as a basis for the development of a software artifact. The prototype created is used to analyze double payments in companies. A double payment is the case when a company pays the full equivalent more than once for a delivery or service [BRW11]. Obviously, double payments pose risks for businesses as they increase costs and reduce profits. The prototypical implementation is based on concrete, implementable DDs derived from the DPs (see Tab. 3).

With SAP S/4HANA, we chose a widely used ERP system as "object of investigation". For data extraction, a Java-based component was developed, supported by the SAP Java Connector[3] (DD3), which provides an API that enables Java applications to communicate with SAP systems via SAP's Remote Function Call (RFC) protocol. For continuous data streaming, Apache Kafka[4] (and Confluent[5]) is used (DD4). Neo4j[6] serves as a graph database for the persistence of the process instance graphs (DD1). Batch import of historical data is accomplished via the Neo4j Bulk Importer[7] (DD4). Aggregation of instances to process models is performed by a Java-based tool that can generate directly-follows graphs (DD2) and provides a REST API (DD5) for business data (document/event level) and process data (event log).

For double payment analysis, mostly journal entry documents are needed. In SAP, journal entries are stored in table BKPF; journal entry items can be found in table BSEG. Fig. 3 shows the data extraction part of the prototypical implementation described above, i.e., the path from the SAP source system to the process instances in Neo4j.

[3] https://support.sap.com/en/product/connectors/jco.html
[4] https://kafka.apache.org/documentation/
[5] https://docs.confluent.io/platform/current/quickstart/ce-docker-quickstart.html
[6] https://neo4j.com, https://github.com/neo4j-contrib/neo4j-streams
[7] https://neo4j.com/developer/guide-import-csv/#batch-importer

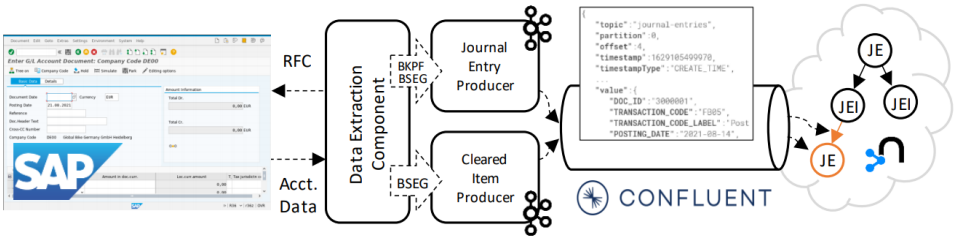


Fig. 3: Part of the prototypical implementation

Our EDT-based application supports the user in detecting double payments. An AI-based tool (an external tool in the sense of the EDT) suggests potential double payments to the user. This is done in form of a list of double payment candidates, each associated with a probability of being a double payment. Based on their (business) experience, the user can assess and judge whether it is a double payment. By rating a double payment case, the user not only trains the AI and thus continuously improves it, but also manages the double payment cases. Since this involves interventions in a critical business system, a reclaim from the supplier or the cancellation of completed double payments does not take place automatically, but through manual actions by the user. The interaction of the user with the AI (training/judgement) and the decision-making support by the AI are good examples of human-in-the-loop. By rating individual candidates, the AI model adapts and with it the probability of the cases. The case study showed that an implementation based on the EDT architecture and the underlying design principles is feasible. Furthermore, a concrete application of the developed prototype was shown.

6 Conclusion and Outlook

In this research paper we showed how DTs can be realized based on a generic EDT architecture. Based on the initial requirements, design principles were determined using design principles induction, which served as the basis for the EDT architecture (RQ1). Using the proposed architecture and the design decisions derived from the DPs, we created a prototype and implemented a real use case (RQ2). Obviously this paper is not without limitations and provides a first step towards a generic and high-level, tool-agnostic EDT. One of the probably biggest challenges was already mentioned in Sect. 4.2: the data exchange with information systems is difficult to generalize, since each system has different, often proprietary interfaces [FLN21]. The same applies to the interfaces of the tools. This is where the OpenPACO (Open Platform for Analytics & Compliance Operations) project comes into play. It aims to close the gap between business information systems such as ERP systems and analytics or modeling tools. The goal is an open-source platform that can be constantly extended with new interfaces to source and target systems. This paper forms a basis for further activities in this context.

References

[Aa16] van der Aalst, W. M. P.: Process Mining. Springer Berlin Heidelberg, 2016.

[Aa21a] van der Aalst, W. M. P.: Concurrency and Objects Matter! Disentangling the Fabric of Real Operational Processes to Create Digital Twins. In: Theoretical Aspects of Computing – ICTAC 2021. Pp. 3–17, Aug. 2021.

[Aa21b] van der Aalst, W. M. P.: Hybrid Intelligence : to automate or not to automate, that is the question. International Journal of Information Systems and Project Management 9/, pp. 5–20, 2021.

[AHW21] van der Aalst, W. M. P.; Hinz, O.; Weinhardt, C.: Resilient Digital Twins. Business & Information Systems Engineering 63/, pp. 615–619, Dec. 2021.

[BCF19] Barricelli, B. R.; Casiraghi, E.; Fogli, D.: A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications. IEEE Access 7/, pp. 167653–167671, 2019.

[Bo21] Bolender, T.; Bürvenich, G.; Dalibor, M.; Rumpe, B.; Wortmann, A.: Self-Adaptive Manufacturing with Digital Twins. In: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). Pp. 156–166, 2021.

[BP22] Becker, M. C.; Pentland, B. T.: Digital Twin of an Organization: Are You Serious? In: Business Process Management Workshops. Springer International Publishing, Cham, pp. 243–254, 2022.

[BRW11] Bönner, A.; Riedl, M.; Wenig, S.: Doppelzahlungen. In: Digitale SAP-Massendatenanalyse. ESV Erich Schmidt Verlag, 2011.

[Ca20] Caporuscio, M.; Edrisi, F.; Hallberg, M.; Johannesson, A.; Kopf, C.; Perez-Palacin, D.: Architectural Concerns for Digital Twin of the Organization. In: Software Architecture. Springer International Publishing, Cham, pp. 265–280, 2020.

[DA05] van Dongen, B.; van der Aalst, W. M. P.: Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets. Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management/, 2005.

[Da21] Dawson, P.: 2021 Hype Cycles: Innovating Delivery Through Trust, Growth and Change. Gartner/, 2021.

[De19a] Dellermann, D.; Calma, A.; Lipusch, N.; Weber, T.; Weigel, S.; Ebel, P.: The Future of Human-AI Collaboration: A Taxonomy of Design Knowledge for Hybrid Intelligence Systems. In: 52nd Hawaii International Conference on System Sciences (HICSS). Jan. 2019.

[De19b] Dellermann, D.; Ebel, P.; Söllner, M.; Leimeister, J. M.: Hybrid Intelligence. Business & Information Systems Engineering 61/, pp. 637–643, Oct. 2019.

[FLN21] Frerichs, M.; Leible, S.; Nüttgens, M.: Towards method- and tool-independent business process modeling. In: INFORMATIK 2021, Lecture Notes in Informatics (LNI). Gesellschaft für Informatik (GI), Bonn, 2021.

[Gr14] Grieves, M.: Digital Twin: Manufacturing Excellence through Virtual Factory Replication, 2014.

[He04] Hevner, A. R.; March, S. T.; Park, J.; Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28/, 2004.

[Jo20] Jones, D.; Snider, C.; Nassehi, A.; Yon, J.; Hicks, B.: Characterising the Digital Twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology 29/, pp. 36–52, 2020.

[KGM12] Koppenhagen, N.; Gaß, O.; Müller, B.: Design Science Research in Action - Anatomy of Success Critical Activities for Rigor and Relevance. 20th European Conference on Information Systems (ECIS 2012)/, 2012.

[KK02] Karagiannis, D.; Kühn, H.: Metamodelling Platforms. In: E-Commerce and Web Technologies. Springer, Berlin, Heidelberg, pp. 182–196, 2002.

[KK21] Kerremans, M.; Kopcho, J.: Create a Digital Twin of Your Organization to Optimize Your Digital Transformation Program. Gartner Research/, 2021.

[Ko16] Koppenhagen, N.; Mueller, B.; Maedche, A.; Li, Y.; Hiller, S.: Designing a supply network artifact for data, process, and people integration. Information Systems and e-Business Management 14/, Aug. 2016.

[NJ82] Naumann, J. D.; Jenkins, A. M.: Prototyping: The New Paradigm for Systems Development. MIS Quarterly 6/, 1982.

[Ös11] Österle, H.; Becker, J.; Frank, U.; Hess, T.; Karagiannis, D.; Krcmar, H.; Loos, P.; Mertens, P.; Oberweis, A.; Sinz, E. J.: Memorandum on design-oriented information systems research. European Journal of Information Systems 20/, pp. 7–10, 2011.

[PA21] Park, G.; van der Aalst, W. M. P.: Realizing A Digital Twin of An Organization Using Action-oriented Process Mining. In: 2021 3rd International Conference on Process Mining (ICPM). Pp. 104–111, 2021.

[PW20] Polyvyanyy, A.; van der Werf, J. M. E. M.: Information Systems Modeling: Playing with the Interplay Between Data and Processes. In: 18th International Conference, BPM 2020. 2020.

[Ri05] Riggio, R.; Ursino, D.; Kühn, H.; Karagiannis, D.: Interoperability in meta-environments: An XMI-based approach. In: Advanced Information Systems Engineering. CAiSE 2005. Springer, Berlin, Heidelberg, pp. 77–89, 2005.

[RSB09] Riege, C.; Saat, J.; Bucher, T.: Systematisierung von Evaluationsmethoden in der gestaltungsorientierten Wirtschaftsinformatik. In: Wissenschaftstheorie und gestaltungsorientierte Wirtschaftsinformatik. Physica-Verlag HD, Heidelberg, pp. 69–86, 2009.

[Sa21] Sadiq, S. W.; Someh, I. A.; Chen, T.; Indulska, M.: Value Creation from Data - Why is this a BPM Problem? In: Proceedings of the International Workshop on BPM Problems to Solve Before We Die (PROBLEMS 2021) co-located with the 19th International Conference on Business Process Management (BPM 2021), Rome, Italy, September 6-10, 2021. Vol. 2938. CEUR Workshop Proceedings, CEUR-WS.org, pp. 11–17, 2021.

[Se21] Semeraro, C.; Lezoche, M.; Panetto, H.; Dassisti, M.: Digital twin paradigm: A systematic literature review. Computers in Industry 130/, p. 103469, 2021.

[VPB12] Venable, J.; Pries-Heje, J.; Baskerville, R.: A Comprehensive Framework for Evaluation in Design Science Research. In: Design Science Research in Information Systems. Advances in Theory and Practice. Springer, pp. 423–438, 2012.

[We12] Weske, M.: Business Process Modelling Foundation. In: Business Process Management: Concepts, Languages, Architectures. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 73–124, 2012.

[We17] Werner, M.: Financial process mining - Accounting data structure dependent control flow inference. International Journal of Accounting Information Systems 25/, pp. 57–80, 2017.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 77

Modellieren mit HERAKLIT

Prinzipien und Fallstudie

Peter Fettke[1], Wolfgang Reisig[2]

Abstract: HERAKLIT ist ein laufendes Forschungsprogramm und Entwicklungsprojekt mit dem Ziel der Schaffung einer Infrastruktur zur Modellierung großer, rechnerintegrierter Systeme. Wir diskutieren die zentralen Anforderungen an solche Modelle (Hierarchien, Nutzersicht, Überführung informaler in formale Ideen, schematische Modelle, gleichrangiger Umgang mit digitalisierten und personengebundenen Prozessen) und erläutern, wie HERAKLIT diese Anforderungen unterstützt. Eine Fallstudie zeigt die Nutzbarkeit von HERAKLIT in der Praxis.

Keywords: systems composition; data modelling; behaviour modelling; composition calculus; algebraic specification; systems mining

1 Einleitung

Informatik hat zwei Gesichter: einerseits die technischen Grundlagen mit den formalen Konzepten zu ihrer Nutzung, andererseits die Anwendung. *Dijkstra* hat vielfach vorgeschlagen, die formale und informelle Sicht zu trennen und zwischen ihnen eine Mauer (engl. „firewall“) zu errichten [Di89]. Begründung: Das „correctness problem“ des Informatikers verlangt ganz andere Herangehensweisen als das „pleasantness problem“ des Anwenders. Modellierung findet in diesem Bild auf der einen oder anderen Seite dieser Mauer statt.

Demgegenüber schlagen wir hier vor, Modellierung als eine Tätigkeit zu verstehen, die einen möglichst bruchlosen Übergang zwischen formal formulierten und lebensweltlichen Sachverhalten erlaubt. Modelle verknüpfen Anwendung und Technik und beruhen auf denselben Grundlagen.

In diesem Beitrag beschreiben und motivieren wir zunächst im zweiten Abschnitt Anforderungen, die heutzutage an die Modellierung rechnerintegrierter Systeme gestellt werden. Kapitel 3 stellt dann die Konzepte vor, mit denen HERAKLIT [FR21a, FR21b] diese Anforderungen bewältigt. Eine umfangreiche Fallstudie zeigt im vierten Kapitel, dass die Erfüllung aller Anforderungen in einem integrierten Framework tatsächlich gelingt. Schließlich wird

[1] Saarland University, Saarbrücken, Germany, und German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany, peter.fettke@dfki.de
[2] Humboldt-Universität zu Berlin, Berlin, Germany, reisig@informatik.hu-berlin.de

doi:10.18420/modellierung2022-005

im fünften Kapitel diskutiert, in welchem Umfang andere Frameworks die Anforderungen aus Kapitel 2 entsprechen.

2 Anforderungen an die Modellierung rechnerintegrierter Systeme

Der Herstellung eines komplexen rechnerintegrierten Systems ist immer ein Planungsprozess vorgeschaltet, in dem die Architektur, Funktionsweise, beabsichtigten Effekte etc. des Produktes formuliert werden. Zentrales Werkzeug und Hilfsmittel dafür sind Modelle. Auch ein gegebenes komplexes System kann man besser verstehen und analysieren, indem man es modelliert. Ein Modell betont einige Aspekte des Systems, oft mit graphischen, standardisierten Darstellungen. Ein gutes Modell ist nicht nur anschaulicher und besser verständlich als eine umgangssprachliche Darstellung; es lässt sich auch vielfältig nutzen: man kann Funktionalität und Performanz-Engpässe erkennen, Kosten abschätzen, Korrektheit gegenüber einer Spezifikation nachweisen, Parameter optimieren und vieles andere mehr.

Moderne und zukünftige digitale Systeme, cyber-physikalische Systeme, digitale informationsbasierte Infrastrukturen, das *Internet of people, things and services* etc., werden zunehmend komplexer und wachsen immer mehr zusammen. Folglich sind bei der Gestaltung rechnerintegrierter Systeme vielfältige Herausforderungen zu überwinden. Nötig sind dafür Frameworks, die insbesondere für große Systeme wirklich Nutzen bringen. Dafür ist eine Reihe von Aspekten besonders wichtig:

Komposition und hierarchische Verfeinerung lokaler Komponenten: Was bedeutet es konzeptuell, dass rechnerintegrierte Systeme „groß“ sind? Es bedeutet zunächst einmal, dass einige Konzepte, die für kleine Systeme durchaus verwendet werden können, nun nicht mehr praktikabel sind. Am augenfälligsten gilt das für globale Zustände und globale Schritte. Und es bedeutet, dass ein großes System im Allgemeinen nicht amorph, gleichförmig gestaltet ist, sondern in irgendeiner Form komponiert ist aus kleineren, mehr oder weniger selbständig agierenden Systemen. Komposition und Verfeinerung sind also fundamentale Konzepte zur Modellierung großer Systeme.

Modelle aus Sicht der Anwender: Es gibt mehrere Weisen, Informatik im Allgemeinen und Modellierung in der Informatik im Besonderen zu fassen. Häufig wird der Zugang über die theoretische Informatik gewählt, mit der Betrachtung von Zeichenketten und ihrer Transformation. Beliebt ist auch der Zugang von der technischen Seite mit digitalen Schaltungen und ihrer Abstraktion, hin zu Software, Datenbanken, etc. Zum Verständnis großer Systeme besonders nützlich erscheint aber der Zugang aus der Sicht der Anwender, Nutzer, Betreiber und der Zwecke, für die große Systeme betrieben werden.

Systematische Überführung informeller, intuitiver Ideen zu einem gegebenen oder intendierten System in ein formales Modell: Dafür bietet die Informatik strukturierte Konzepte. Allerdings münden sie zumeist in Programme, nicht in formale Modelle.

Parametrisierte, schematische Modellierung: Oft soll nicht nur ein einziges System modelliert werden, sondern eine Menge gleichartig strukturierter und verhaltensähnlicher Modelle.

Integrierter, einheitlicher, gleichrangiger Umgang mit digitalisierten und lebensweltlichen Prozessen: Ein Beispiel ist in einem Betrieb die automatisierte digitale Personalverwaltung, zusammen mit der Belehrung eines Arbeitnehmers über gefährliche Güter und dessen Bestätigung durch seine Unterschrift. Neben symbolischen, implementierten Konzepten muss ein Modell also auch pragmatische, nicht zur Implementierung vorgesehene Schritte darstellen können.

3 Von HERAKLIT verwendete Konzepte

Informatik wird üblicherweise von der Rechentechnik her oder vom Symbolischen her aufgefasst, also aus der Sicht der technischen Informatik, oder der klassischen theoretischen Informatik. Demgegenüber betrachtet HERAKLIT die Informatik aus einer dritten Sicht, der Sicht der Anwender. Für HERAKLIT zentral ist die holistische Auffassung von Systemen und ihren Modellen: Es reicht nicht, einzelne Anforderungen an das Systemverhalten, Verfahrensanleitungen für Anwender, Datensammlungen, Algorithmen, Softwarekomponenten, Hardwarekonzepte etc. isoliert zu betrachten, zu verstehen und zu verwenden. Vielmehr muss ihr Zusammenspiel verstanden werden. Dafür braucht man ein Modell, das alle diese Aspekte integriert. HERAKLIT ist auf diese Anforderungen an Modelle zugeschnitten.

3.1 Konzeptionelle Sicht

HERAKLIT unterstützt die Modellierung von Systemen, die aus Teilsystemen zusammengesetzt sind, und die Hierarchien bilden, in denen konkrete Produkte, Maschinen, Menschen, Dokumente, Datenspeicher etc. kombiniert, separiert, transformiert, berechnet, bearbeitet, transportiert, erzeugt, vernichtet werden. HERAKLIT-Modelle können abstrakt, detailliert, schematisch sein; Daten, Gegenstände, Schritte zur Beschreibung dynamischen Verhaltens sind kleinteilig, aber auch umfassend formulierbar. Konkrete und abstrakte Datenstrukturen, sowie die Beschreibung von Verhalten werden integriert und aufeinander bezogen modelliert. HERAKLIT bietet dafür aufeinander abgestimmte Ausdrucksmittel, die den Modellierer bei der Formulierung seiner Ideen unterstützen. Der Modellierer wird nicht eingeschränkt; bei der Wahl des Abstraktionsgrades und der Art der Komposition von Modulen hat er inhaltlich alle Freiheiten; HERAKLIT schlägt aber konzeptionell vereinheitlichende Darstellungen vor. HERAKLIT integriert neue Konzepte mit bewährten, tiefliegend motivierten Konzepten der Modellierung von Software, Geschäftsprozessen und anderen Systemen. Zudem bietet HERAKLIT ein Konzept zur Abstraktion konkreter Daten, und damit die symbolische Modellierung großer Klassen von Systemen, die ähnlich aufgebaut sind und sich ähnlich verhalten.

Mit seinen wenigen, aber ausdrucksstarken Konzepten sind HERAKLIT-Modelle:

- für den Entwickler besser beherrschbar,
- für den Nutzer leicht verständlich,
- weniger fehleranfällig und leichter verifizierbar,
- einfacher änderbar, sowie
- schneller herstellbar und kostengünstiger als andere Methoden, insbesondere auch für wirklich große Systeme.

3.2 Technische Sicht

Technisch und formal neu ist die Integration von Architektur, statischen und dynamischen Konzepten. Dafür kombiniert HERAKLIT bewährte mathematisch basierte und intuitiv leicht verständliche Konzepte wie Prädikatenlogik, abstrakte Datentypen und Petrinetze, die auch bisher schon zur Spezifikation von Systemen verwendet werden; HERAKLIT kombiniert sie neu und ergänzt sie um den Kompositionskalkül. Das HERAKLIT-Framework

- unterstützt die hierarchische Partitionierung eines großen Systems in Module;
- kann technisch einfach, aber inhaltlich flexibel und ausdrucksstark Module zu großen Systemen komponieren;
- beschreibt diskrete Schritte in großen Systemen lokal konzentriert in einzelnen Modulen in frei gewähltem Detaillierungsgrad;
- stellt alle Arten von Modulen integriert mit den gleichen Konzepten dar, egal ob sie zur Implementierung vorgesehen sind oder nicht;
- repräsentiert Daten auf frei gewählter Abstraktionsstufe;
- berücksichtig Datenabhängigkeiten im Kontrollfluss;
- kann von konkreten Daten abstrahieren, um verhaltensgleiche Instanziierungen in einem Schema zu fassen;
- kann Modelle generieren, die skalierbar, systematisch, änderbar und erweiterbar sind;
- unterstützt den Nachweis, dass ein Modell gewünschte Eigenschaften hat.

Mit den beschriebenen technischen Konzepten können Aspekte diskreter Systeme modelliert werden, wie es keine andere Modellierungstechnik ermöglicht:

Module unterschiedlicher Hierarchiestufen können komponiert werden. Beispielsweise kann in einem HERAKLIT-Modell ein Modul eine Systemkomponente auf feinster, operationeller

Stufe modellieren; zugleich werden seine benachbarten Module nur mit ihrer Schnittstelle, ihrem Namen und ggf. Teilen ihrer inneren Struktur repräsentiert.

Lebensweltliche und abstrakte Objekte sind gleichrangige Elemente formaler Argumentation. HERAKLIT folgt der Prädikatenlogik, und fasst lebensweltliche und formale Objekte gleichrangig als Interpretationen von Termen einer Signatur (eines Alphabetes mit getypten Symbolen) auf.

Verhaltensmodelle können parametrisiert werden. Jede Interpretation der symbolischen Beschriftungen eines schematischen HERAKLIT-Moduls generiert ein eigenes Systemmodell. Beispiel: Ein schematisches Modul beschreibt die Prinzipien der Geschäftsprozesse aller Filialen eines Bekleidungsunternehmens; jede Interpretation beschreibt eine konkrete Filiale.

Der Umfang von Modellen wächst linear in der Größe des modellierten Systems. Das gilt insbesondere auch bei der Modellierung dynamischen Verhaltens. Erreicht wird dies mit dem konsequenten Verzicht auf globale Konstruktionen (beispielsweise globale Zustände) bei der Bildung einzelner Abläufe, mit einem lokal beschränkten Kompositionsoperator, etc.

Bewährte Verifikationstechniken werden übernommen. Petrinetze bieten eine Vielzahl effizienter rechnergestützter Verifikationstechniken. Die wichtigsten, insbesondere Platz- und Transitions-Invarianten von Petrinetzen, funktionieren auch auf symbolischer (Signatur-) Ebene und – entsprechend ausdrucksstärker – für die einzelnen Interpretationen einer Signatur. Eine Verifikation auf der Basis des *model checking* ist für einzelne Interpretationen einer Signatur möglich; dafür gibt es effiziente Verfahren. Ein großes, offenes Feld ist die kompositionale Verifikation: Eigenschaften eines komponierten Systems werden aus Eigenschaften der komponierten Module abgeleitet.

Ein einzelner Ablauf kann verteilt sein. In einem Ablauf eines großen Systems treten Ereignisse oft unabhängig voneinander ein. Beispiel: In einer Verkaufsfiliale interagieren die Mitarbeiter unabhängig voneinander mit einzelnen Kunden; sie werden nur gelegentlich synchronisiert beim Zugang zu knappen Ressourcen, beispielsweise beim Bezahlen an der Registrierkasse. Viele Frameworks verstehen Unabhängigkeit als „beliebige Reihenfolge“; das führt zu exponentiell vielen vermeintlich verschiedenen „sequentiellen“ Abläufen. HERAKLIT modelliert Unabhängigkeit von Ereignissen in Abläufen explizit. Das führt zu einem klaren Begriff von Nichtdeterminismus und zu einem Theorem, nachdem die Abläufe eines komponierten Systems aus den Abläufen der Komponenten ableitbar sind.

HERAKLIT ist intuitiv einfach. HERAKLIT verwendet intuitiv einfache Basis-Konzepte: Prädikatenlogik ist vielen Anwendern aus anderen Kontexten ohnehin geläufig. Auch Petrinetze sind weit verbreitet. Der Kompositionskalkül ist an kleinen Beispielen unmittelbar einsichtig. Die graphischen Ausdrucksmittel von Petrinetzen und des *composition calculus* ergänzen sich harmonisch.

3.3 Die drei Dimensionen von HERAKLIT

HERAKLIT unterscheidet drei Dimensionen, die drei unterschiedliche, aber integrierte Aspekte der Modellierung kennzeichnen:

Architektur: Ein HERAKLIT-Modell besteht aus *Modulen*. Jedes Modul hat ein beliebig gestaltetes Inneres; insbesondere ist die Abstraktionsebene der Darstellung frei wählbar. Die Schnittstelle eines Moduls enthält beliebige gelabelte Elemente. Komposition von Modulen ist technisch einfach und immer assoziativ. Formale Grundlage dieses Architekturprinzips ist der Kompositionskalkül aus [Re19].

Statische Aspekte: Für den Umgang mit Daten und Datenstrukturen greift HERAKLIT auf abstrakte Datentypen und algebraische Spezifikationen zurück, wie sie sich in der Informatik von Anfang an bewährt haben und in Spezifikationssprachen wie *VDM* [Jo91] und *Z* [Bo01] seit langem verwendet werden [ST12]. Symbolische Darstellungen (Terme einer Signatur) werden wie in der Prädikatenlogik verwendet. Zu jedem Modul gehört eine Signatur, also eine Menge von Symbolen für Mengen, Konstanten und Funktionen, aus denen zusammen mit Variablen dann Terme gebildet werden. Eine Signatur, und damit ihre Terme, können in ganz unterschiedlichen Lebenswelten instanziiert werden.

Dynamische Aspekte: Ein Modul zur Beschreibung von Verhalten enthält in seinem Inneren ein Petrinetz [Pe77]. Dabei ist jeder Platz des Petrinetzes ein prädikatenlogisches Prädikat, jeder Pfeil ist mit einem Term der Signatur des Moduls beschriftet, und jede Transition mit einer Bedingung. Für eine gegebene Instanziierung der Signatur kann die Menge der Objekte, auf die das Prädikat zutrifft, durch den Eintritt eines Ereignisses wachsen oder schrumpfen. Im Einzelnen wird das durch Terme an den Pfeilen des Petrinetzes beschrieben. Damit kann Verhalten abstrakt auf schematischer Ebene, aber auch konkret für eine „gemeinte“ Instanziierung dargestellt werden.

4 Fallstudie

Gegeben sei ein Restaurant-Betrieb mit mehreren Filialen. Alle Filialen sind nach demselben Schema aufgebaut und verhalten sich nach demselben Muster; sie unterscheiden sich aber in einigen Einzelheiten. Modelliert wird das Schema aller Filialen, ein Beispiel einer Filiale, und ein konkreter Ablauf in dieser Filiale.

4.1 Die Architektur der Filialen

Grundlegendes Konzept der Architektur der Filialen sind Module, wie sie in Abschnitt 3.2 generell für HERAKLIT-Modelle skizziert wurden. Im Inneren der hier verwendeten Module liegen Petrinetze in verschiedenen Ausprägungen. Generell hat ein Modul M zwei

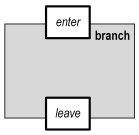


Abb. 1: abstrakte Sicht auf eine Filiale

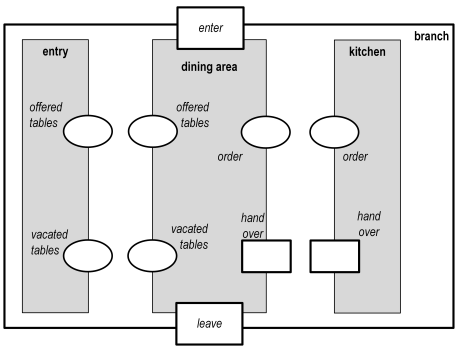


Abb. 2: eine Filiale, bestehend aus drei Modulen

Schnittstellen, die linke Schnittstelle ${}^*M$ und die rechte Schnittstelle M^*. Eine Schnittstelle kann Petrinetz-Plätze und -Transitionen enthalten. Graphisch wird ein Modul rechteckig dargestellt, mit den Elementen der linken Schnittstelle auf dem linken oder oberen Rand des Rechtecks und den Elementen der rechten Schnittstelle auf dem rechten oder unteren Rand.

Abb. 1 zeigt das Schema für die Filialen in größtmöglicher Abstraktion als ein HERAKLIT-Modul, *branch*. Modelliert werden zwei Aktivitäten der Restaurant-Gäste: das Betreten und das Verlassen einer Filiale; technisch als Petrinetz-Transitionen in ${}^*branch$ und $branch^*$ mit den Beschriftungen *enter* bzw. *leave*.

Abb. 2 zeigt, dass jede Filiale drei Komponenten hat: den Eingang, den Gastraum und die Küche. Diese Komponenten werden wiederum als Module modelliert. Die linke Schnittstelle des Eingangs ist leer, die rechte Schnittstelle enthält zwei Petrinetz-Plätze. Die Beschriftungen dieser beiden Plätze stimmen mit den Beschriftungen der Plätze der linken Schnittstelle des Gastraums überein; gleich beschriftete Plätze werden später beim Komponieren der beiden Module miteinander verschmolzen. Die beiden Transitionen der Schnittstellen des *branch*-Moduls liegen auch in den Schnittstellen des Gastraums. Die rechte Schnittstelle des Gastraums enthält einen Platz und eine Transition; entsprechend beschriftete Elemente enthält die linke Schnittstelle des Küchen-Moduls.

Die Beschriftungen der Schnittstellen-Elemente weisen auf das Verhalten und die Funktionalität der Module hin: am Eingang wird jedem Gast ein freier Tisch zugewiesen. Im Gastraum kann der Gast seine Bestellung aufgeben, später das bestellte Gericht entgegennehmen und an seinem Tisch verspeisen.

4.2 Statische Komponenten der Filialen

In jeder Filiale spielen vier Mengen eine zentrale Rolle: die Tische, die Gäste, die Speisekarte mit ihren einzelnen Einträgen und die zubereiteten Gerichte. In jeder Filiale kann jede dieser

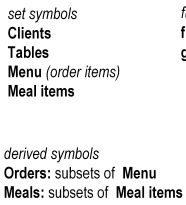


Abb. 3: Signatur Σ_0

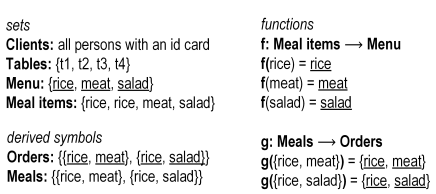


Abb. 4: Σ-Struktur S_0

Mengen jeweils aus anderen Elementen bestehen. Beispielsweise arbeitet eine kleine Filiale mit weniger Tischen und einer anderen Auswahl auf der Speisekarte als eine große Filiale.

Um dennoch für alle Filialen die Abläufe gleich zu formulieren, verwenden wir eine Signatur, Σ_0, die Abb. 3 zeigt. Sie enthält die vier Symbole *Clients, Tables, Menu, Meal items* für die vier oben beschriebenen Mengen, dazu zwei Symbole (*Orders* und *Meals*) für Teilmengen und zwei Symbole (f und g) für Funktionen. *Orders* steht für Bestellungen (eine Bestellung ist eine Teilmenge der Einträge in die Speisekarte), und *Meals* steht für Gerichte (ein Gericht ist eine Teilmenge zubereiteter Speisen). Die Funktion f ordnet jeder Speise den entsprechenden Eintrag in der Speisekarte zu; g ordnet jedem Gericht seine Bestellung zu.

In Abschnitt 3.2 wurde diskutiert, wie eine Signatur auf vielfältige Weise instanziiert werden kann. Jede konkrete Filiale ist durch eine solche Instanziierung der Signatur Σ_0 charakterisiert. In der Fallstudie diskutieren wir die Instanziierung S_0 der Abb. 4.

4.3 Das Verhalten der drei Module auf Schema-Ebene

Zunächst modellieren wir das Verhalten jedes einzelnen der drei Module aus Abb. 2. Abb. 5 zeigt das Verhalten der einzelnen Module auf der Schema-Ebene, also auf Basis der Signatur Σ_0, und nicht einer einzelnen Instanziierung. Als mathematisches Konzept repräsentiert ein Platz ein Prädikat, das auf eine Menge von Objekten zutrifft. Diese Menge kann durch den Eintritt von Transitionen wachsen und schrumpfen.

Das *entry*-Modul in Abb. 5 verwaltet die Tische. Zunächst stellt sich hier die Frage nach einer angemessenen Anfangsmarkierung: Für eine gegebene Instanziierung, also eine konkrete Filiale, enthält der Platz *free tables* anfangs alle Tische als Marken. Im Beispiel der Struktur S_0 aus Abb. 4 sind das die vier Marken t_1, t_2, t_3, t_4. Auf der schematischen Ebene kennen wir nur das Symbol *Tables*, für das jede Instanziierung eine Belegung mit einer Menge von Tischen frei wählen kann. Die zunächst naheliegende Idee, den Platz *free tables* anfangs mit dem Symbol *Tables* zu beschriften, greift allerdings zu kurz: Bei einer Instanziierung des Symbols *Tables* mit einer Menge, beispielsweise $\{t_1, t_2, t_3, t_4\}$, würde anfangs diese Menge als eine einzige Marke entstehen. Wir wollen aber die vier Elemente dieser Menge als einzelne Marken. Das wird mit *elm(Tables)* notiert.

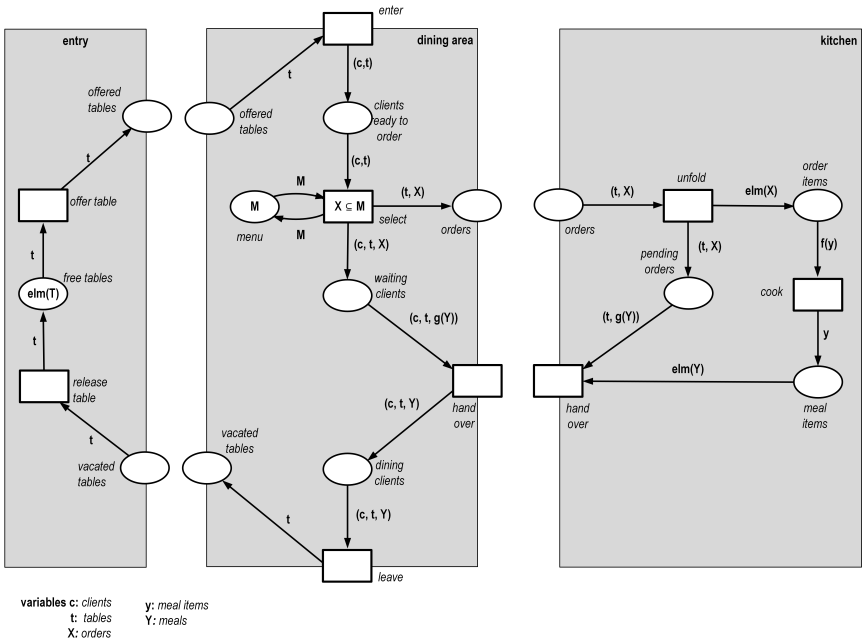


Abb. 5: Verhalten der Module auf der Schema-Ebene

Mathematisch steckt dahinter, dass ein Platz ein Prädikat darstellt, das aktuell auf die Elemente im Platz zutrifft. Der Platz *free tables* mit der Inschrift *elm(Tables)* steht also für den logischen Ausdruck:

$$\forall t \in \mathit{Tables} : \mathit{free\ tables(t)} \tag{1}$$

Die Instanziierung S_0 erzeugt damit in der Anfangsmarkierung den logischen Ausdruck „$\forall t \in \{t_1, t_2, t_3, t_4\}$: *free tables(t)*“. Damit liegen auf dem Platz *free tables* anfangs die vier Marken t_1, t_2, t_3, t_4.

In einer gegebenen Instanziierung S kann im Gastraum-Modul die Transition *enter* eintreten, sobald eine Marke auf *offered tables* vorliegt, indem die Variable t mit dieser Marke belegt wird. Die Belegung der Variablen c kann frei gewählt werden aus der Menge, mit der die Instanziierung S das Mengensymbol *Clients* instanziiert. Die Idee dahinter: Das Modul kann mit einem anderen Modul aus seiner Umgebung komponiert werden, indem die Belegung von c mit einem Gast einen inhaltlichen Sinn hat.

Bei der Transition *select* ist das Verhältnis zwischen dem Symbol *Menu* und der Variablen X interessant: Eine Instanziierung S des Gastraum-Moduls legt *Menu* als eine Speisekarte, also eine Menge einzelner Einträge fest. Hingegen wird beim Eintreten von *select* die Variable X jedes mal neu belegt. Die Bedingung $X \subseteq \mathit{Menu}$ der Inschrift von *select* garantiert, dass die Belegung von X („Bestellung“) nur Einträge enthält, die in der Speisekarte aufgeführt sind.

Eine *order* besteht aus einem Tisch und einer Bestellung. Jede Marke auf dem Platz *waiting* besteht aus einem Kunden c, seinem Tisch t, und seiner Bestellung X. Mit der Transition *hand over* wird dem Gast das bestellte Gericht ausgehändigt.

Im *kitchen*-Modul zerlegt die Transition *unfold* jede eingehende Bestellung in ihre einzelnen Einträge, formuliert mit Hilfe des *elm*-Operators, analog zur Verwendung im *entry*-Modul. Für jeden dieser Einträge $f(y)$, also jede Benennung einer Speise, wird die Speise y gekocht (Transition *cook*). Entsprechend einer vorliegenden Bestellung auf dem Platz *pending orders* werden dann Speisen als Y zusammengestellt und als Gericht an den Kunden übergeben.

4.4 Die Instanziierung S_0 und ihre Verhalten

Abb. 6 zeigt ein Modul, S_0, das aus den Modulen von Abb. 5 in zwei Schritten entsteht: erstens werden die drei Module aus Abb. 5 zu einem einzigen Modul, System S_0, komponiert; zweitens wird die Signatur Σ_0 von Abb. 5 mit der Struktur S_0 instanziiert. Insbesondere enthält der Platz *free tables* jetzt vier Marken, und der Platz *menu* drei Marken.

In der Instanziierung S_0 kann man nun den Eintritt einzelner Transitionen dokumentieren. Im Anfangszustand kann beispielsweise die Transition *offer table* mit der Belegung $t = t_1$ eintreten und dabei die Marke t_1 auf den Platz *offered tables* verschieben. Danach kann *enter* mit der Belegung $t = t_1$ und einem frei gewählten Kunden für c eintreten. Unabhängig davon kann *offer table* auch mit $t = t_2$ eintreten, etc. So kann beispielsweise ein Ablauf entstehen, in dem der Gast *Alice* aus der Speisekarte eine Bestellung für Reis und Fleisch zusammenstellt, für die in der Küche die entsprechenden Speisen gekocht und an Alice übergeben werden. Daneben kann *Bob* ein Gericht mit Reis und Salat bestellen und ausgehändigt bekommen. Solches einzelnes Verhalten wird üblicherweise als Sequenz aus globalen Zuständen und eintretenden Transitionen beschrieben, beginnend mit der Anfangsmarkierung.

HERAKLIT modelliert ein einzelnes Verhalten als verteilten Ablauf (vgl. Abschnitt 3.3); formal gefasst als ein Modul. Damit können verteilte Abläufe einfach komponiert werden. Am laufenden Beispiel wird zudem deutlich, dass verteilte Abläufe Einzelheiten der Zusammenhänge innerhalb eines Ablaufs sehr viel expliziter zeigen als es mit sequentiellen Abläufen möglich ist.

Abb. 7 zeigt einen verteilten Ablauf, A_0, des Systems S_0 in Abb. 6. Zur besseren Lesbarkeit komponieren wir diesen Ablauf aus drei Teilen: Das Modul *Anfang von* A_0 in Abb. 8 besteht aus zwei Strängen. Der obere beginnt mit dem Eintritt der Transition *offer table* aus Abb. 6 im Modus $t = t_1$. Dadurch enthält der Platz *offered tables* die Marke t_1. Diese Marke wiederum aktiviert die Transition *enter* mit der Belegung $t = t_1$, und einer frei wählbaren Belegung von c. Der Ablauf A_0 wählt $c = Alice$ und erzeugt damit die Marke $(Alice, t_1)$ auf dem Platz *clients ready to order*. Unabhängig von diesem Strang beschreibt der untere Strang Schritte der Marke t_2 und das Entstehen der Marke (Bob, t_2) auf dem Platz *clients ready to order*.

