Einsatz von Graphdatenbanken für das Produktdatenmanagement im Kontext von Industrie 4.0

Christopher Sauer, Benjamin Schleich und Sandro Wartzack

Einleitung und Motivation

Im Zuge der digitalen Transformation im Kontext von Industrie 4.0 tun sich eine Vielzahl neuer Datenquellen auf, die im Produktdatenmanagement berücksichtigt werden müssen. Ein Beispiel neuer Datenquellen sind Daten der Industrie 4.0, die zum Beispiel über Sensoren in der Fertigung erhoben werden. Kennzeichen dieser Datenquellen sind die zunehmende Heterogenität der Daten, die nicht mehr in einer Tabelle erfasst werden können. So könnten dies unter anderem Bilder einer optischen Bauteilprüfung sein oder Code zur Bauteilprüfung. Dieser Umstand führt zum Aufbau vieler einzelner neuer Silos, in denen die Daten separat und getrennt vom PDM-System verarbeitet werden müssen. Zudem werden dort abgeschottet von den restlichen Silos Daten gespeichert. Daneben führt eine Vielzahl neuer Autorensysteme (Prüfsoftware, Kundenmanagement, Anforderungsmanagement) zu einer gesteigerten Datenmenge, die nicht mehr in klassischen tabellenbasierten und rein-relationalen Datenbanksystemen sinnvoll erfasst werden können. Um an Informationen zu gelangen, sind im Fall rein-relationaler Datenbanksysteme oft komplizierte Abfragen nötig. Diese greifen dann auf mehrere unterschiedliche Tabellen innerhalb der Datenbank zu und stellen daraus wiederum relevante Informationen bereit. Je mehr größer jedoch diese Datenbanken werden und je mehr Informationen miteinander relational verbunden werden müssen, desto mehr Expertenwissen über das jeweilige Datenbanksystem wird benötigt. Somit büßen rein-relationale (SQL-basierte) Systeme auch einen Großteil der Vorteile ihres logischen strukturellen Aufbaus ein. Um den oben genannten Problemen zu begegnen, können neue Ansätze aus dem Bereich der Linked Data herangezogen werden. Bei

Linked Data werden nicht nur die reinen Daten verwendet, sondern auch beschreibende und verknüpfende Informationen um die Daten zu interpretieren verwendet und weitergegeben. Durch diesen Mehrwert an Information wird es in einem ersten Schritt möglich, heterogene Produkt- und Prozessdaten, also Daten aus verschiedensten Quellen, wie zum Beispiel Konstruktion, Simulation und Qualitätssicherung, miteinander zu verknüpfen. Durch diese Verknüpfung kann eine höherwertige Darstellungsform geschaffen werden, die neben den reinen Daten auch die sinnvolle Verknüpfung enthält und so eine semantisch höherwertige Repräsentation darstellt. Die so entstehende, vernetzte Datenbank kann z.B. über eine graphenorientierte Datenbank oder Graphdatenbank implementiert werden.

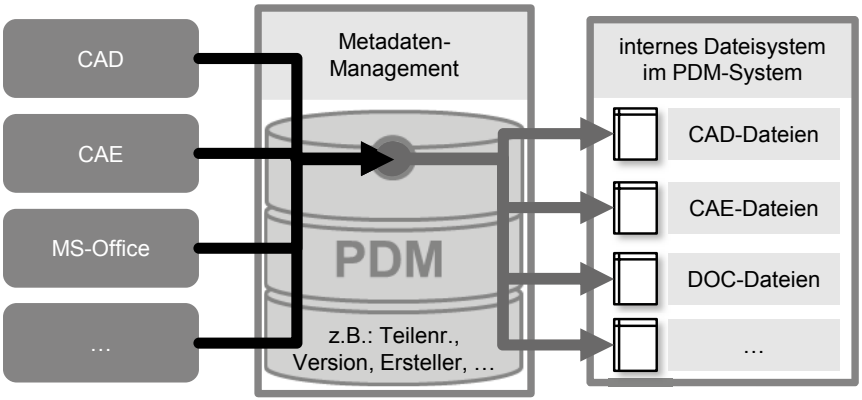
Im vorliegenden Beitrag wird untersucht, inwieweit die Modellierung mit gegenwärtig existierenden Lösungen für Graphdatenbanken möglich ist. Ausgehend von einem Beispiel mit einem vereinfachten Produkt- und Prozessdatenmodell der Blechmassivumformung, wird eine allgemeine Methode vorgestellt, durch die ein SQL-basiertes Datenbanksystem in eine Graphdatenbank überführt werden kann. Anhand dieser Methode wird dargestellt, wie bestehende Lösungen teilweise auch parallel zu neuartigen Linked Data Datenbanken existieren können, um diese Schritt für Schritt in eine Graphdatenbank zu überführen. Die Ergebnisse des Beitrags sind auf der einen Seite das allgemeine Vorgehensmodell zur Einführung von Graphdatenbanken und auf der anderen Seite Aussagen über die Nutzbarkeit der vorgestellten Lösung für das Produkt- & Prozessdatenmanagement.