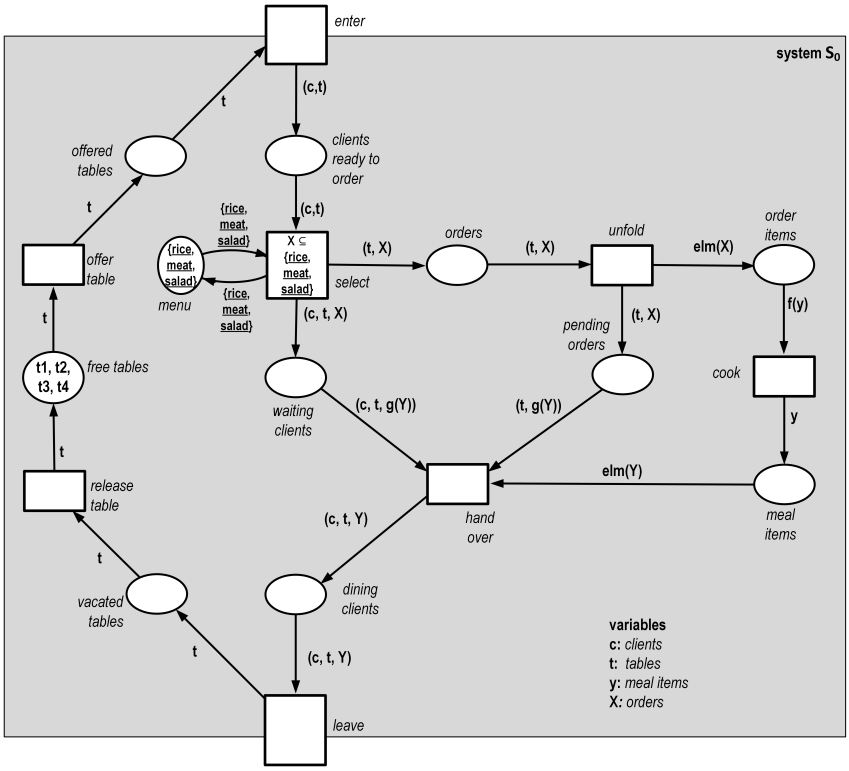


Abb. 6: System S_0

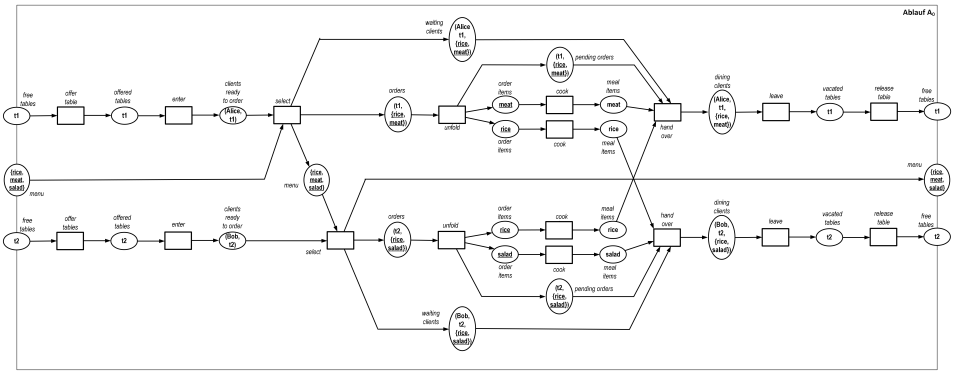


Abb. 7: Ablauf A_0

Das Modul *Mitte von* A_0 in Abb. 10 ergänzt in seiner linken Schnittstelle die rechte Schnittstelle von *Anfang von* A_0 um den Platz *menu* mit der Marke *{rice, meat, salad}*.

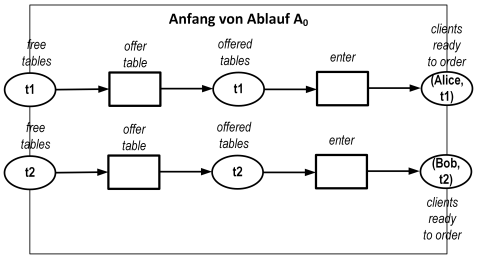


Abb. 8: Anfangsstück des Ablaufs A_0

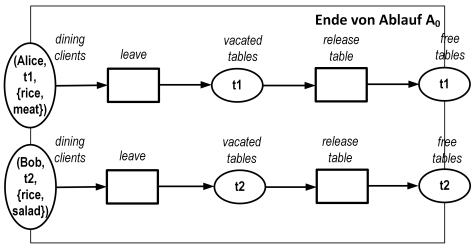


Abb. 9: Endstück des Ablaufs A_0

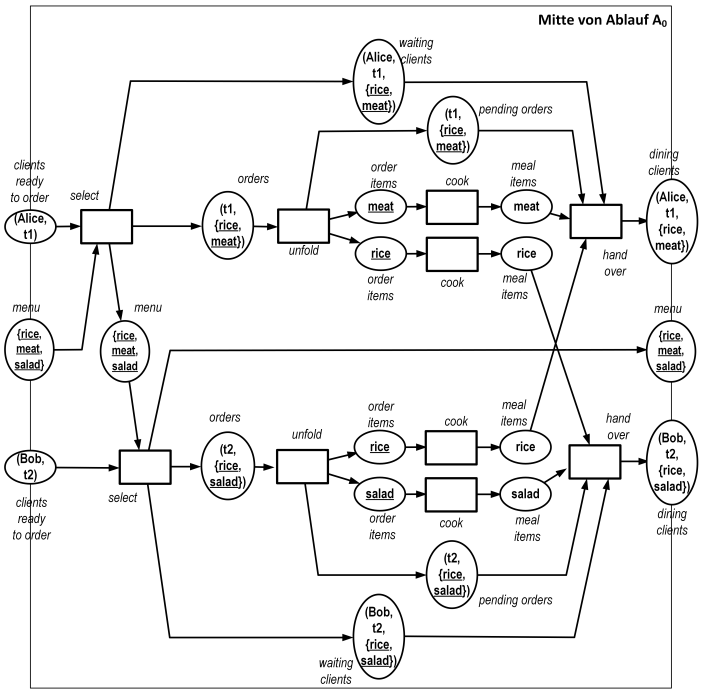


Abb. 10: Mittelstück des Ablaufs A_0

Diese Marke repräsentiert das einzige verfügbare Exemplar der Speisekarte. *Alice* und *Bob* schauen sie nacheinander an, und erzeugen jeweils eine *order*, $(t_1, \{\underline{rice}, \underline{meat}\})$ und $(t_2, \{\underline{rice}, \underline{salad}\})$. An der Transition *unfold* beginnt im Modul *System* S_0 ein Pfeil mit der Inschrift $elm(X)$. Mit $t = t_1$ und $X = \{\underline{rice}, \underline{meat}\}$ zerlegt $elm(X)$ im Modul den Anteil $\{\underline{rice}, \underline{meat}\}$ der Marke $(t_1, \{\underline{rice}, \underline{meat}\})$ in die Bestandteile $\underline{rice}$ und $\underline{meat}$, analog zur Inschrift des Platzes *free tables* im Schema von Abb. 5. Die entsprechenden Speisen *rice* und *meat* werden einzeln gekocht. Mit der Bestellung des Tisches t_2 wird entsprechend verfahren. In beiden Fällen wird *rice* bestellt. Da die Bestellungen nebenläufig vorliegen,

können die beiden gekochten *rice*-Portionen nicht eindeutig den Bestellungen zugeordnet werden. Im Ablauf A_0 werden sie vertauscht.

Das Modul *Ende von Ablauf* A_0 in Abb. 9 beendet nun die beiden Ablauf-Teile in offensichtlicher Weise. Die Komposition *Anfang von Ablauf* $A_0 \bullet$ *Mitte von* $A_0 \bullet$ *Ende von Ablauf* A_0 der drei Module liefert genau das Modul *Ablauf Ablauf* A_0 in Abb. 7.

5 Bezug von HERAKLIT zu anderen Modellierungsinfrastrukturen

Ein *Framework* zur Modellierung beschreibt wesentliche Konzepte zur Modellierung. Darauf aufbauend bietet eine *Modellierungsinfrastruktur* korrespondierende Modelle, Methoden, Techniken und Werkzeugen. Im Vergleich mit anderen Disziplinen wird generell *in der Praxis* der (Wirtschafts-) Informatik wenig mit Modellen gearbeitet. Es sind zahlreiche Frameworks vorgeschlagen worden, aber keines hat sich bisher wirklich durchgesetzt.

Ein Beispiel sind Diagramme der *Unified Modeling Language* (UML, [Ob17]) für Software-Systeme. Die verschiedenen Diagrammtypen veranschaulichen einige Aspekte eines Softwaresystems, aber unterstützen Analysefragen nur wenig. Ein anderes Beispiel sind die in der Wirtschaftsinformatik beliebten Diagramme der *Business Process Model and Notation* (BPMN, [Ob14]) oder *ereignisgesteuerte Prozeßketten* (EPK, [KNS92]). Solche Diagramme beschränken sich auf die Identifikation abstrakter Aktivitäten und die Darstellung des Kontrollflusses. Sie unterstützen Abstraktion und Komposition, helfen aber wenig beim Umgang mit konkreten oder abstrakten Daten und Gegenständen bei der Beschreibung des Verhaltens und beim Nachweis der Korrektheit von Verhaltensmodellen gegenüber einer Spezifikation. Die *Architektur integrierter Informationssysteme* (ARIS, [Sc01]) erlaubt zwar eine integrierte Beschreibung von Daten, Funktionen und Abläufen. Allerdings fehlen leistungsfähige Modularisierungskonzepte. Auch sind ARIS-Modelle nur bedingt formalisiert. Weit verbreitet sind Petrinetze; in ihrer klassischen Form modellieren sie allerdings lediglich verteilten Kontrollfluss. Zahlreiche Varianten und Verallgemeinerungen (insbesondere *Coloured Petri Nets* (CPN, [JK09])) integrieren konkrete Daten (in der Sprechweise von HERAKLIT: einzelne Strukturen). Typische Vorschläge für hierarchische Petrinetze ersetzen einzelne Transition durch ganze Netze. Statecharts [Ha87] verwenden eigenständige Modul- und Kompositionskonzepte; dabei werden gleich gelablete Übergänge endlicher Automaten verschmolzen. [Gr20] schlagen die statecart-basierte Sprache *Gamma* vor und diskutieren eine Vielzahl von Kompositionsoperatoren.

Schließlich gibt es noch eine Reihe von Frameworks, die letztlich als Strukturen einer Signatur, oder als eine Signatur auffassbar sind. Dazu gehören Abstract State Machines (ASM, [Gu00]), event-B [Ab05] und Z [Sp92]. Andere Frameworks orientieren sich an der Prädikatenlogik, beispielsweise TLA [La02], FOCUS [Br97, BS01] und Aloy [Ja87].

Alle diese Frameworks werden zur Analyse und Simulation von Softwarewerkzeugen unterstützt und wurden in größeren Software-Entwicklungsprojekten eingesetzt. Allerdings

hat sich keine wirklich durchgesetzt. Das liegt vorwiegend daran, dass bisher aus Modellen nicht besonders viel Nutzen gezogen werden kann. Derzeitige Frameworks fokussieren jeweils nur spezielle Bereiche eines Systems. Für kleine und mittelgroße Systeme sind die genannten Frameworks mehr oder weniger hilfreich. Aber da, wo man es besonders braucht, bei wirklich großen Systemen, fehlen systematische, strukturierende und abstrahierende Prinzipien zur Modellierung. Komplexe Systeme werden bestenfalls in Teilen modelliert, oft mit ganz unterschiedlichen Frameworks, die nur mühsam zusammenpassen. Es gibt keine umfassenden Modellsichten, in die dann ganz unterschiedliche Teilmodelle integriert werden können.

Im Vergleich mit den genannten Frameworks ist HERAKLIT breiter aufgestellt, indem Petrinetze, Konzepte algebraischer Spezifikation, und ein universeller Kompositionsoperator miteinander integriert sind. Keine der erwähnten Frameworks erreicht die Ausdruckskraft von HERAKLIT. Mit keiner der erwähnten Frameworks können alle Aspekte der Fallstudie des vierten Abschnitts modelliert werden.

Mit dem in der Logik grundlegenden und in der algebraischen Spezifikation bewährten Konzept einer Signatur und ihren unterschiedlichen Instanziierungen beschreibt HERAKLIT auch dynamisches Verhalten, sowohl auf der schematischen Ebene als auch für einzelne Instanziierungen. Das war auch die Intention von Prädikat-Transitionsnetzen [GL81].

HERAKLIT legt Wert auf Ausdrucksmittel für eine integrierte Modellierung lebensweltlicher und formaler Sachverhalte. Dazu nutzt HERAKLIT die wissenschaftstheoretischen Diskussionen um den Bezug zwischen der Welt und ihrer formalen Fassung, seit Aristoteles, die dann in die Prädikatenlogik eingeflossen ist [Su57]. HERAKLIT schlägt vor, dynamische Aspekte in die Prädikatenlogik einzubauen.

Wie bereits in der Einleitung erwähnt, hat *Dijkstra* vorgeschlagen, informelle und formale Argumentationen strikt zu trennen und zwischen ihnen eine „Firewall“ zu errichten. Diese Mauer ist nicht hilfreich zum Verständnis Informatik-basierter Systeme. Vielmehr braucht es eine Brücke, um zwischen der informellen Lebenswelt der Anwendung und der formalen Welt der Technik zu vermitteln. HERAKLIT bietet hierzu ein passendes Framework.

6 Ausblick

Die formalen Grundlagen und die Prinzipien des Einsatzes von HERAKLIT liegen vor; zahlreiche Fallstudien zeigen den Nutzen des umfassenden Ansatzes von HERAKLIT. Für die Verwendung im industriellen Maßstab und als bessere Alternative zu derzeit verwendeten Frameworks zur Modellierung fehlen insbesondere noch Softwarewerkzeuge zur Unterstützung des Entwurfs großer Modelle, und auf HERAKLIT zugeschnittene Analyseverfahren. Für spezielle Anwendungsbereiche werden im Lauf der Zeit Ausprägungen mit verfeinerten Konzepten gebildet. In einigen Anwendungen ist auch die automatische Erzeugung von Programmcode für einige Module wünschenswert.

Literaturverzeichnis

[Ab05] Abrial, J.-R.: The B-Book – Assigning Programs to Meanings. Cambridge University, 2005.

[Bo01] Bowen, Jonathan P: Z: A formal specification notation. In: Software specification methods, S. 3–19. Springer, 2001.

[Br97] Broy, Manfred: Compositional Refinement of Interactive Systems. Journal of the ACM, 44(6):850–891, 1997.

[BS01] Broy, Manfred; Stolen, Ketil: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer, 2001.

[Di89] Dijkstra, E. W.: Reply to comments. Commun. ACM, 32(12):1414, 1989.

[FR21a] Fettke, Peter; Reisig, Wolfgang: Handbook of HERAKLIT. HERAKLIT working paper, v1.1, September 20, 2021, `http://www.heraklit.org`, 2021.

[FR21b] Fettke, Peter; Reisig, Wolfgang: Modelling Service-Oriented Systems and Cloud Services with HERAKLIT. In (Zirpins, Christian; Paraskakis, Iraklis; Andrikopoulos, Vasilios; Kratzke, Nane; Pahl, Claus; El Ioini, Nabil; Andreou, Andreas S.; Feuerlicht, George; Lamersdorf, Winfried; Ortiz, Guadalupe; Van den Heuvel, Willem-Jan; Soldani, Jacopo; Villari, Massimo; Casale, Giuliano; Plebani, Pierluigi, Hrsg.): Advances in Service-Oriented and Cloud Computing. Springer International Publishing, Cham, S. 77–89, 2021.

[GL81] Genrich, Hartmann J.; Lautenbach, Kurt: System modelling with high-level Petri nets. Theoretical Computer Science, 13:109–135, 1981.

[Gr20] Graics, Bence; Molnár, Vince; Vörös, András; Majzik, István; Varró, Dániel: Mixed-Semantics Composition of Statecharts for the Component-Based Design of Reactive Systems. Softwware and Systems Modeling, 19(6):1483–1517, 2020.

[Gu00] Gurevich, Yuri: Sequential abstract-state machines capture sequential algorithms. ACM Transactions on Computational Logic, 1:77–111, 2000.

[Ha87] Harel, David: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 8(3):231–274, 1987.

[Ja87] Jackson, Daniel: Alloy: A Language and Tool for Exploring Software Designs. Communications of the ACM, 62(9):66–76, 1987.

[JK09] Jensen, Kurt; Kristensen, Lars M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, 2009.

[Jo91] Jones, Clifford B.: Systematic software development using VDM. Prentice Hall, 2. Auflage, 1991.

[KNS92] Keller, Gerhard; Nüttgens, Markus; Scheer, August-Wilhelm: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. Bericht 89, Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi) an der Universität des Saarlandes, 1992.

[La02] Lamport, Leslie: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Adison-Wesley, 2002.

[Ob14] Object Management Group: Business Process Model and Notation (BPMN): Version 2.0.2. Bericht formal/2013-12-09, Object Management Group, 2014.

[Ob17] Object Management Group: OMG Unified Modeling Language (OMG UML): Version 2.5.1. Bericht formal/2017-12-05, Object Management Group, 2017.

[Pe77] Petri, C. A.: Non-Sequential Processes. Bericht ISF-77-5, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, Federal Republic of Germany, 1977.

[Re19] Reisig, Wolfgang: Associative composition of components with double-sided interfaces. Acta Informatica, 56(3):229–253, 2019.

[Sc01] Scheer, August-Wilhelm: ARIS: Modellierungsmethoden, Metamodelle, Anwendungen. Springer, 4. Auflage, 2001.

[Sp92] Spivey, John Michael: The Z Notation: A reference manual. Prentice Hall, 2. Auflage, 1992.

[ST12] Sanella, Donald; Tarlecki, Andrzej: Foundations of Algebraic Specification and Formal Software Development. Springer, 2012.

[Su57] Suppes, Patrick: Introduction to Logic. Van Nostrand Reinhold, 1957.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 93

Teaching the Use and Engineering of DSLs with JupyterLab: Experiences and Lessons Learned

Joel Charles[1], Nico Jansen[1], Judith Michael[1], Bernhard Rumpe[1]

Abstract: Domain-Specific Languages (DSLs) are tailored to a specific domain which requires them to provide domain-specific concepts and a sophisticated tooling for their engineering; aspects which we address with the language workbench MontiCore. As we use MontiCore for research and teaching, we are interested in reducing the entry barrier to use and engineer MontiCore DSLs. While there are approaches for ready-to-use learning environments such as web-based editors, only a few provide a tailored solution for specific DSLs. Within this paper, we present our experiences using JupyterLab in combination with the infrastructure of MontiCore for teaching the use and engineering of DSLs in an interactive manner. We have realized three practical courses and one conference tutorial applying this technical approach. The front-end provides immediate feedback and includes supporting explanations in an integrated manner. Initial feedback indicates that this approach can lower the entry barrier for DSL use and engineering for students and practitioners.

Keywords: Education; Domain-Specific Languages; Model-Driven Software Engineering; Software Language Engineering; JupyterLab; Jupyter Notebook

1 Introduction

Domain-Specific Languages (DSLs) become increasingly important in practice, e.g., the German tax forms [Ru21], in systems engineering [Gu21], architecture modeling for safety critical automotive software systems [SBS20], cyber attacks in the automotive domain [Wo21], TV program planning [Dr20], or simulation of marine ecosystems [JH17], just to mention some published examples in addition to the reports from our industry contacts. Thus, universities need to teach DSL engineering and usage. A domain-specific language is tailored to the needs of domain experts [KRR18], thus, often adopting domain terminology for its concrete syntax.

Reusing domain vocabulary within a modeling language mitigates entry barriers and enables its users to create conform models without much familiarization effort. However, learning a DSL is difficult for non-specialists since they first need to get familiar with the terminology of the domain [GM18]. In addition, being able to engineer a high-quality DSL requires a lot of practical hands-on experience [KRR18]. It requires the understanding of how to develop a suitable syntax and the knowledge of using a language workbench [Ba20]. In our context, the sophisticated language workbench MontiCore [HKR21] is used to engineer DSLs.

[1] Software Engineering, RWTH Aachen University, Germany, www.se-rwth.de
{charles,jansen,michael,rumpe}@se-rwth.de

doi:10.18420/modellierung2022-014

Until now, there exists no integrated learning environment to teach MontiCore-based DSLs. In the past, our teaching of the introduction to MontiCore DSLs separated the theoretical concepts from their practical application. Furthermore, the initial setup of the IDEs required manual steps by its users. This distracted them from their actual learning objectives.

In this paper, we present an interactive approach based on Jupyter Notebooks to improve the teaching of Software Language Engineering (SLE) based on our experiences. We have designed the architecture and realized the teaching infrastructure for MontiCore-based DSLs with Jupyter Notebooks. Moreover, we present our lessons learned from concrete labs using this infrastructure.

In Section 2, we explain the basics of SLE, as well as the language workbench MontiCore we use to teach it. Section 3 addresses the requirements we specified for the solution and how the introduction to MontiCore and the use of MontiCore-based languages has been taught until now. In Section 4, we consider related work. Section 5 examines possible approaches and Section 6 describes the selected solution. Section 7 explains the realization in several practical courses and one conference tutorial, whereas Section 8 discusses our lessons learned. The last Section concludes.

2 Preliminaries

Our work is based on the general notion of software language engineering and the particular realization of the MontiCore language workbench, including its generated infrastructure.

Software Language Engineering. The discipline of SLE investigates the efficient development, maintenance, and evolution of modeling languages. In general, modeling languages consist of a concrete and abstract syntax, a semantic domain, and a semantic mapping [Bu19, GRR09, He07]. The concrete syntax defines the possible sentences, while the abstract syntax comprises its structural essence. The semantic domain of a language describes the target application area, which typically depends on mathematical theory, such as Petri nets [Re12] or Focus [BS12] (i.e., formal modeling foundations for describing behavior and processes). A semantic mapping provides a meaning for the sentences, i.e., it maps sentences to the semantic domain, for instance, via formalized mathematical notation or graph transformations [HR04]. We distinguish between two types of modeling languages, General-Purpose Languages (GPLs) that are generally applicable and DSLs that correspond to an application domain [Cl15, Hö19].

MontiCore. MontiCore [HKR21] is a language workbench for designing textual modeling languages. It utilizes context-free grammars (CFGs) to define the abstract as well as the concrete textual syntax of a language in a single effort. The grammars feature an EBNF-like [Wi96] syntax and consist of multiple productions that declare the language. MontiCore processes the grammar and generates abstract syntax classes that describe the language's structure. Additionally, infrastructure for advanced language development

is provided [HKR21], including a parser, a framework for context conditions (CoCos), and a symbol table. The parser processes textual models and creates a corresponding abstract syntax tree (AST) comprising instances of the abstract syntax classes. An AST contains the information of the underlying model without syntactic sugar, which is used for further processing. For a consecutive step, MontiCore provides a CoCo framework, which are additional validation rules for the language. CoCos are constraints on the AST that cannot be defined well within the CFG, such as context-sensitive restrictions. A common example is checking whether a name starts with a capital letter. Furthermore, MontiCore provides a symbol management infrastructure, enabling to derive a symbol table for an AST. This symbol table lifts the tree structure of the AST to a graph structure, enabling cross-referencing, easy type checking, and quick navigation. For further customizability of the generated infrastructure, MontiCore provides the TOP mechanism [HKR21]. It is a realization of the generation gap pattern and allows for overriding generated artifacts. The mechanism automatically recognizes these handwritten artifacts and integrates these seamlessly into the generated architecture. Finally, MontiCore uses template-based code generators [Ad18] to process the AST and to transform the structured information into artifacts of a target language. MontiCore also supports language inheritance, embedding, and aggregation [Ha15], extending existing CFGs and thus reusing their concrete and abstract syntax. These features support more sophisticated language development by leveraging productions of existing languages for new DSLs, thus fostering reusability.

3 Challenges and Requirements for Teaching DSL Use and Engineering

The usage of a DSL and the engineering of a MontiCore-based DSL address different target audiences, resulting in different requirements for its learning materials [St15]. We show specific challenges of learning a DSL, the challenges of teaching the engineering of a MontiCore DSL, and introduce the former teaching approach and its shortcomings. The identified requirements are labeled within the text. While several publications report about requirements for (the engineering of) DSLs [CMP18], our analysis directly relates to teaching how to effectively create and use such modeling languages.

3.1 Using a MontiCore DSL

Since DSLs by their nature are tailored to a specific domain, a modeler must have a basic understanding of the domain to create valid models. To support the progression towards becoming a domain expert, this *basic understanding of the domain* should be taught as part of the intended solution (**Req. U1**). Thus, according to Bloom's Taxonomy [AK01], the learning goal of learning a new DSL is to *apply* gained knowledge concerning the corresponding tasks. Generally, we cannot assume that students and tutorial participants are already domain experts for every DSL they learn. Having internalized the essential key concepts of the domain, a modeler can formulate semantic sound statements within the

domain. In order to model these in a structured way in a DSL, he must *familiarize* himself with the *syntax* of the language (**Req. U2**). Since no special prior knowledge is required for the modeler, their knowledge level varies. Therefore, enabling continuous learning progress requires an educational environment to *provide immediate feedback* on model validity and potential improvements (**Req. U3**). A steep learning curve has to be managed to achieve this level of knowledge, therefore *no initial set-up effort* should distract from the focus on the learning process (**Req. U4**). In order to convey an understanding of the language infrastructure, it is necessary to be able to *integrate further language tooling*, such as a generator, into the solution in an executable manner (**Req. U5**).

3.2 Engineering a DSL with MontiCore

The requirements for developing a DSL are partly similar to those for modeling, but in the case of MontiCore, they are more sophisticated, as it contains various aspects (e.g., grammar, CoCos, etc.) that have to be implemented, often in different meta-languages. Thus, fundamental language quality criteria, as well as the methods for creating a DSL must be learned (**Req. E1**). To define the DSL, the characteristics of the used language workbench must be considered. In our use case we assume that the language engineer has no prior knowledge of MontiCore. Neither should a technical know-how be a prerequisite to get started with DSL engineering (**Req. E2**). Furthermore, it should also be possible to use sophisticated modeling features such as CoCo's and the TOP-mechanism. Although they are technically written in Java, they are an integral part of an advanced DSL development with MontiCore. The TOP mechanism is a programmatic adaptation of the generated code, accordingly a more in-depth understanding of the generation process is required in this case (**Req. E3**). Based on these requirements and an analysis of existing learning and modeling environments, we designed a solution tailored for our use case in MontiCore.

3.3 Previous Approach to Teach DSL Usage and Its Engineering

At RWTH Aachen, we teach DSL usage and engineering, e.g., in the corresponding SLE lecture, which includes the presentation of theory with slides, smaller exercises, and a large group project using MontiCore. The objective of the course is to learn to engineer a DSL. The skills learned are applied in practice as part of exercises. Moreover, the fundamentals of the language workbench are introduced. Weekly assignment sheets are given to students as part of the exercise. They include descriptions of how students need to set up their development environments. Furthermore, they are provided with projects as a starting point for their task, which they work on in an IDE. Consequently, students not only have to familiarize themselves with MontiCore, but also with an IDE that may be unfamiliar to them. Its initial configuration effort obstructs them from achieving the actual learning objective. For groups of people who do not attend the course, we provide them a website with an introduction to the use of the language workbench. For details, we provide the students a

handbook [HKR21] which includes a getting started section explaining configuration steps, and a detailed documentation of MontiCore's features. Consequently, no setup-free access to learn MontiCore was available for any target group.

Previously, we provided generated command line interface (CLI) tools for learning an existing DSL. While CLI-access is arguably an easy and efficient way to use and chain tools for experienced practitioners, it does not foster the initial learning of the language and, thus, is less suitable for inexperienced users [St15]. Default editor features such as syntax highlighting or autocompletion are not available.

In the case of teaching how to engineer a DSL, we generally provided a more sophisticated Gradle project containing the required source files and dependencies for developing a new modeling language. These projects were integrable into an IDE, such as IntelliJ[2] or Eclipse[3], which generally facilitates the engineering process. In MontiCore, generated artifacts (e.g., the parser), as well as handwritten artifacts (e.g., custom CoCos), are based on Java, for which extensive support is automatically provided. While the use of an IDE closest reflects the actual development of a language, it also comes with some issues, especially for novice users. As not all students have a computer science or programming background, an IDE might be quite overwhelming, resulting in practitioners spending more time getting used to the intricacies of the environment than with the language engineering task itself. Thus, using an IDE-based teaching method yielded issues such as dealing with different operating system (OS) distributions, different improper Java or Gradle versions, flawed IDE configurations, and general troubleshooting.

Our target group works with private hardware. Accordingly, a wide variety of OS platforms are used. Due to a lack of resources, we are not able to fully support all systems such that some distributions could not be used. Besides practitioners using incompatible Java or Gradle versions, MontiCore relies on the Java Development Kit (JDK). Even though the provided installation instructions explicitly guide towards the JDK, a common mistake is that users only installed the Java runtime environment during setup, resulting in redundant double-checking efforts for the teachers. Additionally, individual problems, such as reusing IDE configurations from an unrelated project or access rights, require the attention of the exercise instructors. For issues that did not occur during the pilot operation, a root cause must be found in the short term. This is particularly problematic for cases in which the origin of the error is outside of our area of responsibility. Generally, the accumulation of errors and their various causes generated an increased, often unmanageable, support effort resulting in a need for a solution independent of external factors such as hardware, environment setups, and different levels of practitioner knowledge.

[2] https://www.jetbrains.com/idea/
[3] https://www.eclipse.org/

4 Related Work

Other modeling tools are also faced with the challenge of learning them initially. For example, the MetaEdit+ Workbench [KLR96] and ADOxx [FK13] offer workshop events and webinars to help getting started. They are supplemented by videos, reference documentation, as well as slides, and instructions. Furthermore, there are forums where frequently asked questions can be answered and new questions can be posted.

With the Language Server Protocol (LSP) and its graphical extension (GLSP) [Ro18], respectively, with editors that implement these protocols, potential tools are available that can be used for modeling. The one-time implementation of the protocol makes it possible to support all (G)LSP editors, which is an advantage over the AtoMPM [Sy13] and WebGME [Ma14] approaches. Both are tools to create domain-specific editors.

All of the above approaches have in common that they do not take an integrated interactive learning approach. Instead, they only offer solutions to facilitate tool-based modeling. However, our use case considers the engineering and use of DSLs created with MontiCore. Both the DSL engineering and the resulting DSL are purely textual. The goal is for learners to be able to design languages with MontiCore and create textual models. This is not achieved when the learner uses a UI focused on diagram-based abstractions. Furthermore, it does not solve the problem that the editor still has to be mastered by the user. The extensive documentation of the above approaches shows that their use takes a similar time of familiarization as an IDE. For the acceptance of a model-driven approach (in industry), it is necessary to use suitable tools [TK16]. However, the features are often extensive and distract from the actual learning objective.

Another paper [BVG18] analyzes teaching the Object Constraint Language (OCL) [RG02] using different modeling tools (such as MagicDraw, Papyrus, etc.). This work aims at analyzing different modeling tools for a specific language. It confirms that direct feedback during modeling can positively affect the final result. Furthermore, their observations support that immediate feedback in an integrated learning environment improves the overall learning success.

Furthermore, [Te19] presents a browser-based modeling tool. Its main objective is to analyze practitioners and their learning behavior during modeling. The application tracks interesting events, such as interactions with an editor. The main goal of this approach is to gather data about how novice modelers obtain knowledge about the modeling language and modeling itself. While our platform focuses more on education than on gathering information about the learning experience, it is notable that the presented solution also uses a web-based application as a foundation for a proper learning environment. While future discoveries of this research observatory will provide relevant information for our work, the described setup already indicates that a directly functional application, without initial effort, is a suitable foundation for learners.

In general, there are several approaches documented that focus on teaching modeling

languages or their convenient provision via corresponding tools. The sources range from literature studies [RTS19] over DSL design guidelines [Ul13] to experience reports on the transition to distance learning [Bo21], with special concerns on educating practitioners. Our analyses indicate that providing extensive material in a self-learnable manner and an integrated learning environment with immediate feedback has positive effects on the learning results. Applications such as the bigER tool [GB21] suggest that integrated modeling with different views, here presented on textual and graphical representations of entity relationship diagrams, can additionally increase modeling efficiency and understanding of models simultaneously. In summary, it appears that many of our requirements for learning a modeling language or creating one, established in Section 3, is part of current research. Recurring features include immediate feedback, minimal initial set-up effort, a convenient look and feel for familiarizing with the language or modeling tool, and further integrated functionality, such as well-formedness checks or code generation.

5 Learning Environments for Teaching DSL Usage and its Engineering

When choosing a development environment, many considerations must be taken into account. First, a single cross platform IDE should be chosen to ensure a consistent user experience. Furthermore, the focus is on learning a DSL or its engineering. Accordingly, the initial setup process should be minimal. Only web-based solutions allow for a setup-free user experience, so we limit the candidates to these only [LH01, Ag00](see Req. U4).

One category of online environments are so-called playgrounds. The TypeScript Playground[4] and JSFiddle[5] are exemplary representatives of the category. Their main objective is rapid prototyping (see Req. U3). Applications are for instance to showcase a reproducible minimal example or to test a new feature of a programming language. Usually a collaboration is possible, e.g., via sharing a link. Since this category of editors is more aimed at providing a basis for discussions using small code snippets, it is not suitable for our use case [St15].

The next category is represented by online IDEs. Solutions like GitPod[6] and CodePen[7] are designed to streamline the development process by providing an always configured online IDE for developers (see Req. U4). They can be used to develop entire applications, while facilitating collaboration between developers.

With the Meta Programming System (MPS) [VP12] is a tool from JetBrains available to engineer DSLs. However, the tool is a editor only, which does not offer the integrated learning approach (see Req. U1, U2) we are targeting [Pr21].

These approaches do not include introductions to syntax and therefore require a pre-existing basic understanding of the respective language (see Req. E2). Additionally, they contain

[4] TypeScript Playground, Microsoft: `https://www.typescriptlang.org/play` (accessed: 2022-03-08)
[5] JSFiddle, JSFiddle: `https://jsfiddle.net` (accessed: 2022-03-08)
[6] GitPod GmbH, Gitpod: `https://www.gitpod.io` (accessed: 2022-03-08)
[7] CodePen, CodePen: `https://codepen.io` (accessed: 2022-03-08)

more functionality than needed for our use cases. Therefore, they are less suitable for newcomers. Furthermore, they fail to combine the code with the explanations for learning the language.

Another option is to develop a tailored IDE ourselves, using the IDE Platform Theia[8] for example. However, the development effort for this is high and involves long-term maintenance costs.

The open-source project Jupyter [PG15, GP21] focuses on web-based interactive computing for a wide variety of programming languages. It originates from the IPython project [PG07], which has pushed interactive computing for python. The core concept of the approach are so-called Jupyter Notebooks. The Jupyter Notebook is an UI which combines interactive computing with additional annotations in a tutorial-based approach (see Req. U1, U2, U3, E1). In the Jupyter context, a *tutorial* integrates information and input fields on one screen following the Read-Eval-Print-Loop (REPL) concept [Va20]. Figure 1 shows a sample notebook. It is composed of a set of typed cells. Cells are executable and are either of type

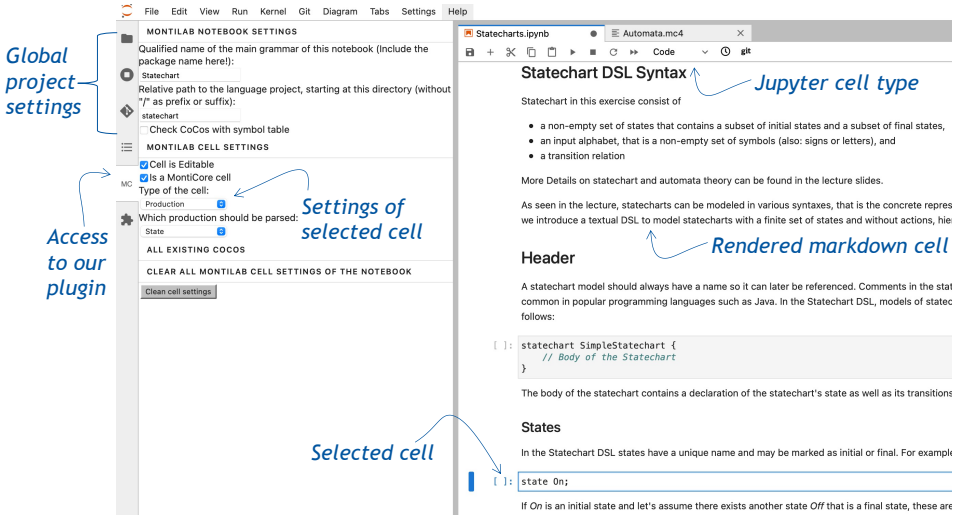


Abb. 1: A Jupyter Notebook edited in JupyterLab.

code, markdown or raw. The latter two types allow to insert additional information before, in between or after code cells. The execution of a markdown cell causes the markdown to be rendered in the cell. The raw cell only becomes relevant when the notebook is converted to another format, so this type will not be discussed in detail. Jupyter achieves the notebooks' language independence through the concept of kernels. The kernel contains all language-specific functionality and runs independent of the client used.

Within the project, JupyterLab represents the latest client. It is a web-based development

[8] Eclipse Foundation, Theia: `https://github.com/eclipse-theia/theia` (accessed: 2022-03-08)

environment (see Req. U4). The user interface is highly customizable through so-called plugins. Even the notebook itself in the right part in Figure 1 is such a plugin. The content of a code cell in the notebook is transferred to the kernel when it is requested to be executed. A kernel processes the request and transmits the result to the client. JupyterLab then renders the output under the executed cell (see Req. U5, E3). In this manner, domain specifics are separated from the language unspecific client. A notebook can be persisted in .ipynb format (json) and re-uploaded into JupyterLab. Thus, an executable documentation of the working process is available.

Compared to the previous solutions presented, JupyterLab is also suitable for less non-technical newcomers, as the functions are tailored to an interactive learning process [Bl19, Ma20]. In contrast, classic IDEs are tailored for the efficient development through experienced engineers. Compared to a custom solution with Theia, project Jupyter offers the advantage to build on an already established platform. At the same time, we benefit from enhancements at no cost to ourselves. For these reasons, we decided to use project jupyter for the realization. Table 1 summarizes the degree of requirements fulfillment of the available alternatives.

Approach	U1	U2	U3	U4	U5	E1	E2	E3
Playground	X	0	Y	Y	X	X	0	0
IDE	X	0	Y	Y	Y	X	X	0
Jupyter	Y	Y	Y	Y	Y	Y	Y	Y

Tab. 1: Fulfilled requirements by solution approach

6 Building the Jupyter Infrastructure

Jupyter Notebooks can be used both for engineering DSLs and for learning them. For the editing of Jupyter Notebooks we rely on JupyterLab. To enable multi-user support we use the JupyterHub which is part of project Jupyter. JupyterHub is a is a containerizable solution that provides user authentication and serves the configured JupyterLab. It suggests a deployment to an kubernetes cluster if more than 100 simultaneous users are targeted. However, otherwise the use of a single machine is suggested. We use the latter option as we initially assumed fewer than 100 users and could not justify the cost of a kubernetes cluster. However, we did not want to miss out on the advantages of a containerized deployment. Figure 2 shows the deployment we used. JupyterHub is executed inside a docker container, which is accessible via Https from the internet. When a user successfully authenticates, a container is created using JupyterHub's docker spawner. The spawner uses the same image for each user, which was created in advance on the host system. The image contains all the materials needed for the exercises. If the users manage to set their environment to a bad state while working on an exercise, the entire container can be restored. On the one hand, this is possible for administrators with access to the host system, but also for the users themselves, since communication of the user container with the JupyterHub container is established via a docker network. In principle, it is possible to store the files of the user

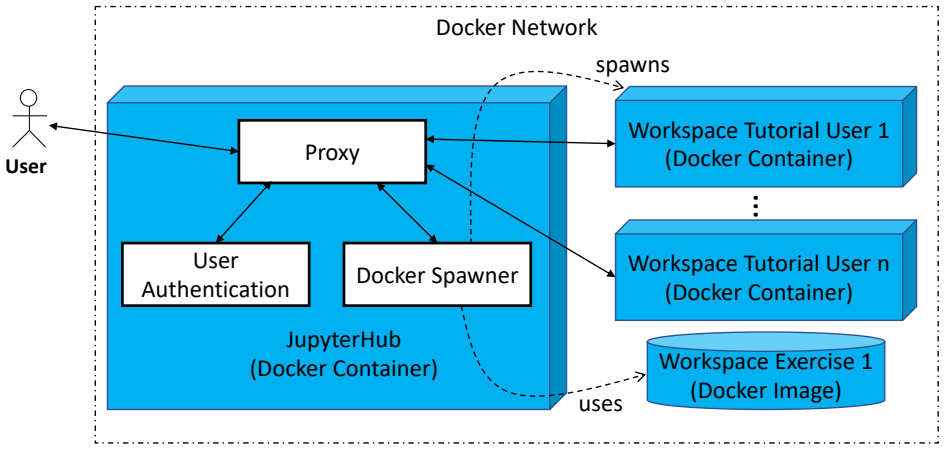


Abb. 2: The deployed JupyterHub architecture

containers via docker volumes. Then it is possible to remove the container in idle phases and save resources. As soon as the user returns, his data can be restored by mounting the volume. In our specific case, however, we have opted not to do so. On the one hand, we are able to keep the containers running for the period of the exercise with our user counts. On the other hand the files are no longer needed after the exercise is finished. However, if the learner want to persist the data, he can do so using the export function of JupyterLab. To avoid having to download individual files, a plugin is installed which downloads the files bundled in a zip archive.