Grundlagen und Stand der Technik

1.1 Produktdatenmanagement

Gegenwärtige Produktdatenmanagement-Systeme (PDM) verwalten die Produktstruktur und deren zugrundeliegende Daten hierarchisch und statisch (Conrad et al. 2007). Dabei steht das Projekt an oberster Stelle der Struktur, gefolgt von Produkten, die sich dann jeweils in einzelne Baugruppen gliedern. Mit dieser Struktur lassen sich vor allem CAD-Daten sehr gut verwalten, da CAD-Modellbäume ähnlich hierarchisch und statisch aufgebaut sind, wie aktuelle PDM-Systeme. Hier ist schon erkennbar, dass für die Integration von weiteren Daten, neue Funktionen im PDM-System notwendig sind, um zum

Beispiel einen anderen strukturellen Aufbau zu unterstützen als die oben genannten Hierarchien. Dabei ist es zum Beispiel möglich, CAE-Daten mit der zugehörigen CAD-Geometrie zu verknüpfen. Die CAE-Eingangsdaten werden dabei semantisch an die CAD-Geometrie gekoppelt. Darüber hinaus stellen nach Eigner & Stelzer (2009) PDM-Systeme unter anderem folgende Funktionalitäten bereit: Datenstrukturmanagement, Versionsverwaltung, Projektmanagement, Arbeitsablauf- und Prozessverwaltung. Gerade die Integration von Arbeitsabläufen und Prozessen ermöglicht neben dem reinen Produktdatenmanagement auch das Management von unternehmensinternen Prozessen. Dabei stehen vor allem Geschäftsprozesse, wie ein Zeichnungsfreigabeprozess oder ein Angebotserstellungsprozess im Fokus. Grundsätzlich werden durch PDM-Systeme alle Informationen, die im Produktentstehungsprozess anfallen, transparent und nachvollziehbar abgelegt und miteinander verknüpft (Vajna et al. 2018). Gegenwärtig erfolgt diese Verknüpfung über die Definition von Metadaten, siehe Abbildung 1. Zu verknüpfende Informationen können dabei zum Beispiel CAD-Daten, Anforderungslisten oder Simulationsdaten sein. Die zur Verknüpfung verwendeten Metadaten sind vor allem Daten, wie etwa die Teilenummer, der Ersteller oder die aktuelle Versionsnummer des Bauteils. Die gesamte Datenverwaltung wird dabei meist über ein SQL-basiertes relationales Datenbankmanagementsystem realisiert.



2.2 Datenbanken

Die Grundlage aktueller Produktdatenmanagement-Systeme bilden in der heutigen Zeit meist relationale Datenbanksysteme von großen Herstellern wie Microsoft, Oracle, usw. (Vajna et al. 2018). In relationalen Datenbanksystemen werden die einzelnen Tupel in Tabellenform gespeichert. Ein Tupel kann dabei wie ein Datensatz betrachtet werden. Darüber hinaus werden die einzelnen Tabellen dann über Schlüssel miteinander verknüpft. Die eindeutige Zuordnung zwischen den einzelnen Elementen der Tabellen findet dann mithilfe dieser Schlüssel statt. Innerhalb dieser Tabellen werden später Abfragen oder Transaktionen erstellt und durchgeführt. Dies geschieht anhand sogenannter Metadaten, welche die Informationen über die gespeicherten Daten beinhalten. Neben den Abfragen an eine Datenbank, bei denen gezielt Datensätze aus der Datenbank abgefragt werden können, sind Transaktionen eine weitere wichtige Funktion. Unter einer Transaktion werden verschiedene Interaktionen wie Speichern, Löschen oder Verändern von Datensätzen verstanden (Wiese 2015). Nach Wiese (2015) erstreckt sich eine Transaktion immer von einem konsistenten Zustand zum nächsten. Somit sind alle Bedingungen einer ACID gemäßen Datenbank nach Maier & Kaufmann (2016) gegeben.