Our objective is to process different artifacts within a single notebook (e.g. MontiCore grammars, handwritten Java classes). The fundamental idea is that JupyterLab orchestrates a language project in the background. This task is performed by a custom kernel developed by us. However, the code cell type is the only cell type that is passed to the kernel for execution. By default, JupyterLab does not distinguish between different types of code cells. Therefore a possibility had to be created either to send additional meta information about the cell content or to interpret cell's artifact type by the kernel. For some artifacts we need additional meta information in any case, which the kernel is unable to compute. For example, if a partial model is to be parsed in a cell, the kernel needs the meta-information which production of the grammar it should try to parse. This information should not be visibly included in the cell in advance, nor should the learner be forced to write it into the cell. For this reason we decided to develop a front-end plugin instead for a more robust solution. The plugin can be seen on the left in Figure 1 and ensures that the required metadata is sent to the kernel in addition to the cell content. In this way, only internally necessary information is hidden for the tutorial user.

7 Jupyter Notebook in SLE Teaching

By integrating explanations of concepts into an environment where they can be immediately applied, JupyterLab unlocks new opportunities to teach SLE. The following sections explain how JupyterLab was used to contribute to the teaching of SLE. Section 7.1 states details on building Jupyter notebooks for learning a DSL. Section 7.2 takes it a step further by explaining how engineering of DSLs can also be supported.

7.1 Using Jupyter Notebook for using a DSL

We have used Jupyter Notebook in the practical courses of the lectures Model-Based Software Engineering, and Model-Based Systems Engineering in the winter semesters 2020/21 and 2021/22 with about 60 students in each course. Complex domains were already modeled with MontiCore DSLs. It has been shown that expressive languages can result from this process, whose domain should be introduced iteratively according to our experience. For this reason, we have introduced the tutorial with markdown cells in which the domain is motivated. Building on this, terminology was introduced gradually. By transferring fragments of semantic statements of the domain into the syntax of the DSL via explanatory texts, keywords of the language and the syntax in which they are applied was taught. In code cells following the explanation, learners were able to immediately put the explanations into practice. By running the cell, they got instant feedback on whether their partial model conforms to the language. This procedure was repeated until the complete syntax was learned.

At this point, the tutorial user was able to create syntactically correct models, but those might be semantically invalid. Explanations in the notebook should point out this fact. By configuring a CoCo check in the front-end plugin by the tutorial creator in the subsequent cells, in addition to parsing the model, the semantic violations were experienced. The creators had the possibilities to deactivate individual CoCos. After the tutorial user created valid models, he was introduced to language tooling. Our plugin allows to configure a number of different operations on cell execution. The operations were sequenced one after the other. For example, if the parsing of a model was successful, CoCos were executed and upon success, the cell contents was used as input for a script or other language tooling. In this way, the generator generated artifacts based on the model. This can be multimedia content that is rendered below the cell or other kind of files that could be inspected via the file browser of JupyterLab. The notebook created during this process was saved by the tutorial user (e.g., as a pdf) and served as documentation for his learning achievements.

7.2 Using Jupyter Notebook for the engineering of a DSL

We have used Jupyter Notebook in the practical course of the lecture Software Language Engineering in the summer semesters 2021 with about 60 students and in a conference tutorial

at the Modellierung 2020 [Hö20]. Teaching the engineering of a DSL is fundamentally more challenging than using one (see differences in the cognitive processes for apply and create [AK01]). The language developer must consider not only the syntax of the resulting language, but also its maintainability [FR07, KRR18]. The language workbench MontiCore, whose handling must be trained, contains concepts to accomplish such a DSL. We used a similar approach as in Section 7.1 to getting started with MontiCore.

The learning process starts with a motivating introduction to SLE and the relevance of tailored DSLs. The objective of developing a DSL with the help of the notebook is formulated. A transition to the language workbench MontiCore follows. The MontiCore grammar is introduced as a central artifact of the engineering process. Through further explanations, the syntax of productions is introduced. To make the effect of a production experience-able, a code cell follows in which a grammar with a simple production is already given. Using our plugin it is possible to protect the cell from user modifications. When the cell is executed, MontiCore generates the infrastructure associated with the grammar. In the cell output area, a report appears showing the changes made to the underlying project by the generation step. The generated files are inspected via the file browser on demand. The subsequent markdown cell motivates the tutorial user to enter a partial model that matches the production. Upon execution, the learner is informed whether the input is valid and otherwise faced with a clear error message of the parser. In this way, the basic syntax of the grammar definition is introduced iteratively. At the same time, the tutorial user understands the impact of grammar changes on the generated infrastructure. The learner extends the given grammar with his own productions. The user investigates the correctness of the user-defined grammar by using predefined cells containing valid models. The underlying language project is in a valid state at all time which is ensured by our kernel.

After the grammar definition is completed, context sensitive conditions are introduced. The reason why these cannot be expressed in a context-free grammar is explained. In this advanced part of the tutorial, one can refer to Jupyter Notebooks which explains Java in its basics. Furthermore, inexperienced tutorial users can be supported by partially pre-filled code blocks in the code cell. The level of support was adjusted to the experience of the participants (see Req. E2). For example, our kernel is able to add package definitions or imports on its own. Furthermore, syntax highlighting and an integrated Java parser is used to communicate the source of errors. To avoid that a syntactically incorrect CoCo interrupts the generation process, the kernel adds it to the language project only if the Java parser accepts it. Handwritten extensions via the TOP mechanism are handled analogously. Finally, predefined models are used to check whether all checks have been implemented correctly. The language project created in this manner can be viewed by the tutorial user both in JupyterLab and exported to an IDE of his choice.

8 Lessons Learned and Discussion

The concept of Jupyter Notebooks allows to introduce the domain alongside the associated syntax of the DSL iteratively in markdown cells in an integrated manner. The insertion of code cells allow the practitioner to get immediate feedback when applying the acquired knowledge (see Req. U1, U2, U3). In the same manner language quality criteria can be teached without the need for prior tool experience (see Req. E1, E2). The custom developed kernel allows to respond upon a cell execution as needed and thus enables the integration of further language tooling (see Req. U5). For example, the kernel optionally outputs details of the generation process. JupyterLab, in turn, allows the generated artifacts to be inspected directly (see Req. E3). At the same time, the web-based approach eliminates the initial setup effort for the learner (see Req. U4).

The approach presented has already been used in three practical courses and in a conference tutorial. Lessons could be learned in the process of their realization. The lessons learned can be categorized into the perspective of the tutorial creator and the tutorial user.

Lesson learned 1: Initial setup effort. On the one hand the initial setup effort is increased for the tutorial creator. On the other hand, the setup process is completely eliminated for the tutorial user (see Req. U4). A custom kernel and a front-end extension have been implemented. An initial setup effort for the infrastructure is also required. However, once the containerized environment is established, it can be immediately reused in other contexts. This significantly reduces the effort required for follow-up courses which want to make use of the Jupyter infrastructure.

In addition, the tutorial content also needs to be created. A tutorial creator needs to get familiar with JupyterLab and the concept of interactive teaching of knowledge. However, a strategy for communicating the topics must be found in any case.

Lesson learned 2: Effort during execution. The unified development environment has resulted in significantly fewer inquiries from participants regarding the setup. In case of technical problems, the tutorial user have either been able to reset their environment themselves or have been able to ask an instructor to do so. The kernel ensures a valid state of the language project, therefore this case generally rarely occurs.

In one case, we accessed a user's container to reproduce his error pattern. It was caused by a bug in the kernel, which could be fixed by a redeployment. Our software architecture allowed us to apply the fix immediately for all users. In previous teaching units without JupyterLab, this would have required active actions from users. Furthermore, it would have been difficult to determine whether all participants had correctly applied the fix.

Lesson learned 3: Maintenance effort. The hosting of a server is accompanied by necessary maintenance work. Updates for JupyterLab, JupyterHub and other packages have to be installed. Since we want to provide the latest functionalities of MontiCore to the participants,

the kernel has to be maintained as well. Accordingly, the tutorial content may need to be extended in the future. However, this would also be the case in the traditional exercise mode.

Lesson learned 4: Required hardware resources. Developing models for a given DSL is not very resource intensive. However, as soon as sophisticated language infrastructure is used in the notebook, the CPU requirement increases significantly. Accordingly, the use of the MontiCore generator leads to an increase in the load of the system. The format of the exercise operation causes that at certain times a large number of users work on the system simultaneously. However, in other time frames, the server's capacity is idle. The load scenario advocates the use of a scalable cloud infrastructure.

Lesson learned 5: Overall feedback. Feedback on the use of Jupyter has been positive, both from tutorial creators and tutorial users. There are evaluations of the courses realized, which indicate a positive acceptance of the Jupyter Notebooks. Nevertheless, a full evaluation of the teaching approach has not yet been conducted by us. It has also been shown that at a certain experience level of the tutorial users a migration to a real IDE is desired. This is reasonable, for example, when complex language infrastructure is to be realized.

The availability of an interactive solution to teach SLE has prompted requests for more tutorial content. Since the expected user base is growing, we decided to move the solution to a kubernetes cluster.

9 Conclusion

Within this paper, we have shown how a technical infrastructure, namely JupyterLab, and its usage to teach the use and engineering of domain-specific languages. We have presented requirements for teaching DSLs and discussed them. Moreover, we have discussed lessons learned from its application in several practical courses and one conference tutorial.

To sum up, we think that JupyterLab is well useable for teaching the use and engineering of DSLs if one can cope with the increased need for server resources. The main advantages of this approach for *students* are that they have less set-up of the technology and can focus on the main teaching goals, they have information and input fields integrated on one screen, and they get interactive response of their solutions. The main advantages for *teachers* are that they need less technical supervision after the initial set-up and can focus on teaching DSL engineering and use. Moreover, they can reuse the Jupyter tutorial for following instances of the same practical course.

Literaturverzeichnis

[Ad18] Adam, Kai; Butting, Arvid; Kautz, Oliver; Pfeiffer, Jerome; Rumpe, Bernhard; Wortmann, Andreas: Retrofitting Type-safe Interfaces into Template-based Code Generators. In: 6th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD'18). SciTePress, S. 179 – 190, 2018.

[Ag00] Aggarwal, Anil K: Web-based Learning and Teaching Technologies: Opportunities and Challenges, Idea Group Publishing. Inf. Soc., 20(2):153–154, 2000.

[AK01] Anderson, Lorin W; Krathwohl, David R: A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives. Longman, 2001.

[Ba20] Barash, Mikhail: Example-driven software language engineering. In: Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering. S. 246–252, 2020.

[Bl19] Blank, Douglas S; Bourgin, David; Brown, Alexander; Bussonnier, Matthias; Frederic, Jonathan; Granger, Brian; Griffiths, Thomas L; Hamrick, Jessica; Kelley, Kyle; Pacer, M et al.: nbgrader: A tool for creating and grading assignments in the Jupyter Notebook. The Journal of Open Source Education, 2(11), 2019.

[Bo21] Bork, Dominik; Fend, Andreas; Scheffknecht, Dominik; Kappel, Gerti; Wimmer, Manuel: From In-Person to Distance Learning: Teaching Model-Driven Software Engineering in Remote Settings. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). IEEE, S. 702–711, 2021.

[BS12] Broy, Manfred; Stølen, Ketil: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer Science & Business Media, 2012.

[Bu19] Butting, Arvid; Eikermann, Robert; Kautz, Oliver; Rumpe, Bernhard; Wortmann, Andreas: Systematic Composition of Independent Language Features. Journal of Systems and Software, 152:50–69, June 2019.

[BVG18] Burgueño, Loli; Vallecillo, Antonio; Gogolla, Martin: Teaching UML and OCL models and their validation to software engineering students: an experience report. Computer Science Education, 28(1):23–41, 2018.

[Cl15] Clark, Tony; Brand, Mark van den; Combemale, Benoit; Rumpe, Bernhard: Conceptual Model of the Globalization for Domain-Specific Languages. In: Globalizing Domain-Specific Languages. LNCS 9400. Springer, S. 7–20, 2015.

[CMP18] Czech, Gerald; Moser, Michael; Pichler, Josef: Best Practices for Domain-Specific Modeling. A Systematic Mapping Study. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, S. 137–145, 2018.

[Dr20] Drave, Imke; Henrich, Timo; Hölldobler, Katrin; Kautz, Oliver; Michael, Judith; Rumpe, Bernhard: Modellierung, Verifikation und Synthese von validen Planungszuständen für Fernsehausstrahlungen. In: Modellierung 2020. GI, S. 173–188, 2020.

[FK13] Fill, Hans-Georg; Karagiannis, Dimitris: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. Enterprise Modelling and Information Systems Architectures (EMISAJ), 8(1):4–25, 2013.

[FR07] France, Robert; Rumpe, Bernhard: Model-driven Development of Complex Software: A Research Roadmap. Future of Software Engineering (FOSE '07), S. 37–54, May 2007.

[GB21] Glaser, Philipp-Lorenz; Bork, Dominik: The bigER Tool-Hybrid Textual and Graphical Modeling of Entity Relationships in VS Code. In: 2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW). IEEE, S. 337–340, 2021.

[GM18] Gonnord, Laure; Mosser, Sébastien: Practicing domain-specific languages: from code to models. In: 21st ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Companion Proceedings. S. 106–113, 2018.

[GP21] Granger, Brian E.; Pérez, Fernando: Jupyter: Thinking and Storytelling With Code and Data. Comput. Sci. Eng., 23(2):7–14, 2021.

[GRR09] Grönniger, Hans; Ringert, Jan Oliver; Rumpe, Bernhard: System Model-Based Definition of Modeling Language Semantics. In: Formal techniques for distributed systems, S. 152–166. Springer, 2009.

[Gu21] Gupta, Rohit; Kranz, Sieglinde; Regnat, Nikolaus; Rumpe, Bernhard; Wortmann, Andreas: Towards a Systematic Engineering of Industrial Domain-Specific Languages. In: IEEE/ACM 8th Int. WS on Software Eng. Research and Industrial Practice (SE&IP). IEEE, 2021.

[Ha15] Haber, Arne; Look, Markus; Mir Seyed Nazari, Pedram; Navarro Perez, Antonio; Rumpe, Bernhard; Völkel, Steven; Wortmann, Andreas: Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components. In: Model-Driven Engineering and Software Development Conf. (MODELSWARD'15). SciTePress, 2015.

[He07] Herrmann, Christoph; Krahn, Holger; Rumpe, Bernhard; Schindler, Martin; Völkel, Steven: An Algebraic View on the Semantics of Model Composition. In: Conf. on Model Driven Arch. - Foundations and Applications (ECMDA-FA'07). LNCS 4530. Springer, 2007.

[HKR21] Hölldobler, Katrin; Kautz, Oliver; Rumpe, Bernhard: MontiCore Language Workbench and Library Handbook: Edition 2021. Aachener Informatik-Berichte, Software Engineering, Band 48. Shaker Verlag, May 2021.

[Hö19] Hölldobler, Katrin; Michael, Judith; Ringert, Jan Oliver; Rumpe, Bernhard; Wortmann, Andreas: Innovations in Model-based Software and Systems Engineering. The Journal of Object Technology, 18(1):1–60, July 2019.

[Hö20] Hölldobler, Katrin; Jansen, Nico; Rumpe, Bernhard; Wortmann, Andreas: Komposition Domänenspezifischer Sprachen unter Nutzung der MontiCore Language Workbench, am Beispiel SysML 2. In: Modellierung 2020. GI, S. 189–190, 2020.

[HR04] Harel, David; Rumpe, Bernhard: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 37(10):64–72, October 2004.

[JH17] Johanson, Arne N.; Hasselbring, Wilhelm: Effectiveness and Efficiency of a Domain-Specific Language for High-Performance Marine Ecosystem Simulation: A Controlled Experiment. Empirical Software Engineering, 22(4):2206–2236, 2017.

[KLR96] Kelly, Steven; Lyytinen, Kalle; Rossi, Matti: Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In: International Conference on Advanced Information Systems Engineering. Springer, S. 1–21, 1996.

[KRR18] Kautz, Oliver; Roth, Alexander; Rumpe, Bernhard: Achievements, Failures, and the Future of Model-Based Software Engineering. In: The Essence of Software Engineering, S. 221–236. Springer, 2018.

[LH01] Lin, Binshan; Hsieh, Chang-tseh: Web-based teaching and learner control: A research review. Computers & Education, 37(3-4):377–386, 2001.

[Ma14] Maróti, Miklós; Kecskés, Tamás; Kereskényi, Róbert; Broll, Brian; Völgyesi, Péter; Jurácz, László; Levendovszky, Tihamer; Lédeczi, Ákos: Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. MPM@ MoDELS, 1237:41–60, 2014.

[Ma20] Manzoor, Hamza; Naik, Amit; Shaffer, Clifford A; North, Chris; Edwards, Stephen H: Auto-grading jupyter notebooks. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education. S. 1139–1144, 2020.

[PG07] Pérez, Fernando; Granger, Brian E: IPython: a system for interactive scientific computing. Computing in science & engineering, 9(3):21–29, 2007.

[PG15] Perez, Fernando; Granger, Brian E: Project Jupyter: Computational narratives as the engine of collaborative data science. Retrieved September, 11(207):108, 2015.

[Pr21] Prinz, Andreas: Teaching Language Engineering Using MPS. In: Domain-Specific Languages in Practice, S. 315–336. Springer, 2021.

[Re12] Reisig, Wolfgang: Petri Nets: An Introduction, Jgg. 4. Springer Science & Business Media, 2012.

[RG02] Richters, Mark; Gogolla, Martin: OCL: Syntax, semantics, and tools. In: Object Modeling with the OCL, S. 42–68. Springer, 2002.

[Ro18] Rodriguez-Echeverria, Roberto; Izquierdo, Javier Luis Cánovas; Wimmer, Manuel; Cabot, Jordi: Towards a language server protocol infrastructure for graphical modeling. In: 21th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems. 2018.

[RTS19] Rosenthal, Kristina; Ternes, Benjamin; Strecker, Stefan: Learning Conceptual Modeling: Structuring Overview, Research Themes and Paths for Future Research. In: 27th European Conference on Information Systems - Information Systems for a Sharing Society, ECIS 2019, Stockholm and Uppsala, Sweden, June 8-14. 2019.

[Ru21] Rumpe, Bernhard; Michael, Judith; Kautz, Oliver; Krebs, Roland; Gandenberger, Sabine; Standt, Janos; Weber, Uli: Digitalisierung der Gesetzgebung zur Steigerung der digitalen Souveränität des Staates, Jgg. 19 in Berichte des NEGZ. Nationales E-Government Kompetenzzentrum e. V., June 2021.

[SBS20] Schlichthaerle, Stefan; Becker, Klaus; Sperber, Sebastian: A Domain-Specific Language Based Architecture Modeling Approach for Safety Critical Automotive Software Systems. In: Software Engineering Workshops 2020. CEUR-WS.org, 2020.

[St15] Staubitz, Thomas; Klement, Hauke; Renz, Jan; Teusner, Ralf; Meinel, Christoph: Towards practical programming exercises and automated assessment in Massive Open Online Courses. In: 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE). IEEE, S. 23–30, 2015.

[Sy13] Syriani, Eugene; Vangheluwe, Hans; Mannadiar, Raphael; Hansen, Conner; Van Mierlo, Simon; Ergin, Huseyin: AToMPM: A web-based modeling environment. In: Joint Proc. of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition. S. 21–25, 2013.

[Te19] Ternes, Benjamin; Strecker, Stefan; Rosenthal, Kristina; Barth, Hagen: A browser-based modeling tool for studying the learning of conceptual modeling based on a multi-modal data collection approach. In: Human Practice. Digital Ecologies. Our Future. 14. Internationale Tagung Wirtschaftsinformatik (WI 2019), February 24-27, 2019, Siegen, Germany. University of Siegen, Germany / AISeL, S. 1984–1988, 2019.

[TK16] Tolvanen, Juha-Pekka; Kelly, Steven: Model-driven development challenges and solutions: Experiences with domain-specific modelling in industry. In: 4th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD). IEEE, S. 711–719, 2016.

[Ul13] Ulrich, Frank: Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In: Domain engineering, S. 133–157. Springer, 2013.

[Va20] Van Binsbergen, L Thomas; Verano Merino, Mauricio; Jeanjean, Pierre; Van Der Storm, Tijs; Combemale, Benoit; Barais, Olivier: A principled approach to REPL interpreters. In: ACM SIGPLAN Int. Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software. S. 84–100, 2020.

[VP12] Voelter, Markus; Pech, Vaclav: Language modularity with the MPS language workbench. In: 34th Int. Conf.e on Software Engineering (ICSE). IEEE, S. 1449–1450, 2012.

[Wi96] Wirth, Niklaus: Extended Backus-Naur Form (EBNF). Iso/Iec, 14977(2996):2–21, 1996.

[Wo21] Wolschke, Christian; Marksteiner, Stefan; Braun, Tobias; Wolf, Markus: An Agnostic Domain Specific Language for Implementing Attacks in an Automotive Use Case. In: 16th Int. Conf. on Availability, Reliability and Security (ARES 2021). ACM, 2021.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 111

Open-World Loose Semantics of Class Diagrams as Basis for Semantic Differences

Imke Nachmann, Bernhard Rumpe, Max Stachon and Sebastian Stüber [1]

Abstract: Class Diagrams (CDs) model data structures in object-oriented systems and evolve throughout the course of the development process. Analyzing the semantic differences between consecutive versions of a CD is crucial to detect unintended changes of the modeled structures and involves comparing the sets of valid object models of both CDs. Established definitions of CD-semantics employ a closed-world assumption for the validity of object structures, which may not fit all stages of the development process. In this paper, we provide different definitions of CD-semantics and discuss their validity, relationship and limitations in the context of semantic evolution analysis. We show that the closed-world semantics of a CD is a subset of its open-world semantics and how this can be used for analyzing model-evolution. We also consider objects both as simple datastructures, as well as instances of (super-)classes and interfaces, and analyze how these approaches affect refinement and refactoring.

Keywords: Class Diagrams; Open-World Semantics; Closed-World Semantics; Semantic Differences

1 Introduction

Class Diagrams (CDs) are the most widely used Unified Modeling Language (UML)-Models in industry and research [DP06, La14, Hu11]. A CD describes the structure of an object-oriented system by means of classes, attributes and associations. In Model-Driven Engineering (MDE) CDs are used to generate code in general programming languages such as Java or C# [SN11]. Often CDs are used in combination with other UML models such as Statecharts or Sequence Diagrams [PSB11, SKM07, SS19, Sw12]. Furthermore, CDs can be used to generate graphical interfaces [Ge21].

As CDs are one of the primary artifacts in the development process, they are also frequently modified. Syntactic changes are made due to changing requirements, bug fixes, or when moving to a next design level. Effective change management is a major concern in MDE and not yet fully addressed. Nowadays, only syntactic differencing operators are widely established [AP03, KKT11, T.13, KGE09, Kü08, Ta14, TK09].

However, they do not show the impact on the model's semantics, i. e. its meaning [HR04]. The UML-Definition [Ma15] contains a syntactical description of CDs, but is vague on

[1] RWTH Aachen, Lehrstuhl für Software-Engineering, Ahornstraße 55, 52074 Aachen, Germany,
nachmann@se-rwth.de, rumpe@se-rwth.de, stachon@se-rwth.de, stueber@se-rwth.de
Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) - Projektnummer 250902306

doi:10.18420/modellierung2022-016

the meaning of the diagrams. The semantics of a CD is the set of all object structures representing legal instances of the CD [MRR11a]. A semantic difference of two CDs is the set of all object structures representing a legal instance of the first and no legal instance of the second CD [MRR10, MRR11a]. The absence of such object structures implies that the first CD *refines* the latter [MRR10].

The *closed-world* assumption [Re78] requires legal instances to only instantiate classes and associations introduced in the CD (e. g. [BM13, MRR11a]). This becomes problematic during the analysis of systems, when the CDs are underspecified, as not all information is captured. Following analysis steps can add new classes or associations, which where not part of the initial analysis. In the closed-world semantics, this is not a refinement and leads to a semantic difference.

In contrast, the *open-world* assumption allows, whatever is not explicitly forbidden, i. e. legal instances may instantiate classes or associations not used in the CD as long as all CD constraints are satisfied. Adding new information (e. g. an additional class) leads to a refinement in open-world semantics.

Furthermore, in addition to the distinction between open-world and closed-world approaches, CD-semantics definitions may differ in their semantic domain, e. g., in [MRR11a], objects within an object structure are viewed simply as data structures and only instantiate a single class each. The superclasses and implemented interfaces of this instantiated class impact the semantics only indirectly via inherited attributes and associations. This definition of semantics is valid and useful, especially under a closed-world assumption, as it permits meaningful syntactical changes that do not change the semantics of a CD, as we will show in the following section. However, this approach becomes problematic when we consider the use of objects as method arguments. There, the instantiated superclasses and interfaces are relevant for type checking. In this situation, a semantic domain where objects are instances of multiple classes and interfaces is more appropriate.

This paper presents and compares open- and closed-worlds semantics definition for CDs with both single-instance objects, as well as multi-instance objects. We give examples that show the limitations of a closed-world approach and the utility of open-world CD-semantics for system-analysis in early developments stages.

In the remainder, Sect. 2 presents motivating examples, which demonstrate the limitations of closed-world semantics and motivates inheritance information in object structures. Sect. 3 introduces the abstract syntax of CDs, which is used in Sect. 4 to define multiple semantic mappings. These different semantics are compared in Sect. 5 and Sect. 6 debates related work. Finally, we conclude in Sect. 7.

2 Motivating Examples

Consider the CD $cd1$ in Fig. 1, it consists of the classes `Professor`, `Lecture` and `Student`. `Professors` hold at least one `Lecture` and each `Lecture` is held by exactly one `Professor`. A `Lecture` may be attended by multiple `Students` and a `Student` may attend multiple `Lectures`. We modify the CD by adding another class `Room`. Each `Lecture` is now required to take place in at least one *Room* and each `Room` can host multiple *Lectures*. In early development this new CD $cd2$ would be understood as a refinement of $cd1$, since adding new elements simply adds new requirements on top of the already existing requirements regarding valid object structures. We would therefore consider the semantics of $cd2$ a subset of the semantics of $cd1$. This is the case under an open-world assumption. However, under a closed-world assumption the semantics of $cd1$ prohibits instances of any class not explicitly modeled in the CD. Thus, $om1$ is in the closed-world semantics of $cd1$ but not $cd2$, and $om2$ is in the closed-world semantics of $cd2$ but not $cd1$. We therefore say that the closed-world semantics of $cd1$ and $cd2$ are incomparable.

Now, consider the CDs $cd3.1$ and $cd3.2$ in Figure 2. If we view objects simply as data-structures that instantiate a single class, these two CDs would be semantically equivalent under a closed-world assumption. $cd3.1$ would be considered a refactoring of $cd3.2$ and vice-versa. If instead we consider a semantics definition that views objects as instances of multiple classes, we would find $cd3.1$ and $cd3.2$ to have incomparable semantics under a closed-world assumption, as $om3.1$ would be in the closed-world semantics of $cd3.1$ but not $cd3.2$ and $om3.2$ would be in the closed-world semantics of $cd3.2$ but not $cd3.1$. However, under an open-world assumption $cd3.1$ would be considered a strict refinement of $cd3.2$, as the open-world semantics of $cd3.1$ does not restrict instances of the class *Employee*.

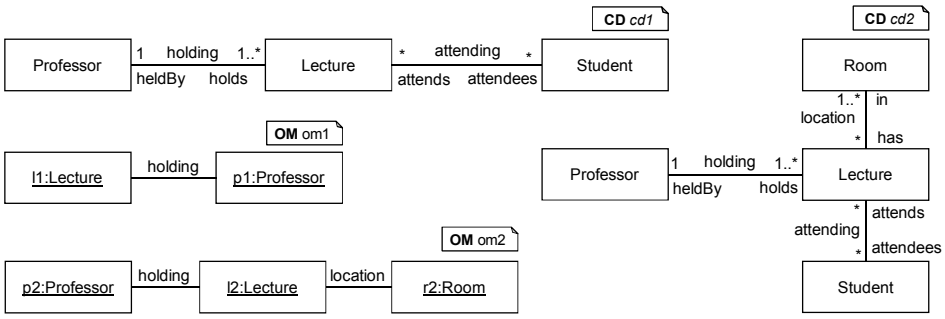


Fig. 1: Running examples of CDs and object structures

3 Class Diagrams and Object Structures

In the following, we use a variant of CDs which describes object structures in terms of classes, associations, roles, extends relations, and multiplicity constraints. An object

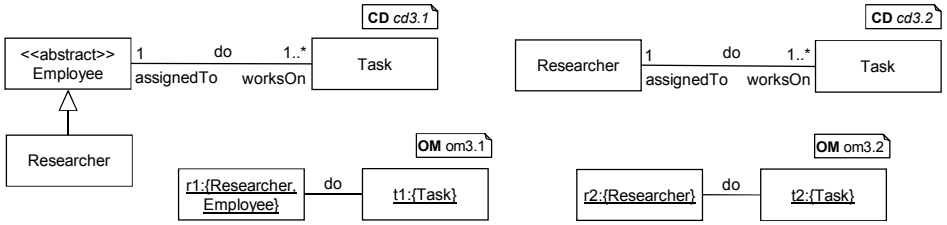


Fig. 2: Examples of CDs. Object structures include inheritance information

structure represents a possible data state of a system by means of objects and links between them [Ru16]. Those object structures that satisfy all constraints prescribed by a CD, are the elements of the CD's semantics.

Notation. Let A be a set. Then, $|A|$ denotes the cardinality of the set A. Further, $\wp_2(A)$ denotes the set of two-element subsets of A and $\wp_{\text{fin}}(A)$ denotes the set containing all non-empty finite subsets of A. We denote the set of natural numbers including 0 by $\mathbb{N}_0$, and the set of natural numbers including 0 and the infinity symbol by $\mathbb{N}_\infty$. The relation $\leq$ is the natural total ordering on this set. Given a binary relation R, we denote the reflexive transitive hull of that relation by R^*.

3.1 An Abstract Syntax for Class Diagrams

From now on, let $\mathcal{C}^U$ be a universe of class names, $\mathcal{R}^U$ be a universe of role names, and $\mathcal{A}^U$ be a universe of association names. The following function assigns two roles to every association. We assume that associations describe exactly two roles. Roles are not shared among different associations. To this effect, $rls : \mathcal{A}^U \to \wp_2(\mathcal{R}^U)$ maps each association to its roles. Since roles are not shared, we require $rls(a_1) \cap rls(a_2) = \emptyset$ for every two associations $a_1, a_2 \in \mathcal{A}^U$ with $a_1 \neq a_2$. The assumption that role names are not shared does not imply a loss of generality, as, e. g. prepending the association's name to every role name of the association yields the required property. For example, consider the CD $cd3.1$ in Figure 2 and its association $\texttt{do} \in \mathcal{A}^U$. The two roles of do are $rls(\texttt{do}) = \{\texttt{assignedTo}, \texttt{worksOn}\}$. A CD consists of a finite set of classes, a finite set of associations with multiplicity constraints on both association ends, and a finite set of extends relations between the classes. The CD assigns the roles of each associations a unique class. We need not consider attributes, as they can be simulated by corresponding 1-to-1 associations. Similarly, interfaces are not explicitly regarded, as they would be semantically equivalent to abstract classes in our semantics definitions. Finally, our syntax and semantics definitions omit methods, since they require behavioural analysis, and we are primarily concerned with object structures.

Definition 3.1 (Class Diagram) *A class diagram is a tuple* $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$*, where* ***(1)*** $C \subseteq C^U$ *is a finite set of classes,* ***(2)*** $\mathcal{A}bs \subseteq C$ *is a set of abstract classes,* ***(3)*** $\mathcal{A} \subseteq \mathcal{A}^U$ *is a finite set of associations,* ***(4)*** *the function* $cls : rls(\mathcal{A}) \to C$ *assigns each role of each association a unique class,* ***(5)*** C *contains all classes of all roles of all associations, i. e. for all associations* $a \in \mathcal{A}$ *and all roles* $r \in rls(a)$ *of a, it holds that* $cls(r) \in C$*,* ***(6)*** $\mathcal{M} = \{min_a^{cd}, max_a^{cd} \mid a \in \mathcal{A}\}$ *is a finite set containing a minimum multiplicity constraint* $min_a^{cd} : rls(a) \to \mathbb{N}_0$ *and a maximum multiplicity constraint* $max_a^{cd} : rls(a) \to \mathbb{N}_\infty$ *for each association* $a \in \mathcal{A}$*,* ***(7)*** $\mathcal{H} \subseteq \{c_1 \preccurlyeq c_2 \mid c_1, c_2 \in C\}$ *is a finite set of extends relations between the classes in* C *such that* ***(8)*** $\mathcal{H}^*$ *is a partial ordering on* C*,* ***(9)*** *and for any* $c_1 \preccurlyeq c_2 \in \mathcal{H}$*, it holds that* $c_1 \in \mathcal{A}bs \implies c_2 \in \mathcal{A}bs$*.*

In a CD $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, for each association $a \in \mathcal{A}$, and each role $r \in rls(a)$ of the association, $min_a^{cd}(r) \in \mathbb{N}_0$ defines the minimum multiplicity of the class $cls(r)$ with role r required by association a in the CD cd. Similarly, $max_a^{cd}(r) \in \mathbb{N}_\infty$ defines the maximum multiplicity of the class $cls(r)$ with role r required by the association a in the CD cd. For instance, the abstract syntax of the CD $cd3.1$ depicted in Figure 2 can be formalized as $cd3.1 = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, where $C = \{\texttt{Employe}, \texttt{Task}, \texttt{Researcher}\}$, $\mathcal{A}bs = \{Employee\}$, $\mathcal{A} = \{\texttt{do}\}$, $cls = \{\texttt{assignedTo} \mapsto \texttt{Employee}, \texttt{worksOn} \mapsto \texttt{Task}\}$, $\mathcal{M} = \{min_{\texttt{do}}^{cd3.1}, max_{\texttt{do}}^{cd3.1}\}$, and $\mathcal{H} = \{\texttt{Researcher} \preccurlyeq \texttt{Employee}\}$. The multiplicity constraints in $\mathcal{M}$ are given by the functions $min_{\texttt{do}}^{cd1} = \{\texttt{assignedTo} \mapsto 1, \texttt{worksOn} \mapsto 1\}$, $max_{\texttt{do}}^{cd1} = \{\texttt{assignedTo} \mapsto 1, \texttt{worksOn} \mapsto \infty\}$.

3.2 Strict Expansion of CDs

We now define the notion of a strict expansion of a class diagram that will serve as the basis for an open-world semantic definition in Sect. 4.2. Under an open-world assumption, CDs are considered underspecified. Adding a new class, association or extends relation to a CD should therefore be considered a refinement, since we are specifying previously underspecified parts of the CD. This should also be the case for moving an association to a superclass and changing a non-abstract class into an abstract class.

Definition 3.2 *We say that a class diagram* $cd' = (C', \mathcal{A}bs', \mathcal{A}', cls', \mathcal{M}', \mathcal{H}')$ *is a strict expansion of another class diagram* $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ *iff* ***(1)*** $C \subseteq C'$*,* $\mathcal{A}bs \subseteq \mathcal{A}bs'$*,* $\mathcal{A} \subseteq \mathcal{A}'$*,* $\mathcal{M} \subseteq \mathcal{M}'$*,* $\mathcal{H} \subseteq \mathcal{H}'$ *and* ***(2)*** *for all* $a \in \mathcal{A}$ *and* $r \in rls(a)$*, we find that* $cls(r) \preccurlyeq cls'(r) \in (\mathcal{H}')^*$ *Furthermore, we use* $EXP(cd)$ *to denote the set of all strict expansions of* cd*.*

3.3 Semantic Domain

Similar to [MRR11a], we define the semantics of a CD as the set of object structures, which contains all object structures permitted by the CD. The abstract syntax of object

structures comprises objects and links between them. Unlike [MRR11a], we consider both a semantic domain based on single-instance objects, as well as one consisting of multi-instance objects. To this effect, let O^U denote a universe of objects. A *link* of type $a \in \mathcal{A}^U$ connects two objects via the roles $rls(a)$. For example, consider the object structure $om3.2$ depicted in Figure 2. Formally, the link connecting the `Researcher`-object `r2` to the `Task`-object `t2` is represented by the tuples (`r2`, `worksOn`, `do`, `t2`, `assignedTo`) and (`t2`, `assignedTo`, `do`, `r2`, `worksOn`). The two are necessary to obtain undirected links.

Definition 3.3 (Object Structure) *An object structure is a tuple* $om = (O, \text{class}, \mathcal{L})$*, where (1)* $O \subseteq O^U$ *is a finite set of objects, (2)* $\text{type} : O \to \wp_{\text{fin}}(C^U)$ *is a function that assigns a finite set of classes to each object. (3)* $\mathcal{L} \subseteq O \times \mathcal{R}^U \times \mathcal{A}^U \times O \times \mathcal{R}^U$ *is a finite set of links such that for all* $(o, r, a, o', r') \in \mathcal{L}$ *it holds that a)* $r, r' \in rls(a)$ *and* $r \neq r'$*, and b)* $(o', r', a, o, r) \in \mathcal{L}$*.*

The constraint (a) assures that the roles used by a link that instantiates an association $a \in \mathcal{A}^U$, correspond to the roles of the association. The constraint (b) assures undirectedness by requiring both link directions to be included in the set of links. For instance, the abstract syntax of the object structure $om3.1$ depicted in Figure 2 is given by $om3.1 = (O, \text{class}, \mathcal{L})$ where $O = \{\texttt{r1}, \texttt{t1}\}$, type $= \{\texttt{r1} \mapsto \{\texttt{Researcher}, \texttt{Employee}\}, \texttt{t1} \mapsto \{\texttt{Task}\}, \}$, $\mathcal{L} = \{(\texttt{r1}, \texttt{worksOn}, \texttt{do}, \texttt{t1}, \texttt{assignedTo}), (\texttt{t1}, \texttt{assignedTo}, \texttt{do}, \texttt{r1}, \texttt{worksOn})\}$

Multiplicity constraints affiliated with an association restrict, for every object, the number of outgoing links of this association type. In an object structure om, the auxiliary function $\mathbb{L}^{om}$ captures these links: $\mathbb{L}^{om}(o, a, r) \stackrel{\text{def}}{=} \{(o, r, a, o', r') \in \mathcal{L} \mid o' \in O \wedge r' \in rls(a) \setminus \{r\}\}$.

Consider the object structure $om1$ in Figure 1. The set of links of association type `holding` connecting the object `l` via role `heldBy` to `Professor`-objects is $\mathbb{L}^{om2}(\texttt{l}, \texttt{holding}, \texttt{holds}) = \{(\texttt{l}, \texttt{heldBy}, \texttt{holding}, \texttt{p}, \texttt{holds})\}$. The set of `attending`-links connecting the object `l` to `Student`-objects is the empty set: $\mathbb{L}^{om2}(\texttt{l}, \texttt{attending}, \texttt{attends}) = emptyset$.

A *simple object structure* $om = (O, \text{class}, \mathcal{L})$ is an object structure with $|class(o)| = 1$ for all $o \in O$. We may use simple object structure as the semantic domain of a CD-semantics definition that does not consider the superclasses instantiated by an object. The term "simple" is in reference to simple graphs from graph theory.

4 Semantics of Class Diagrams

The semantics of a CD cd, denoted by $[\![cd]\!]$, is the set of all valid object structures. A constraint-based approach to semantics considers an object structure as valid with respect to a CD iff the object structure satisfies the constraints prescribed by the CD. The semantic mapping involves a definition of which constraints a CD prescribes and how an element

of the semantic domain satisfies a constraint. Depending on the interpretation of what the elements of the semantic domain represent, these definitions may differ, which impacts the outcome of a semantic differencing operator, i. e. the set of valid instances of one CD that are not instances of another CD depends strongly on these definitions. This section formally defines a variety of open-world and closed-world semantics for CDs based on this notion.

4.1 Closed-World Semantics of Class Diagrams

This section introduces CD semantics that apply the closed-world assumption. Given a CD $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, an $om = (O, \text{type}, \mathcal{L})$ satisfies the closed-world assumption iff **CW-1** $\forall o \in O$ it holds that $\text{type}(o) \subseteq C$, **CW-2** $\forall (o, r, a, o', r') \in \mathcal{L} : a \in \mathcal{A}$ **CW-3** for all $o \in O$ it holds that $\text{type}(o) \notin \mathcal{A}bs$

That is, objects may only instantiate classes in the CD, links may only instantiate associations in the CD, and no object may instantiate only abstract classes. This section introduces two closed-world semantics of CDs that interpret object structures as simple data structures in which objects are considered instances of a single class. The other definition considers objects as instances of possibly many classes reflecting the notion that if an object instantiates a subclass, it also instantiates the superclass. Considering the semantic domain to represent simple datastructures with single-instance objects, the semantics of a CD only contains simple object structures.