- Atomarität (Atomicity) – eine Transaktion wird entweder komplett durchgeführt oder überhaupt nicht
- Konsistenz (Consistency) – nach Ende einer Transaktion müssen alle vordefinierten Konsistenzbedingungen auch wieder erfüllt sein
- Isolation (Isolation) – gleichzeitig ablaufende Transaktionen führen zu denselben Resultaten wie in einer Einbenutzerumgebung
- Dauerhaftigkeit (Durability) – Datenbankzustände können nur durch Transaktionen verändert werden

Für relationale Datenbanken gelten diese vier Bedingungen immer. Aufgrund ihrer hohen Verbreitung sind relationale Datenbanken immer noch der quasi-Standard im Einsatz befindlicher Datenbanksysteme.

2.3 Graphenorientierte Datenbanken oder Graphdatenbanken

Graphdatenbanken wurden zusammen mit anderen neuartigen und postrelationalen Datenbanken aufgrund folgender Schwächen relationaler Datenbanken entwickelt (Wiese 2015).

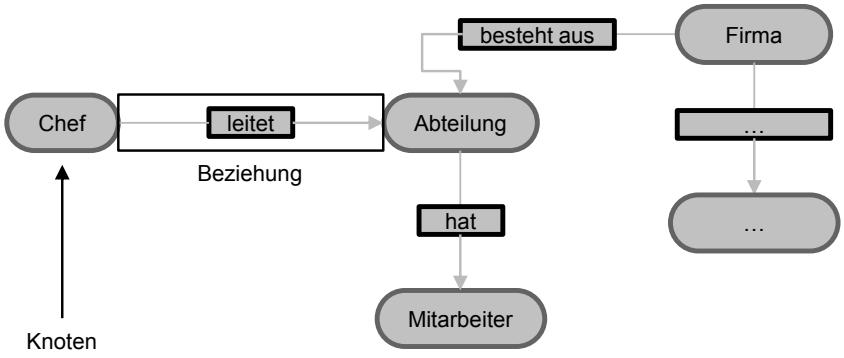
- Inadäquate Repräsentation von beliebigen Daten – die strikte Konvention Daten nur tabellarisch zu erfassen, ist nicht für jeden Anwendungstyp geeignet.
- Semantische Überbeschreibung – relationale Datenbanken müssen sowohl die Instanzen als auch deren Beziehungen tabellarisch erfassen, so entstehen unnötig viele Tabellen zur Beschreibung der direkten Beziehungen mit Hilfe von Beziehungstabellen und Schlüsseln
- Schwache Unterstützung für rekursive Anfragen – Datenbankanfragen, die verschiedene Datensätze rekursiv zueinander in Beziehung setzen müssen, benötigen relativ lange für diese Aufgabe
- Zwang zur Homogenität – Datensätze müssen immer alle Attribute der Tabelle belegen

Zudem erkennt Maier & Kaufmann (2016), dass der Aufwand zur Ermittlung von referenzierten Tupel in relationalen Datenbanken mit der Anzahl der Tupel wächst, dies ist für andere Datenbanktypen wie Graphdatenbanken nicht der Fall, da bei diesen die direkten Nachbarn immer abgefragt werden können und nicht vorher alle Relationen durchlaufen werden müssen. Die oben aufgeführten Schwächen führten in den letzten Jahren unter anderem zur Entwicklung von NoSQL-Datenbanken und speziell Graphdatenbanken. Dabei werden Daten innerhalb einer mathematischen Struktur, dem namensgebenden Graphen abgelegt. Dies bringt nach Angles & Gutierrez (2008) unter anderem folgende Vorteile mit sich.

- Die Daten werden natürlich modelliert. Dieser Umstand bezieht sich auf die einfache Formulierung von Graphen, so kann ein Datenbankmodell auch von auf diesem Gebiet Produktentwicklern, ohne vertieftes PDM Wissen schnell an einem Whiteboard entworfen und später umgesetzt werden.
- Bei Abfragen kann direkt auf die Datenstruktur zugegriffen werden, auch müssen keine Relationen durchlaufen werden

— Es sind effiziente Graph-Algorithmen aus dem Gebiet der mathematischen Graphentheorie verfügbar. Zum Beispiel zur Suche der kürzesten Verbindung zwischen zwei Knoten

Grundsätzlich speichert eine Graphendatenbank Informationen und Daten anhand von Knoten und Kanten wie in Abbildung 2 zu erkennen ist.



Im Beispielmodell aus Abbildung 2 lässt sich die natürliche Modellierung besonders gut erkennen. So ist diese als recht intuitiv zu betrachten, in einer relationalen Datenbank müssten die einzelnen Datensätze über Beziehungstabellen und Schlüssel miteinander verknüpft werden, in einer Graphdatenbank erfolgt diese Modellierung direkt über die Kanten des zugrundeliegenden Graphens.

3 Vorgehensmodell zur Einführung von Graphdatenbanken

3.1 Problemstellung

Für das in diesem Beitrag vorgestellte Vorgehensmodell wurde ein Anwendungsfall aus dem Bereich der Blechmassivumformung gewählt. Bei der Blechmassivumformung handelt es sich um ein neuartiges Fertigungsverfahren, das die Vorteile der Blechumformung mit der Massivumformung vereint (Hetzner et al. 2011). Für die Blechmassivumformung wurde zur Erfassung aller relevanten Fertigungs-, Produkt- und Prozessdaten ein Datenbankmodell für die Ablage entwickelt. Innerhalb der Beiträge von Breitsprecher et al.

(2011 und 2014) ist dabei ein rein-relationales Produkt- und Prozessdatenmodell (PPDM) entstanden. Ein Ausschnitt des Datenbankschemas, also der zugrundeliegenden Formulierung ist in Abbildung 3 zu sehen. Umgesetzt wurde diese Datenbank mit der Datenbanksoftware Sqlite, die die Funktionalität einer relationalen Datenbank wie in Abschnitt 2.2 beschrieben, implementiert.

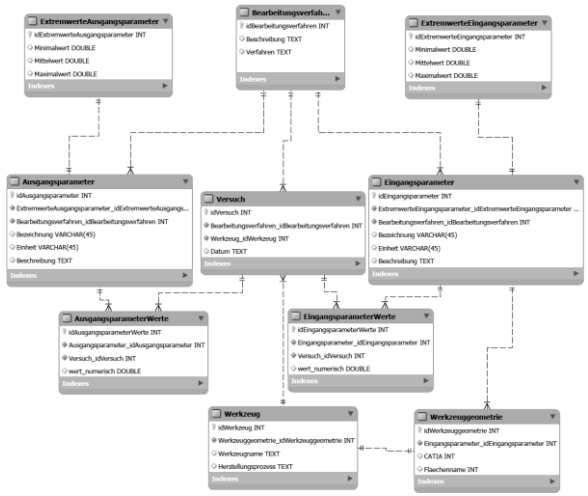


Abbildung 3: Schema des Produkt- und Prozessdatenmodells für die BMU (Breitsprecher 2014)

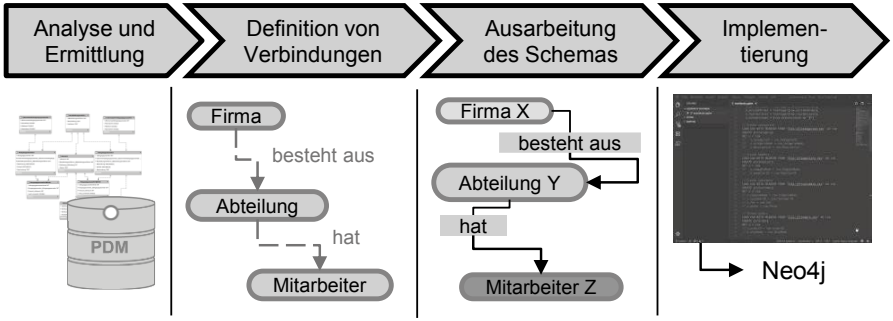
Aufgrund der Erkenntnis, Daten nicht mehr rein-relational erfassen zu können, also zum Beispiel Bilddateien oder ortsaufgelöste Metamodelle (Sauer 2018), benötigt dieses Datenbankmodell jedoch eine Erneuerung. Die zu erfassenden Daten sind nicht mehr im klassischen Schema der relationalen Datenbank ablegbar und über die zahlreichen Referenz-IDs in Beziehungstabellen (Schlüssel) immer unübersichtlicher geworden. Zudem wird ein Einfügen neuer, bisher unbekannter Datenquellen immer schwieriger, da das bestehende Schema bei jeder Ergänzung neu migriert werden muss. In einem ersten Schritt soll in diesem Beitrag eine Eignung von Graphdatenbanken überprüft werden. Der vorliegende Anwendungsfall soll sich aber nicht nur auf die Blechmassivumformung alleine beschränken, sondern eine allgemeine Vorgehensweise zur Einführung oder Übertragung von relationalen Datenbankmodellen in eine Graphdatenbank darstellen. Graphdatenbanken kommen aufgrund der in Abschnitt 2.3 beschriebenen Vorteile zum Einsatz,

da eine möglichst flexible und natürliche Formulierung der Datenstruktur für den Erfolg als essentiell erachtet wird.