Definition 4.1 (Simple Closed-World CD-Semantics) *Consider a CD in the sense above, i. e.* $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$. *The* simple closed-world semantics*, denoted* $[\![cd]\!]^{sc}$ *is the set of all simple object structures* $om = (O, \text{type}, \mathcal{L})$ *such that* ***(1)*** *om satisfies the closed-world assumptions, i. e. CW-1 to CW-3.* ***(2)*** *for all* $o \in O$ *and* $c_1, c_2 \in C$ *such that* $c_1 \in \text{type}(o)$*, it holds that* $c_1 \preccurlyeq c_2 \in \mathcal{H}^* \implies \forall a \in \mathcal{A}, (r, r') = rls(a)$*, such that* $cls(r') = c_2$*, it holds that* $min_a^{cd}(r) \leq |\mathbb{L}^{om}(o, a, r)| \leq max_a^{cd}(r)$*.* ***(3)*** *for all* $o \in O$ *and* $c_1, c_2 \in C$ *such that* $c_1 \in \text{type}(o)$*, it holds that* $c_1 \preccurlyeq c_2 \notin \mathcal{H}^* \implies \forall a \in \mathcal{A}, r \in rls(a), r' \in rls(a) \setminus \{r\}$*, such that* $cls(r') = c_2$*, it holds that* $\mathbb{L}^{om}(o, a, r) = \emptyset$

The extends relations affect simple object structures only inderectly, i. e. objects and their links have to also satisfy the multiplicity constraints of inherited associations. The Conditions (2), and (3) formalize this: Consider an association a that a subclass c_2 inherits from its superclass c_1 in a CD. Then an object of type c_2 has to fulfill the multiplicity constraints of a, i. e. the number of outgoing links of type a is greater than or equal to $min_a^{cd}(c_2)$ and smaller than or equal to $max_a^{cd}(c_2)$, cf. Condition (2). On the other hand, an object that is not of a subclass type may not have outgoing links that instantiate associations of another class, cf. Condition (3). Simple object structures do not incorporate the notion that instances of a subclass type also instantiate the superclass type. When it comes to semantic

differencing, this may yield results that are incorrect with respect to the understanding of the developer. Therefore, we define a closed-world semantics, that considers the types of an object to be a *set* of classes. Applying the closed-world semantics, the type of an object must include a class and exactly all of its superclasses as specified by the extends relations of the CD.

Definition 4.2 (Multi-Instance Closed-World CD-Semantics) *Consider the CD* $cd = (\mathcal{C}, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$*. The multi-instance closed-world semantics* $[\![cd]\!]^{mc}$ *is the set of all object structures* $om = (\mathcal{O}, \text{type}, \mathcal{L})$ *such that* ***(1)*** *om satisfies the closed–world assumptions, i. e. CW-1 to CW-3.* ***(2)*** *for all* $c_1 \preccurlyeq c_2 \in \mathcal{H}, o \in \mathcal{O}$ *it holds that* $c_1 \in \text{type}(o) \Rightarrow c_2 \in \text{type}(o)$*,* ***(3)*** *for all associations* $a \in \mathcal{A}$*, all roles* $r \in rls(a), r' \in rls(a) \setminus \{r\}$*, and all objects* $o \in \mathcal{O}$ *such that* $cls(r') \in \text{type}(o)$*, it holds that* $min_a^{cd}(r) \leq |\mathbb{L}^{om}(o, a, r)| \leq max_a^{cd}(r)$*.* ***(4)*** *for all associations* $a \in \mathcal{A}$*, all roles* $r \in rls(a), r' \in rls(a) \setminus \{r\}$*, and all objects* $o \in \mathcal{O}$ *such that* $cls(r') \notin \text{type}(o)$*, it holds that* $\mathbb{L}^{om}(o, a, r) = \emptyset$*, and* ***(5)*** $\forall o \in O$ *it holds that* $\forall c_1, c_2 \in \text{type}(o) : c_1 \preccurlyeq c_2 \in \mathcal{H}^*$ *or* $c_2 \preccurlyeq c_1 \in \mathcal{H}^*$*.*

In the above definition, we consider objects, whose type is a *set* of classes. Condition (2) states that objects whose class type includes a subclass must also include all superclasses according to the extends relation of the CD. Condition (3) assures that any object that instantiates a subclass that inherits an association from its superclass obeys the multiplicity constraints prescribed by that association, while Condition (4) assures that an object does not have outgoing links of an association that its class-type does not inherit. Condition (5) assures the correct instantiation of the class-hierarchy prescribed by the extends relations in the CD.

4.2 Open-World Semantics of Class Diagrams

The closed-world assumption reflects the notion that everything which is not modeled by the CD is not allowed. In contrast, the open-world assumption promotes that whatever is not modeled is not restricted and therefore allowed. Open-world semantics definitions therefore, do not restrict object structures to only instantiate elements from the CD. As for the closed-world, there are multiple ways to define open-world semantics. This section introduces two kinds of open-world semantics.

Since we have used a constraint-based approach for defining the closed-world semantics, we might approach the definition of an open-world semantics by simply removing/loosening some of these constraints.

Definition 4.3 (Loose CD-Semantics) *Let* $cd = (\mathcal{C}, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ *be a CD in the above sense. The loose semantics* $[\![cd]\!]^{l}$ *is the set of all object structures* $om = (\mathcal{O}, \text{type}, \mathcal{L})$

such that ***(1)*** *for all* $o \in \mathcal{O}$ *it holds that* $\text{type}(o) \not\subseteq \mathcal{A}bs$ ***(2)*** *for all* $c_1 \preccurlyeq c_2 \in \mathcal{H}, o \in \mathcal{O}$ *it holds that* $c_1 \in \text{type}(o) \Rightarrow c_2 \in \text{type}(o)$, ***(3)*** *for all associations* $a \in \mathcal{A}$, *all roles* $r \in rls(a), r' \in rls(a) \setminus \{r\}$, *and all objects* $o \in \mathcal{O}$ *such that* $cls(r') \in \text{type}(o)$, *it holds that* $min_a^{cd}(r) \leq |\mathbb{L}^{om}(o, a, r)| \leq max_a^{cd}(r)$.

The loose semantics does not require objects and links to be of class, or association types that are elements of the CD. However, instantiating only abstract classes is not allowed (cf. Condition (1)). We consider objects to instantiate multiple classes, therefore the Condition (2) assures that objects of subclass type are also of superclass type. Condition (3) requires valid object structures to obey the multiplicity constraints prescribed by the CD.

For instance, the object structure $om3.1$ in Figure 2 is an element of the loose open-world semantics of $cd3.2$ as it obeys all extends relations and all multiplicity constraints of $cd3.2$. Because $cd3.2$ does not restrict the class hierarchy of the class `Researcher`, by the open-world assumption, objects of this type may also instantiate other classes. Since the class `Employee` is not an element of $cd3.2$, the CD does not pose any restrictions on the instantiation of this class. Therefore, objects whose class type includes `Employee` may exist and have outgoing links of the association type `do`.

An issue regarding the previous definition of open-world semantics is that, given a CD cd, there may be object structures in $[\![cd]\!]$ that cannot occur in the closed-world semantics of any CD. Consider, for example, the CD $cd4$ and the object structure $om4$ in Fig. 3: we find that $om \in [\![cd4]\!]^l$. However, there exists no CD $cd' = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ such that $om \in [\![cd']\!]^{mc}$. This is because the object $o3$ instantiates both classes A and B, thus either $A \preccurlyeq B \in H^*$ or $A \preccurlyeq B \in H^*$ would have to hold, but since $o1$ and $o2$ only instantiate A and B respectively, $A \preccurlyeq B \notin H^*$ and $A \preccurlyeq B \notin H^*$ would have to hold, as well, which is impossible.

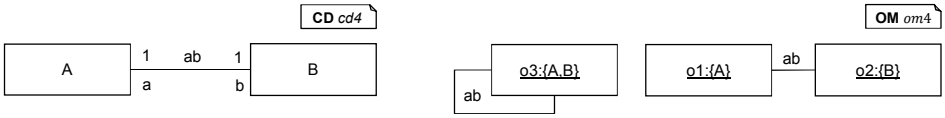


Fig. 3: object structure $om4$ in the loose semantics of CD $cd4$ is not in the closed-world semantics of any CD.

Under the open-world assumption a CD is considered underspecified. This notion implies that any object structure within the open-world semantics of a CD cd, should also be in the closed-world semantics of some CD cd' that is an (open-world) refinement of cd. We define a new open-world semantics accordingly:

Definition 4.4 (Expansion-Based Open-World CD-Semantics) *We define the simple* expansion-based *open-world semantics as*

$$[\![cd]\!]^{so} := \bigcup_{cd' \in EXP(cd)} [\![cd']\!]^{sc}.$$

Analogously, the multi-instance expansion-based *open-world semantics is defined as*

$$[\![cd]\!]^{mo} := \bigcup_{cd' \in EXP(cd)} [\![cd']\!]^{mc}.$$

An object structure that is an element of the expansion-based open-world semantics is therefore valid, iff there exists a strict expansion of the CD such that the object structure is a valid closed-world instance of this strict expansion. The notion of refinement requires that the properties of the elements of a CD also hold for every refining CD. The cd' in Definition 4.4 always refines the original cd, because for any $cd'' \in EXP(cd')$, it holds that $cd'' \in EXP(cd)$, as well, and thus any object structure in the semantics of cd' must also be in the semantics of cd''.

5 Comparison of Semantics with Regards to Semantic Differencing

CDs evolve naturally during the development process, and semantic evolution analyses, such as semantic differencing, allow a systematic tracking of these changes with respect to the modeled system. A successor version containing all previously defined classes and associations can be interpreted as a refinement of the predecessor version even if new classes or associations are added, as long as the constraints prescribed by the previous version are not contradicted. In this case, the addition corresponds to enhancing the model by further information about the system. During specification, semantic evolution analysis should interpret the addition of classes and associations as well as extends relations as the process of removing underspecification. The definition of an open-world CD semantics reflects this interpretation. In contrast, closed-world CD semantics are not (and do not need to be) compatible with the above interpretation. Using a closed-world semantics from above, each instance of a CD must not contain instances of classes and associations that are not used in the CD. Therefore, adding a class, an association, or an extends relation to a CD, results in a CD with a closed-world semantics that is incomparable to the closed-world semantics of the original CD. While this is a limitation in the context of semantic evolution analyses, it is no limitation, e. g. in the context of finite satisfiability, which is not concerned with comparing the semantics of the two CDs. In fact, a class that is used in a CD is satisfiable using the closed-world semantics of [LN94, BM13] iff it is satisfiable in the CD using the open-world definitions given above.

As seen above, open- and closed-world are not defined independently, but use a paradigm to define the set of valid object structures. In general, the multi-instance closed-world

semantics of a CD is a subset of both multi-instance open-world semantics (cf. Definition 4.3 and Definition 4.4), and the simple closed-world semantics of a CD is a subset of the expansion-based simple open-world semantics. In the context of semantic differencing, the semantics-defining paradigm also influences the output of a semantic difference. This section outlines the relations, differences and application scenarios of the above definitions of CD semantics.

5.1 Simple vs Multi-Instance Semantics

First, let us compare simple and multi-instance semantics under a closed-world approach. We find that two CDs may be semantically equivalent under the simple closed-world semantics, but not under the multi-instance closed-world semantics and vice versa. Consider the CDs in Figure 4 and the object structures in Figure 5: it holds that $[\![cd5.1]\!]^{sc} = [\![cd5.2]\!]^{sc}$, but $om5 \in [\![cd5.1]\!]^{mc} \setminus [\![cd5.2]\!]^{mc}$. On the other hand, $[\![cd5.2]\!]^{mc} = [\![cd5.3]\!]^{mc}$, but $om5 \in [\![cd5.2]\!]^{sc} \setminus [\![cd5.3]\!]^{sc}$. For all three of our open-world approaches we find that $om7$ is in the semantic difference of $cd5.1$ to $cd5.2$ and $cd5.2$ is a strict refinement of $cd5.1$. It makes sense that adding an additional abstract super-class would further restrict the semantics and thus refine an under-specified CD. Note also that $om6 \in [\![cd5.2]\!]^{mo} \setminus [\![cd5.3]\!]^{mo}$, as well as $om6 \in [\![cd5.2]\!]^{l} \setminus [\![cd5.3]\!]^{l}$.

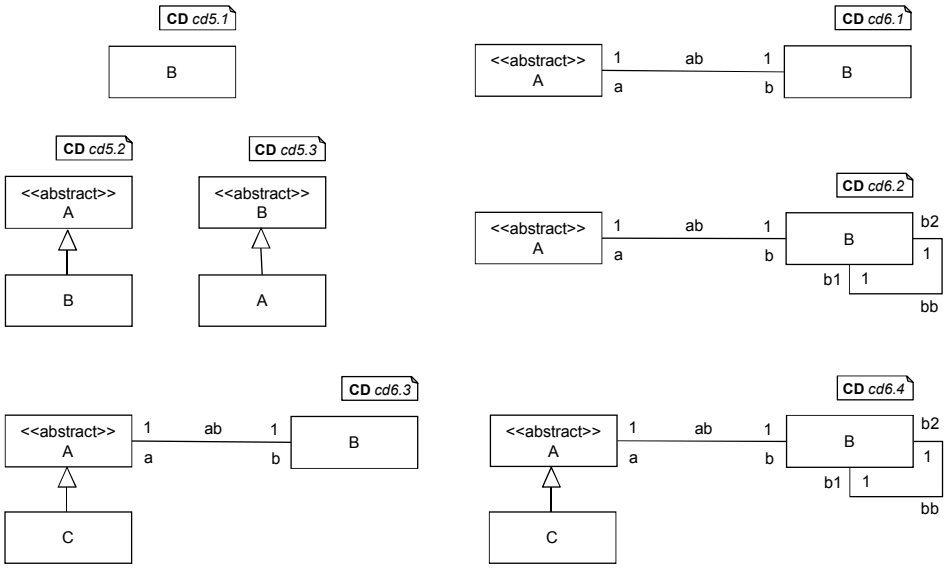


Fig. 4: CDs to illustrate the difference between the semantics definitions.

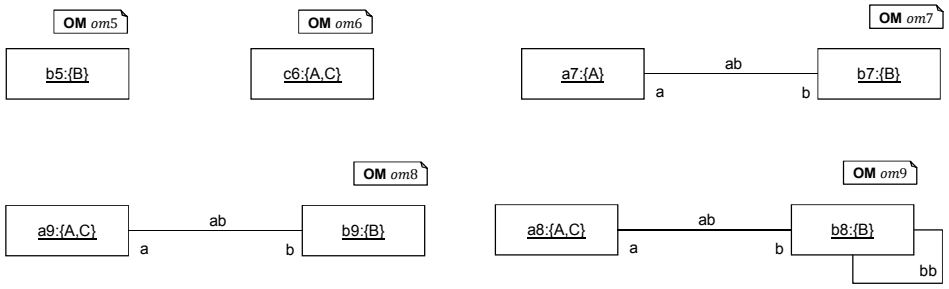


Fig. 5: Object structures to illustrate the difference between the semantics definitions.

5.2 Closed-World vs Open-World Semantics

To outline further differences between closed-world and open-world semantics, consider the CDs $cd6.1$ and $cd6.2$: under both simple and multi-instance closed-world semantics these two CDs are semantically equivalent, as none of their classes can be instantiated. By adding an additional class C as a subclass to A in both CDs, we get $cd6.3$ and $cd6.4$, which are no longer semantically equivalent. In fact, they are semantically incomparable: $om8$ is in the closed-world semantics of $cd6.3$ but not $cd6.4$ and $om9$ is in the closed-world semantics of $cd6.4$ but not $cd6.3$. This situation in which two previously semantically equivalent, models become semantically incomparable by adding the same new element to both, calls into question the well-formedness of these close-world CD-semantics. The expansion-based open-world semantics, by their very Definition 4.4, do not suffer from this flaw, and regarding the previous example, we find that $cd6.2$ is a strict refinement of $cd6.1$ under open-world semantics.

5.3 Relation Between Constraint-Based Open- and Closed-World Semantics

Compared, to the open-world constraint-semantics Definition 4.3, the constraint-based closed-world Definition 4.2 additionally requires instances of cd to solely contain objects and links, typed with classes and associations used in the CD cd. These definitions are related in the sense that removing all objects and links from an instance of the CD, which are not typed with classes and associations of the CD, yields an object structure that instantiates the CD considering either definition of semantics.

To this effect, let $cd = (\mathcal{C}, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ be a CD and let $om = (\mathcal{O}, \text{class}, \mathcal{L})$ be an object structure. We denote by $om \downarrow cd \stackrel{\text{def}}{=} (\mathcal{O}', \text{type}', \mathcal{L}')$ the restriction of om to instances of classes and associations used in cd, i. e. $\mathcal{O}' = \{o \in \mathcal{O} \mid \text{type}(o) \cap \mathcal{C} \neq \emptyset\}$, $\mathcal{L}' = \{(o, r, a, o', r') \in \mathcal{L} \mid a \in \mathcal{A}\}$, $\text{type}' : \mathcal{O}' \to \wp_{\text{fin}}(\mathcal{C}), o \mapsto \text{type}(o) \cap \mathcal{C} \setminus (\{c_2 \in \mathcal{C} \mid \nexists c_1 \in \text{type}(o) : c_1 \preccurlyeq c_2 \in \mathcal{H}^*\}) \cup \{c_1 \in \mathcal{C} \mid \nexists c_2 : c_1 \preccurlyeq c_2 \in \mathcal{H}^*\}$.

The following lemma formalizes the relation between the two semantic mappings: If an object structure is an element of the semantics of a CD, then its restriction to the elements of the CD is an element of the closed-world CD semantics.

Lemma 5.1 *Let cd be a CD and let om be an object structure. If* $om \in [\![cd]\!]'$*, then* $om \downarrow cd \in [\![cd]\!]^{mc}$.

Proof sketch The definitions of O', type′, and $\mathcal{L}'$ rule out the existence of objects in $om \downarrow cd$ of class types or links of association-types that are not defined in the CD. Since an om in the loose open-world semantics does not include objects of abstract types, $om \downarrow cd$ does not either. Therefore the closed-world assumptions hold for $om \downarrow cd$. Loose open-world instances do not include objects whose class type does not include all super-classes of included sub-classes. Therefore Condition (2) of Theorem 4.2 holds. Instances of the loose open-world semantics obey the multiplicity constraints defined in the CD. Since, the restriction removes links that instantiate associations which are not defined in the CD and since instances of the loose open-world semantics already obey the multiplicity constraints, the restriction $om \downarrow cd$ does not include objects with outgoing links that instantiate associations which the object's class type does not inherit and all objects continue to be consistent with the extends relations prescribed in the CD. Therefore, also Conditions (3) and (4) of Theorem 4.2 hold. The definition of type′ assures that the classes in the type of an object in $om \downarrow cd$ are related by extends relations defined in cd which implies Condition (5) of Theorem 4.2.

By Lemma 5.1, the closed-world semantics is a restriction of the open-world semantics, i. e. every closed-world instance of a CD is also an open-world instance of the CD. This does not necessarily hold vice-versa. Further, from every open-world instance om of cd , we can construct a closed-world instance by the transformation $om \downarrow cd$.

6 Related Work

While UML does not define formal semantics of CDs, every tool working with CDs uses a semantic definition. Commonly, the semantics for generators [SN11, PSB11, SKM07, SS19, Sw12] is not explicitly defined and the behavior for certain UML-constructs such as "composition" might vary. Generators use a closed-world semantics and only translate modeled elements.

An important analysis on CDs are *consistency* checks. Especially when "nontrivial multiplicity constraints (different than 0, 1, *)" [BM13] occur, there might not exist a valid object structure. In [Sz06] proves the inconsistency of an exemplary CD. [ACIG10] shows that the complexity of full-satisfiability problems is ExpTime-complete. With restrictions on used CD concepts, the complexity can be significantly decreased.

Consistency analysis is often performed using a translation to Alloy [MRR11b, MGB04, Ka17, SAB09, CGR15] or SMT [Wu17, Pr16, PWD16]. Because the closed-world semantics is always a subset of the open-world semantics, consistency checks on closed-world semantics can lead to useful results in open-world semantics.

Like consistency analysis, *semantic differencing*[FLW11, LMK14, Ka17, MRR11a] searches for a valid witness object structure. In previous work, we have presented a semantic differencing tool using a translation to Alloy [Ka17, MRR11a].

Refinement is an interesting property in many domains. For example, the refinement calculus [BW12] can prove the correctness of program modifications. [BS01] proves refinement properties over distributed systems. Open-world models are useful in combination with a merge operator [Br06, LWD11], which reduces underspecification.

7 Conclusion

In this paper we have defined multiple closed-world and open-world semantics for CDs that consider an object either as an instance of a single class or additionally as an instance of all superclasses. We have analyzed and compared these CD-semantics regarding semantic differencing, as well as model-evolution.

We found that an open-world semantics definition based on strict expansions adheres to the notion of underspecification in CDs better than a semantic based on relaxed constraints, and that it is therefore well-suited for semantic differencing and model evolution, especially during analysis in the early stages of development, where the addition of new elements and features should be seen as a refinement. In later stages of the development or when considering code generation, a closed-world approach might be more appropriate.

Moreover, we consider both the simple as well as the multi-instance semantics to be valid and useful depending on the circumstances. If one is, for instance, primarily concerned with development and evolution of data structures, the simple semantics might be the better choice. However, if the usage of objects as method arguments has to be considered (e. g. code generation) a multi-instance semantics would seems more appropriate.

Finally, we found that for our semantics definitions a closed-world instance is always an open-world instance, as well, and that any open-world instance can be transformed into a corresponding closed-world instance by removing not explicitly modeled elements. However, we have yet to develop a transformation that reliably produces non-empty instances for certain edge cases. Nevertheless, we do believe that a reduction of multi-instance expansion-based open-world semantic differencing to a closed-world semantic differencing should be possible. Furthermore, it might be interesting to consider a multi-instance semantics that distinguishes sub- and super-classes instantiated by an object.

Bibliography

[ACIG10] Artale, Alessandro; Calvanese, Diego; Ibáñez-García, Angélica: Full satisfiability of UML class diagrams. In: International Conference on Conceptual Modeling. Springer, pp. 317–331, 2010.

[AP03] Alanen, Marcus; Porres, Ivan: Difference and Union of Models. In: «UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications. 2003.

[BM13] Balaban, Mira; Maraee, Azzam: Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. ACM Trans. Softw. Eng. Methodol., 22(3), 2013.

[Br06] Brunet, Greg; Chechik, Marsha; Easterbrook, Steve; Nejati, Shiva; Niu, Nan; Sabetzadeh, Mehrdad: A manifesto for model merging. In: Proceedings of the 2006 international workshop on Global integrated model management. pp. 5–12, 2006.

[BS01] Broy, Manfred; Stølen, Ketil: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer Science & Business Media, 2001.

[BW12] Back, Ralph-Johan; Wright, Joakim: Refinement calculus: a systematic introduction. Springer Science & Business Media, 2012.

[CGR15] Cunha, Alcino; Garis, Ana; Riesco, Daniel: Translating between Alloy specifications and UML class diagrams annotated with OCL. Software & Systems Modeling, 14(1):5–25, 2015.

[DP06] Dobing, Brian; Parsons, Jeffrey: How UML is used. Communications of the ACM, 49(5):109–113, 2006.

[FLW11] Fahrenberg, Uli; Legay, Axel; Wąsowski, Andrzej: Vision Paper: Make a Difference! (Semantically). In: Model Driven Engineering Languages and Systems. 2011.

[Ge21] Gerasimov, Arkadii; Michael, Judith; Netz, Lukas; Rumpe, Bernhard: Agile Generator-Based GUI Modeling for Information Systems. In (Dahanayake, Ajantha; Pastor, Oscar; Thalheim, Bernhard, eds): Modelling to Program (M2P). Springer, pp. 113–126, March 2021.

[HR04] Harel, David; Rumpe, Bernhard: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 37(10):64–72, October 2004.

[Hu11] Hutchinson, John; Whittle, Jon; Rouncefield, Mark; Kristoffersen, Steinar: Empirical assessment of MDE in industry. In: Proceedings of the 33rd international conference on software engineering. pp. 471–480, 2011.

[Ka17] Kautz, Oliver; Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: CD2Alloy: A Translation of Class Diagrams to Alloy. Technical Report AIB-2017-06, RWTH Aachen University, July 2017.

[KGE09] Küster, Jochen M.; Gerth, Christian; Engels, Gregor: Dependent and Conflicting Change Operations of Process Models. In: Model Driven Architecture - Foundations and Applications. 2009.

[KKT11] Kehrer, Timo; Kelter, Udo; Taentzer, Gabriele: A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning. In: International Conference on Automated Software Engineering (ASE'11). 2011.

[Kü08] Küster, Jochen M.; Gerth, Christian; Förster, Alexander; Engels, Gregor: Detecting and Resolving Process Model Differences in the Absence of a Change Log. In: Business Process Management. 2008.

[La14] Langer, Philip; Mayerhofer, Tanja; Wimmer, Manuel; Kappel, Gerti: On the usage of UML: Initial results of analyzing open UML models. Modellierung 2014, 2014.

[LMK14] Langer, Philip; Mayerhofer, Tanja; Kappel, Gerti: A Generic Framework for Realizing Semantic Model Differencing Operators. In: PSRC@MoDELs. CEUR Workshop Proceedings, 2014.

[LN94] Lenzerini, Maurizio; Nobili, Paolo: On the interaction between ISA and cardinality constraints. In: Proceedings of 1994 IEEE 10th International Conference on Data Engineering. 1994.

[LWD11] Lutz, Rainer; Wurfel, David; Diehl, Stephan: How humans merge UML-models. In: 2011 International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 177–186, 2011.

[Ma15] Management Group, Object: , OMG Unified Modeling Language (OMG UML) Version 2.5, 2015.

[MGB04] Massoni, Tiago; Gheyi, Rohit; Borba, Paulo: A UML class diagram analyzer. TUM, p. 100, 2004.

[MRR10] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: A Manifesto for Semantic Model Differencing. In: Proceedings Int. Workshop on Models and Evolution (ME'10). LNCS 6627. Springer, pp. 194–203, 2010.

[MRR11a] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: CDDiff: Semantic Differencing for Class Diagrams. In (Mezini, Mira, ed.): ECOOP 2011 - Object-Oriented Programming. Springer Berlin Heidelberg, pp. 230–254, 2011.

[MRR11b] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Semantically Configurable Consistency Analysis for Class and Object Diagrams. In: Conference on Model Driven Engineering Languages and Systems (MODELS'11). LNCS 6981. Springer, pp. 153–167, 2011.

[Pr16] Przigoda, Nils; Hilken, Frank; Peters, Judith; Wille, Robert; Gogolla, Martin; Drechsler, Rolf: Integrating an SMT-Based ModelFinder into USE. In: MoDeVVa@ MoDELS. pp. 40–45, 2016.

[PSB11] Parada, Abilio G.; Siegert, Eliane; Brisolara, Lisane B. de: Generating Java Code from UML Class and Sequence Diagrams. In: 2011 Brazilian Symposium on Computing System Engineering. pp. 99–101, 2011.

[PWD16] Przigoda, Nils; Wille, Robert; Drechsler, Rolf: Ground setting properties for an efficient translation of OCL in SMT-based model finding. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 261–271, 2016.

[Re78] Reiter, Raymond: On Closed World Data Bases. In: Logic and Data Bases. 1978.

[Ru16] Rumpe, Bernhard: Modeling with UML: Language, Concepts, Methods. Springer International, July 2016.

[SAB09] Shah, Seyyed; Anastasakis, Kyriakos; Bordbar, Behzad: From UML to Alloy and back again. In: International Conference on Model Driven Engineering Languages and Systems. Springer, pp. 158–171, 2009.

[SKM07] Sarma, Monalisa; Kundu, Debasish; Mall, Rajib: Automatic test case generation from UML sequence diagram. In: 15th International Conference on Advanced Computing and Communications (ADCOM 2007). IEEE, pp. 60–67, 2007.

[SN11] Sejans, Janis; Nikiforova, Oksana: Practical Experiments with Code Generation from the UML Class Diagram. In: MDA/MDSD. pp. 57–67, 2011.

[SS19] Sunitha, EV; Samuel, Philip: Automatic code generation from UML state chart diagrams. IEEE Access, 7:8591–8608, 2019.

[Sw12] Swain, Ranjita; Panthi, Vikas; Behera, Prafulla Kumar; Mohapatra, Durga Prasad: Automatic test case generation from UML state chart diagram. International Journal of Computer Applications, 42(7):26–36, 2012.

[Sz06] Szlenk, Marcin: Formal semantics and reasoning about uml class diagram. In: 2006 International Conference on Dependability of Computer Systems. IEEE, pp. 51–59, 2006.

[T.13] T. Kehrer, U. Kelter, and G. Taentzer: Consistency-Preserving Edit Scripts in Model Versioning. In: International Conference on Automated Software Engineering (ASE). 2013.

[Ta14] Taentzer, Gabriele; Ermel, Claudia; Langer, Philip; Wimmer, Manuel: A fundamental approach to model versioning based on graph modifications: from theory to implementation. Software & Systems Modeling, 13(1), 2014.

[TK09] Thüm, Thomas; , Don; Kästner, Christian: Reasoning about Edits to Feature Models. In: Proceedings of the 31st International Conference on Software Engineering. 2009.

[Wu17] Wu, Hao: MaxUSE: a tool for finding achievable constraints and conflicts for inconsistent UML class diagrams. In: International Conference on Integrated Formal Methods. Springer, pp. 348–356, 2017.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 129

Projektorgansisationsmodellierung mittels Matrizen und Kausalnetzen in universitären Softwareentwicklungsprojekten

Laif-Oke Clasen, Daniel Moldt, Matthias Feldmann[1]

Abstract:

Damit Studierende erlernen, wie man Software in größeren Gruppierungen entwickelt, müssen die richtigen Bedingungen in Lehrprojekten geschaffen werden. Zur Vermittlung von Strukturen und Prozessen der Projektorganisation und -durchführung sowie deren Abhängigkeiten sind Methoden, Techniken und Werkzeuge notwendig.

Der PAOSE-Ansatz (Petri net-based, Agent- and Organization-Oriented Software Engineering) liefert hierfür sowohl die Modellierungs- als auch die Umsetzungsmittel. In diesem Beitrag wird erläutert, wie, in Anlehnung an die bisher verwendete Modellierungstechnik der Use-Cases und Matrizen, Studierende ein tieferes Verständnis der Aufgabenstrukturierung im Softwareentwicklungsprozess erlangen können, so dass zum einen eine koordinierte Arbeit erfolgt und zum anderen ein bewußter Lernprozess unterstützt wird. Zur Erläuterung wird beispielhaft anhand der bekannten Rollen des Scrum Ansatzes das Verfahren diskutiert.

Keywords: Fallstudie; Softwareprojekt; Lehrprojekt; Scrum; Projektmanagement; Matrix; Kausalnetz

1 Einleitung

Damit sich Teilnehmende im Kontext eines Projektes zurechtfinden, müssen die organisatorischen Strukturen und Abläufe bekannt sein. Die erfolgreiche Durchführung der Entwicklungstätigkeiten erfordert bei größeren Projekten, dass immer eine Orientierung in Hinblick auf die Aufgaben und ihre Durchführungsmöglichkeiten gegeben ist.

Eingesetzte Werkzeuge und Modelle im Rahmen der Softwareentwicklung – als unterstützende Anwendungssysteme betrachtet – sollten nach Conway's Law[Con68] die Projektstrukturen widerspiegeln. In diesem Beitrag verwenden wir Matrizen um die Projektorganisation zu modellieren. Hierbei werden Präzedenzen in den Zellen der Matrix notiert, wodurch aus diesen Präzedenzgraphen konstruiert werden können, welche sich leicht in Kausalnetze überführen lassen. Die Matrizen, die als Ausgangspunkt dieser Arbeit verwendet werden, stammen aus dem PAOSE-Ansatz, hier sind diese als *coarse desgin*

[1] Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30 22523 Hamburg, Deutschland 7clasen/moldt/4feldman@informatik.uni-hamburg.de

doi:10.18420/modellierung2022-013

diagram kurz CDD bekannt (siehe [Cab10]). CDDs stellen eine agentenorientierte Variante der Use-Case Diagramme der UML dar.

Die Modellierung der Projektorganisation mittels Matrizen ermöglicht es ein Projekt auf seine Durchführbarkeit hin zu überprüfen. Außerdem kann durch sie die Kommunikation über Teamgrenzen hinweg sichtbar gemacht werden. Jedoch müssen hierfür die Matrizen sowohl übergreifend erfasst, als auch stetig aktualisiert werden.

Als Fallbeispiel wird in diesem Beitrag das RENEW Lehrprojekt verwendet, das sowohl im Bachelor als auch im Master belegt werden kann. In diesem wurde das gleichnamige Werkzeug entwickelt.[Kum+20] RENEW basiert auf der Programmiersprache Java und wurde am Fachbereich Informatik der Universität Hamburg über den Zeitraum von 25 Jahren entwickelt. Somit erfordert und bietet es eine hinreichend komplexe Umgebung für das kollaborative Arbeiten in großen Teams an einer sehr großen Java-Codebasis von RENEW.

Insgesamt stellen wir eine praktisch mehrfach verfeinerte, grundlegende Modellierungsperspektive vor, die Studierenden in größeren, kollaborativen Softwareentwicklungsteams eine Orientierung für ihre Handlungsstrukturierung bietet. Dafür werden zunächst in Abschnitt 2 die CDD Matrizen aus dem PAOSE-Ansatz vorgestellt. Die Matrix ergibt sich dabei aus einer Aufteilung in funktionale (in Form von Rollen) und prozessorientierte Perspektiven (in Form von Interaktionen), die für jedes Projekt letztlich integriert werden müssen. Deren Interpretation als partiell geordnete Prozesse mit synchronisierenden Tätigkeitselementen durch Kausalnetze mit synchronen Kanälen als Modell wird in Abschnitt 3 vorgestellt. In dem Abschnitt 4 werden die von uns vorgeschlagenen Erweiterungen der CDD Matrix erörtert. Die entwickelten Erweiterungen lassen sich auf rollenbasierte Projekte anwenden. RENEW dient hierfür in diesem Beitrag abschließend als Fallstudie in dem Abschnitt 5.

2 CDD Matrizen des PAOSE-Ansatzes

In diesem Abschnitt wird der PAOSE-Ansatz umrissen. Hierfür wird in dem Abschnitt 2.1 ein Sprint im Kontext des PAOSE-Ansatzes beschrieben. Anschließend wird in dem Abschnitt 2.2 das *Coarse Design Diagram* kurz CDD vorgestellt. Hiernach wird die erweiterte CDD in dem Abschnitt 2.3 erläutert. Abschließend wird in dem Abschnitt 2.4 eine genauere Betrachtung der Präzedenzen innerhalb der Matrizen vorgenommen.

2.1 Sprint

Wie andere agile Softwareentwicklungsansätze ist der PAOSE-Ansatz [Cab10] (Petri net-based Agent- and Organization-oriented Software Engineering) in Sprints unterteilt. Die Synchronisation der Teammitglieder findet am Anfang und am Ende eines Sprints statt. Im

universitären Kontext ist die Verwendung von Sprints durch die vorgegebenen Zeitabschnitte der Studierenden besonders sinnvoll.

Ein zeitlich fixierter Sprint lässt sich nach dem PAOSE-Ansatz grob als Mehrfachausführung der drei Phasen Anforderungserstellung, Grobentwurf und Umsetzung mittels Rollen-, Interaktions- und Ontologieerstellung unterteilen.

2.2 CDD Matrix

Im Folgenden werden wir uns auf die Phase des *Grobentwurfs* konzentrieren. In dieser Phase wird ein *coarse design diagram* kurz CDD erstellt, das an die Use-Case Diagramme der UML angelehnt ist. Statt der Diagramme können jedoch auch Tabellen (in Form einer Matrix) verwendet werden. In den Multiagentensystemen (MAS), die dem PAOSE-Ansatz zugrunde liegen, werden Zeilen den Interaktionen (Anwendungsfälle in UML) und Spalten den anwendungsgetriebenen Rollen (Akteure in UML) zugeordnet. Für jede Interaktion existiert ein Auslöser, der als Nachricht an eine der Rollen geht. Die Abarbeitung des Auslösers erfolgt dann über mehrere Rollen verteilt hinweg. Den Rollen und Interaktionen werden in einem Projekt dann Personen zur Erstellung zugeordnet. In der CDD Matrix werden so die Schnittstellen zwischen der Interaktion mit allen jeweils betroffenen Rollen festgelegt / identifiziert und damit auch die zuständigen Personen. So haben Rollen in der Regel mehrere Interaktionen an denen sie beteiligt sind und Interaktionen haben mehrere Rollenbezüge. Die Ontologie liefert die gemeinsame Terminologie innerhalb der Anwendung bzw. der Agenten. Details zur konkreten Ausgestaltung der Techniken und Werkzeuge des PAOSE-Ansatzes finden sich insbesondere in [Cab10].

Bezüglich des Vorgehens ist festzuhalten, dass die Schnittpunkte in der Matrix somit notwendige Koordination anzeigen. In Softwareentwicklungsprojekten erfolgt diese üblicherweise synchron als Treffen, da Schnittstellen (wie API, Ontologie) ausgehandelt werden müssen. Entwicklungstätigkeiten vor und nach diesen Treffen sind selbstverständlich ebenfalls notwendig, werden hier aber nicht explizit modelliert.

In der Tabelle 1 ist ein Beispiel für eine CDD Matrix zu sehen. Hieran ist zu erkennen, dass an der Interaktion A sowohl die Rollen A als auch C beteiligt sind. Somit ist × sowohl ein Indikator dafür, dass hier in der Entwicklung Kommunikationsaufwand zwischen den Entwicklergruppen der Rollen A und C und der Entwicklergruppe der Interaktion A notwendig ist, als auch dafür, welche Gruppen welche Interfaces bereitstellen müssen.

Rollen / Interaktionen	Rolle A	Rolle B	Rolle C
Interaktion A	×		×
Interaktion B		×	
Interaktion C		×	×

Tab. 1: Beispiel einer CDD Matrix (siehe [Cab10])

2.3 Erweiterte CDD Matrix

Die CDD Matrix kann nach [Sch+18; Sie16] um die Darstellung von Meilensteinen und Präzedenzen erweitert werden. Hierdurch wird es ermöglicht durch die CDD Matrix Sprints oder Phasen zu planen, je nach Projektart.
Ein Beispiel einer erweiterten CDD Matrix ist in Tabelle 2 zu sehen. Hierbei wird der ungefärbte Block in einem Durchgang / Sprint entwickelt und der graue Block in einem weiteren.

Rollen / Interaktionen	Rolle A	Rolle C	Rolle B
Interaktion A	1	2	3
Interaktion B	2	1	5
Interaktion C	3		4
Interaktion D		3	4

Tab. 2: Beispiel einer erweiterten CDD Matrix (siehe [Sie16])

Hierzu wurden Präzedenzen (Zahlen) anstelle der ✕ eingefügt. Die bisherige Betrachtung der Präzedenzen in den Matrizen geht von den Interaktionen aus. Also aus Sicht der Interaktion: Welche Rollen sind in welcher Reihenfolge an mir beteiligt? Durch diese Präzedenzen lässt sich die Kommunikation über Teamgrenzen hinweg sichtbar machen. Diese geben letztlich am Ende eines Sprint-Pokers die Reihenfolge der geplanten Abarbeitung an.[2] Dabei bedeuten z.B. gleiche Zahlen, dass es keine Präzedenzen zwischen den zugehörigen Aktionen bestehen, während die Reihenfolge (z.B. 1 vor 2) bedeutet, das alle Aktionen von 1 präzedent zu den Aktionen von 2 sind. Diese Abhängigkeiten werden jeweils aus der Perspektive der Rollen und der Interaktionen separat bestimmt. Anschließend werden dann die Gesamtpräzedenzen über alle Aktivitäten hinweg festgelegt. Zyklen können bzw. müssen dabei erkannt und aufgelöst werden.Insgesamt ergibt sich so für das gesamte Projekt über alle Teams hinweg eine eindeutige, partiell geordnete Menge von Aktionen.[3]

Auch die erweiterte CDD Matrix nach [Sie16] wurde für Agentenanwendungen entwickelt. Weshalb der Fokus auf einer Akteursperspektive liegt. Somit beschreiben die Rollen die Struktur und die Interaktionen das Verhalten. Das hat zur Folge, dass die Matrix nach [Sie16] im Laufe der Entwicklung sowohl um Rollen als auch um Interaktionen ergänzt wird. Deshalb wächst die Matrix mit der Zeit sowohl in X- als auch in Y-Richtung.

[2] Die zeitliche Dauer von Aktivitäten der Softwareentwicklung wird hier bewusst nicht mit abgebildet. Hierfür werden dann weitere Konzepte, insbesondere Zeitdauern von Aktivitäten wichtig. In [Sie16] werden hierfür die internen Entwicklungsaktivitäten mit angegeben. In [Sch+18] werden hierfür dann die unter anderem in [Röd16] entwickelten Netzplantechnik-Erweiterungen vorgestellt. Zusätzlich hierzu hat [Sch+18] gezeigt, dass sich NPT Diagramme bzgl. der Präzedenzen durch Kausalnetze beschreiben lassen, wodurch die erweiterte CDD Matrix eine formal fundierte Semantik besitzt.

[3] Teaminterne Aktionen können ebenfalls mit in die Matrix aufgenommen werden, vermindern jedoch die Übersichtlichkeit. Hier helfen Werkzeuge, die entsprechend Filter auf den dargestellten Aktionen anbieten.

2.4 Genauere Betrachtung der Präzedenzen

An dieser Stelle gehen wir genauer auf die Präzedenzen ein. Bisher wurden lediglich die Präzedenzen aus Sicht der Interaktion betrachtet. Hinzu kommen Präzedenzen aus Sicht der Rolle. Grund hierfür ist, dass Rollen pro Sprint/Meilenstein an mehreren Interaktionen beteiligt sind. Deshalb müssen die Rollen die einzelnen Interaktionen für sich selbst ordnen. Zudem müssen die funktionalen Anteile der Rolle als rolleninterne Aktivitäten der zuständigen Entwickler bereit gestellt werden. Ein Beispiel hierfür ist die Priorisierung eines Entwicklers von zwei Treffen mit unterschiedlichen Teams.