3.2 Vorgehen

Grundsätzlich lässt sich das Vorgehen in vier Schritte unterteilen. Abbildung 4 dient zur Verdeutlichung dieser Abfolge:

- Schritt 1: Analyse und Ermittlung: Zu erfassenden Daten und Datenquellen erkennen und Tabellen extrahieren
- Schritt 2: Definition von Verbindungen: Relationen innerhalb der in Schritt 1 erfassten Daten und Datenquellen herausstellen
- Schritt 3: Ausarbeitung des Schemas: Erfassung zugrundeliegender Graphdatenbankstruktur und Anlegen erster Knoten
- Schritt 4: Implementierung: Anlegen der kompletten Graphdatenbank innerhalb einer Graphdatenbanksoftware (hier: Neo4j)



Für eine komplette Neueinführung von Graphdatenbanken werden dabei alle vier Schritte durchlaufen. Handelt es sich lediglich um eine Überführung einer bestehenden relationalen Datenbank in eine graphenorientierte können gegebenenfalls die Schritte eins und zwei entfallen. Für die Überführung lassen sich zwei hilfreiche Überführungsregeln erkennen: Eine Reihe oder ein Datensatz einer relationalen Datenbank ist immer ein Knoten in der resultierenden Graphdatenbank. Ein Tabellenname einer relationalen Datenbank ist meist eine Bezeichnung für eine Kante innerhalb einer graphenorientierte Datenbank. Es lohnt sich auch in der Überführung nochmals grundlegend zu analysieren ob die beiden obigen Regeln zum Einsatz kommen können.

Ausgehend vom Einführungsfall werden im ersten Schritt alle zugrundeliegenden Daten erfasst. Im Fall der Blechmassivumformung sind es vor allem die Folgenden:

- Werkzeugkonzept (Wz.-Konzept) – welches Werkzeug wird für die Herstellung verwendet. Darunter lassen sich folgende Informationen erfassen: Name des Werkzeugs, Beschreibung des Werkzeugs, Prozessfolge, Schrittzahl (Wie viele Arbeitsschritte werden auf dem Werkzeug ausgeführt)
- Halbzeugkonzept (Hz.-Konzept) – welches Halbzeug wird für die Herstellung eines Bauteils verwendet? Dies umfasst: Name des Halbzeugs und Beschreibung, eventuell auch Rückschlüsse über den Vendoren (wenn zum Beispiel eine Anbindung an ein ERP System vorhanden ist)
- Fertigungsverfahren – das zum Einsatz kommende Fertigungsverfahren, welches Werkzeug und Halbzeug zusammenbringt, um ein Bauteil zu erzeugen. Hierunter fällt: Name des Fertigungsverfahrens, Beschreibung des Fertigungsverfahrens und ggf. verfügbare Fertigungsmaschinen.
- Bauteil – das Bauteil welches gefertigt werden soll. Hierfür lassen sich der Name, eine Beschreibung, ein Versionsstand (Verknüpfung mit CAD-Dateien) und ggf. eine Konfiguration der Elemente erfassen.
- Formelemente – eine spezifische Größe für die Blechmassivumformung. Bauteile besitzen immer eines oder mehrere Formelemente. Hierunter lassen sich der Name, die Bezeichnung, die Art (Hauptformelemente entsprechen Massivteilen, Nebenformelemente entsprechen Blechteilen) und eine Parameterkonfiguration der Geometrie erfassen.
- Zusatzeigenschaften – etwaige Zusatzeigenschaften eines Bauteils, die durch spezifische Fertigungsprozess- oder Materialeigenschaften entstehen. Informationen wie Name und Beschreibung sowie Wert der Zusatzeigenschaften können hierunter erfasst werden.
- Versuch – werden Bauteile zum ersten Mal gefertigt, benötigen diese einen Versuch, der die Bauteileigenschaften validiert. Dabei

kann der Versuch sowohl experimentell als auch numerisch/simulativ durchgeführt werden. Im Versuch werden Zielgrößen betrachtet, basierend auf den Eingangsparametern, die meist durch Geometriegrößen ausgedrückt werden.

- Versuchsdaten – entstehen aufgrund eines Versuchs und enthalten für die vorher definierten Zielgrößen gefundene Werte, Parameternamen und weitere Ergebnisse.
- Metamodelle – Anhand von Versuchsdaten werden Vorhersagemodelle trainiert, die auf Basis bekannter Versuchspunkte bisher unbekannte neue Versuchspunkte vorhersagen können, dies umfasst auch die oben erwähnten lokalen Metamodelle. Diese enthalten ihren Namen, Beschreibung, einen Pfad innerhalb einer Datenablage, ein Qualitätskriterium zur Beschreibung der Vorhersagegüte und die eigentliche erzielte Vorhersagegüte nach dem Aufbau des Metamodells.

Ist die Analyse der zu erfassenden Datenquellen abgeschlossen, werden im nächsten Schritt Relationen und Beziehungen zwischen den Daten beschrieben. Hierzu wird grundlegend versucht, die Erkenntnisse aus der unstrukturierten Analyse weiter zu strukturieren. Im vorliegenden Fall wurden unter anderem folgende Beziehungen identifiziert.

- Werkzeug- und Halbzeugkonzept beeinflussen das Fertigungsverfahren, bzw. definieren auch die Anzahl und Art der zur Verfügung stehenden Fertigungsverfahren
- Bauteile besitzen Formelemente und Zusatzeigenschaften
- Fertigungsverfahren wirken sich auf den Versuch aus
- Formelemente und Zusatzeigenschaften der Bauteile bedingen die im Versuch zu erhebenden Größen
- Versuche liefern Versuchsdaten
- Versuchsdaten erzeugen Metamodelle

Es ist zu erkennen, dass schon anhand der wörtlichen Formulierung erste Beziehungen zwischen den Größen erkennbar sind.

In Schritt drei sollen diese nun in eine grafische Form überführt werden, dies kann zum Beispiel über Diagramme geschehen. Das aus den bisherigen Erkenntnissen gewonnene Diagramm ist in Abbildung 5 zu erkennen.

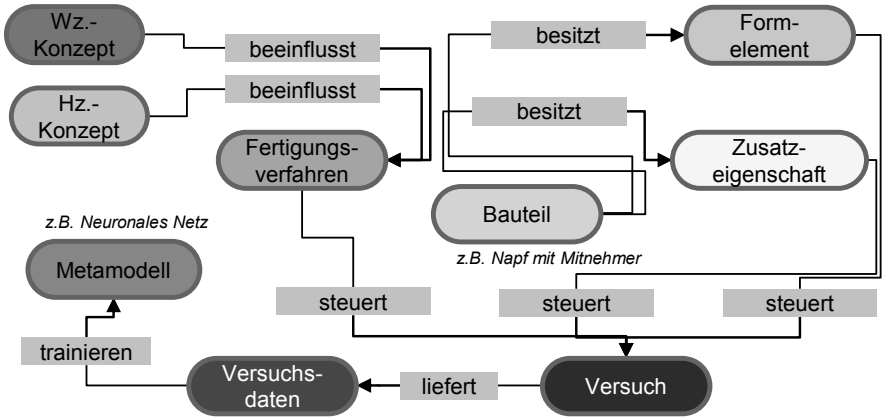


Abbildung 5: Schema der Graphdatenbank

Anhand des in Abbildung 5 dargestellten Schemas kann nun in Schritt vier in die Implementierung der Graphdatenbank übergegangen werden. Im vorliegenden Beitrag ist dies mit der Graphdatenbanksoftware Neo4j durchgeführt worden. Ist Graphdatenbank erzeugt und mit Datensätzen befüllt worden, lohnt ein Blick auf die Abfragemöglichkeiten der zugrundeliegenden Daten.

3.3 Abfragen

Um eine Abfrage wie in Abbildung 6 zu erzeugen, bei der gezielt auf die mit einem Versuch zusammenhängenden Knoten eingegangen wurde, werden Abfragesprachen benötigt. Neo4j unterstützt dabei die quasi Standardsprache GraphQL (Graph Query Language): GraphQL macht es dabei möglich, die Abfragen ebenfalls in einer graphenartiger Struktur zu verfassen, dadurch bleibt die Abfrage innerhalb der grapheigenen Syntax (He & Singh 2010). Eine beispielhafte Abfrage, um die Versuchsbeschreibung für Versuchs #1 zu erhalten, könnte in GraphQL lauten:

```
{
  "Versuch#1"{
    Versuchsbeschreibung
  }
}
```

Die Abfrage würde dann die entsprechende Versuchsbeschreibung zu Versuch#1 zurückgeben und dem Benutzer oder der Anwendung zur Verfügung stellen. Das in diesem Beitrag eingesetzte Neo4j unterstützt neben GraphQL auch die Abfragesprache Cypher, diese wurde extra für Graphdatenbanken innerhalb von Neo4j entwickelt (Wiese 2015). Cypher orientiert sich dabei eher am klassischen SQL, so verwendet es eine ähnliche Syntax. Sowohl mit GraphQL als auch mit Cypher lassen sich ähnliche Abfragen realisieren, übersetzt man die abgefragten Daten in ihre bildliche Form, entsteht zum Beispiel eine Struktur wie in Abbildung 6.