Damit ergeben sich zwei Einträge für Erwartungen bezüglich der Abarbeitung von Aufgaben innerhalb des Projekts. Treten Zyklen auf, dann müssen diese zwingend beseitigt werden, da ein Zyklus inhärent widersprüchlich zu den Erwartungen aller Beteiligten wäre. Sobald keine Zyklen vorliegen, kann die Gesamtplanung für das Team erfolgen. Hierbei ergibt sich aus den Reihenfolgen der Aufgaben (und auch deren Zeitschätzung und Priorisierung der jeweiligen Produktmerkmale (Feature)) ein übergeordneter Präzedenzgraph aller modellierten Aufgaben in den Zellen.

Somit existieren zwei ggf. unterschiedliche Präzedenzen für jede Zelle der Matrix. diese können durch (Präzedenz-)Tupel in den Zellen dargestellt werden. Hierfür nehmen wir fortlaufend an, dass das Tupel wie folgt aussieht: [Präzedenz der Interaktion, Präzedenz der Rolle]. Ein Beispiel hierfür ist in Tabelle 3 zu sehen, die einen Zyklus, bedingt durch die eingetragenen Präzedenzen, enthält.[4]

Rolle / Interaktionen	Rolle A	Rolle B
Interaktion C	[1,2]	[2,1]
Interaktion D	[2,1]	[1,2]

Tab. 3: Ausschnitt einer CDD Matrix, die sowohl Präzedenzen bzgl. Rollen als auch Interaktionen zur Illustration eines Zykluses enthält

3 Kausalnetze

Ein Kausalnetz (siehe [Rei13, S. 39]) ist ein azyklisches Petrinetz, für das gilt, dass sich sowohl im Vor- als auch im Nachbereich der Plätze maximal eine Transition befindet. Hieraus folgt unter anderem, dass ein Kausalnetz konfliktfrei ist. Präzedenzen lassen sich in einfacher Weise auf Kausalnetze abbilden: Die Aktionen sind die Transitionen, die Präzedenzen werden durch die Verbindung über die Stellen dargestellt.

[4] Diese Zyklen müssen aufgelöst werden. Dies geht nur unter Berücksichtigung der fachlichen Notwendigkeiten. Zyklen deuten auf inhärente Widersprüche bzgl. der Abläufe hin, die sich aus den unterschiedlichen Perspektiven ergeben. Wenn kein Versehen bei der Angabe vorliegt, dann helfen diese Zyklenangaben bei teamübergreifenden Identifikationen von fehlerhaften bzw. problematischen Ablaufplanungen. Für den Gesamtablauf mit nur einer eindeutigen Angabe der Präzedenzen müssen diese Tupel in eine zyklenfrei Form überführt werden.

In [Sie16; Sch+18] sind Kausalnetze um synchrone Kanäle erweitert worden, um damit unterschiedliche Abläufe von verschiedenen Teams zu modellieren. Für jedes Team (Rollen und Interaktionen) bilden alle Präzedenzen ein Kausalnetz, was einfach möglich ist durch die vorherige Überführung in einen Präzedenzgraphen. Diese Kausalnetze bilden durch die Synchronisation über die synchronen Kanäle größere Kausalnetze. Die Synchronisationen repräsentieren genau diejenigen Treffen, die gruppenübergreifend notwendig sind.[5] Die Synchronisation von zwei Transitionen mit passenden synchronen Kanälen entspricht dann genau einer Handlung / Aktivität / Aktion, je nach Interpretation der Transitionen des Kausalnetzes. Durch die Verschmelzung der Transitionen von Kausalnetzen ergeben sich übergreifende Gesamtnetze, die, wie oben angeführt, wieder nur Kausalnetze ergeben müssen, damit es keine Verklemmungen bei dem logischen Ablauf gibt. Erreicht wird somit mittels der Präzedenztupel eine eindeutige, partiell geordnete Menge von Aktionen im Projekt.

Jedoch ist die Planung von einem einzigen Kausalnetz vom Beginn bis Ende eines Projektes nicht sinnvoll. Das Problem hierbei ist die zeitliche Zerlegung in bspw. Sprints und fachliche Zerlegung in *handhabbare* Teile bspw. durch Scrum oder Kanban. Ein einziges Kausalnetz ist aufgrund der notwendigen Größe nicht mehr sinnvoll nutzbar. Deshalb ist dies nur in einer gewissen Größenordnung sinnvoll. Fachlich ergibt sich eine sinnvolle Strukturierung, die sich für die drei zuvor beschriebenen Tabellen ergibt.

Die Kausalnetze (mit den synchronen Kanälen) liefern eine formale Basis für die Präzedenzrelation. Sonst für Projekte verwendete Workflownetze [Aal97] liefern erheblich komplexere Modelle. Diese lassen sich unmittelbar als Faltungen von Kausalnetzen (als Operationen auf Petrinetzen) verwenden. Dabei sind dann aber die Präzedenzen nicht mehr direkt ersichtlich.

4 Erweiterung der CDD Matrix für rollenbasierte Projekte

In diesem Abschnitt wird das Projekt mit der CDD Matrix betrachtet. Zuerst wird in dem Abschnitt 4.1 eine modifizierte Interpretation der CDD Matrix vorgestellt. Anschließend wird die CDD Matrix in dem Abschnitt 4.2 auf ein Team ohne externe Rollen angewendet. Im Anschluss wird diese CDD Matrix in dem Abschnitt 4.3 auf ein Team mit externen Rollen erweitert. Außerdem wird in dem Abschnitt 4.4 eine neue CDD Matrix durch Aggregation von mehreren CDD Matrizen geschaffen. Damit kann das gesamte Projekt grundsätzlich durch eine einzige CDD Matrix dargestellt werden. Zusätzlich können durch die Aggregation unterschiedliche Abstraktionsebenen betrachtet werden, womit verschiedene Sichtweisen auf das Projekt realisiert werden können. In dem Abschnitt 4.5 wird beschrieben wie Kommunikation über Teamgrenzen sichtbar gemacht werden kann. Abschließend wird die Diskussion des Einsatzes von CDD Matrizen in dem Abschnitt 4.6 geführt.

[5] Interessanterweise lassen sich auch teaminterne Arbeiten mittels dieser Modellierung erfassen. Jede Person muss letztlich für eigene Aufgaben eine Präzedenzrelation erstellen, die dann teamintern für die kooperative Arbeit maßgeblich ist. Dies wird aber hier nicht weiter ausgeführt. Details dazu finden sich in [Sie16].

4.1 Modifizierte Interpretation der CDD Matrix

Wie bereits in dem Abschnitt 2.2 beschrieben, lässt sich die Projektorganisation ebenfalls als MAS betrachten. Hieran knüpft der Vorschlag dieses Beitrags an indem die Unterstützung eines Projektes durch die Softwareentwicklungsumgebung wiederum als MAS betrachtet wird. Mit der Matrix-Perspektive lassen sich so Aufgaben ablaufgerecht für einen Sprint priorisieren. Die beiden Perspektiven einer Matrix (Interaktionen und Rollen) werden auf die Projektstruktur gelegt.
Dazu werden die Interaktionen, die sich bezüglich der Ausführung der Aufgaben zur Softwareentwicklung ergeben, in den Zeilen und die jeweiligen Rollen der Beteiligten, wie in der CDD Matrix, in den Spalten angeordnet. Dabei wird auf die projektbezogenen Aufgaben der Beteiligten abgezielt. Damit wird ein Projekt im Prinzip als MAS betrachtet.

Eine Zuordnung der damit verbunden Aufgaben in den Zellen der Matrix werden dann später tatsächlichen Personen flexibel zugeteilt. Aus der Perspektive der Interaktionen ergeben sich Präzedenzen aus den inhärenten Abhängigkeiten der (Teil)Aufgaben, wobei hier die rolleninhärenten Präzedenzen vorerst nicht berücksichtigt sind. Mittels einer partiellen Ordnung wird eine maximale Nebenläufigkeit für die Ausführungen erlaubt, die sich mithilfe von Kausalnetzen modellieren lässt (siehe [Sch+18]). Möglichkeiten zur zeitlichen Interpretation im Sinne der Netzplantechnik[Alt96] ergeben sich ebenfalls[Sch+18].

4.2 CDD Matrix für ein Team ohne externe Rollen

Zunächst betrachten wir lediglich ein Team, wie in Tabelle 4 exemplarisch dargestellt. Die Zuordnung der Personen auf die Rollen sowie die Zuordnung der Rollen auf Teams können beliebig geschehen. Für Analysezwecke muss die Zuordnung von Personen auf Rollen festgehalten werden. Für die Interaktionen werden in der CDD Matrix Aufgaben verwendet. Dieses Beispiel entspricht Tabelle 2 und enthält somit lediglich die Präzedenzen der Interaktionen. Als Rollen werden hier Verantwortlichkeitsrollen verwendet. Jedoch sind die Rollen hier nicht auf Verantwortlichkeitsrollen beschränkt. Sie dienen hier lediglich als Beispiel. Somit lässt sich die CDD Matrix auf sämtliche rollenbasierte Projekte anwenden.

Verantwortlich-keitsrollen / Interaktionen	Anforderungs-management (AM) Feature X	Teamleitung Feature X	Entwicklung Feature X
Ticket für Feature X erstellen	[1,1]		
Feature X planen		[1,1]	[2,2]
Feature X implementieren			[1,1]

Tab. 4: Ausschnitt einer CDD Matrix für ein Team ohne externe Rollen

Die Matrix wird im Projekt fortlaufend erweitert werden. Jedoch vergrößert sich die Matrix in diesem Beispiel nur vertikal und nicht wie bei [Sie16] vertikal und horizontal, da die

Rollen hier statisch festgelegt sind. Allgemein ist die CDD Matrix wie bei [Cab10; Sie16] darauf ausgelegt, dass dynamisch neue Rollen dazu kommen können.

Durch dieses Vorgehen kann für jedes Team eine CDD Matrix mit formal fundierter Semantik angelegt werden. In der praktischen Umsetzung ist zu beachten, dass Rollen auch von unterschiedlichen Personen gleichzeitig belegt werden können. Zum Beispiel gibt es nicht nur eine Person, die in der Entwicklung in einem Team tätig ist, sondern mehrere. Hierdurch ist es nun möglich die Koordination im Team konzeptionell sauber zu beschreiben. Erweiterungen der Matrix sind für die inhaltlichen und zeitlichen Anpassungen notwendig. Mehrere Matrizen sind für einzelne (Sub)Teams sinnvoll.

Die Matrizen dienen der besseren Fokussierung der Beteiligten und einer expliziten Repräsentation der wichtigsten Dinge, die als nächstes zu tun sind. Hieraus folgt eine ständige Begleitung des Projektes durch die Matrix zur Unterstützung.

4.3 CDD Matrix für ein Team mit externen Rollen

Um aber nachvollziehen zu können, welche Aktivitäten innerhalb eines Meilensteins/Sprints von einem Team durchgeführt werden, ist es nicht ausreichend die teaminternen Rollen zu betrachtet. Weshalb zusätzlich die (team)externe Rollen betrachtet werden. Somit könnte die Matrix für einen Meilenstein/Sprint mehr oder weniger Rollen enthalten als in einem anderen Sprint, obwohl dasselbe Team betrachtet wird. Ein beispielhafter Ausschnitt einer solchen Matrix ist in Tabelle 5 zu sehen.

Zusätzlich werden jetzt auch Aktivitäten mit in Betracht gezogen, welche entweder nur Rollen oder nur Interaktionen betreffen. Solche Aktivitäten wurden bereits in [Sie16; Röd16] mit in der CDD Matrix verankert, wodurch alle Aufgaben und Synchronisationen abgebildet werden können.

Verantwortlich-keitsrollen / Interaktionen	AM Feature A	Teamleitung Feature A	Entwicklung Feature A	AM Feature B
Ticket für Feature A erstellen	[1,2]			
Feature A planen		[1,1]	[2,1]	
Feature A implementieren			[1,2]	
Treffen B durchführen	[1,1]			[2,1]

Tab. 5: Ausschnitt einer CDD Matrix für ein Team mit externen Rollen

4.4 CDD Matrix für ein Projekt

Bis hierhin gibt es für jedes Team für jeden Meilenstein/Sprint eine Matrix. Um für das gesamte Projekt einen Überblick zu erreichen, wird eine aggregierte Sichtweise auf die

Matrizen benötigt. Die aggregierte Sichtweise bedeutet, dass in den Spalten Teams betrachtet werden, da ein Team mehrere Rollen enthält, und in den Zeilen Interaktionen eines Teams betrachtet werden. Hierdurch wird insbesondere die Kommunikation über Teamgrenzen hinweg sichtbar. Interaktionen lassen sich zu komplexeren Interaktionen gruppieren. Die Auslöser sind dann entsprechend abstrakter und haben komplexere Parameter.

Ein beispielhafter Ausschnitt einer solchen Matrix ist in Tabelle 6 zu sehen. Auf der Diagonalen dieser Matrix wären jetzt genau die Meilensteine/Sprints der Teams zu finden. Auf dieser Abstraktionsebene ist bisher lediglich zu sehen, dass es Interaktionen gibt, die mehrere Teams betreffen. Hier dargestellt durch ×.

Teams / Interaktionen	Team A	Team B	Team C
Interaktionen von Team A	×	×	
Interaktionen von Team B	×	×	
Interaktionen von Team C			×

Tab. 6: Grobe Sicht: Ausschnitt einer CDD Matrix für ein Projekt

Dadurch, dass die Matrix für das Projekt eine aggregierte Sichtweise auf die Matrizen darstellt, lässt sich eine Verfeinerung vorstellen. Durch dieses würden auf der X-Achse die Rollen der Teams sichtbar werden und auf der Y-Achse die einzelnen Interaktionen des Teams. Wodurch genau die Team Matrizen mit den externen Rollen auf der Diagonale ablesbar werden.

Zusätzlich wäre es möglich, dass nicht nur eine Verfeinerung angewendet wird, sondern auch ein Aus- bzw. Einblenden von Rollen oder Interaktionen. Hierdurch könnte man unter anderem eine Sicht auf die Matrix schaffen, die alle Tätigkeiten der Teamleitung anzeigt.

Außerdem lässt sich vorstellen, dass man die Zeilen und Spalten der Matrix miteinander vertauscht oder auch mehrfach einblendet, wodurch ein oder mehrere Sprints eines oder mehrerer Teams in der Matrix betrachten werden können.

Ergänzend ist auch die Anwendung von Filtern auf die Matrix vorstellbar. Hierdurch wird es zum Beispiel möglich nur die Treffen zwischen Projektteilnehmenden anzuzeigen.

4.5 Beispiel: Kommunikation über Teamgrenzen

Wie in dem vorherigen Abschnitt beschrieben, lassen sich unterschiedliche Sichten auf die Matrix definieren indem man z.B. unterschiedliche Abstraktionsgrade annimmt. Dazu werden gewisse Zeilen und Spalten mehrfach eingeblendet oder Rollen bzw. Interaktionen ein- oder ausgeblendet. Hierdurch lassen sich insbesondere Sichten auf die Matrix definieren, die Kommunikation über Teamgrenzen hinweg sichtbar machen.

Ein Beispiel hierfür wurde in der Tabelle 6 abgebildet. Hierbei sind die Teams als Spalten abgebildet und ihre Interaktionen als Zeilen. Es ist offensichtlich, dass jedes Team an seinen eigenen Interaktionen beteiligt ist. Somit befinden sich alle internen Interaktionen ausschließlich auf der Diagonalen. Im Gegensatz zu externen Interaktionen, welche nicht auf der Diagonalen liegen. Hieran lässt sich erkennen, dass Interaktionen zwischen den Teams A und B stattfinden.

Um genauer sehen zu können, an welchen Interaktionen welche Teams beteiligt waren, müssen die einzelnen Interaktionen betrachtet werden. Hierfür kann die bereits beschriebene Verfeinerung verwendet werden. Zusätzlich zur Verfeinerung wird das Team C herausgefiltert und die Interaktionen werden nach Treffen gefiltert. Wodurch eine Sicht auf die Matrix definiert werden kann, wie sie in Tabelle 7 zu sehen ist. In der Tabelle ist ein tägliches Treffen A zu sehen. Anhand der Präzedenzen wird deutlich, dass diese Treffen lediglich teamintern sind.

Teams / Interaktionen	Team A	Team B
Tägliches Treffen A	[1,1]	
Treffen X	[1,2]	[1,2]
Tägliches Treffen B		[1,1]
Treffen X	[1,2]	[1,2]

Tab. 7: Verfeinerte Sicht: Ausschnitt einer CDD Matrix für ein Projekt

Anhand der Präzedenzen des Treffens X ist zu erkennen, dass beide Teams beteiligt sind und es sich somit um Kommunikation über Teamgrenzen hinweg handelt. Hierbei könnte es sich zum Beispiel um inhaltliche Überschneidungen der Teams handeln, die diskutiert werden müssen. Solche Überschneidungen treten auf, da mit mehreren Teams an einem Stück Software gearbeitet wird.

Außerdem fällt bei der Betrachtung der Matrix auf, dass das Treffen X als Interaktion zweimal vorhanden ist, einmal in dem Block von Team A und einmal in dem Block von Team B. Entstanden ist dies durch die Verfeinerung in die Interaktionen der Teams, wodurch deutlich wird, dass das Ticket in beiden Sprints der Teams Bestand hatte. Jedoch wurde zuvor schon beschrieben, dass solche mehrfachen Einblendungen in Sichten auf die Matrix möglich sind.

Abschließend wird deutlich, dass es sich bei allen Interaktionen, deren Präzedenzen nicht ausschließlich auf der Diagonale, in Tabelle 7 hellgrau dargestellt, liegen, um Interaktionen handelt, die teamübergreifende Aktionen benötigen.

4.6 Diskussion des Einsatzes von CDDs

Wie bereits in dem Abschnitt 4.5 beschrieben, ist es durch CDDs möglich die Kommunikation zwischen Teamgrenzen zu explizieren. Zusätzlich lassen sich Zyklen innerhalb von CDDs erkennen, wie in Abschnitt 2.4 beschrieben. Jedoch stellen Zyklen im Kontext von Projekten anwendungsbezogene Widersprüche zu den Erwartungen der Beteiligten dar. Diese Widersprüche müssen zur korrekten und konsistenten Bearbeitung aufgelöst werden. Letztlich kann durch das Festlegen der Präzedenzen verifiziert werden, ob das Projekt in Hinblick auf die geplanten Abläufe durchführbar ist, also keine Zyklen enthält.

Dadurch, dass sich die Präzedenzen der Matrizen bei jedem Treffen ändern können, kann es aufwendig werden die CDDs zu pflegen. Außerdem wächst das Problem, dass die Matrix mit der Größe und der Zeitdauer des Projekts zunimmt und es somit schwieriger wird alle Matrizen zu überblicken. Deshalb sollte der Umgang mit den Matrizen durch ein Werkzeug unterstützt werden.

Bei dem Umgang mit CDDs ist es wichtig zu beachten, dass diese keine Termine abbilden. Durch die Meilensteine/Sprints bekommen CDDs zwar einen temporalen Charakter, aber dieser ist nicht an feste Termine gebunden, sondern lediglich an den Meilenstein/Sprint. Im Gegensatz zu den starrer angelegten Ablaufplänen der Netzplantechnik[Alt96] werden in agilen Projekten im Wesentlichen Abhängigkeiten von Tätigkeiten verwendet. Durch die Aufwandsschätzung der Features ergibt sich dann eine zeitliche Anordnung. Hier kann für die Zukunft in Werkzeugen eine Option für eine Planungskomponente gesehen werden, um kritische Pfade zu berücksichtigen. Zusätzlich ist zu beachten, dass eine CDD Matrix nicht das Projektmanagement ersetzt. Es kann aber vom Projektmanagement als Technik werkzeugunterstützt verwendet werden.

Der Ansatz erlaubt die gezielte Ausbildung im Kontext von Lehrveranstaltungen, da Fähigkeiten-spezifisch die Personen auf Aufgaben zugewiesen werden können. Über die Zeit hinweg können die beteiligten Personen ihre Fähigkeiten immer wieder selbst erproben und so vertiefen.

5 Fallstudienkontext

Als Beispiel verwenden wir als Fallstudie die Entwicklungsprojekte von RENEW an unserem Fachbereich Informatik [Kum+20]. RENEW ist ein Werkzeug, mit dem es möglich ist, unterschiedliche Netzarten zu modellieren, zu analysieren und zu simulieren [Kum+20]. Bei diesen Netzarten handelt es sich unter anderem um Platz-/Transitionsnetze [Rei13], Workflownetze [Aal97] oder Referenznetze [Kum02]. In den letzten vier Jahren wurde die Entwicklung von RENEW selbst ausschließlicher Gegenstand der Entwicklungstätigkeit im Lehrprojekt. Im Fokus stand insbesondere die Modularisierung der Codebasis, ca. 80 Java Code Plugins. Basis war dafür die jahrzehntelange Erfahrung bei der Entwicklung von komplexen Multiagentensystemanwendungen auf Basis des PAOSE-Ansatzes.

Das Lehrprojekt wird hier als Fallbeispiel für die in diesem Beitrag weiterentwickelte CDD Matrix genutzt, da in diesem Kontext die Konzepte kontinuierlich von den Veranstaltenden entwickelt wurden. Die Matrizen wurden im Laufe des Projektes von den Veranstaltern als Gedankenmodell verwendet. Hierfür führen wir eine Ex-Post-Evaluation als Gedankenexperiment durch. Hierdurch soll überprüft werden, ob die Weiterentwicklungen der Matrizen sich auf unser Projekt anwenden lassen. Die Evaluation wurde nach dem Projekt durch die Projektorganisatoren durchgeführt.

Das Lehrprojektmodul wird in mehreren Studiengängen angeboten. Zusätzlich hinzu kommen Teilnehmende, die ihre Abschlussarbeit im Kontext von RENEW schreiben. Die Unterrichtenden des Projekts setzen sich aus den Veranstaltern, Wissenschaftlichen Mitarbeitern und Studentischen Hilfskräften zusammen. Durch diese Konstellation konnte im Wintersemester 2021/2022 eine Teilnehmendenanzahl von ca. 40 Personen erreicht werden, die somit nur geringfügig größer war als die letzten Jahre.

Somit wurde die Entwicklung von RENEW durch ein heterogenes Team durchgeführt. Als Projektmanagementansatz wurde eine abgewandelte Version des PAOSE-Ansatzes verwendet, die an Scrum @ Scale nach [Sut22] angelehnt ist. Damit erfolgte die Konkretisierung der hier vorgestellten CDD Matrix. In jedem Team sind mindestens ein Product Owner (PO), ein Scrum Master (SM) und mehrere Entwickler vertreten. Aufgrund der Scrum @ Scale Skalierung auf sechs Teams gab es einen Scrum of Scrums Master (SoSM) und einen Chief of Product Owner (CPO), deren Rollen durch die Veranstalter eingenommen wurden. Mit einer Einführungsphase werden Teilnehmende auf die gemeinsame Terminologie mittels Schulungen für ihre Aufgaben im Projekt vorbereitet. So werden die Scrum Rollen Entwicklung, Product Owner und Scrum Master Tätigkeiten systematisch geschult, sofern das jeweils vorhandene Wissen nicht reicht.

In der zweiten Phase wurde die Entwicklung von RENEW durch alle Beteiligten vorangetrieben. Die Teilnehmenden wurden für die zweite Phase in sechs Teams (jeweils Scrum Master (SM), Product Owner (PO) und 4 bzw. 5 Developer) eingeteilt, welche miteinander koordiniert werden müssen. Scrum Master und Product Owner sollten und konnten aus didaktischen Gründe auch als Entwickelnde tätig sein, da als ein Lernziel letztlich alle Beteiligten alle Rollen verstehen und auch übernehmen können sollen.[6]

Für alle Beteiligten ergaben sich schon durch die große Anzahl an Beteiligten Schwierigkeiten der Orientierung. Hinzu kam das Problem der komplexen Anwendung. Insgesamt ist daher die Herausforderung, dass die Aufgaben und deren Reihenfolge eine zentrale Bedeutung erlangten. Die Unterstützung nur durch die Werkzeuge Confluence [Atl22a] und Jira [Atl22b] reichte dabei nicht aus. Die üblicherweise eindeutige Zuordnung von den drei Scrum-Rollen sollte vor dem Hintergrund der Lehre nicht fix sein. Dem PAOSE-Ansatz folgend konnten Teammitglieder im Anschluss an Treffen kurzfristig nicht nur andere inhaltliche Aufgaben übernehmen, sondern auch in andere Rollen schlüpfen.

[6] An dem Lehrprojekt nahmen auch Studierende teil, die ihre Abschlussarbeit in dem Rahmen des Projekts erstellten. Diese wurden meist als PO eingesetzt. Die Masterstudierenden übernahmen SM Rollen.

In der Tabelle 8 ist ein Ausschnitt aus der CDD Matrix des RENEW-Projektes zu sehen. Dadurch, dass der CPO und SoSM keinem Team zugeordnet sind bilden sie jeweils ihr eigenes Team.

Wie schon erwähnt, lassen sich unterschiedliche Sichten auf die Matrix definieren. Beispielsweise könnte eine Verfeinerung und ein Filtern nach Treffen durchgeführt werden. Hierdurch würde man zu der Ansicht in der Tabelle 9 kommen. In dieser Ansicht ist die Kommunikation über Teamgrenzen hinweg sichtbar.

Zusätzlich lässt sich wie bereits beschrieben aus den Präzedenzen ein Präzedenzgraph bilden, welcher einfach als Kausalnetz dargestellt werden kann. Durch die Darstellung als Kausalnetz wiederum lassen sich Zyklen und somit Widersprüche in der Projektdurchführung identifizieren. Wodurch eine Prüfung der Durchführbarkeit eines Projekt durch die Modellierung der Projektorganisation mittels Matrizen möglich ist.

Hieraus ergibt sich, dass, wenn die Konzepte des Papiers in einem Werkzeug implementiert werden, kann dieses sowohl für die Prüfung der Durchführbarkeit genutzt werden als auch damit jeder Projektteilnehmer zu jedem Zeitpunkt weiß was zu tun ist und welche Zyklen in den Aufgaben zu lösen sind. Weshalb eine Verwendung in einem unserer nächsten Projekte sinnvoll scheint.

Interaktionen \ Teams	Team A	Team B	CPO	SoSM
Interaktionen von Team A	×	×	×	×
Interaktionen von Team B	×	×	×	×
Interaktionen des CPO	×	×	×	
Interaktionen des SoSM	×	×		×

Tab. 8: Grobe Sicht: Ausschnitt aus der CDD Matrix für das RENEW-Projekt

Interaktionen \ Teams	Team A	Team B	CPO	SoSM
Daily Scrum A	[1,1]			
Treffen X	[1,3]	[1,3]		
Treffen CPO und alle POs	[1,2]	[1,2]	[2,1]	
Treffen SoSM und allen SMs	[1,2]	[1,2]		[2,1]
Daily Scrum B		[1,1]		
Treffen X	[1,3]	[1,3]		
Treffen CPO und alle POs	[1,2]	[1,2]	[2,1]	
Treffen SoSM und allen SMs	[1,2]	[1,2]		[2,1]
Treffen CPO und alle POs	[1,2]	[1,2]	[2,1]	
Treffen SoSM und allen SMs	[1,2]	[1,2]		[2,1]

Tab. 9: Verfeinerte Sicht: Ausschnitt aus der CDD Matrix für das RENEW-Projekt

6 Schluss

Die Frage, *Wie lassen sich komplexe Projektabläufe modellieren und an Studierende so vermitteln, dass sie eine Orientierung über die Abläufe haben?*, wird mithilfe der hier vorgestellten Matrizen beantwortet. Innerhalb eines Team werden lokal Präzedenzen festgelegt und diese werden übergreifend abgestimmt, so dass sich für das gesamte Projekt klare Präzedenzen ergeben. Damit können sich alle Beteiligten sehr gut an der CDD-Matrix orientieren. Haben Studierende / Projektbeteiligte mehr Erfahrung und Wissen, dann können sie bekannte Metaphern und Konzepte des Projektmanagement mit einer formalen Semantik von Petrinetzen unterlegen. Dies ist jedoch, wie sich gezeigt hat, nicht zwingend notwendig, da das Grundverständnis für Präzedenzen aufgrund der erstrebten Kausalnetzstrukturen der Aktivitäten für einfache Koordination ausreicht.

Für eine Übersicht über ein Projekt mit 40 Teilnehmenden sind dazu allerdings eine systematische Arbeitsweise und Erfahrungen beim Projektmanagement notwendig. Den Studierenden gelang die Koordination ihrer teamlokalen Aufgaben und die Einordnung in das Gesamtprojekt sehr gut, eine Übersicht über das Gesamtprojekt wurde jedoch in in den allermeisten Fällen nicht erreicht, wurde aber auch nicht direkt angestrebt. Die POs und SMs konnten aufgrund der übergeordneten Treffen eine hinreichende Übersicht gewinnen, um teamübergreifende Abhängigkeiten zu identifizieren und zu klären.

Gerade im Kontext der Lehre, bei der die grundlegende Theorie zusätzlich zu den praktischen Entwicklungstätigkeiten im Projekt reflektiert wird, ergeben sich vertiefte Einsichten in das Projektmanagement. Das Verständnis des kollaborativen gemeinsamen, eigenverantwortlichen, gesamtverantwortlichen Handelns in einem großen Team wird durch das konkrete Erleben der Situationen erheblich vertieft. Die Verknüpfung von zielgerichteter Arbeit (Anwenden von Erlerntem) und zielgerichtetem umfassenden Lernen wird durch die gemeinsamen Erfahrungen von Stärken und Schwächen der anderen Beteiligten in praktischen Kontexten erreicht.

Durch diesen Beitrag wird eine Möglichkeit entwickelt, mithilfe von Matrizen die Projektorganisationen und deren Tätigkeitsabhängigkeiten zu modellieren. Hierfür wurde die CDD Matrix aus dem PAOSE-Ansatz auf ein Projekt angewendet und erweitert. Als Erweiterungen sind hier sowohl die unterschiedlichen Sichten auf die Matrix, die durch Verfeinerung, Ein- oder Ausblenden von Zeilen oder Spalten oder die Anwendung von Filtern zustande kommen, als auch die Verwendung von Präzedenz-Tupeln, welche es ermöglichen Widersprüche zu identifizieren, zu erwähnen. Die möglichen Sichten auf die Matrizen verbessern die Übersicht über die verschiedenen Aufgaben und Abhängigkeiten zwischen den Aufgaben für die Projektmitglieder. Durch die Klarheit über die Abhängigkeiten ist die notwendige Abarbeitungsreihenfolge für alle verständlich.

Die hinter den Matrizen stehende formale Semantik in Form der Petrinetze wird bei uns intern als Denkmodell benutzt. Weiterhin werden Werkzeuge, die wir intern selbst erstellt haben, generell auf Petrinetze abgebildet. Dabei kommt unsere RENEW Umgebung zum Einsatz,

die die Basis für den PAOSE-Ansatz liefert. Erweiterungen in Richtung Branching Prozesse (Ereignisnetze) und deren Faltung bei Wiederholungen zu speziellen Workflow-Netzen sind Gegenstand unserer weiteren Untersuchungen bei der Entwicklung von Werkzeugen.

Literatur

[Aal97] Wil van der Aalst. „Verification of Workflow Nets“. In: *Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, Proceedings*. Hrsg. von Pierre Azéma und Gianfranco Balbo. Lecture Notes in Computer Science 1248. Berlin Heidelberg New York: Springer Verlag, 1997, S. 407–426.

[Alt96] Günter Altrogge. *Netzplantechnik*. 3. Aufl. De Gruyter Oldenbourg, 1996. ISBN: 978-3486237283.

[Atl22a] Atlassian. *Confluence Homepage*. https://www.atlassian.com/de/software/confluence. [last accessed: 2022-03-08]. 2022.

[Atl22b] Atlassian. *Jira Homepage*. https://www.atlassian.com/de/software/jira. [last accessed: 2022-03-08]. 2022.

[Cab10] Lawrence Cabac. „Modeling Petri Net-Based Multi-Agent Applications“. Dissertation. Vogt-Kölln Str. 30, D-22527 Hamburg: Universität Hamburg, Department Informatik, Apr. 2010. URL: https://ediss.sub.uni-hamburg.de/handle/ediss/3691.

[Con68] Melvin E. Conway. „How Do Committees Invent?“ In: *Datamation* (1968).

[Kum+20] Olaf Kummer u. a. *Renew – The Reference Net Workshop*. Release 2.5.1. Nov. 2020. URL: http://www.renew.de/.

[Kum02] Olaf Kummer. *Referenznetze*. Berlin: Logos Verlag, 2002. ISBN: 978-3-8325-0035-1. URL: http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=.

[Rei13] Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Berlin, Heidelberg, New York: Springer-Verlag, 2013.

[Röd16] Christian Röder. „Entwicklung eines Netzplantechnik-Plugins für Renew und eine prototypische Einbettung von Netzplantechnik in den PAOSE-Softwareentwicklungsansatz“. Diplomarbeit. Vogt-Kölln Str. 30, D-22527 Hamburg: Universität Hamburg, Fachbereich Informatik, Okt. 2016.

[Sch+18] Dennis Schmitz u. a. „Team Coordination Based on Causal Nets with Synchronous Channels“. In: *18th International Conference on Application of Concurrency to System Design, ACSD 2018, Bratislava, Slovakia, June 27-29, 2018*. IEEE Computer Society, 2018, S. 60–69. ISBN: 978-1-5386-7013-2. URL: https://doi.ieeecomputersociety.org/10.1109/ACSD.2018.00009.

[Sie16] Joanna Sieranski. „Diskussion und Weiterentwicklung der Rollen-Interaktionsmatrix zur Unterstützung des Projektmanagements in der PAOSE“. Masterarbeit. Vogt-Kölln Str. 30, D-22527 Hamburg: Universität Hamburg, Fachbereich Informatik, Okt. 2016.

[Sut22] Jeff & Scrum Inc. Sutherland. *The Scrum@Scale Guide*. https://www.scrumatscale.com/wp-content/uploads/2020/12/official-scrum-at-scale-guide.pdf. [last accessed: 2022-02-11]. 2022.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 145

Supporting Requirements Engineering and Development with Event Modeling - an Overview

Sebastian Copei [1], Clemens Emme [2], Maximilian Freiherr von Künßberg [3], Adam Malik [4], Natascha Nolte [5], Ulrich Norbisrath [6], Albert Zündorf [7]

Abstract: Workflow based systems are frequently modeled using BPMN diagrams for requirements engineering and UML diagrams for analysis and design. While such diagrams are great input for the software development, non-ICT people like domain experts, customers, end users or business people usually do not understand their content. Therefore, non-ICT people are frequently not able to contribute to these diagrams nor able to identify missing or incorrect parts. This paper proposes the very informal event storming approach as a means to involve users into the requirements engineering activities. We then refine the event storming results via Event Modeling by adding elaborated user interface scenarios. These event models then serve as input for integrated data modeling steps.

Keywords: Requirements Engineering; Design; Event Storming; Wireframes; Data Modeling

1 Introduction

This paper focuses on workflow based systems like Enterprise Resource Planning (ERP), management, or administrative systems. Generally, a software project consists of requirements engineering phases, design phases, development phases, and deployment and operation phases. For workflow based systems, there exists a large body of knowledge, methods, languages, and tools for requirements engineering especially in the area of Business Process Models and Notations (BPMN) [OPM11, Sc00]. Similarly, there is a large body of knowledge, methods, languages, and tools for software architecture, design, and modeling – notably the Unified Modeling Notation [Bo96], the Unified Modeling Process [JBR99], and Design Pattern [Ga95]. In spite of these achievements, there are still a lot of open problems to be solved:

Requirements engineering often uses formal notations like BPMN diagrams to specify workflows precisely. Such notations are good for the communication towards the development

[1] Software Engineering, Kassel University, Germany, sco@uni-kassel.de
[2] Software Engineering, Kassel University, Germany, clemens@uni-kassel.de
[3] Software Engineering, Kassel University, Germany, maximilian-kuenssberg@uni-kassel.de
[4] Adam Malik Consulting, Germany, am@adam-malik.de
[5] Software Engineering, Kassel University, Germany, natascha@uni-kassel.de
[6] Distributed Systems, University of Tartu, Estonia, ulino@ut.ee
[7] Software Engineering, Kassel University, Germany, zuendorf@uni-kassel.de

doi:10.18420/modellierung2022-007

team, but formal notations like BPMN are usually hard to understand for non-ICT people like problem owners, domain experts, customers, end users, or business persons.

The same problems also persists in analysis and design: formal notations like UML diagrams are hard to understand and use effectively for non-ICT people. Therefore, it is hard for customers and end users to contribute to analysis and software design.

Frequently, requirement, analysis, and design documents are not well integrated nor aligned. Implementing, matching, and verifying of such requirements is hard and again contributing to the problem of non-ICT people feeling or actively seeking to be excluded from the software analysis and design process.

Due to our experience, non-ICT people are easily involved into the development when it comes to user experience (UX) e.g. via wireframes or mockups. Unfortunately, UX activities are often considered as a minor and late activity that is done after analysis and design. This causes user feedback to be quite late. In addition, UX activities are often not well integrated with the requirements elicitation activities. Again this is a difficulty for the contribution of non-ICT people.

To address some of these problems, we propose a slightly adapted software development process: iterate through 1. do some event storming (analog), 2. model events, wireframes, and read models for some steps and discuss these with users, 3. derive data modeling from the wireframes for the developers, 4. implement the modeled steps, 5. deploy. The following sections explain these steps.

2 Related Work: Event Storming

Event Storming has been proposed by Alberto Brandolini [Br13, Br21b]. Figure 1 shows an example of the result of an Event Storming workshop, which he teaches through workshops [Br21a] or presents in social media (eg. Youtube) [Br19]. Brandolini proposes to invite a group of relevant domain experts, customers and end users from all parts of your customer's enterprise. Each participant contributes to the workshop, by using sticky notes and a board marker on a large empty wall paper. By doing so, the participants create a contributive model containing of domain events and actions triggering such events.

Brandolini states that with the help of some moderation such an Event Storming workshop achieves a high involvement of non-ICT people. He claims that you should come up with the identification of all relevant workflows within a one or two day workshop. In addition, the grouping of people in front of the Event Storming Board already reveals some organisational structures and candidates for bounded contexts [EE04].

A great contributor to the facilitating effect is that Event Storming uses just pencil and paper and no formal notation, thus everybody can participate and gets easily involved. The

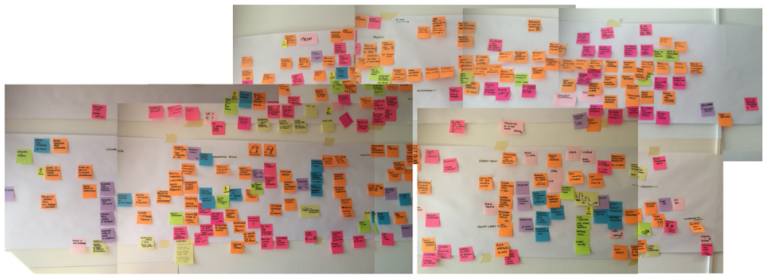


Fig. 1: Event Storming Big Picture ala Brandolini from [Br21b]

"Story Telling" part also fits very natural to the people. This is an experience that we share [Zü05, Zü19, NJZ13], and that attracted us to Event Storming.

The notation of BPMN diagrams is slightly related to event storming diagrams. BPMN diagrams provide decision points where workflows may branch into different directions. This helps to address all possibilities within a single diagram. Event Storming focuses on representative example workflows. If there are decisions that result in different behavior, we need to provide alternative scenarios for each case. The advantage of the alternative scenarios is that they outline in which situation which behaviour is chosen. This has the risk that some special case scenarios might not be covered. However, in our experiences domain experts are really good in pointing us to missing cases and help coming up with scenarios for these special cases. Thus, we believe that event storming scenarios are more likely to cover all relevant cases as BPMN diagrams that have been created by requirements experts. Still, we might summarize alternative Event Storming scenarios into a common BPMN diagram in order to have a condensed input for the analysis and design phases.

3 Related Work: Event Modeling

Once we have done some requirements elicitation e.g. with Event Storming, we need to become a little bit more formal in order to do analysis and design. However, we still want the non-ICT people to be involved and we want analysis and design integrated with the requirements steps. Therefore, the idea is to add wireframes to the Event Storming that show scenario steps from the user interface perspective. This refinement step is called *Event Modeling*. Thus, Event Modeling integrates UX design into Event Storming.

On writing this paper we learned that Adam Dymitruk has coined the phrase *Event Modeling* in 2018 [Dy18]. Actually, Dymitruk's ideas are quite close to ours. He also proposes to use wireframes and read models to refine Event Storming stories. Adding details to them allow for better user understanding and more preciseness and are a good starting point to the design and implementation phase. However, Dymitruk uses Event Modeling still in a very informal way on a whiteboard. In addition, he uses explicit command notes and read

model notes that we omit as they provide little extra information. Finally, we use a formal notation for the description of an event model. This formal notation allows us to generate different views on the system, e.g. a clickable mockup prototype, see below.