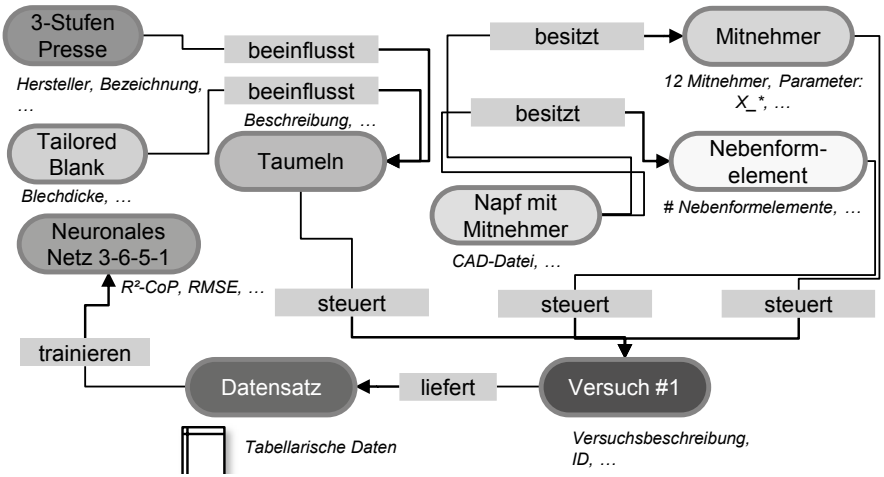


Abbildung 6: Resultierender Graph (durch Abfrage eines Versuchsknotens „Versuch#1" und aller verknüpften weiteren Knoten erzeugt)

Grundsätzlich lässt sich festhalten, dass die Abfragen ähnlich flexibel sind wie mit bestehenden SQL-basierten Systemen. Besonderes Augenmerk gilt dabei der Sprache GraphQL, da sie aufbauend auf der grapheigenen Syntax Abfragen ermöglicht, so entfallen Schlüsselwörter und Sprachkonstrukte wie sie SQL und Cypher vorsehen.

4 Bewertung und Diskussion

4.1 Bewertung der Nutzbarkeit

Im Rahmen dieses Beitrags hat sich herausgestellt, dass Graphdatenbanken neue Wege in der Entwicklung von Produkt- und Prozessdatenmodellen aufzeigen können. Dabei können schnell und unkompliziert komplette Datenbanken implementiert werden und auch eine Änderung und Anpassung an neue Gegebenheiten wird ermöglicht. Gerade im Kontext von Industrie 4.0, unter welchem viele neue Datenquellen an ein bestehendes System angebunden werden wollen, ist die Verwendung von Graphdatenbanken nötig. Ein Vorteil von Graphdatenbanken ist ihre Unterstützung für neue Datentypen, wie der angesprochenen lokalen Metamodelle, diese basieren auf neuronalen Netzen, welche mit Hilfe ihrer Struktur und ihrer Gewichte innerhalb der Graphdatenbank erfasst werden können. Dabei wird ein direkter Zugriff ermöglicht. Darüber hinaus wird es den Produkt- und Prozessentwicklern aufgrund der weniger aufwendigen Struktur der Datenbank möglich, ohne vertiefte Kenntnisse in Datenbanktechnik neue Modelle zu entwickeln. Die „Whiteboard"-freundliche Formulierung von Graphdatenbanken vereinfacht zudem Kommunikation und Austausch der Mitarbeiter und PDM-Verantwortlichen über das zugrundeliegende Datenbankmodell.

4.2 Integration in bestehende Umgebungen

Wie anfangs erwähnt verwenden bestehenden PDM-Systeme rein-relationale SQL-basierte Datenbanken. Dieser Umstand wird sich aufgrund der bewährten Technologie auch in der kommenden Zeit nur langsam ändern. Für neuartige Systeme und Produkt- und Prozessdatenmodelle lohnt jedoch ein Blick auf die Graphdatenbanken. So ist es durchaus denkbar, dass sich bestehende PDM-Systeme über Graphdatenbanken erweitern lassen, um zum Beispiel die angesprochene Flexibilität in der Anbindung neuer Datenquellen zu ermöglichen.