4 Event Modeling with FulibWorkflows

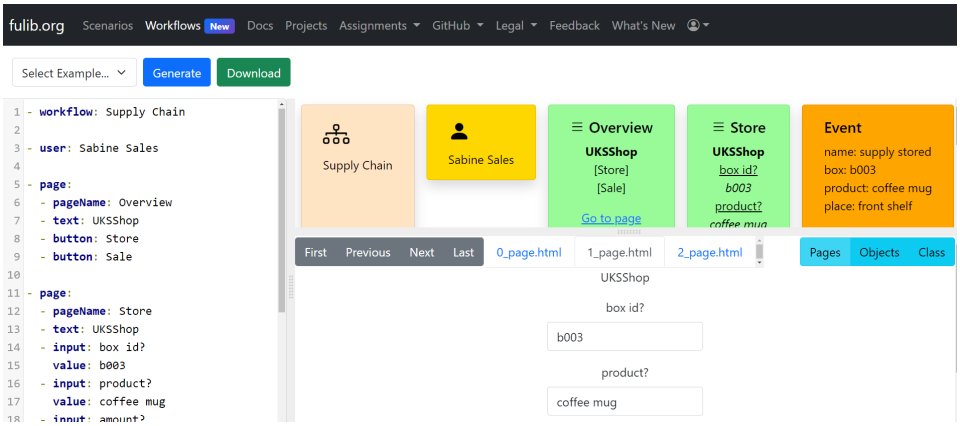


Fig. 2: The fulibWorkflows Tool

Figure 2 shows a screenshot of our *fulibWorkflows tool* with an example for a small event model for a University Kassel (UKS) web shop. The tool is available at our main tool site fulib.org [KFvK22]. In Figure 2 we have just started to formalize some domain events and added some wireframes. The left panel is an editor for our textual event modeling notation. In the upper right panel you see a graphical representation of the domain events in an event storming like event board. The lower right panel shows the mockups of a simple click prototype that is derived from our event model.

The fulibWorkflows tool uses a very simple Yaml based notation as input, cf. Listing 1, 2, and 3. We call this our *Event Modeling Notation*. Although, our event modeling notation tries to be as simple as possible, we do not expect that non-ICT people will be able to read and write event models in this notation. Thus, writing event model descriptions shall be done by requirements engineers or other software people. However, from the event modeling notation, we can derive the event storming board view containing events and simple wireframes as simple clickable mockups. These additional views allow us to integrate non-ICT people in our event modeling activities.

Workflows and Events

Listing 1 shows our Event Modeling notation for workflows and events. Lines 1 and 10 introduce the workflows “Supply” and “Selling”, respectively. Lines 2 to 4 introduce a

"supply ordered" event with the additional fields "product" and "amount". These additional fields are used to provide details on the information that is provided by an event and it will be input for our data modeling step. Usually, these additional fields are derived from input forms, see below. Other events are shown at lines 5, 11 and 14.

```
1 - workflow: Supply
2 - event: supply ordered
3   product: coffee mugs
4   amount: 50
5 - event: supply stored
6   box: b003
7   product: coffee mug
8   amount: 50
9   place: front shelf
10 - workflow: Selling
11 - event: prod offered
12   product: coffee mugs
13   price: Euro 1
14 - event: prod ordered
15   orderId: O_001
16   product: coffe mugs
17   amount: 1
18   customer: Alice
19   address: Wonderland 1
20 # ...
```

Listing 1: Events

```
1 - workflow: Supply
2 # ...
3 - user: Sabine Sales
4 - page:
5   - pageName: Overview
6   - text: UKSShop
7   - button: Store
8   - button: Sale
9     targetPage: Prices
10  - button: "..."
11 - page:
12  - pageName: Store Box
13  - text: UKSShop
14  - input: box id?
15    value: b003
16  - input: product?
17    value: coffee mug
18  - input: amount?
19    value: 50
20  - input: place?
21    value: shelf1
22  - button: add
23 # ...
```

Listing 2: Pages

```
1 # ...
2 - data: Box b003
3   product: mug
4   amount: 50
5   place: shelf1
6 - data: Box b002
7   product: mug
8   amount: 10
9   place: shelf2
10 - data: Box b001
11   product: tshirt
12   amount: 42
13   place: shelf1
14 - data: Product mug
15   boxes: [b002, b003]
16 - data: Product tshirt
17   boxes: [b001]
18 - data: Shelf shelf1
19   boxes: [b003]
20 - data: Shelf shelf2
21   boxes: [b002]
22 # ...
```

Listing 3: Data

We render the described events into a board of workflows. Ideally, the resulting event board resembles the analog board created in our event storming workshop. As explained in the industrial expert interview [MFvK22], participants in such an event storming workshop often keep a good spatial memory of the analog event board with its workflows and their steps. For the discussion of details, it helps tremendously to point to the physical location of the overall board where the corresponding events are shown. The former participants will immediately know what part of the overall workflow you are referring to and if they are not experts in that area themselves they will at least point you to the people that work in that area and that may provide you with detailed information to answer your detailed questions. Enabling even more communication, we stick to the ubiquitous language that we have developed during the event storming workshops.

Pages

Listing 2 focuses on the "supply stored" event of our example. The "user" entry of line 3 specifies that the following activities are executed by "Sabine Sales". This is rendered

by a yellow note in our event board, cf. Figure 3. Now we have added some very simple wireframes to our event model using “page” entries cf. lines 4 to 10 and lines 11 to 22 of Listing 2, producing the green notes in Figure 3.

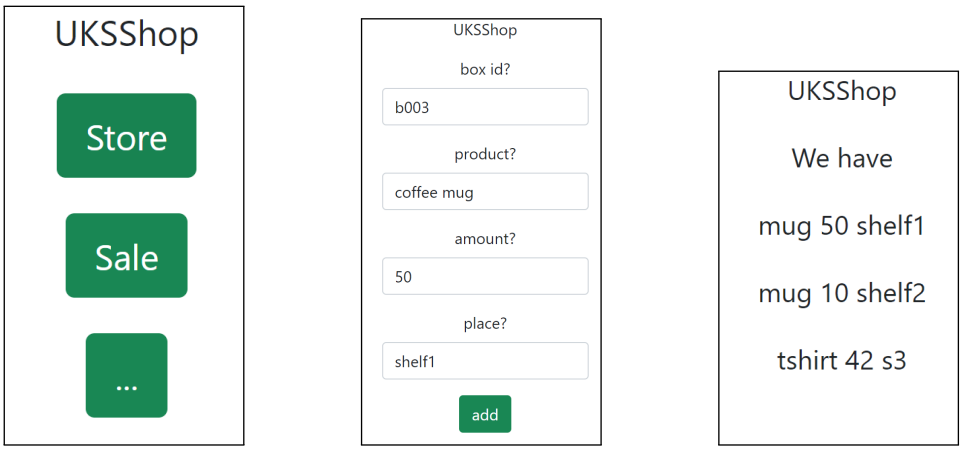
Fig. 3: Store Supply Pages

These two green pages of Figure 3 try to explain the user interaction that results in the orange “supply stored” event. The first page object describes the “Overview” page of our employee app with a [Store] button and a [Sale] button. A [...] button indicates that we expect more buttons to be added later. The [Store] button shall lead to the “Store Box” page. Per default, buttons just switch to the next page. Alternatively, line 9 of Listing 2 tells that the [Sale] button of the “Overview” page switches to a “Prices” page that is not shown in our Figures. The “Store Box” page is a form with four input fields containing the needed information for storing a new box. In our Yaml notation, you may specify example values for the input fields.

While the event board shows page details only as some ASCII art wireframe, the lower right compartment of fulibWorkflows shows an HTML based mockup of the described page. Figure 4 shows how a user might click through a scenario of 3 steps.

Figure 4a shows the mockup for the “Overview” page. If we click on the [Store] button, the lower right compartment of fulibWorkflows will switch to a mockup of our “Store Box” page. Depending on the scenario, you may first show an empty form and on another click switch to a filled form, cf. Figure 4b.

Walking users through such a little click dummy also facilitates users evaluating mockups. This possibility is also triggered by our expert interview[MFvK22]. When the users click through the mockups following the current scenario, the user will soon point you to missing information, missing inputs, missing alternatives and to superfluous or repeated inputs. In our example, Sabine Sales would probably ask for a feedback page that shows that adding the coffee mugs was successful. Thus we may add an “Inventory” page as shown in Figure 4c to our scenario. From the addition of another page and the potential discussion about the design of said page event more pages or changes to the existing workflow can emerge. Overall our experience shows that clickable mockups result in good involvement even of non-ICT people.



(a) "Overview" page (b) "Store Box" page (c) "Inventory" page

Fig. 4: Store Supply Mockups

Instead of our event modeling notation, one may use any other User eXperience (UX) approach, wireframe tool, or mockup library for the same purpose. However, due to our experience it is very important to link the UX steps to the corresponding parts of the event storming board. This reference helps the non-ICT people to recall what workflow parts are addressed and helps them to contribute to the discussion.

5 Data Model

While our detailed mockups and usage scenarios require quite some work, the resulting forms and "Inventory" pages do not only allow for excellent user involvement but they also provide a very good input for the data modeling and design phases. For example, the "Store Box" page shown in Figure 4b contains input fields for a box id, a product name, the amount of the product that the box contains and the shelf where the box is stored. These input fields will result in an event handler method with corresponding parameters and eventually in an object and class within our system model. Actually, the "Inventory" page in Figure 4c shows that there may be multiple boxes with the same product and that there are multiple products and shelves.

To allow the integration of data modeling into Event Modeling our Yaml notation uses e.g. "- data: Box b003" notes to describe objects (i.e. an object of type Box with id b003) and their attributes, cf. Listing 3. In our event board we render data notes in dark green color, cf. Figure 5.

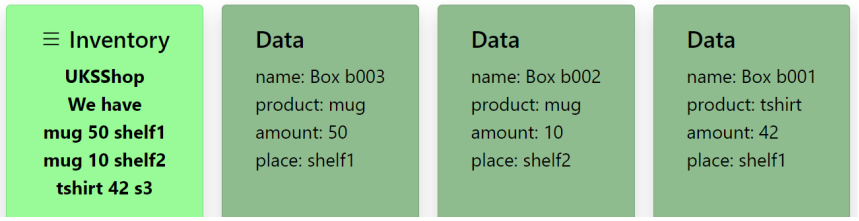


Fig. 5: FulibWorkflows Event Board with Data Notes

6 Conclusions and Future Work

As discussed, BPMN diagrams and UML diagrams are great means for providing the software development steps with concise information. However, these diagrams are hard to grasp for non-ICT people. Therefore, we propose to complement classical requirements engineering and software engineering diagrams with informal techniques like event storming. Similarly, we exploit wireframes and GUI mockups for better involvement of non-ICT people. Event modeling integrates wireframes and mockups into the event storming board. This again helps the involvement of non-ICT people. Detailed event models then serve as input for the data modeling steps. Again, our approach integrates the data modeling activities into the event storming notation in order to support consistency during iterative development.

We can even derive class diagrams for some of the core data models, we can employ code generation like using the FULib code generator [ZKE21] (or any other capable UML tool) to derive a Java implementation. However, the automatically derived class diagram is usually only a first conceptual model and thus data modeling remains an explicit engineering activity. In addition, the common class diagram tends to grow larger with the number of workflows that are incorporated. An idea for tackling the complexity of our data models is to split the whole system into a number of microservices, cf. [Co22]. [Co22] outlines that the workflows defined by Event Storming and refined through Event Modeling are a good start for splitting up monolithic models into a number of smaller micromodels.

Our approach exploits the experiences and insights of our co-author Adam Malik who uses many of our ideas in industrial projects. We have used our event modeling notation with great success in two courses at Kassel University [Zü21b, Zü21a]. We plan further evaluation of our approach within scientific as well as within industrial projects.

With fulibWorkflows [KFvK22], we have developed a first simple tool support for our event modeling approach. We plan to continue the development of the fulibWorkflows tool in order to support more enhanced wireframes and mockups. In addition, we need to develop mechanisms to connect analog event storming boards with the content of fulibWorkflows descriptions.

Bibliography

[Bo96] Booch, Grady; Jacobson, Ivar; Rumbaugh, James et al.: The unified modeling language. Unix Review, 14(13):5, 1996.

[Br13] Brandolini, Alberto: Introducing event storming. blog, Ziobrando's Lair, 18, 2013.

[Br19] Brandolini, Alberto: , 100,000 Orange Stickies Later. https://www.youtube.com/watch?v=fGm62ra_mQ8, 2019.

[Br21a] Brandolini, Alberto: , Event Storming. https://www.eventstorming.com/, 2021.

[Br21b] Brandolini, Alberto: , Introducing EventStorming. https://leanpub.com/introducing_eventstorming, 2021.

[Co22] Copei, Sebastian; Eickhoff, Christoph; Malik, Adam; Nolte, Natascha; Norbisrath, Ulrich; Sorgalla, Jonas; Weber, Jens H.; Zündorf, Albert: From Monolithic Models to Agile Micromodels. In: MODELSWARD. 2022.

[Dy18] Dymitruk, Adam: , Event Modeling: What is it. https://eventmodeling.org/posts/what-is-event-modeling/, 2018.

[EE04] Evans, Eric; Evans, Eric J: Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, 2004.

[Ga95] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John; Patterns, Design: Elements of Reusable Object-Oriented Software. Design Patterns. massachusetts: Addison-Wesley Publishing Company, 1995.

[JBR99] Jacobson, Ivar; Booch, Grady; Rumbaugh, James: The unified process. Ieee Software, 16(3):96, 1999.

[KFvK22] Kunz, Adrian; Freiherr von Künßberg, Maximilian: , fulib.org. https://fulib.org/workflows, 2022.

[MFvK22] Malik, Adam; Freiherr von Künßberg, Maximilian: , Expert interview on using Event Storming for Requirements Engineering in an industrial context (In German). https://www.youtube.com/watch?v=2hYlh_xENDY, 2022.

[NJZ13] Norbisrath, Ulrich; Jubeh, Ruben; Zündorf, Albert: Story driven modeling. CreateSpace Independent Publ. Platform, 2013.

[OPM11] Omg, OMG; Parida, R; Mahapatra, S: Business process model and notation (bpmn) version 2.0. Object Management Group, 1(4), 2011.

[Sc00] Scheer, August-Wilhelm: ARIS—business process modeling. Springer Science & Business Media, 2000.

[ZKE21] Zündorf, Albert; Kunz, Adrian; Eickhoff, Christoph: Addressing Industrial Needs with the Fulib Modeling Library. In: MODELSWARD. pp. 155–162, 2021.

[Zü05] Zündorf, Albert: Story driven modeling: a practical guide to model driven software development. In: Proceedings of the 27th international conference on Software engineering. pp. 714–715, 2005.

[Zü19] Zündorf, Albert; Copei, Sebastian; Diethelm, Ira; Draude, Claude; Kunz, Adrian; Norbisrath, Ulrich: Explaining Business Process Software with Fulib-Scenarios. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). IEEE Computer Society, pp. 33–36, 2019.

[Zü21a] Zündorf, Albert: , Microservices Course WT 21-22, Kassel University. https://www.youtube.com/playlist?list=PLohPa1TMsVqpdFQ8a2fPw1goIwwTMRolJ, 2021.

[Zü21b] Zündorf, Albert: , Programming and Modeling Course WT 21-22, Kassel University. https://www.youtube.com/playlist?list=PLohPa1TMsVqoveiQzR6nPqkV9OaJWzTCI, 2021.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 155

Verhalten und Ausführungssemantik von erweiterten Sequenzkanten in Geschäftsprozessen

Thomas Bauer[1]

Abstract: Bei der Modellierung von Geschäftsprozessen (GP) werden Aktivitäten üblicherweise als atomare Einheiten betrachtet. Dies führt zu unnötigen Einschränkungen, wenn z.B. Sequenzen von Aktivitäten modelliert werden. Hier kann die Flexibilität erhöht werden, indem sich Sequenzkanten beliebig auf die Start- und Endeereignisse ihrer Quell- und Zielaktivitäten beziehen können. Dadurch werden zur Ausführungszeit der GP (Runtime) zusätzliche Ausführungsreihenfolgen möglich, d.h. die Benutzer haben mehr Flexibilität bei der Prozessbearbeitung. Dennoch werden selbstverständlich alle modellierten Kontrollflussbedingungen ebenso eingehalten, wie zeitliche Constraints zwischen Aktivitäten (z.B. Mindestzeitabstände). Damit eine Prozess-Engine einen GP entsprechend dieser Bedingungen (automatisch) steuern kann, ist eine formale Ausführungssemantik erforderlich. Sie wird in diesem Beitrag ebenso vorgestellt, wie Maßnahmen, mit denen die Prozess-Engine den Start bzw. die Beendigung von Aktivitäten verzögern und beschleunigen kann, um so das gewünschte Verhalten bei der GP-Ausführung zu erreichen.

Keywords: Geschäftsprozess; Flexibilität; Kontrollfluss; Sequenz; Zeit; Workflow-Engine

1 Einleitung

Ziel des Projektes CoPMoF (Controlable Pre-Modeled Flexibility) ist, die Flexibilität von PAIS (Process-aware Information System) zu erhöhen. Allerdings werden hierzu keine dynamischen Änderungen (vgl. [RW12]) eingesetzt. Stattdessen wird in einem Geschäftsprozess (GP) zu erwartender Flexibilitätsbedarf bereits zur Buildtime vor-modelliert. Zur Runtime muss diese Flexibilität dann nur noch genutzt (angewandt) werden, was für den Endbenutzer deutlich einfacher und mit weniger Aufwand durchzuführen ist. Außerdem ist Flexibilität dadurch nur an tatsächlich erwünschten (d.h. vom Prozessverantwortlichen vorgesehenen) Stellen verfügbar und es können Benutzerrechte für deren Anwendung vergeben werden. Im Projekt CoPMoF wurde untersucht, welche Arten von Flexibilität üblicherweise benötigt und vormodelliert werden können. Dabei wurden sowohl Flexibilitätsbedarfe für den Kontrollfluss von GP (z.B. optionale und alternative Aktivitäten) [Ba21b] als auch für andere Prozessperspektiven (z.B. alternative Bearbeiterzuordnungen) [Ba19b] betrachtet.

Eine der Möglichkeiten, um die Flexibilität des Kontrollflusses zu erhöhen, ist es, die Mächtigkeit von Sequenzkanten zu erweitern: Normalerweise werden Aktivitäten von GP bei der Modellierung des Kontrollflusses als atomare Einheiten betrachtet [RH06]. Bei einer Sequenz muss deshalb die Vorgängeraktivität A abgeschlossen sein, bevor

[1] Hochschule Neu-Ulm, Fakultät Informationsmanagement, Wileystr. 1, 89231 Neu-Ulm, thomas.bauer@hnu.de

doi:10.18420/modellierung2022-003

die Nachfolgeraktivität B gestartet werden kann (vgl. Abb. 1a). Detaillierter betrachtet besteht eine Aktivität jedoch aus einem Start- und einem Endeereignis. Auf diesem Detaillierungsgrad erfolgt bei Prozess-Management-Systemen (PMS) üblicherweise die Protokollierung (Log-File) und solche Ereignisse werden zum Process-Mining verwendet [Da18, ZSA21]. Allerdings können diese Ereignisse normalerweise nicht beliebig zur Modellierung von Kontrollflusskanten verwendet werden [RH06]. Stattdessen müssen Sequenzkanten immer an einem Endeereignis starten und ein Startereignis als Ziel haben (vgl. Abb. 1a). Dies soll so erweitert werden, dass sie an beliebigen Ereignissen beginnen und enden können (Abb. 1b). Im dargestellten Beispiel ermöglicht dies zusätzlich die Ausführungsreihenfolge ii) aus Abb. 1c. Außer den Reihenfolgebeziehungen können bei dem in diesem Beitrag vorgestellten Ansatz auch noch zeitliche Abhängigkeiten beliebig zwischen den Start- und Endeereignissen definiert werden.

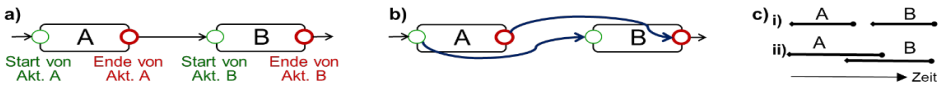


Abb. 1: a) Klassische Sequenzkante b) Sequenzkanten mit erweiterter Semantik c) dadurch erlaubte Ausführungsreihenfolgen

Der in Abb. 2a dargestellte GP zur Entwicklung von Elektronikkomponenten (z.B. eines Fahrzeugs) zeigt, wie die zusätzlichen Kantentypen verwendet werden können. Von der Akt. A zur Akt. B verläuft die klassische Sequenzkante ①. In dem GP findet nach dem Entwurf der Gesamtarchitektur (Akt. B) die Entwicklung der Steuergeräte (Akt. C) statt. Um Entwicklungszeit zu sparen (Concurrent Engineering), sollen diese Aktivitäten jedoch zeitlich überlappend ausgeführt werden. Deshalb wurde für die Kante ② der Typ „StartBeforeStart“ verwendet. Das bedeutet, dass die Bearbeitung der Akt. B vor dem Start von Akt. C beginnen muss, d.h. die Akt. C darf bereits ab dem Start von B gestartet werden, sie muss also nicht auf die Beendigung von Akt. B warten. Dasselbe gilt für die Akt. D in Bezug auf Akt. C (Kante ③). Durch die Kante ④ wird jedoch zusätzlich festgelegt, dass der Test der entwickelten Steuergeräte erst dann abgeschlossen werden darf, wenn zuvor deren Entwicklung (Akt. C) beendet wurde. Schließlich legt die Kante ⑤ eine Maximalzeit für die Ausführung des gesamten GP fest, weil zwischen dem Start der Akt. A und dem Ende der Akt. D maximal 120 Tage vergehen dürfen.

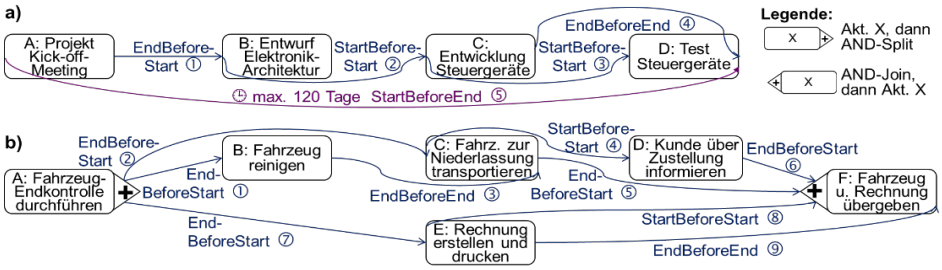


Abb. 2: Beispielprozesse mit erweiterten Sequenz- und Zeitkanten

Die zusätzlichen Kantentypen können auch in Kombination mit komplexeren Ablaufstrukturen verwendet werden. Abb. 2b zeigt einen GP zur Fahrzeugauslieferung, der stark verdichtet ist, um insb. den Zweck solcher Kanten zu erläutern. Die Fahrzeug-Herstellung wird mit der Endkontrolle (Akt. A) abgeschlossen. Danach findet parallel die Fahrzeug-Auslieferung und die Rechnungserstellung statt. Ein auszulieferndes Fahrzeug wird in Akt. B vom Fahrer endgereinigt (z.B. im Fahrzeug verbliebener Müll entfernen). Dies muss abgeschlossen sein, bevor der Transport (Akt. C) abgeschlossen wird, weil es nicht möglich ist, das Fahrzeug danach noch zu reinigen. Es ist jedoch erlaubt, dass die Aktivitäten B und C überlappend ausgeführt werden, z.B. indem das Fahrzeug während einer Transportpause gereinigt wird. Deshalb wird für die Kante ③ der Typ EndBeforeEnd verwendet. Außerdem muss der Fahrzeugtransport (Akt. C) beginnen, bevor die Akt. D (Kunde über anstehende Zustellung informieren) beginnt (StartBeforeStart bei Kante ④). Im Falle eines früheren Informierens wäre die Gefahr einer Fehlinformation zu groß, weil vor Beginn des Fahrzeugtransports die Wahrscheinlichkeit noch hoch ist, dass der Transport doch nicht stattfindet (z.B. weil der LKW nicht verfügbar oder defekt ist). ⑧ ist ein Beispiel für einen neuartigen Kantentyp, der zudem in einen AND-Join (Ende der Parallelität) mündet: Bevor die Akt. F (Übergabe an Kunde) gestartet werden darf, müssen die Aktivitäten C und D beendet sein. Außerdem muss die Akt. E zumindest gestartet worden sein, d.h. ein Mitarbeiter wurde mit der Rechnungserstellung beauftragt. Damit der Kunde nicht unnötig warten muss, darf mit Akt. F aber bereits begonnen werden (z.B. Erläuterung der Fahrzeugfunktionen). Abgeschlossen werden kann diese Akt. F aber erst nach der Übergabe der Rechnung. Da diese davor fertig erstellt und gedruckt sein muss, erfordert die Beendigung der Akt. F die Beendigung von Akt. E (EndBeforeEnd bei Kante ⑨).

Ähnliche Anforderungen sind auch aus dem Projektmanagement bekannt [Bu18]. So können sich auch dort Reihenfolgeabhängigkeiten auf beliebige Start- und Endeereignisse von Aufgaben beziehen. Dadurch ergeben sich vier Kombinationsmöglichkeiten: Normalfolge, Anfangsfolge, Endfolge, Sprungfolge [Bu18]. Diese entsprechen den vier Typen von Sequenzkanten in CoPMoF (vgl. Abschnitt 2). Außerdem können auch beim Projektmanagement minimale und maximale Zeitabstände zwischen den Start- und Endeereignissen definiert werden. Diese haben dort den Zweck, den kritischen Pfad eines Projektes, also die benötige Ausführungsdauer, zu ermitteln. Dahingegen haben die zusätzlichen Kantentypen (erweiterte Sequenzkanten) bei CoPMoF den Zweck, die Ausführungsdauer von GP zu reduzieren, indem mehr Parallelität zwischen den Aktivitäten ermöglicht wird. Hierbei sollen natürlich auch die zeitlichen Constraints eingehalten werden. Ähnliche zeitliche Reihenfolgen, wie sie durch die bei CoPMoF neu eingeführten Kantentypen ermöglicht werden, sind zudem in Allens Intervall Algebra [Al83] beschrieben (z.B. overlaps, during).

In [Ba19a] wurden Beispiele für GP vorgestellt, in denen erweiterte Sequenzkanten nützlich sind. Außerdem wurden die daraus resultierenden Anforderungen dargestellt. Allerdings wurde bisher nicht betrachtet, wie es einem PMS möglich ist, die Ausführung entsprechender GP zu steuern (zur Runtime). Diese Forschungslücke wird durch den vorliegenden Beitrag geschlossen: Er beschreibt, wie ein PMS den Start bzw. das Ende von Aktivitäten verzögern

bzw. beschleunigen kann. Außerdem wird eine Ausführungssemantik für GP mit erweiterten Sequenzkanten und Zeitbedingungen entwickelt. Diese wird von PMS benötigt, um die ausführbaren Aktivitäten und die zugehörigen Zeitpunkte zu ermitteln.

In Abschnitt 2 werden relevante Vorarbeiten aus dem Projekt CoPMoF vorgestellt und der allgemeine Stand der Literatur betrachtet. Abschnitt 3 beschreibt, wie sich das PMS zu Runtime verhält, insb. wird eine formale Ausführungssemantik definiert. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick.

2 Grundlagen und Literatur

Vorarbeiten: Die Grundidee erweiterter Sequenzkanten ist, dass sich diese beliebig auf die Start- und Endeereignisse ihrer Quell- und Zielaktivität beziehen können. Diese Idee wurde bereits in [Ba19a] publiziert. Dort wurden außerdem Beispielszenarien vorgestellt, in denen erweiterte Sequenzkanten benötigt werden. Aus diesen wurden (ähnlich wie das mit den in Abb. 2 dargestellten GP möglich wäre) die Anforderungen abgeleitet, dass vier Typen von Sequenzkanten, zeitliche Constraints mit minimalen und maximalen Zeitdauern und die Kombination daraus erforderlich sind.

Allerdings wurde in [Ba19a] die Bedeutung der vier Kantentypen lediglich informell festgelegt. So wie in Abb. 3 dargestellt, wurden für jeden Typ die erlaubten Ausführungsreihenfolgen aufgezählt und die Funktionsweise wurde erläutert:

1. EndBeforeStart	Ende von Akt. A muss vor dem Start von Akt. B erfolgen (das entspricht einer klassischen Sequenzkante, vgl. [RH06])
2. EndBeforeEnd	Ende von Akt. A muss vor dem Ende von Akt. B erfolgen
3. StartBeforeStart	Start von Akt. A muss vor dem Start von Akt. B erfolgen
4. StartBeforeEnd	Start von Akt. A muss vor dem Ende von Akt. B erfolgen

Ebenso wurde die Bedeutung der zeitlichen Constraints nur textuell beschrieben, da die Bedeutung eines minimalen bzw. maximalen Zeitabstandes intuitiv klar ist.

Die Prozess-Engine eines PMS benötigt einen Algorithmus zur Steuerung der auszuführenden GP-Instanzen. Eine nur für Menschen verständliche Erläuterung genügt hierfür nicht. Deshalb werden in dem vorliegenden Beitrag maschinell auswertbare Regeln entwickelt (d.h. formale Regeln), die eine Ausführungssemantik für Aktivitäteninstanzen definieren. Diese berücksichtigen sowohl alle vier Typen von Sequenzkanten als auch zeitliche Minimal- und Maximalabstände zwischen beliebigen Start- und Endeereignissen der Aktivitäten.

Stand von Forschung und Technik: Kommerzielle PMS basieren häufig auf standardisierten GP-Modellierungssprachen wie BPEL [OA07] und BPMN [Ob11]. Diese Standards definieren Sequenzkanten, die jedoch nur eine rein sequentielle Bearbeitung der Aktivitäten ermöglichen (Typ 1). Parallelitäten ermöglichen eine zeitlich überlappende Ausführung, jedoch ist dann jede beliebige Reihenfolge für Aktivitäten aus unterschiedlichen parallelen Zweigen erlaubt. Es gibt keine Konstrukte, die eine wie im vorherigen Abschnitt für die Typen 2 bis 4 vorgestellte Bedeutung haben (z.B. StartBeforeStart).

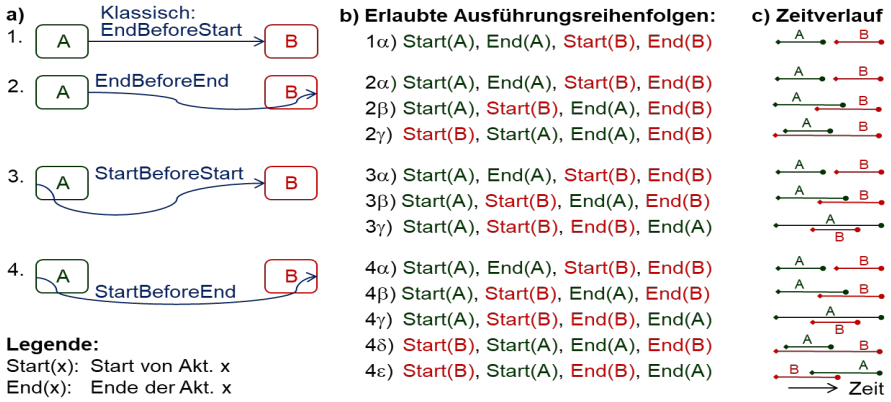


Abb. 3: a) Typen von Sequenzkanten zwischen den Aktivitäten A und B b) erlaubte Reihenfolgen von Start-/Endeereignissen der Aktivitäten c) graphische Darstellung dieser Reihenfolgen, vgl. [Al83]

In BPMN können Maximalzeiten mit einem intermediate Timer-Event [Ob11] realisiert werden. Hierzu müssen zusätzliche Ausführungspfade modelliert werden, die bei einer Zeitüberschreitung ausgeführt werden. Nach dem Timer-Event wird eine speziell für diesen Zweck vorgesehene Eskalationsaktivität modelliert, die beim Eintreten des Events (d.h. bei einer Zeitüberschreitung) automatisch ausgeführt wird. Dadurch können maximale Zeitabstände erzwungen werden. Durch eine solche Vorgehensweise ergeben sich jedoch komplexe Abläufe, mit deren Erstellung „normale GP-Modellierer“ evtl. überfordert sind.

Die in [RH06] vorgestellten Kontrollfluss-Pattern ermöglichen zahlreiche Ausführungsreihenfolgen von Aktivitäten. Die Aktivitäten werden hierbei als atomare Einheiten betrachtet, weswegen sich Sequenzkanten nicht auf beliebige Start- und Endeereignisse der verbundenen Aktivitäten beziehen können. Die Typen 2 bis 4 werden also nicht berücksichtigt.

Bei Ansätzen zum Case Handling [AWG05] definiert die Zustand der Eingabedaten einer Aktivität, ob diese gestartet werden kann. Dadurch kann ein Bearbeiter selbst entscheiden, eine Aktivität zu starten, sobald die erforderlichen Daten vorhanden sind. [HW16] erweitert solche Ansätze um die Möglichkeit, einen Lifecycle für Daten zu modellieren, hierbei eigene Ausführungszustände einzuführen und mittels dieser zu definieren, welche Aktivitäten startbar sind. Dies wird also nicht durch Kontrollflusskanten definiert. Dadurch kann man Abhängigkeiten vom Typ StartBeforeStart (Typ 3) realisieren: Für die Akt. F aus Abb. 2b kann z.B. modelliert werden, dass sie startbar ist, sobald die Rechnung den (benutzerdefinierten) Zustand „angelegt“ erreicht hat. Wenn eine Rechnung unmittelbar nach dem Start der Akt. E diesen Zustand erreicht, wird ein Verhalten wie bei der StartBeforeStart-Kante ⑧ erreicht. Für rein „physische“ Aktivitäten ist dies nicht so direkt umsetzbar: So erzeugt die Akt. C aus Abb. 2b keine Daten, sondern das PMS registriert nur, dass diese Transportaktivität gestartet wurde. Die Kante ④ ist also nicht so einfach realisierbar, weil die Startbarkeit der Akt. D nicht direkt von Daten abhängig ist.

Bei Constraint-basierten Ansätzen (ein Überblick findet sich in [RW12]) wird der Kontrollfluss mittels Regeln festgelegt, welche die Menge der erlaubten Ausführungsreihenfolgen einschränken. Auch diese beziehen sich auf Aktivitäten als Ganzes, weshalb hiermit ebenfalls keine Sequenzen der Typen 2 bis 4 modelliert werden können.

[He00, He01] ermöglicht die Definition beliebiger Abhängigkeiten zwischen dem Start und dem Ende von Aktivitäten (auch die Typen 2 bis 4). Allerdings werden bei diesem Ansatz keine Abhängigkeiten zwischen Aktivitäten derselben Prozessinstanz festgelegt, sondern zwischen Aktivitäten unterschiedlicher Prozessinstanzen und sogar unterschiedlicher Vorlagen. Da sich solche Abhängigkeiten nicht auf einen einzelnen Prozessgraphen beziehen, können sie dort auch nicht als graphische Kontrollflusskonstrukte modelliert werden. Stattdessen werden sie mittels speziell hierfür erstellter regulärer Ausdrücke definiert.

In [LWR10] werden Entwurfsmuster (Pattern) vorgestellt, welche die Festlegung von minimalen und maximalen Zeitabständen zwischen Aktivitäten erlauben. Da hierbei die Start- und Endezeitpunkte der Vorgänger- und der Nachfolgeraktivitäten beliebig verwendet werden können, sind alle vier vorgestellten Kantentypen abgedeckt. Allerdings werden die zeitlichen Constraints nicht im Zusammenhang mit Kontrollfluss-Kanten betrachtet und es wird keine Ausführungssemantik für die vorgestellten Entwurfsmuster definiert.

Zusammenfassend lässt sich also feststellen, dass es zwar Arbeiten zu Zeitabständen zwischen Aktivitäten gibt, die vorgestellten Kontrollfluss-Kantentypen 2 bis 4 bisher jedoch nicht betrachtet wurden. Allerdings existieren Arbeiten aus anderen Bereichen (Abhängigkeiten zwischen GP [He00, He01], Projektmanagement [Bu18]), die eine beliebige Verwendung der Start- und Endeereignisse von Aktivitäten bei der Modellierung erlauben.

3 Verhalten eines PMS zur Ausführungszeit (Runtime)

In diesem Abschnitt wird erläutert, wie ein PMS Prozessinstanzen steuert, die erweiterte Sequenzkanten enthalten. Zuerst wird beschrieben, wie der Start bzw. die Beendigung von Aktivitäten generell verzögert oder beschleunigt werden kann. Nach der Definition einiger Grundlagen wird dann für jeden Kantentyp, die zugehörige Ausführungssemantik festgelegt.

3.1 Beeinflussung des Start- und Endezeitpunkts von Aktivitäten

Wenn eine Aktivität noch nicht gestartet werden soll (z.B. aufgrund eines zeitlichen Mindestabstands im Fall A2 in Abb. 4), dann wird noch kein entsprechender Eintrag in die Arbeitslisten der Benutzer eingestellt (wie immer bei PMS). Ist der Start früher möglich (z.B. aufgrund eines neuen Typs von Sequenzkante im Fall B1) so wird ein solcher Eintrag bereits erstellt. Muss der Start früher erfolgen (Fall B2), so „erzwingt“ dies das PMS durch Eskalationen [ARD07, Ba21a]. Bei dieser bereits heute von vielen PMS angebotenen Vorgehensweise werden Nachrichten mit einer Aufforderung zur Aktivitätenbearbeitung

(automatisch und rechtzeitig) an die potentiellen Bearbeiter, deren Vorgesetzte oder Prozessadministratoren versendet. Ebenso kann die rechtzeitige Beendigung von Aktivitäten (D2) durch Eskalationen „erzwungen“ werden. Darf eine Aktivität noch nicht beendet werden (C), so kann z.B. der Ende-Button des entsprechenden Aktivitätenprogramms deaktiviert („ausgegraut“) sein. Da dies für Benutzer irritierend sein kann, sollte zusätzlich eine gut verständliche Information angezeigt werden, warum die Beendigung dieser Aktivität noch nicht möglich ist.

Zur Realisierung einiger der in Abb. 4 dargestellten Fälle muss die Ausführungssemantik der Prozess-Engine um zusätzliche Regeln erweitert werden (z.B. für Fall B1). Diese Regeln werden im Abschnitt 3.2 erläutert. Zur Verbesserung der Übersichtlichkeit sind die Nummern solcher neuer Regeln in Abb. 4 ebenfalls angegeben. Ist für einen Fall ein zusätzlicher Ausführungszustand für Aktivitäteninstanzen erforderlich, so ist auch dieser aus Abb. 4 ersichtlich (bei A2 und C1/2).

wegen	**Start von Akt. B**		**Ende von Akt. B**	
	A: später	**B:** früher	**C:** später	**D:** früher
1: Sequenzkante	Fall existiert nicht	StartBeforeStart A B Regel 3, 8	StartBeforeEnd A EndBeforeEnd B Regel 5, 6, neuer Zustand Running-Completable	Fall existiert nicht
2: Zeit-Constraint	min. x B Regel 1, 2, 8 neuer Zustand WaitingForTime	max. x B Regel 4, durch Eskalation erzwungen	min. x B Regel 5', neuer Zustand Running-Completable	max. x B Regel 7, durch Eskalation erzwungen

Abb. 4: Übersicht über die verschiedenen Fälle

3.2 Definitionen und Ausführungsregeln

Im Folgenden werden die üblicherweise verwendeten (z.B. [RD98, We19, Ob11]) Definitionen für Geschäftsprozesse und die Menge der Ausführungszustände von Aktivitäteninstanzen so erweitert, dass sie als Grundlage für die Festlegung von Ausführungsregeln für erweiterte Sequenzkanten geeignet sind.

Def. 1: Prozesstemplate, Prozessinstanz und Aktivitätenzustände
Ein Prozesstemplate PT=(N, C, T) besteht aus einer Menge von Knoten N (engl.: Node), einer Menge von Kontrollflusskanten C (Control Flow) und einer Menge von Zeitkanten T (Time).
Eine Prozessinstanz PI=(PT, State) besteht aus einem Prozesstemplate PT und einer Menge von Aktivitätenzuständen (State). Jede Aktivität a $\in$ N kann einen unterschiedlichen Zustand State(a) haben. Aktivitäten können die in Abb. 6 dargestellten Zustände einnehmen, d.h.
$\forall$ a $\in$ N gilt: State(a) $\in$ {Inactive, WaitingForTime, Active, Running, RunningCompletable, Completed}

Das PMS kennt den aktuellen Zustand jeder Aktivität und verändert diese Zustände während der Ausführung der Prozessinstanz mittels vorgegebener Ausführungsregeln. Dadurch kann das PMS die Ausführungsreihenfolge der Aktivitäten steuern, Einträge in Arbeitslisten einfügen, Aktivitätenprogramme (z.B. Formulare) starten, automatisch ausgeführte Services aufrufen, etc.

Def. 2: Eine Kontrollflusskante c ∈ C ist definiert als
c = (SourceAct, TargetAct, Type) mit:
SourceAct: die Quellaktivität der Kante
TargetAct: die Zielaktivität der Kante
Type: der Typ der Kante mit
Type ∈ {StartBeforeStart, StartBeforeEnd, EndBeforeStart, EndBeforeEnd}

Wie in Def. 1 erkennbar ist, stellen Kontrollfluss- und Zeitkanten unterschiedliche Kantentypen dar. Der Grund für diese Trennung ist, dass die Existenz eine Zeitkante zwischen zwei Aktivitäten nicht immer auch eine Sequenz-Abhängigkeit impliziert: Wie in Abb. 5 dargestellt, kann gefordert sein, dass der Start einer Akt. B spätestens (d.h. max.) 10 Stunden nach dem Ende der Akt. A erfolgen muss (d.h. klassischer Typ EndBeforeStart). Um den Ablauf zusätzlich zu beschleunigen, kann aber erlaubt sein, dass der Start der Akt. B bereits vor der Beendigung der Akt. A erfolgen darf, d.h. es existiert zwischen den beiden Aktivitäten keine Kontrollflusskante dieses Typs EndBeforeStart, sondern des Typs StartBeforeStart (vgl. Abb. 5).

Abb. 5: Aktivitäten mit Zeitkante, aber ohne „klassische“ Kontrollflusskante

Def. 3: Eine Zeitkante t ∈ T ist definiert als
t = (SourceAct, TargetAct, Type, MinTime, MaxTime) mit:
SourceAct, TargetAct und Type: wie in Def. 2 festgelegt
MinTime: die für die Kante festgelegte minimale Zeitdauer
MaxTime: die für die Kante festgelegte maximale Zeitdauer,
wobei bei Kanten, für die kein entsprechendes Zeit-Constraint definiert wurde,
MinTime bzw. MaxTime den Wert undef haben

Um diese Informationen in den Ausführungsregeln nutzen zu können, kann mit gleichnamigen Funktionen auf sie zugegriffen werden. So liefert z.B. SourceAct(c) die Quellaktivität der Kante c und State(a) den aktuellen Ausführungszustand der Akt. a.

Die Menge der Zustände einer Aktivitäteninstanz wurde erweitert, um die neuen Typen von Sequenzkanten realisieren zu können. Die beiden zusätzlich erforderlichen Zustände sind in Abb. 6 farbig hinterlegt. Außerdem sind die Nummern der für erweiterte Sequenzkanten zusätzlich erforderlichen Regeln den entsprechenden Kanten zugeordnet. Die aus klassischen PMS bereits bekannten Regeln (vgl. [RD98, We19, Ob11]) werden im Folgenden nicht wiederholt, weswegen in Abb. 6 auch Kanten ohne Beschriftung existieren. Auf das Verhalten

bei Verzweigungen, Parallelitäten und Schleifen wird nur kurz eingegangen. Stattdessen stehen erweiterte Sequenzkanten im Fokus. Außerdem werden keine Zustände betrachtet, die hierfür nicht direkt relevant sind. Solche Zustände sind z.B. für einen Abbruch einer Aktivität, deren Kompensation, erfolglos beendete Aktivitäten (Zustand Failed in [Ob11]) oder bei Sprüngen zu anderen Aktivitäten einer Prozessinstanz [Ba18] erforderlich.

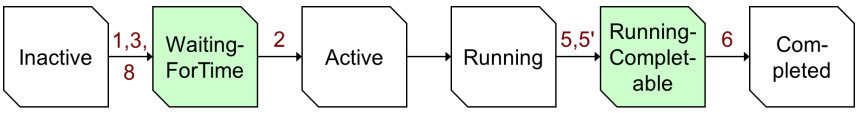


Abb. 6: Zustandsübergangsdiagramm für Aktivitäteninstanzen

Späterer Start einer Aktivität - Sequenzkante: Es kann nicht vorkommen, dass eine Akt. B durch einen der neuen Sequenzkanten-Typen später startbar ist, als bei klassischen Sequenzkanten: Der einzig neue Typ, der sich auf den Start einer Zielakt. B bezieht, ist StartBeforeStart. Existiert eine Kante dieses Typs z.B. von Akt. A zu Akt. B (Fall B1 in Abb. 4), so ist die Akt. B jedoch früher startbar als bei einer klassischen Kante (Typ EndBeforeStart) – und nicht später.

Zeitkante: Eine Akt. B kann später startbar sein, weil sie die Zielaktivität einer Zeitkante mit vorgegebener Minimalzeit ist (A2 in Abb. 4). Da die Verzögerung den Start der Aktivität betrifft, kann die Kante den Typ EndBeforeStart oder StartBeforeStart haben.

Damit das PMS solche Zeitkanten zur Runtime verarbeiten kann, ist der zusätzliche Aktivitätenzustand WaitingForTime erforderlich (vgl. Abb. 6): Dass die Vorgängeraktivität von Akt. B bereits abgeschlossen ist, „merkt“ sich die Prozess-Engine, indem Akt. B den Startzustand Inactive verlässt. Würden ausschließlich Kontrollflusskanten berücksichtigt werden, so wäre die Akt. B bereits startbar, d.h. bei den „klassischen“ Ausführungsregel würde sie in den Zustand Active wechseln. Dies ist jedoch aufgrund des Zeit-Constraints noch nicht erlaubt, weswegen dieser Fall durch den neuen Zustand WaitingForTime unterschieden wird. Die „klassische“ Ausführungsregel wird so verändert, dass auf Inactive nicht direkt in der Zustand Active folgt. Stattdessen erfolgt zuerst ein Wechsel in den Zustand WaitingForTime:

Regel 1: Die Kante $c \in C$ sei eine in die Aktivität a mündende „normale“ Kontrollflusskante, d.h. TargetAct(c) = a $\wedge$ Type(c) = EndBeforeStart. Nach Beendigung der Quellaktivität q dieser Kante, wechselt die Akt. a in den Zustand WaitingForTime:
Wird bei der Akt. q = SourceAct(c) der State(q) = Completed erreicht, dann ändert sich der Zustand von Akt. a: State(a) = WaitingForTime

Ein Wechsel vom Zustand WaitingForTime in den Zustand Active erfolgt mittels der Ausführungsregel 2, sobald alle minimalen Wartezeiten der eingehenden Zeitkanten erreicht sind, so dass die Akt. a nun tatsächlich bearbeitet werden darf.

Regel 2: T_a sei die Menge der für die Berechnung des frühesten Startzeitpunkts relevanten Zeitkanten einer Akt. a mit State(a) = WaitingForTime:

$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeStart, EndBeforeStart\} \wedge MinTime(t_i) \neq undef\}$

Dann ergibt sich die früheste Startzeit $EarliestStartingTime_a$ der Akt. a als der größte (d.h. späteste) Zeitpunkt, der aus einer dieser Kanten resultiert:

$EarliestStartingTime_a = Max(ResultingTime_{Min,Start}(t_i)) \; \forall \; t_i \in T_a$

Hierfür wird der aus der Kante t resultierende Zeitpunkt $ResultingTime_{Min,Start}(t)$ berechnet, indem (abhängig vom Kantentyp) zur Start- bzw. Endezeit der Quellaktivität der Kante t die für die Kante t festgelegte Minimalzeit addiert wird:[2]

$ResultingTime_{Min,Start}(t_i) =$

$$\begin{cases} StartTime(SourceAct(t_i)) + MinTime(t_i), \text{ falls } Type(t_i) = StartBeforeStart \\ EndTime(SourceAct(t_i)) + MinTime(t_i), \text{ falls } Type(t_i) = EndBeforeStart \end{cases}$$

Sobald die Zeit $EarliestStartingTime_a$ erreicht ist (d.h. $CurrentTime \geq EarliestStartingTime_a$) wird der Zustand von Akt. a in Active geändert: State(a) = Active.

Bemerkung: Falls es keine solchen Zeitkanten gibt (d.h. $T_a = \{\}$), ergibt die Berechnung des Maximums (Max) eine $EarliestStartingTime_a = -\infty$. Da stets $CurrentTime \geq -\infty$ gilt, wird direkt in State(a) = Active gewechselt, sobald State(a) = WaitingForTime erreicht ist. Der Zustand WaitingForTime ist dann also nicht relevant und wird sofort wieder verlassen.

Die nachfolgend erläuterten Varianten von ResultingTime werden später in den Ausführungsregeln 4, 5′ und 7 benötigt: $ResultingTime_{Max,Start}(t_i)$ wird wie in Regel 2 berechnet, wobei jedoch $\underline{Max}Time(t_i)$ (anstatt $MinTime(t_i)$) addiert wird. Außerdem werden $ResultingTime_{Min,End}(t_i)$ und $ResultingTime_{Max,End}(t_i)$ auf dieselbe Weise berechnet, wobei jedoch die Menge T_a Kanten t_i mit $Type(t_i) \in \{StartBefore\underline{End}, EndBefore\underline{End}\}$ enthält.

Früherer Start einer Aktivität - Sequenzkante: Außer durch klassische Sequenzkanten (d.h. Typ EndBeforeStart) kann eine Akt. B auch durch eine Kante vom Typ StartBeforeStart ausführbar werden (Fall B1 in Abb. 4). Auch dann muss ihr Zustand von Inactive zu WaitingForTime wechseln (noch nicht Active, wie im vorherigen Abschnitt erläutert). Dies wird durch die folgende Ausführungsregel realisiert:

Regel 3: Die Akt. a mit State(a) = Inactive sei das Ziel der Kontrollflusskante $c \in C$ vom Typ StartBeforeStart, d.h. $TargetAct(c) = a \wedge Type(c) = StartBeforeStart$.
Nach Start der Quellakt. q dieser Kante, wechselt die Akt. a in den Zustand WaitingForTime: Wird bei der Akt. q = SourceAct(c) der State(q) = Running gesetzt, dann ändert sich der Zustand von Akt. a: State(a) = WaitingForTime

Zeitkante: Hat eine Akt. B eine eingehende Zeitkante mit einer Maximalzeit (B2 in Abb. 4), so muss ihr Start bis zu einem gewissen Zeitpunkt ($LatestStartingTime_a$, s.u.) durch Eskalationen erzwungen werden. Dabei sind nur Zeitkanten relevant, die sich auf den Start der Akt. B beziehen (also Typen ..BeforeStart). Der Zeitpunkt $LatestStartingTime_a$ berechnet sich wie folgt:

[2] StartTime(a) einer Akt. a ist der Zeitpunkt, an dem diese Aktivität gestartet wurde. EndTime(a) ist der Zeitpunkt ihrer Beendigung. CurrentTime ist der aktuelle Zeitpunkt.

Regel 4: T_a sei die Menge der für die Berechnung des spätesten Startzeitpunkts von Akt. a relevanten Zeitkanten:
$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeStart, EndBeforeStart\} \wedge MaxTime(t_i) \neq undef\}$
Dann ergibt sich die späteste Startzeit $LatestStartingTime_a$ der Akt. a als der kleinste (d.h. früheste) Startzeitpunkt, der aus einer dieser Kanten t_i als späteste (d.h. Max) Startzeit resultiert:
$LatestStartingTime_a = Min(ResultingTime_{Max,Start}(t_i)) \; \forall \; t_i \in T_a$

Wie bereits erwähnt, soll dieser späteste Startzeitpunkt z.B. durch Eskalationen sichergestellt werden. Da nach der Auslösung einer Eskalation eine gewisse Zeit vergeht, bis diese vom Benutzer erkannt und die Aktivität auch tatsächlich gestartet wird, sollte die Eskalation rechtzeitig vor Erreichen der $LatestStartingTime_a$ ausgelöst werden. In gewissen Anwendungsfällen können auch mehrstufige Eskalationen [ARD07] sinnvoll sein: Beispielsweise kann zuerst eine eMail an die potentiellen Bearbeiter der Aktivität gesendet werden. Wurde die Aktivität nach einer gewissen Zeit immer noch nicht gestartet, so wird ein Prozessverantwortlicher (Administrator, Vorgesetzter) informiert. Auch dies muss noch rechtzeitig vor Erreichen von $LatestStartingTime_a$ erfolgen.

Späteres Ende einer Aktivität - Sequenzkante: Durch die für den Fall C1 in Abb. 4 dargestellten Typen (..BeforeEnd) von Sequenzkanten, kann die Beendigung der Akt. B verzögert werden. Das bedeutet, dass der Benutzer die Akt. B evtl. bereits bearbeitet hat, diese aufgrund des vorgegebenen Prozesstemplates aber noch nicht beenden darf. Damit die Prozess-Engine diesen Fall erkennen kann, wird der neue Zustand RunningCompletable eingeführt. Die Akt. B befindet sich während ihrer Bearbeitung zuerst im Zustand Running. Erst wenn die mit einer solchen Kante verbundene Vorgängerakt. A gestartet bzw. beendet wurde, darf Akt. B abgeschlossen werden, weshalb sich ihr Zustand zu RunningCompletable ändert. Die nun folgende Regel 5 realisiert diesen Zustandsübergang.

Regel 5: C_a sei die Menge der für die Beendigung der Akt. a mit State(a) = Running relevanten Kontrollflusskanten:
$C_a = \{c_i \in C \mid TargetAct(c_i) = a \wedge Type(c_i) \in \{StartBeforeEnd, EndBeforeEnd\}\}$
Ein Zustandswechsel der Akt. a ist erlaubt, sobald alle Quellaktivitäten dieser Kanten gestartet (i) bzw. beendet (ii) wurden, d.h. alle folgenden Bedingungen erfüllt sind:
i) Falls $\forall c_i \in C_a$ mit $Type(c_i) = StartBeforeEnd$ gilt:
$State(SourceAct(c_i)) \in \{Running, RunningCompletable, Completed\}$ und
ii) falls $\forall c_i \in C_a$ mit $Type(c_i) = EndBeforeEnd$ gilt:
$State(SourceAct(c_i)) = Completed$,
dann wechselt der Zustand von Akt. a in: State(a) = RunningCompletable

Bemerkung: Wenn es keine Kontrollflusskanten dieses Typs gibt, deren Zielaktivität die Akt. a ist (d.h. $C_a = \{\}$), dann sind die Bedingungen i) und ii) automatisch erfüllt und der Zustand geht unmittelbar nach Erreichen von Running in RunningCompletable über. Der

Grund dafür ist, dass die Bedingung „für alle“ ($\forall$) c_i erfüllt ist, wenn es gar keine solchen c_i gibt.

Die nachfolgende Regel 6 ersetzt die klassische bei der Beendigung einer Aktivität ausgeführte Regel. Die wesentliche Änderung ist, dass Regel 6 den (neuen) Zustand RunningCompletable verwendet, anstatt den Zustand Running.

Regel 6: Befindet sich eine Akt. a im Zustand State(a) = RunningCompletable, dann darf der Benutzer diese Aktivität abschließen. Dadurch ändert sich ihr Zustand in:
State(a) = Completed

Wenn eine Aktivität noch nicht beendet werden darf (d.h. noch im Zustand Running anstatt RunningCompletable ist), dann kann dies dem Benutzer signalisiert werden, indem z.B. der Button „Aktivitätenprogramm beenden“ oder ein entsprechender Eintrag im Menü deaktiviert ist. Zusätzlich sollte ihm (zuvor im Prozesstemplate modellierte) Information angezeigt werden, warum eine Beendigung noch nicht möglich ist. Damit ein Aktivitätenprogramm entsprechend gestaltet werden kann, muss die Schnittstelle (API) des PMS Funktionen anbieten, um solche Hilfetexte abzufragen und um zu ermitteln, ob eine bestimmte Aktivität bereits beendet werden darf. Die Funktion zum Beenden der Akt. a soll einen Fehler zurückliefern, wenn dies bereits im Zustand State(a) = Running versucht wird.

Zeitkante: Außer durch die eben betrachteten Kontrollflusskanten kann auch durch Zeitkanten mit vorgegebener Minimalzeit die Beendigung der Akt. B verzögert werden (vgl. C2 in Abb. 4). Da sie sich auf das Ende der Akt. B beziehen sollen, muss es sich um Zeitkanten der Typen ..BeforeEnd handeln. Um diese Kanten zu berücksichtigen, wird die oben dargestellte Regel 5 um eine weitere Bedingung (iii) erweitert, die ebenfalls erfüllt sein muss:

Regel 5′: T_a sei die Menge der für die Berechnung des frühesten Endezeitpunkts relevanten Zeitkanten der Akt. a (mit State(a) = Running):
$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeEnd, EndBeforeEnd\} \wedge$
$MinTime(t_i) \neq undef \}$
Der früheste erlaubte Zeitpunkt für die Beendigung der Akt. a ist dann:
$EarliestCompletionTime_a = Max(ResultingTime_{Min,End}(t_i)) \; \forall \; t_i \in T_a$
iii) Ein Zustandswechsel der Akt. a ist erlaubt, sobald gilt:
$CurrentTime \geq EarliestCompletionTime_a$

Für einen Zustandswechsel müssen also alle 3 Bedingungen erfüllt sein, d.h. i) und ii) aus Regel 5 und iii) aus Regel 5′. Wenn allerdings keine entsprechenden Zeitkanten existieren (d.h. $T_a = \{\}$), dann ergibt die Berechnung des Maximums wieder den Wert $-\infty$ und die $\geq$-Bedingung für CurrentTime ist stets erfüllt, und damit auch Bedingung iii).

Es ist auch erlaubt, dass eine Zeitkante $t_i \in T_a$ vom Typ StartBeforeEnd die Akt. a sowohl als Quell- wie auch als Zielaktivität verwendet. Dies stellt den durchaus relevanten Sonderfall dar, dass für eine Aktivität eine Mindest-Bearbeitungsdauer festgelegt wird,

z.B. die Zeitdauer zum Trocknen einer Verklebung. Dieser Sonderfall ist in Regel 5′ auch eingeschlossen.

Früheres Ende einer Aktivität - Sequenzkante: Es kann nicht vorkommen, dass durch die neuen Typen von Kontrollflusskanten eine Akt. a früher beendet werden kann, als in klassischen Prozessmodellen. Bei letzteren kann die Akt. a unmittelbar nach ihrem Start beendet werden. Eine frühere Beendigung (d.h. vor dem Starten) macht keinen Sinn und kann deshalb auch nicht aus den neuen Kontrollflusskanten-Typen resultieren.

Zeitkante: Durch Zeitkanten mit einer Maximal-Zeit kann gefordert sein, dass eine Akt. B bis zu einem bestimmten Zeitpunkt beendet wird (vgl. D2 in Abb. 4). Dabei muss es sich um Kanten handeln, die sich auf die Beendigung der Akt. B beziehen (d.h. die Typen ..BeforeEnd) und der Beendigungszeitpunkt wird vom PMS wieder durch Eskalationen sichergestellt. Die Regel 7 dient zur Berechnung dieses Zeitpunkts.

Regel 7: T_a sei die Menge der für die Berechnung des spätesten Endezeitpunkts von Akt. a relevanten Zeitkanten:
$T_a = \{t_i \in T \mid TargetAct(t_i) = a \wedge Type(t_i) \in \{StartBeforeEnd, EndBeforeEnd\} \wedge MaxTime(t_i) \neq undef\}$
Dann ergibt sich der späteste Endezeitpunkt $LatestCompletionTime_a$ der Akt. a als der kleinste (d.h. früheste) Maximal-Endezeitpunkt, der aus einer dieser Kanten t_i resultiert:
$LatestCompletionTime_a = Min(ResultingTime_{Max,End}(t_i)) \; \forall \; t_i \in T_a$

Gateways: Im Folgenden wird speziell auf das Verhalten bei Gateways eingegangen, da dies durch die bisher vorgestellten Regeln nicht vollständig abgedeckt ist. Für nach einem Split liegende Aktivitäten (z.B. die Akt. B, C und E in Abb. 2b) sind keine Erweiterungen notwendig, da diese nur eine einzige Vorgängeraktivität haben (die Split-Aktivität A in Abb. 2b). Außerdem decken die vorgestellten Regeln bereits den Fall mehrerer eingehender Kanten der Typen StartBeforeEnd und EndBeforeEnd (beide durch Regel 5) sowie mehrere Zeitkanten (Regeln 2, 4, 5′ und 7) ab.

Um die Lesbarkeit zu verbessern, wurde bei den Regeln 1 und 3 bisher der Fall mehrerer eingehender Kanten der Typen EndBeforeStart sowie StartBeforeStart ausgeklammert. Diese treten z.B. bei AND-Joins[3] auf (Akt. F in Abb. 2b)[4]. Bei diesen muss auf alle Vorgängeraktivitäten gewartet werden. Hierzu werden die Regeln 1 und 3 durch die nachfolgend dargestellte Regel 8 ersetzt, bei der die Menge Q_{End} alle mit Kanten des Typs EndBeforeStart und Q_{Start} mit StartBeforeStart verbundene Vorgängeraktivitäten enthält.

[3] Auch OR- und XOR-Join-Knoten haben mehrere eingehende Kanten. Bei diesen sind nur diejenigen Vorgängeraktivitäten des Join-Knotens zu berücksichtigen, die in bei dieser Prozessinstanz tatsächlich ausgeführten Pfaden liegen. Die Mengen Q_{End} und Q_{Start} enthalten also keine Aktivitäten aus nicht ausgeführten Pfaden.

[4] Im Gegensatz zu BPMN verzichten wir Abb. 2b auf separate Gateway-Knoten (ebenso wie z.B. ADEPT [RD98, Re00]), weil Gateways keine nennenswerte Zeit zum „Durchschalten“ benötigen. Deswegen fallen ihr Start- und Endeereignis quasi zusammen, so dass der separate Knoten im betrachteten Kontext ebenso irrelevant wäre, wie die zusätzlich entstehende Kante. (In Abb. 2b ergäbe sich ein Join-Knoten mit den eingehenden Kanten ⑤, ⑥ und ⑧, sowie eine zusätzliche ausgehende Kante zur Akt. F, deren Typ End/StartBeforeStart irrelevant ist, weil die entsprechenden Ereignisse fast gleichzeitig stattfinden.)

Neu ist, dass nicht eine einzelne Vorgängeraktivität q den erforderlichen Zustand erreichen muss, sondern alle Aktivitäten dieser beiden Mengen. So ergeben sich bei Abb. 2b für die Akt. F die Mengen $Q_{End} = \{C, D\}$ und $Q_{Start} = \{E\}$ (letztere wegen der Kante ⑧, die Kante ⑨ spielt für die Startbarkeit der Akt. F keine Rolle).

Regel 8: C_{End} sei die Menge der in Aktivität a mündenden „normalen" Kontrollflusskanten:
$C_{End} = \{c_i \in C \mid TargetAct(c_i) = a \wedge Type(c_i) = EndBeforeStart\}$
C_{Start} sei die Menge der in Aktivität a mündenden Kanten des Typs StartBeforeStart:
$C_{Start} = \{c_i \in C \mid TargetAct(c_i) = a \wedge Type(c_i) = StartBeforeStart\}$
Nach Beendigung aller zugehöriger Quellaktivitäten $q \in Q_{End}$ der Kanten aus C_{End} und nach dem Starten aller Aktivitäten $q \in Q_{Start}$ der Kanten aus C_{Start} wechselt die Akt. a in den Zustand WaitingForTime:
Wenn $\forall\, q \in Q_{End}$ mit $Q_{End} = \{q \in N \mid \exists\, c_i \in C_{End} \wedge q = SourceAct(c_i)\}$ gilt State(q) = Completed und $\forall\, q \in Q_{Start}$ mit $Q_{Start} = \{q \in N \mid \exists\, c_i \in C_{Start} \wedge q = SourceAct(c_i)\}$ gilt State(q) = Running, dann ändert sich der Zustand von Akt. a: State(a) = WaitingForTime

4 Zusammenfassung und Ausblick

Der vorgestellte Ansatz erweitert Sequenzkanten, indem sich diese nun beliebig auf die Start- und Endeereignisse ihrer Quell- und Zielaktivitäten beziehen dürfen. Außerdem wird die Festlegung von minimalen und maximalen Zeitabständen ermöglicht, die sich ebenfalls auf diese Ereignisse beziehen. Dieser Beitrag erläutert, wie ein PMS (z.B. durch Eskalationen) die Benutzer so beeinflussen kann, dass all diese modellierten Bedingungen eingehalten werden. Außerdem wird die formale Ausführungssemantik von Prozess-Engines erweitert, indem zusätzlich erforderliche Zustände für Aktivitäteninstanzen eingeführt und neue Ausführungsregeln definiert werden. Damit wird es einem PMS ermöglicht, solche GP automatisch zu steuern, die die eben erwähnten Konstrukte enthalten.

Die vorgestellten Regeln müssen mittels einer prototypischen Implementierung noch technisch evaluiert werden. Ideal wäre hierfür deren Integration in ein existierendes und bereits praktisch nutzbares PMS, weil dies zusätzlich eine (fachliche) Evaluation ihrer Eignung für GP-Modellierer und Endanwender ermöglicht. Allerdings ist eine solche Integration sehr aufwendig und, wegen der Komplexität von Prozess-Engines, eigentlich nur durch den jeweiligen Hersteller realisierbar. Andererseits ist eben diese Erweiterung existierender PMS um die hier beschriebenen Funktionalitäten das längerfristige Ziel, weil diese so einem großen Nutzerkreis bereitgestellt werden können. Selbst deren Integration in GP-Modellierungswerkzeuge zur reinen (fachlichen / semantischen) GP-Dokumentation und -Optimierung ist erstrebenswert, weil sie eine detailliertere GP-Modellierung ermöglichen. Durch Analyse der dann entstehenden GP-Modelle kann später ermittelt werden, wie häufig die vorgestellten Konstrukte in realen GP benötigt werden.

Literaturverzeichnis

[Al83] Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11):832–843, 1983.

[ARD07] Aalst, W.M.P. van der; Rosemann, M.; Dumas, M.: Deadline-based Escalation in Process-Aware Information Systems. Decision Support Systems, 43(2):492–511, 2007.

[AWG05] Aalst, W.M.P. van der; Weske, M.; Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data & Knowledge Engineering, 53(2):129–162, 2005.

[Ba18] Bauer, T.: Ausführungssemantik für Sprünge in Geschäftsprozessen. Datenbank-Spektrum, 18(2):99–111, 2018.

[Ba19a] Bauer, T.: Modellierung erweiterter Beziehungen zwischen dem Start und dem Ende von Aktivitäten in Geschäftsprozessen: Szenarien, Anforderungen und Darstellungsvarianten. In: Proc. Informatik 2019, Gemeinsamer Workshop IT-Governance und Strategisches Informationsmanagement. S. 353–366, 2019.

[Ba19b] Bauer, T.: Pre-modelled Flexibility for Business Processes. In: Proc. 21th Int. Conf. on Enterprise Information Systems. S. 547–555, 2019.

[Ba21a] Bauer, T.: Effiziente Modellierung von Eskalationen und Stellvertretungen in Geschäftsprozessen. In: Proc. Informatik 2021, Workshop zum Stand, den Herausforderungen und Impulsen des Geschäftsprozessmanagements. S. 1503–1516, 2021.

[Ba21b] Bauer, T.: Pre-modelled Flexibility for the Control-Flow of Business Processes: Requirements and Interaction with Users. In: Enterprise Information Systems. S. 833–857, 2021.

[Bu18] Burghardt, M.: Projektmanagement: Leitfaden für die Planung, Überwachung und Steuerung von Projekten. Publicis Publishing, Erlangen, 10. Auflage, 2018.

[Da18] Dakic, D.; Stefanovic, D.; Cosic, I.; Lolic, T.; Medojevic, M.: Business Process Mining Application: A Literature Review. In: Proc. 29th DAAAM International Symposium on Intelligent Manufacturing and Automation. S. 866–875, 2018.

[He00] Heinlein, C.: Workflow- und Prozeßsynchronisation mit Interaktionsausdrücken und -graphen: Konzeption und Realisierung eines Formalismus zur Spezifikation und Implementierung von Synchronisationsbedingungen. Dissertation, Universität Ulm, Fakultät für Informatik, 2000.

[He01] Heinlein, C.: Workflow and Process Synchronization with Interaction Expressions and Graphs. In: Proc. 17th Int. Conf. on Data Engineering. S. 243–252, 2001.

[HW16] Hewelt, M.; Weske, M.: A Hybrid Approach for Flexible Case Modeling and Execution. In: Proc. 14th Int. Conf. on Business Process Management, Business Process Management Forum. S. 38–54, 2016.

[LWR10] Lanz, A.; Weber, B.; Reichert, M.: Workflow Time Patterns for Process-Aware Information Systems. In: Proc. Enterprise, Business-Process, and Information. S. 94–107, 2010.

[OA07] OASIS: Web Services Business Process Execution Language Version 2.0. Standard, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[Ob11] Object Management Group: Business Process Model and Notation (BPMN) 2.0. Standard, 2011. http://www.omg.org/spec/BPMN/2.0.

[RD98] Reichert, M.; Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2):93–129, 1998.

[Re00] Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. Dissertation, Universität Ulm, Fakultät für Informatik, 2000.

[RH06] Russell, N.; Hofstede, A.H.M. ter: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, 2006.

[RW12] Reichert, M.; Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer, 2012.

[We19] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, 3. Auflage, 2019.

[ZSA21] Zerbino, P.; Stefanini, A.; Aloini, D.: Process Science in Action: A Literature Review on Process Mining in Business Management. Technological Forecasting and Social Change, 172, 2021.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 171

Ein Modellierungskonzept zur Prozessstrukturierung für Soziale Dienstleister

Felix Holz, Michael Fellmann, Birger Lantow [1]

Abstract: Die Geschäftsprozesse von sozialen Dienstleistungsunternehmen weisen Eigenschaften von flexiblen, schwach strukturierten und wissensintensiven Prozessen auf. Für die modellbasierte Unterstützung solcher Prozesse wird die Verwendung von deklarativen Notationen empfohlen, wie z.B. CMMN oder ConDec. Im praktischen Einsatz haben sich jedoch Schwächen dieser Modellierungssprachen gezeigt. Im Zuge dieses Beitrages werden domänenspezifische Konzepte zur zielgerichteteren Modellierung vorgestellt, um die Tätigkeiten der Akteure besser strukturieren zu können. Sowohl Arbeitsdokumentationen von Mitarbeitenden eines Unternehmens im Bereich der Sozialhilfe, als auch durchgeführte Expertenworkshops bilden die Grundlage dieser Konzepte. Zwar können bereits existierende Sprachen zur Modellierung genutzt werden, geeigneter wäre jedoch eine Sprache mit Fokus auf die drei wesentlichen Konzepte der sozialen Arbeit: Ziele, Situationen und Maßnahmen.

Keywords: Notation; Personenbezogene Dienstleistungen; Soziale Dienstleistungen; CMMN; Wissensintensive Prozesse; Mensch-zentrierte Prozesse

1 Einleitung

Die Digitalisierung innerhalb Deutschlands ist je nach Branche unterschiedlich fortgeschritten. Für bessere Digitalisierung ist eine Formalisierung verschiedener Unternehmensaspekte erforderlich, um diese wiederum computergestützt zu verwalten und zu steuern. Soziale Dienstleistungsunternehmen können stark von der Digitalisierung profitieren, jedoch gestaltet sich die Formalisierung entsprechender Unternehmensaspekte schwierig. So sind die Prozesse von sozialen Dienstleistern gemeinhin flexibel, schwach strukturiert und die Tätigkeiten wissensintensiv. [LBL19, HLF21]
Hermann und Kurz [HK11] sehen das Case Management als vielversprechenden Ansatz zur Strukturierung von wissensintensiven Prozessen. Die Case Management Model and Notation (CMMN) [OM14] sei dafür eine geeignete Sprache. Allerdings geht die Umsetzung von CMMN nur schwerlich voran, was z.B. in fehlenden Workflow-Engines [Ro21] erkennbar ist. Ebenfalls scheinen den entsprechenden Zielgruppen die Vorteile einer Prozessrepräsentation nicht klar zu sein, da CMMN tendenziell in Arbeitsbereichen eingesetzt werden soll, in denen weniger Modellierungskompetenzen existieren. In [HL17] ist zudem erkennbar gewesen, dass die Notation nicht gänzlich den Wünschen und Bedarfen der Zieldomäne sozialer Dienstleister entspricht, da die Modellierung der Kernprozesse nur schwerlich möglich

[1] Universität Rostock, Albert-Einstein-Straße 22, 18059 Rostock, Deutschland [felix.holz,michael.fellmann,birger.lantow]@uni-rostock.de

doi:10.18420/modellierung2022-015

sei. Weiterhin bietet das deklarative Prozessmodellierungsparadigma eine Möglichkeit zur Darstellung von hoher Prozessflexibilität [MGP15], da hier – statt eines vordefinierten Ablaufs – Beschränkungen des Handelns modelliert werden; somit wären alle Tätigkeiten erlaubt, die nicht beschränkt sind [Ca16, Pv06]. Der große Nachteil deklarativer Sprachen (wie bspw. ConDec [Pv06]) liegt jedoch in der erschwerten Lesbarkeit und Zugänglichkeit für Nicht-Methodenexperten [MGP15].
In diesem Beitrag werden Konzepte der Prozessflexibilität und des Case Management vorgestellt, die gemeinsam mit Eigenschaften von wissensintensiven Prozessen Anforderungen einer Prozessdarstellung an die Domäne bilden. Da sich die bereits existierenden Prozessbeschreibungssprachen nicht in der Domäne etablieren konnten, werden drei bedeutende Konzepte zur möglichen Modellierung vorgestellt: zu erreichende Ziele, vorkommende Situationen und durchzuführende Maßnahmen. Dabei steht die zielgerichtete Unterstützung der inhaltlichen Arbeit mit den Klienten im Vordergrund. Zur Untermauerung und Verfeinerung der Erkenntnisse sind Expertenworkshops durchgeführt worden.

2 Zum Stand der Prozessflexibilität

Soziale Dienstleister sind den personenbezogenen Dienstleistern zuzuordnen. Dabei handelt es sich um Dienstleistungen, die auf die Verbesserung von Körper und/oder Geist eines Klienten gerichtet sind. Der wichtigste Aspekt liegt in der engen Zusammenarbeit von Klient und Leistungserbringer sowie der Verfolgung eines langfristigen Ziels. [LBL19]
Die durchzuführenden Tätigkeiten müssen je nach Situation und Kontext des Klienten ausgewählt werden. Anpassungen der Tätigkeiten an unvorhergesehene Umstände treten ebenfalls häufig auf, sodass der Leistungserbringer Erfahrung und Wissen in die Arbeit einfließen lassen muss, um die vorgegebenen Ziele unter unvorhersehbaren Umständen zu erreichen. Daher sind die Abläufe in sozialen Dienstleistungen den wissensintensiven Prozessen ähnlich und erfordern in der Prozessunterstützung ein hohes Maß an Flexibilität. Im Folgenden werden Konzepte der Prozessflexibilität erläutert, sowie das Case Management, welches Wissensarbeitern die nötige Flexibiltät zur Erfüllung ihrer Aufgaben bieten soll. [MLv15]

2.1 Flexibilität

Sadiq et al. [SSO01] definieren die Flexibilität als die Fähigkeit eines Workflow-Prozesses auf der Basis eines „losen“ oder nur teilweise spezifizierten Modells ausgeführt zu werden, während die volle Prozessspezifikation erst zur Laufzeit entsteht; jede Prozessinstanz kann dabei einzigartig sein. Schonenberg et al. [M.07] definieren vier Arten, wie Flexibilität in Prozessdarstellungen erreicht werden kann: *(1) Flexibility by Design* soll vorhersehbare Abweichungen des Prozesses abdecken und ermöglicht Änderungen an der Prozessspezifikation nur zur Design-Zeit. *(2) Flexibility by Change* ermöglicht hingegen Änderungen an der Prozessspezifikation zur Laufzeit, um den Prozess an unvorhergesehenes Verhalten

anzupassen. *(3) Flexibility by Deviation* beschreibt eine reine Unterstützung zur Laufzeit, indem es erlaubt, zur Laufzeit von der Prozessspezifikation abzuweichen, ohne Änderungen an dieser durchzuführen. *(4) Flexibility by Underspecification* beschreibt die Fähigkeit, zur Designzeit unvollständig definierte Prozesse dennoch zur Laufzeit ausführen zu können.
Reichert und Weber [RW12] stellen hingegen Anforderungen an die Leistungsfähigkeit flexibler Prozessspezifikationen im Kontext von Process Aware Information Systems (PAIS) als Taxonomie dar. Die vier Hauptkriterien sind dabei *(1) Adaption*, die Anpassung der Prozessinstanz an die Prozessdefinition, um geplante und ungeplante Abweichungen auffangen zu können. Für die *(2) Variabilität* muss die Prozessspezifikation mehrere Prozessvarianten abdecken; dies kann über konfigurierbare Prozessmodelle geschehen, in welchen je nach Kontext bestimmte Varianten ausgewählt werden können. Die *(3) Looseness* beschreibt eine „lockere“, zur Design-Zeit unvollständige Prozessdefinition, um Prozessentscheidungen und Abweichungen zur Laufzeit zu erlauben. Ein übergeordnetes Prozessziel steht dabei jedoch fest. Die Fähigkeit eines PAIS, die Prozessdefinition zu verändern, sobald sich der Prozess entwickelt, wird mit der *(4) Evolution* beschrieben.
Das wichtigste Kriterium für den Bereich Sozialer Dienstleister stellt die „Looseness“ bzw. Flexibility by Underspecification dar, also eine freie Prozessspezifikation, die zur Laufzeit auch ungeplante Aktivitäten unterstützt [RW12]. Diese Looseness-Flexibilität kann durch Unterspezifikation der Prozessdefinition [M.07], Case-Based- und Pattern-Mechanismen [Co18], sowie über Modularität [Co18], Low Fidelity Modelle [No03], den Pockets of Flexibility [SSO01], generische Prozessmodelle und Platzhalterelemente [va99] erreicht werden. Da der Sozialarbeiter während der Bearbeitung seiner Aufgaben vollen Handlungsspielraum benötigt und situationsspezifische Entscheidungen treffen muss, soll weniger ein stoisch zu folgender Prozess modelliert, sondern eine Auswahl sinnvoller Aktivitäten geboten werden, die für die aktuelle Situation angebracht sind.

2.2 Case Management

In [Ro21] wird das Case Management als ein kollaborativer Prozess beschrieben, in welchem die Möglichkeiten und Dienste beurteilt, geplant, implementiert, koordiniert, überwacht und evaluiert werden, um die Bedürfnisse der Klienten zu erfüllen. Das Case Management beinhalte Interessenvertretung, Kommunikation und Ressourcenmanagement für qualitäts- und kosteneffektive Ergebnisse. Alle relevanten Informationen werden zentral abgelegt, um organisationale Anforderungen erfüllen zu können. Diese zentrale Struktur stellt einen Fall dar, welcher im Kontext der Sozialhilfe alle Informationen zu einem Klienten beinhaltet. Ebenso wie Prozesse, bestehen Fälle als Falldefinitionen auf der Schema-Ebene, wie auch als Instanzen [HL17].
Die Case Management Model and Notation (CMMN) ist eine von der OMG entwickelte Notation zur Unterstützung des Case Managements und als deklarative Alternative zur Business Process Modelling and Notation (BPMN). [OM14] Die hauptsächlichen Elemente des CMMN sind der **Fall**, dort werden alle weiteren Elemente untergebracht, die zu einem Fall gehören. Ein **Task** beschreibt eine Aktivität, die durchgeführt werden kann. Eine **Stage**

kann als „Episode“ des Falls angesehen werden, mit deren Hilfe Tasks modularisiert werden können. **Event Listener** und **Meilensteine** sind ebenfalls modellierbare Elemente, die jedoch nach [Ro21] weniger Relevanz in der Modellierung einnehmen. Im Case Management wird nicht zwischen Design-Zeit und Laufzeit unterschieden; jedem Wissensarbeiter sollte erlaubt sein, die Fallinstanz so zu gestalten und auszuführen, wie er es für angebracht hält [Ca16]. Dementsprechend bietet die Notation zur Unterstützung der Strukturierung des Modells und der Planung des Ablaufs diskrete Modellelemente, welche nur visuelle Zwecke erfüllen und von der zugrundeliegenden Workflow-Engine ignoriert werden können. Darüber hinaus können Modellelementen zur weiteren Spezifizierung Annotationen hinzugefügt werden, die sich sowohl symbolisch abheben, als auch von der Workflow-Engine interpretiert werden können. Beispielsweise werden hiermit erforderliche oder sich wiederholende Tasks näher beschrieben. Für Stages wird die Möglichkeit des Ein- und Ausklappens gegeben, um die Komplexität des Modells zu reduzieren. Weiterhin werden Verbindungen von Modellelementen **Sentries** angefügt. Diese definieren zum einen die Leserichtung der Relationen, zum anderen fungieren sie als Ein- und Ausgangsvoraussetzungen der Modellelemente.

3 Modellierung wissensintensiver Prozesse in sozialen Dienstleistungen

In [HL17] hat sich bei der domänenspezifischen Prozessmodellierung die Einsicht ergeben, dass CMMN nicht geeignet sei, „Kernaufgaben der betrachteten Domäne“ abzubilden. Das erstellte CMMN-Modell bildete fokussiert die ersten Phasen der „Hilfe zur Erziehung“ ab, konnte jedoch nicht ausreichend die inhaltliche Ausgestaltung der tatsächlichen Betreuung darstellen [HL17]. CMMN wird in der Domäne zwar als ein valides Werkzeug der Modellierung angenommen, die Schwächen der Notation – wie die komplexe Modellierung, die symbolische Überfrachtung und die Nicht-Nachvollziehbarkeit von Übergangskriterien – führten jedoch zu einer geringeren Nutzungsabsicht. Aus diesen Erkenntnissen hat sich der Bedarf ergeben, die inhaltliche Auseinandersetzung mit den Klienten modellieren zu können, sodass dort Muster, Gemeinsamkeiten und Handlungsempfehlungen ersichtlich werden [HLF21].