5 Fazit und Ausblick

Zusammenfassend ist zu sagen, dass sich Graphdatenbanken für den Einsatz zur Produkt- und Prozessdatenverwaltung grundsätzlich eignen. Der in die-

ser Arbeit vorgestellte Ansatz dient für einen ersten Versuch ein SQL-basiertes Modell in eine Graphdatenbank zu überführen. Die Stärke von SQL-basierten Datenbanken, die in den Grundlagen vorgestellten ACID-Transaktionen, werden aus Sicht der Autoren eher bedingt im PDM-Bereich benötigt. Wohlüberlegte Graphdatenbanken sind aufgrund ihrer Flexibilität und Anpassbarkeit für eine Vielzahl von Verwendungsmöglichkeiten geschaffen. Hierfür sind Graphdatenbanken prädestiniert, da diese in Anlehnung zu den Programmiersprachen eine Formulierung angelehnt an eine Hochsprache erlauben. SQL-basierte Modelle zeichnen sich demgegenüber durch einen höheren Programmier- und Instandhaltungsaufwand und eine geringe Flexibilität aus. Für die Zukunft sollte das hier vorgestellte Modell weiter auf seine Eignung im produktiven PDM-Einsatz untersucht und ausgebaut werden. PDM-Systeme werden auf lange Sicht nicht ohne den Einsatz oder der teilweisen Unterstützung von graphbasierten Datenbankmodellen auskommen, da vor allem eine Unterstützung der oben genannten Datenquellen unter dem Kontext Industrie 4.0 immer wichtiger wird. Jedoch wird auch die Kompatibilität zu SQL-basierten Modellen nicht einfach wegfallen können, so dass in Zukunft wahrscheinlich beide Systeme parallel existieren und eine Interoperabilität immer wichtiger wird.

Danksagung

Die Autoren danken der Deutschen Forschungsgemeinschaft (DFG) für die Förderung des Teilprojekts B1 „Entwicklung eines selbstlernenden Assistenzsystems“ im Rahmen des Sonderforschungsbereiches Transregio 73.

Literaturverzeichnis

Abramovici, M, Gerhard, D (1997): Use of PDM in Improving Design Processes – State of the Art, Potentials and User Perspectives. In: Proceedings of the 11th International Conference on Engineering Design (ICED) 1997, Tampere. Design Society.

Angles, R, Gutierrez, C (2008): Survey of graph database models. In: ACM Computing Surveys (CSUR), 40 (1). *https://doi.org/10.1145/1322432.1322433*

Breitsprecher, T, Meinel, A, Thummet, M, Wartzack, S (2014): Produkt- und Prozessdatenmodellierung im Kontext der Blechmassivumformung. In: Entwerfen Entwickeln Erleben – EEE2014 (551 – 564). Dresden: Verlag der Wissenschaften.

Breitsprecher, T, Westphal, C, Meintker, N, Wartzack, S (2011): Formalisierung und Verwaltung von Entwicklungswissen im Kontext des Integrierten Produktmodells. In: Tagungsband 22. DfX Symposium. Hamburg: TuTech Verlag. *https://www.designsociety.org/publication/32437/*

Conrad, J, Deubel, T, Köhler, C, Wanke, S, Weber, C (2007): Comparison of Knowledge Representation in PDM and by Sematic Networks. In: Proceedings of the 16th International Conference on Engineering Design (ICED) 2007, Paris. Design Society. *https://www.designsociety.org/publication/25624/*

Eigner, M, Stelzer R (2009): Product Lifecycle Management. Berlin: Springer-Verlag. *https://doi.org/10.1007/B93672*

He, H, Singh, A. K. (2010): Query language and access methods for graph databases. In: In: Aggarwal, C.C., Wang, H. (Hrsg.) Managing and Mining Graph Data, S. 125–160. Springer. *https://doi.org/10.1007/978-1-4419-6045-0_4*

Hetzner, H, Koch, J, Tremmel, S, Wartzack, S, Merklein, M (2011): Improved Sheet Bulk Metal Forming Processes by Local Adjustment of Tribological Properties. In: JMSE, 133. *https://doi.org/10.1115/1.4005313*

Meier, A, Kaufmann, M (2016): SQL- & NoSQL-Datenbanken. Berlin/Heidelberg: Springer-Verlag. *https://doi.org/10.1007/978-3-662-47664-2*

Sauer, C, Schleich, B, Wartzack, S (2018): Deep learning in sheet-bulk metal forming part design. In: Proceedings of the 15th International Design Conference (DESIGN) 2018, Dubrovnik. Design Society. *https://doi.org/10.21278/idc.2018.0147*

Vajna, S, Weber, C, Zeman, K, Hehenberger, P, Gerhard, D, Wartzack, S (2018): CAx für Ingenieure. 3. Auflage. Berlin/Heidelberg: Springer-Verlag. *https://doi.org/10.1007/978-3-662-54624-6*

Wiese, L (2015): Advanced Data Management. Berlin/Boston: Walter de Gruyter GmbH. ISBN: 978-3110441406

Kontakt

Christopher Sauer, M.Sc.
Lehrstuhl für Konstruktionstechnik
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 9
91058 Erlangen
www.mfk.tf.fau.de