3.1 Wissensintensive soziale Arbeit

Die tatsächlichen Tätigkeiten personenbezogener und sozialer Dienstleister bilden Eigenschaften der wissensintensiven Arbeit ab. Geschäftsprozesse in der Wissensarbeit zeichnen sich zum einen durch Unvorhersehbarkeit und zum anderen durch Wissensintensität aus – der Abhängigkeit des Prozessesergebnisses und -verlaufs von dem impliziten Wissen und den Erfahrungen der Akteure sowie der Schaffung neuen Wissens [SL20].
Boissier et al. [BRL19] haben die Eigenschaften von wissensintensiven Prozessen auf Grundlage von [IMv13] zusammengefasst: So kann der Prozesserfolg sowohl vor als auch

nach der Ausführung nicht abgesehen werden. Jede Kundenanfrage wird individuell behandelt und bildet eine eigene Prozessinstanz ab; dabei hängt die Ausführung des Prozesses sowohl von den Daten ab, die in dem derzeitigen Verlauf erhoben werden als auch vom Wissen des ausführenden Akteurs. Die Tätigkeiten sind komplex und erfordern Kreativität, Kollaboration und das simultane Einbeziehen vieler Informationen. Insbesondere Rahmenbedingungen der Ausführung sind zu beachten. Wissensintensive Prozesse sind schwierig zu automatisieren, zu kontrollieren und zu verbessern, da die Prozessausführung keine universelle Sequenz von Aktionen darstellt, die zur Design-Zeit erstellt und getestet werden kann; ungeplante Aktivitäten und Situationen kommen häufig vor. Das wichtigste Konzept für wissensintensive Prozesse ist das *Ziel*.
Diese Eigenschaften treffen auf die Soziale Arbeit ebenfalls zu. Nach [MGP15] erfordern solche dynamischen, nicht-standardisierten und wissensintensiven Prozesse die Laufzeitflexibilität der deklarativen Prozessmodellierung. Der große Nachteil des deklarativen Paradigmas liegt jedoch in der erschwerten Lesbarkeit und Verständlichkeit, insbesondere für Nicht-Methodenexperten. [MGP15]
Eine weitere Besonderheit in deutschen Sozialen Dienstleistungsunternehmen zeigt sich zumeist in den Reglementierungen, die zu befolgen sind (z.B. Sozialgesetzbüchern) und den staatlich geprägten Kunden/Trägern, wie Sozial- und Jugendämtern. Diese Faktoren führen zu einer Berichtspflicht, in dessen Zuge die Sozialarbeiter die Tätigkeiten mit ihren Klienten dokumentieren. Wie in jedem Dienstleistungsunternehmen treten auch in Sozialen Dienstleistungen sowohl administrative Prozesse (wie die Personalplanung), als auch Routineprozesse (z.B. die Aufnahme eines neuen Klienten) auf; diese lassen sich auch mit bereits bekannten Sprachen abbilden und digital unterstützen. Die Fallbearbeitung und die Interaktion mit dem Klienten bilden den wissensintensiven Kern der sozialen Arbeit und somit die Hauptaufgabe eines unterstützenden Prozessmodells.

3.2 Durchgeführte Workshops

Der Bedarf einer Prozessstrukturierung ergab sich aus einer Reihe von Workshops, in welchen sich Akteure aus der Praxis mit Forschern und Softwareentwicklern zur Digitalisierung der Kinder-, Jugend- und Sozialhilfe austauschen. Diese Workshops werden seit Mitte 2019 im einem 5 bis 8-wöchentlichen Rhythmus abgehalten, in welchen die Domänenexperten aus drei deutschen Unternehmen der Sozialhilfe partizipieren. Die Unternehmen agieren nach den Sozialgesetzbüchern (SGB) 8, 9 und 12 und bieten sowohl ambulante, als auch stationäre Betreuungsleistungen an. Der Expertenkreis besteht hauptsächlich aus SozialarbeiterInnen in leitenden Positionen oder Geschäftsführern, welche 10-25 Jahre Berufserfahrung vorweisen können. Wegen unvorhergesehener Umstände und Termindiskrepanzen wechseln die Teilnehmer jedoch, allerdings sind immer mindestens zwei Praktiker zu diesen Treffen zugegen. Der Workshopstil gestaltet sich frei in einem ein- bis zweistündigen Zeitrahmen und Erhebungen werden erfahrungsgeleitet-intuitiv durchgeführt. Weiterhin stehen als zusätzliche Quelle der zu modellierenden Informationen anonymisierte Berichtsdokumentationen über Klientenfälle zur Verfügung, welche die inhaltliche Auseinandersetzung

mit dem Klienten, also die tatsächlichen Tätigkeiten der Sozialarbeiter, beschreiben. Die Berichte weisen jedoch schwankende Qualität auf, da sie individuell verfasst und nur selten überprüft werden. Es existieren zwar amtsseitige Vorgaben, wie die Berichte auszusehen haben, aber auch diese unterscheiden sich stark z.B. je nach Gesetzbuch, Kostenträger oder Landkreis. In den kooperierenden Unternehmen werden auch interne Dokumentationen angefertigt, welche zumeist mehr Fallwissen beinhalten [HLF21].

4 Vorschlag eines domänenspezifischen Modellierungskonzeptes

In diesem Kapitel wird das Konzept einer graphischen Modellierungssprache zur Unterstützung der sozialen Arbeit mit Fokus auf die Klienteninteraktion vorgestellt. Ein entsprechendes Modell soll den Sozialarbeiter in der Vor- und Nachbereitung durch eine Auswahl möglicher Maßnahmen unterstützen, die für den Kontext oder die Situation des Klienten angemessen ist. Es soll sich dabei nicht um die Unterstützung von Routineprozessen handeln (wie die Neuaufnahme von Klienten [HL17]), da diese zum einen deutlich strukturierter sind und zum anderen bereits in einer textuellen Form vorliegen (z.B. TODO-Listen). Im Zuge dieses Kapitels werden zunächst die während der Workshops erhobenen, domänenspezifischen Zwecke von Modellen aufgeführt. Daraufhin werden zentrale Modellelemente und Relationen für die Strukturierung der klientenbezogenen sozialen Arbeit beschrieben.

4.1 Zweck der Modellierung für Soziale Dienstleister

Aus den durchgeführten Workshops sind folgende Aspekte von Modellen entstanden und mittels Experten validiert worden, welche den Sozialarbeitern einen Mehrwert bieten können: (M1) Mögliche Agenda zur Unterstützung bei der Fall- und Maßnahmenplanung, sowie Abschätzung der Erwartungshaltung an die Klienteninteraktion; (M2) Anschaulichkeit des aktuellen Standes des Falls; (M3) Überblick und Vergleichsmöglichkeit der von Kollegen durchgeführten Maßnahmen als Handlungsempfehlung und zur Einarbeitung neuen Personals; (M4) Diskussionsgrundlage zur Unterstützung der Teamarbeit und (M5) Hilfestellung zur Selbstreflexion nach Abschluss des Falls.
Dabei wurden die Domänenexperten zu Problemen und Besonderheiten in der Prozessausführung bzw. Fallbearbeitung befragt. Die Ausführungen der Experten wurden zusammengefasst und mit den Prozessmodellzwecken aus [va16][S.29] abgeglichen. Die Validierung selbst hat ergeben, dass jeder dieser Mehrwerte wünschenswert sei und unterschiedliche Phasen der sozialen Arbeit beschreibe. Auch implizieren die genannten Vorteile weitere Aspekte, wie die Weiterentwicklung von Prozessen. Auch wenn jeder dieser Punkte als ein valider Mehrwert erscheint, muss der Fokus des Modells auf einen Zweck abgestimmt werden. So muss ein Modell zur Hilfestellung der Selbstreflexion nur eine Prozessinstanz darstellen können, während zur Diskussionsgrundlage oder zur Planungsunterstützung mehrere Fälle zusammengefasst abgebildet werden müssen.

4.2 Modellelemente und Relationen

Um die klientenbezogenen Tätigkeiten eines Sozialarbeiters zu modellieren, sollten die Modellelemente so genau wie möglich die tatsächlichen Konzepte der Zieldomäne abbilden [Fr13], dabei jedoch im Sinne der Übersichtlichkeit abstrakt genug bleiben. Die Tabellen 1 und 2 zeigen die notwendigen Modellelemente und Relationen.
Im Case Management ist zum Beispiel das „Ziel“ ein zentrales Konzept [Ro21], ebenso

Modellelement	Beschreibung
Fallklasse	beschreibt das Wurzelelement eines Falls und bildet eine Gruppe von Fällen ab, die ähnliche Eigenschaften besitzen. Es bildet den Rahmen um das Modell und kann auch als das „Oberziel“ des Falls fungieren.
Ziel	beschreibt „Unterziele“ innerhalb der Fallbearbeitung und vereint zielspezifische Situationen und Maßnahmen in sich. Je nach Klient unterscheidet sich die Wichtigkeit einzelner Ziele, spiegelt dennoch mögliche „Arbeitsbereiche“ der gesamten Fallklasse wider.
Situation	beschreibt den Zustand, in welchem sich ein Klient befinden kann. Ein Sozialarbeiter muss auf solche Situationen reagieren können, sie stellen also häufig einen Auslöser dar.
Maßnahme	beschreibt eine klientenbezogene Tätigkeit. Aus den amtsseitig geforderten Berichtsdokumentationen geht hervor, dass die möglichen Tätigkeiten zwar endlich, jedoch abstrakt sind (z.B. Beratung). Um den Sozialarbeitern gezieltere Hilfestellungen bieten zu können, sollten diese abstrakten Tätigkeiten mit einer inhaltlichen Spezifikation (Thema) verbunden werden (z.B. Beratung Abstinenzverhalten).

Tab. 1: Zentrale Konzepte (Modellelemente) für die klientenbezogene soziale Arbeit.

Relation	Beschreibung
Zugehörigkeit	Fallklassen und Ziele können mehrere Modellelemente beinhalten. In diesem Fall entspricht die Zugehörigkeit der Bedeutung, dass die entsprechenden Elemente bei der Bearbeitung des zugehörigen Ziels relevant sind, also zum Ziel gehören. Die Zugehörigkeit ist transitiv.
Voraussetzung	Wenn zwei Modellelemente gerichtet miteinander verbunden sind (z.B. durch einen gerichteten Pfeil), fungiert das Element, von welchem der Pfeil ausgeht als Voraussetzung für das Element, in welches der Pfeil eingeht. Zweiteres kann nur ausgeführt werden, sofern die Voraussetzungen erfüllt sind (z.B. Abschluss einer Maßnahme, Eintreten einer Situation).
Empfehlung	dient zur Darstellung von empfohlenen Verbindungen von Modellelementen, die jedoch keine Voraussetzung füreinander bilden. Diese Relation soll einen sinnvollen Zusammenhang zweier Elemente verdeutlichen – sie sollen jedoch auch unabhängig voneinander auftreten können.

Tab. 2: Zentrale Konzepte (Relationen) für die klientenbezogene soziale Arbeit.

wie es in der Flexibility by Underspecification [M.07] beschrieben wird, sollte auch ein lose definierter Prozess einem Ziel folgen. Über die Zugehörigkeitsrelation können Zielhierarchien gebildet werden, um den Kontext des Klienten als zu erreichenden Zustand näher eingrenzen zu können. Des Weiteren sehen sich Wissensarbeiter - und somit auch

Sozialarbeiter - häufig unvorhergesehenen Situationen gegenüberstehen [SL20]. Situationen und Ziele bilden die Rahmenbedingungen der wissensintensiven Arbeit ab [BRL19]. Diese Rahmenbedingungen können über die gegebenen Relationen verfeinert werden, indem notwendige Abhängigkeitsbeziehungen oder empfohlene Zusammenhänge dargestellt werden.
Obwohl sich die angegebenen Konzepte auch rein textbasiert abbilden ließen (z.B. durch „Stichpunkthierarchien"), bietet eine graphische Modellierungssprache mehr Vorteile. Besonders in der Domäne der sozialen Arbeit steigt die Nutzungsabsicht als Nachschlagewerk oder Diskussionsgrundlage (M5), wenn es sich um beschriftete „Bilder" handelt, statt um strukturierte Texte; die Informationen lassen sich schneller erfassen, auch wenn sie nicht so präzise sein mögen.
Die vorgestellten Konzepte können durchaus mit CMMN modelliert werden. So können CMMN-Tasks die Maßnahmen und CMMN-Stages die Ziele darstellen. Event Listener könnten behelfsmäßig Situationen abbilden. Jedoch ist es für eine graphische Sprache wichtig, dass die Symbole die realen Konzepte widerspiegeln [Fr13, Mo09]; die Bedeutung der Event Listener ist zum einen allgemein weniger verständlich [Ro21], zum anderen würde durch die geringere Größe des Symbols das entsprechende Konzept weniger wichtig wirken. Die existierende CMMN-Relation erschwert durch die Unterscheidung von Ein- und Ausgangssentries das Verständnis für Nicht-Methodenexperten und erlaubt weniger eine Unterscheidung zwischen empfohlenen und erforderlichen Verbindungen.

5 Fazit und Ausblick

Sozialarbeiter sehen sich in der Interaktion mit den Klienten oftmals unvorhergesehenen *Situationen* gegenüberstehen und müssen basierend auf ihrem Wissen und ihrer Erfahrung entscheiden, welche *Maßnahme* notwendig ist. Dabei muss das möglicherweise längerfristige Klienten*ziel* bearbeitet werden. Diese drei in den Expertenworkshops entstandenen Kernaspekte bilden die Grundlage, um den Akteuren eine verbesserte Arbeitsunterstützung und -Strukturierung zu ermöglichen und das Prozesswissen zu externalisieren. Der Teilnehmerkreis der Expertenworkshops ist jedoch nicht unbedingt repräsentativ. Weitere, methodisch genauere Untersuchungen mit praktizierenden Sozialarbeitern zur Validierung der Erkenntnisse sind somit notwendig. CMMN eignet sich behelfsmäßig, um die zentralen Konzepte darzustellen. Jedoch ist es auf der einen Seite zu allgemein und bietet zu viele für die Domäne nicht benötigte Möglichkeiten, welche der Zielgruppe das Verständnis erschweren. Auf der anderen Seite kann die Sprache nicht alle zentralen Konzepte angemessen reflektieren (wie die Situation). Demnach muss sich in Zukunft eine eigene domänenspezifische Notation bilden, simpler und weniger visuell überfrachtet als CMMN. Die ersten Anforderungen wurden in diesem Beitrag gegeben; das Erstellen einer formalen Spezifikation, das Finden von passender Symbolik sowie eine Evaluation der Konzepte bilden die nächsten Schritte, um die klientenbezogene soziale Arbeit mehr strukturieren und formalisieren zu können [Fr13].

Literaturverzeichnis

[BRL19] Boissier, F.; Rychkova, I.; Le Grand, B.: Challenges in knowledge intensive process management. Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOCW, 2019-October, 2019.

[Ca16] de Carvalho, Renata M.; Mili, Hafedh; Gonzalez-Huerta, Javier; Boubaker, Anis; Leshob, Abderrahmane: Comparing ConDec to CMMN. 2016.

[Co18] Cognini, Riccardo; Corradini, Flavio; Gnesi, Stefania; Polini, Andrea; Re, Barbara: Business process flexibility - a systematic literature review with a software systems perspective. Information Systems Frontiers, 20(2):343–371, 2018.

[Fr13] Frank, Ulrich: Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In (Reinhartz-Berger, Iris; Sturm, Arnon; Clark, Tony; Cohen, Sholom; Bettin, Jorn, Hrsg.): Domain engineering, S. 133–157. Springer, Berlin and Heidelberg, 2013.

[HK11] Herrmann, C.; Kurz, M.: Adaptive case management: Supporting knowledge intensive processes with IT systems. Communications in Computer and Information Science, 213 CCIS:80–97, 2011.

[HL17] Herzog, P.; Lantow, B.: Adaptive case management in social institutions [Adaptive Case Management in sozialen Einrichtungen]. Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI), 275, 2017.

[HLF21] Holz, Felix; Lantow, Birger; Fellmann, Michael: Towards a Content-Based Process Mining Approach in Personal Services. In (Augusto, Adriano; Gill, Asif; Nurcan, Selmin; Reinhartz-Berger, Iris; Schmidt, Rainer; Zdravkovic, Jelena, Hrsg.): ENTERPRISE, BUSINESS-PROCESS AND INFORMATION SYSTEMS MODELING, Jgg. 421 in Lecture Notes in Business Information Processing, S. 62–77. SPRINGER NATURE, [S.l.], 2021.

[IMv13] Işik, Öykü; Mertens, Willem; van den Bergh, Joachim: Practices of knowledge intensive process management: quantitative insights. Business Process Management Journal, 19(3):515–534, 2013.

[LBL19] Lantow, Birger; Baudis, Tom; Lambusch, Fabienne: Mining Personal Service Processes. In (Abramowicz, Witold; Corchuelo, Rafael, Hrsg.): Business Information Systems Workshops, Jgg. 373 in Springer eBook Collection, S. 61–72. Springer International Publishing and Imprint Springer, Cham, 2019.

[M.07] M.H. Schonenberg; R.S. Mans; N.C. Russell; N.A. Mulyar; Aalst, van der, W.M.P.: Towards a taxonomy of process flexibility (extended version). BPM reports. BPMcenter. org, 2007.

[MGP15] Mertens, Steven; Gailly, Frederik; Poels, Geert: Enhancing Declarative Process Models with DMN Decision Logic. In (Nurcan, Selmin; Guerreiro, Sérgio; Ma, Qin; Schmidt, Rainer, Hrsg.): Enterprise, Business-Process and Information Systems Modeling, Jgg. 214 in Springer eBook Collection Computer Science, S. 151–165. Springer, Cham, 2015.

[MLv15] Marin, Mike A.; Lotriet, Hugo; van der Poll, John A.: Metrics for the Case Management Modeling and Notation (CMMN) Specification. In (Barnett, Richard J., Hrsg.): Proceedings of the 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists. ACM Digital Library, ACM, New York, NY, S. 1–10, 2015.

[Mo09] Moody, D.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering, 35(6):756–779, 2009.

[No03] Noll, John: Flexible process enactment using low-fidelity models. In: Proceedings of the International Conference on Software Engineering and Applications (SEA 03). Jgg. 45, 2003.

[OM14] OMG: , Case Management Model and Notation, version 1.0. Technical Report May, OMG, May 2014, 2014.

[Pv06] Pesic, M.; van der Aalst, W. M. P.: A Declarative Approach for Flexible Business Processes Management. In (Eder, Johann; Dustdar, Schahram, Hrsg.): Business process management workshops, Jgg. 4103 in Lecture Notes in Computer Science, S. 169–180. Springer, Berlin, 2006.

[Ro21] Routis, Ioannis; Bardaki, Cleopatra; Dede, Georgia; Nikolaidou, Mara; Kamalakis, Thomas; Anagnostopoulos, Dimosthenis: CMMN evaluation: the modelers' perceptions of the main notation elements. Software and Systems Modeling, 20(6):2089–2109, 2021.

[RW12] Reichert, Manfred; Weber, Barbara: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer Science & Business Media, 2012.

[SL20] Szelagowski, M.; Lupeikiene, A.: Business Process Management Systems: Evolution and Development Trends. Informatica (Netherlands), 31(3):579–595, 2020.

[SSO01] Sadiq, Shazia; Sadiq, Wasim; Orlowska, Maria: Pockets of Flexibility in Workflow Specification. In (Kunii, Hideko S., Hrsg.): Conceptual modeling, Jgg. 2224 in Lecture Notes in Computer Science, S. 513–526. Springer, Berlin, 2001.

[va99] van der Aalst, W.M.P.: Generic workflow models: how to handle dynamic change and capture management information? In: Proceedings / 1999 IFCIS International Conference on Cooperative Information Systems. IEEE Computer Soc, Los Alamitos, Calif., S. 115–126, 1999.

[va16] van der Aalst, Wil, Hrsg. Process mining: Data science in action. Springer, Berlin and Heidelberg and New York and Dordrecht and London, second edition. Auflage, 2016.

M. Riebisch, M. Tropmann-Frick (Hrsg.): Modellierung 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 181

A Cross-Disciplinary Process Modelling Language for Validating Reconfigured Production Processes[1]

Sandro Koch[2], Tim Wunderlich[3], Jonas Hansert[4], Thomas Schlegel[4], Steffen Ihlenfeldt[3], Robert Heinrich[2]

Abstract: Modelling and reconfiguration of production processes require knowledge across different domains. This in-depth knowledge is necessary to avoid possible side effects that could threaten the production plant, the workpiece or the worker. Therefore, process modelling approaches allow adding additional data to the steps of a process. Such additions can be constraints, which need to be fulfilled before a step can be executed. Upon reconfiguration of production processes, these constraints need to be validated to ensure that the objective of the process is still met. However, this task demands expertise in the field of process modelling as well as in the domain of the production process and the production plant. To the best of our knowledge, state-of-the-art production process modelling approaches are unable to determine the semantic validity of a reconfigured production process. In this paper, we introduce a domain-specific modelling language dedicated to model and validate constraints between production steps. With this approach, we aim to assist the operator in reconfiguring production processes. We evaluate this approach in three case studies and show that our approach can detect violated constraints in production processes.

Keywords: domain-specific modelling language; process modelling; process validation; resilience; cyber-physical production systems

1 Introduction

The increasing prevalence and availability of Internet of Things (IoT) technology like sensors, actuators, nodes, and tags create a much higher and more dynamic integration into the physical world and new challenges for future software systems. The interconnection of IoT components increases the complexity of such systems and the potential for errors and failures grows, as well. Hence, also the risk of errors and failures increases. Due to the complexity of IoT systems and the human interaction with such systems, not all errors can be predicted. IoT are frequently confronted with unknown input and output values due to the constantly changing and uncontrollable execution context of the system. As a result, processes have to deal with highly volatile resources and states at runtime, which can fail at any time and result in the need for reconfiguration of process. Reconfiguring a failed process is complex, because the partial state of the system until the failure needs to be considered.

[1] This work was partially funded by the German Federal Ministry of Education and Research under grant 01IS18067A/B/D (RESPOND), and the KASTEL institutional funding.

[2] Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, GER, firstname.lastname@kit.edu
[3] Fraunhofer IWU, Reichenhainer Str. 88, 09126 Chemnitz, GER, firstname.lastname@iwu.fraunhofer.de
[4] Karlsruhe University of Applied Sciences, Moltkestr. 30, 76133 Karlsruhe, GER, firstname.lastname@h-ka.de

doi:10.18420/modellierung2022-004

In this paper, we present a *Domain-Specific Modelling Language* (DSML) that enables people who are not experts on the production system but are experts on the production process to model and modify production processes in an IoT system. In the development of the DSML, besides computer scientists, experts of mechanical and electrical engineering and experts in the field of modelling production processes were involved. The DSML allows the domain expert to model sub-processes with conditions that must be fulfilled. These conditions can be pre-conditions that must be fulfilled before a sub-process can be executed as well as post-conditions that are fulfilled after a sub-process has been executed. In case the conditions are not met, the automated verification detects such errors. As a result, non-domain experts can use the DSML given in this paper to model production processes in an IoT system.

Our contributions in this paper are: 1. A DSML that allows to model production processes in an IoT system. In contrast to general-purpose process modelling approaches, our approach focuses on the conditions and constraints of production processes. 2. A verification process that allows to validate reconfigured production processes. More details regarding the state-of-the-art is provided in Sect. 2.

The remainder of the paper is structured as follows: State-of-the-art of process verification is presented in Sect. 2. The requirements for the DSML are discussed in Sect. 3. The DSML is presented in Sect. 4. In Sect. 5, we evaluate the DSML by modelling three production processes based on two real world production lines and on one laboratory case study. The paper concludes in Sect. 6.

2 State of the Art

Business Process Model and Notation 2 (BPMN2) is a historically grown, monolithic DSML for modelling and simulating business processes. Due to the strong coupling of the entities, Business Process Model and Notation (BPMN) is hard to maintain [Ro17] and to extend [HSR19]. However, the BPMN2 specification and other process modelling approaches like DECLARE [PSA07] do not support the validation of pre- and post-conditions of process models and extending these approaches would increase their complexity. With increasing complexity of process models, the need for automatic and reliable testing of these models increases, as well. This arises from the necessity to follow regulations in certain fields, e. g., patient care [Ly12] or pharmaceutical quality assurance [Al03], but also from the need for validating rescheduled manufacturing processes as a result of risk mitigation in Cyper-physical Production Systems (CPPS) [Ih21].

Validating process models during the modelling process can help to find failures and improve the quality [Kü10]. Previous research focuses on annotating business processes with semantic information regarding the constraints between certain process steps [WGH08; WHM08] and considers general business processes such as sales orders. Furthermore, declaring constraints between process steps has been examined as a vehicle to synthesise emergency

response workflows on the basis of predefined templates [ML17] and to dynamically generate and adapt process models describing software engineering workflows [GOR10a; GOR10b; GOR11]. Processes can also be represented as a combination of Unified Modeling Language (UML) and Object Constraint Language (OCL). This can be converted into logical expressions, which in turn can be used for validation [EST15]. Furthermore, Petri nets were used to allow verification of Event-driven Process Chain (EPC) models [Va99], but EPC does not provide a concise syntax for modelling constraints between process steps. Instead, linking multiple events to a process step requires the combination with logical connectors. It is also possible to model production processes using imperative languages such as Prolog, RuleML [BPS10] or Stanford Research Institute Problem Solver (STRIPS) [FN71]. The process structure is modelled as a set of facts and rules and the Prolog or RuleML engine is then used to validate the modelled process. In STRIPS, a set of states and actions with pre- and post-conditions can be defined. Due to the imperative nature of these languages, they are harder to comprehend than our approach.

While the risk of modelling processes that are not valid are reduced by these approaches, domain knowledge is still required to model these processes. In addition, the applicability of these approaches has not been empirically examined in the context of CPPS, yet. Finally, UML and OCL are General-purpose programming languages (GPPLs) and therefore significantly more difficult than a dedicated DSML for modelling and evaluating production processes.

3 Requirement Elicitation

We identified the following requirements that the language needs to comply with: Certain processes cannot be executed when certain conditions are not fulfilled at the beginning of the process. Thus, the first requirement **R1** for the language is to specify the preconditions of a process. After a processes is executed, the state of the plant or the workpiece can be changed, thus we derive **R2** that the language must be able to also model the postconditions of a process. It is also possible that after a process is executed, previously specified conditions are no longer relevant, thus we derive **R3** that the language must be able to remove conditions. A process can also explicitly exclude a condition, thus we derive **R4** that the language must be able to exclude conditions. Besides the syntactic requirements, we also require the language to verify whether a modelled process meets the specified condition, thus we derive **R5** that the language must be able to verify whether a process is valid.

4 A Language for Validating Reconfigured Production Processes

4.1 DSML Implementation

In this section, we introduce the language engineering framework used in our approach. Furthermore, we will explain key parts of the grammar of our language.

Xtext[5] is a framework for language engineering that allows to design and implement editing tools for textual *Domain-Specific Language*s (DSLs). The DSL is based on a grammar, like the Extended Backus–Naur Form (EBNF). The Xtext framework generates plug-ins for the Eclipse[6] *Integrated Development Environment* (IDE) without any additional effort. We chose Xtext, as it is integrated with the *Eclipse Modelling Framework* (EMF)[7], which is a modelling framework for code generation.

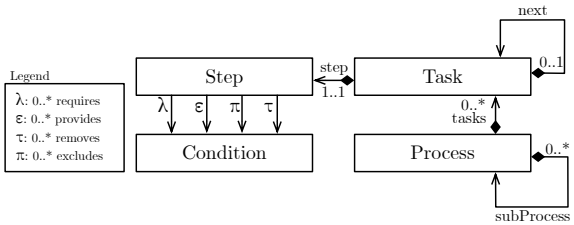


Fig. 1: Reduced illustration of the Process Model Validation Language (PMVL) syntax.

Fig. 1 shows a reduced model of the PMVL. A process consists of a sequence of further (sub)processes or tasks. We decided to introduce subprocesses to enable modularisation of processes and to allow reducing the size of a process. Thus, it is possible to reuse already existing processes and to reduce code duplication. The tasks in a process represent a sequence of steps that are executed in a certain order. One task of a process consists of a step. A step can have conditions that are either required (λ) and must be fulfilled, so that the step can be executed, or a step can provide (ϵ) conditions that are fulfilled after a step is executed. A step can also remove (τ) conditions that were fulfilled until the step was executed. Finally, a step can also exclude (π) conditions that would prevent the execution of the task.

The PMVL is divided in the declaration of processes, tasks, steps, and states. Conditions are part of steps and cannot be declared separately. Due to space limitations, the textual syntax of a process it described in more detail in our supplementary material [Ko22].

4.2 Production Process Validation

Besides the language definition, the implementation of validation steps is also supported by the Xtext framework. We distinguish between the following kinds of validation: 1) Inconsistent preconditions: Due to the backward references, the algorithm traverses through all steps leading to the current step. By traversing through these steps, contradicting preconditions can be identified and the corresponding steps are highlighted. 2) Inconsistent preconditions: Due to the forward references, the algorithm traverses through all steps following the current

[5] https://www.eclipse.org/Xtext/
[6] https://www.eclipse.org/eclipse/
[7] https://www.eclipse.org/modeling/emf

step. By traversing through these steps, contradicting postconditions can be identified and the corresponding steps are highlighted. 3) Branch inconsistency: If processes are determined to run in parallel, the algorithm checks contradicting conditions of the steps in a branch. If errors are found, the corresponding steps are highlighted. 4) Conditions already met: If a step does provide the same set or a sub-set of postconditions as the preceding step, the algorithm identifies the previous step via the backward reference. If the conditions are already met by a preceding step, the redundant step is highlighted. 5) Cycle detection: Via a depth-first search utilising the forward and backward references, the algorithm checks whether the process reaches the last step. If the depth-first search does not reach the last step in the processes, the corresponding processes that form a cycle are highlighted.

5 Evaluation

This section presents the evaluation of the PMVL based on three case studies. The evaluation design is explained in Sect. 5.1. The case studies are presented in Sect. 5.2. The evaluation results are presented in Sect. 5.3 and discussed in Sect. 5.4.

5.1 Evaluation Design

The evaluation of the PMVL follows the Goal Question Metric (GQM) approach [Ba94]. The goal is to analyse the PMVL to evaluate the accuracy of the PMVL instances with respect to precision and completeness of the identified errors from the point of view of the production plant domain experts in the context of reconfiguring and validating production processes.

To evaluate the accuracy of the PMVL instances, we compare a list of the identified warnings and errors to a reference list. The list of errors has been created using PMVL. The reference list has been created manually by the best available experts in developing and maintaining the respective case study production plant and its production processes. For the filling line case study, we asked the co-authors from the Karlsruhe University of Applied Sciences. For the assembly line case study, we asked the co-authors from the Fraunhofer Institute for Machine Tools and Forming Technology IWU. For the production line case study, we asked the plant experts from SITEC Industrietechnologie GmbH. Where the co-authors fulfill also the role of the domain experts who provide a reference list, we consider the list as correct and complete.

For evaluating the accuracy of the PMVL, we ask the research question: **How is the accuracy of our approach?** To answer this question, we use the metric $\boldsymbol{F_1}$ **score**. F_1 is a harmonic mean of precision and recall, by aggregating the amount of true positives, false positives, and false negatives. The amount of true positives, false positives, and false negatives is calculated by comparing the list of identified errors with the reference list. An error derived

by PMVL is considered as a true positive, if the error is also marked in the reference list (i. e., there is an error in the reference list for the same line as in the derived list). An error derived by PMVL is considered as a false positive, if the error is not marked in the reference list (i. e., there is no error in the reference list for an error found by PMVL). An error not derived by PMVL is considered a false negative, if the error is marked in the reference list (i. e., there is an error in the reference list with no corresponding error in the derived list).

After calculating the numbers for true positives (t_p), false positives (f_p), and false negatives (f_n), precision and recall are calculated as follows: $precision = \frac{t_p}{t_p + f_p}$ and $recall = \frac{t_p}{t_p + f_n}$. F_1 score is calculated as the harmonic mean of precision and recall: $F_1 = 2\,\frac{precision \times recall}{precision + recall}$.

5.2 Case Studies

This section gives an overview of the three case studies used to evaluate PMVL.

Filling Line: For evaluation and demonstration purposes, the filling line was set up in the Karlsruhe University of Applied Sciences laboratory. It has nine pump stations and a carriage that transports a cup to them. The carriage and the pump stations are considered separate systems. There is a pump and a scale to weigh the bottle from which the liquid is pumped. A stepper motor moves the carriage horizontally and a scale holds the cup. The Workflow Management System (WfMS) sends Message Queuing Telemetry Transport (MQTT) messages to control the filling line. The pumps' positions must be measured in motor steps before the carriage can be utilised. The carriage is moved to the start point, then two liquids are filled at distinct stations, and finally the carriage is moved to the end place.

Assembly Line: The second case study comprises a six-station assembly line connected by a conveyor belt. The assembly line handles an aluminium piece that will be cut by a mill and dot peened by a dot peen marking machine. The dot peen marking machine engraves an Data Matrix code (DMC), allowing traceability of the manufacturing process. The workpiece also features a hole for a steel ball to be pressed into. The hole is measured either mechanically by inserting a cone or visually using a camera. A successful manufacturing process requires four criteria: 1. The engraving is cut into the workpiece. 2. The DMC is engraved. 3. A steel ball is forced into the workpiece's hole. 4. The finished object is put into the tray. While the engravings are independent of the other procedures, the steel ball may only be pressed in once the hole diameter has been confirmed. Moreover, the steps can be performed in any order as long as the skid on the conveyor belt, the mill, and the dot peen marking machine can fit a workpiece that has already been joined with the steel ball. As a result, certain production steps can be rescheduled to make the process more resilient while still meeting the criteria mentioned above.

Electrochemical Machining Line: The Electrochemical Machining (ECM) case study is based on a production line of our industrial partner SITEC Industrietechnologie GmbH. In

terms of non-disclosure agreements, details about the exact functionality of the stations are changed, but the overall process stays the same. The ECM line handles one workpiece at a time in three stations. The workpiece is sealed at the start of the process. At the line's entry point, there must be at least one workpiece. A conveyor belt transports the workpiece to the first station to remove the sealing. The workpiece is then conveyed to the second station. Station 2 takes workpieces from the conveyor belts and performs ECM. The workpiece is then conveyed to the third station to re-seal the workpiece and transports it out. To finish the production process, the workpiece must be machined and resealed.

5.3 Evaluation Results

In Tab. 1 and Tab. 2, we present the evaluation results for each process model. The evaluation data can be found in our supplementary material [Ko22]. In Tab. 1, the numbers of true positives (t_p), false negatives (f_n), and false positives (f_p) are shown for each plant. The table includes precision, recall, and the F_1 metric. The table separates the results for inconsistent preconditions, inconsistent postconditions, conditions already met, branch inconsistency, and cycle detection. The results of using PMVL are compared to the experts' estimation.

	Filling Station					**Assembly Line**					**Sealing Line**				
	Pr.	**Po.**	**Met.**	**Br.**	**Cy.**	**Pr.**	**Po.**	**Met.**	**Br.**	**Cy.**	**Pr.**	**Po.**	**Met.**	**Br.**	**Cy.**
t_p	4	5	3	1	2	4	2	3	1	1	4	4	3	2	1
f_n	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
f_p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tab. 1: Results for the individual case studies (Pr.: preconditions, Post.: postconditions, Met.: conditions already met, Br.: branch, Cy.: Cycle).

The results for the filling line case study show that all pre- and postcondition errors, already-met-condition-errors, branching-errors, and cycling-errors identified by the domain experts were also correctly identified by PMVL (t_p). No additional errors were identified by PMVL (f_p). None of the errors from the experts' list were missing (f_n).

The results for the assembly line case study show that all pre- and postcondition errors, already-met-condition-errors, branching-errors, and cycling-errors identified by the domain experts were also correctly identified by PMVL (t_p). No additional errors were identified by PMVL (f_p). None of the errors from the experts' list were missing (f_n).

The results for the ECM line case study show that all postcondition errors, already-met-condition-errors, and cycling-errors identified by the domain experts were also correctly identified by PMVL (t_p). No additional errors were identified by PMVL (f_p). Two of the errors the experts identified were missing (f_n). First, one precondition error required a time constraint, which the PMVL is currently not able to model, and one branching error was masked by a second branching error. When the second branching error was fixed, the second

error was identified by PMVL, as well. Although the first error could be identified after a fix, we still consider it as a f_n.

	Prec.	**Postc.**	**Met con.**	**Branch**	**Cycle**	**Total**
t_p	12	11	9	4	4	**40**
f_n	1	0	0	1	0	**2**
f_p	0	0	0	0	0	**0**
Precision	1.0	1.0	1.0	1.0	1.0	**1.0**
Recall	0.92	1.0	1.0	0.8	1.0	**0.95**
F_1	0.96	1.0	1.0	0.89	1.0	**0.98**

Tab. 2: Evaluation results for the case studies combined (Prec.: preconditions, Post.: postconditions, Met con.: conditions already met).

In Tab. 2, the precision, recall, and F-measure are depicted. The results show that PMVL does only identify errors that are also present in the reference list (Precision). When modelling and validating production processes, we do not highlight problems that are not true errors, so we do not mislead the user. However, we are unable to identify all of the errors in the reference list (Recall).

5.4 Discussion and Limitations

The results of the evaluation show that all postcondition errors, already-met-condition-errors, and cycle errors have been identified correctly by PMVL. The missing precondition error could not be modelled with PMVL. We have to extend the language in order to be able to add runtime constraints and thereby model such errors. The missing branching error could be identified after fixing the error, masking the missing error. The user of PMVL must be aware that after fixing the identified errors, more errors could occur. Overall, PMVL shows promising results regarding validating reconfigured production processes. Regarding the requirement **R5**, we come to the conclusion that in general, PMVL is able to model and validate reconfigured production processes, but needs to be extended to model and validate time-constrained production processes.

6 Conclusion and Future Work

In this paper, we introduced PMVL, a language for the modelling and validation of production processes in the context of IoT and CPPS. It allows domain experts to specify sub-processes with conditions that can be used by non-domain experts. The design process of PMVL is driven by the input of industry partners with the need to model and validate production processes. The evaluation of PMVL demonstrates that the DSML can be used to specify real-word production processes and identify errors such as inconsistent pre- and postconditions, branching errors or cycles. Future work includes the application of PMVL to further disciplines and augmenting PMVL with additional features like modelling production processes with time constraints and to include more complex production processes.

References

[Al03] Aleem, H.; Zhao, Y.; Lord, S.; McCarthy, T.; Sharratt, P.: Pharmaceutical process validation: an overview. Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering 217/2, pp. 141–151, 2003.

[Ba94] Basili, V. R. et al.: The Goal Question Metric Approach. In: Encyclopedia of Software Engineering. Wiley, 2:528–532, 1994.

[BPS10] Boley, H.; Paschke, A.; Shafiq, O.: RuleML 1.0: the overarching specification of web rules. In: International Workshop on Rules and Rule Markup Languages for the Semantic Web. Springer, pp. 162–178, 2010.

[EST15] Estañol, M.; Sancho, M.-R.; Teniente, E.: Verification and Validation of UML Artifact-Centric Business Process Models. In (Zdravkovic, J.; Kirikova, M.; Johannesson, P., eds.): Advanced Information Systems Engineering. Springer International Publishing, Cham, pp. 434–449, 2015, ISBN: 978-3-319-19069-3.

[FN71] Fikes, R. E.; Nilsson, N. J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence 2/3-4, pp. 189–208, 1971.

[GOR10a] Grambow, G.; Oberhauser, R.; Reichert, M.: Employing semantically driven adaptation for amalgamating software quality assurance with process management. In: Second International Conference on Adaptive and Self-Adaptive Systems and Applications 2010 (ADAPTIVE 2010). Pp. 58–67, 2010.

[GOR10b] Grambow, G.; Oberhauser, R.; Reichert, M.: Semantic workflow adaption in support of workflow diversity. In: The Fourth International Conference on Advances in Semantic Processing (SEMAPRO 2010). Pp. 158–165, 2010.

[GOR11] Grambow, G.; Oberhauser, R.; Reichert, M.: Semantically-driven workflow generation using declarative modeling for processes in software engineering. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops. IEEE, pp. 164–173, 2011.

[HSR19] Heinrich, R.; Strittmatter, M.; Reussner, R. H.: A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis. IEEE Transactions on Software Engineering/, 2019, ISSN: 0098-5589.

[Ih21] Ihlenfeldt, S.; Wunderlich, T.; Süße, M.; Hellmich, A.; Schenke, C.-C.; Wenzel, K.; Mater, S.: Increasing Resilience of Production Systems by Integrated Design. Applied Sciences 11/18, p. 8457, 2021.

[Ko22] Koch, S.: A Cross-Disciplinary Process Modelling Language for Validating Reconfigured Production Processes - Supplementary Material, version v1.0, Mar. 2022, URL: https://doi.org/10.5281/zenodo.6334307.

[Kü10] Kühne, S.; Kern, H.; Gruhn, V.; Laue, R.: Business process modeling with continuous validation. Journal of Software Maintenance and Evolution: Research and Practice 22/6-7, pp. 547–566, 2010.

[Ly12] Ly, L. T.; Rinderle-Ma, S.; Göser, K.; Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems. Information Systems Frontiers 14/2, pp. 195–219, 2012.

[ML17] Marrella, A.; Lespérance, Y.: A planning approach to the automated synthesis of template-based process models. Service Oriented Computing and Applications 11/4, pp. 367–392, 2017.

[PSA07] Pesic, M.; Schonenberg, H.; van der Aalst, W. M.: DECLARE: Full Support for Loosely-Structured Processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). Pp. 287–287, 2007.

[Ro17] Rostami, K.; Heinrich, R.; Busch, A.; Reussner, R.: Architecture-based Change Impact Analysis in Information Systems and Business Processes. In: 2017 IEEE International Conference on Software Architecture (ICSA2017). IEEE, pp. 179–188, 2017, ISBN: 978-1-5090-5729-0, URL: https://doi.org/10.1109/ICSA.2017.17.

[Va99] Van der Aalst, W. M.: Formalization and verification of event-driven process chains. Information and Software technology 41/10, pp. 639–650, 1999.

[WGH08] Weber, I.; Governatori, G.; Hoffmann, J.: Approximate compliance checking for annotated process models. In: 1st International Workshop on Governance, Risk and Compliance-Applications in Information Systems (GRCIS'08). 2008.

[WHM08] Weber, I.; Hoffmann, J.; Mendling, J.: Semantic business process validation. In: Proceedings of the 3rd International Workshop on Semantic Business Process Management (SBPM'08), CEUR-WS Proceedings. Vol. 472, Citeseer, 2008.