ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme
Direktor: Prof. Dr. Manfred Reichert

Integration heterogener Produktdaten in verteilten, komplexen Entwicklungsprozessen

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Vorgelegt von:
Julian Tiedeken
aus Friesoythe

2018

Amtierender Dekan: Prof. Dr. Maurits Ortmanns
Gutachter: Prof. Dr. Manfred Reichert
Prof. Dr. Richard Lenz
Tag der Promotion: 17.07.2019

Inhaltsverzeichnis

Abkürzungsverzeichnis

AF	Anwendungsfall
Cor	Korrepondenz (engl. correspondence)
DMU	Digital Mock-Up
ECU	elektronisches Steuergerät (engl. electronic control unit)
EDI	Electronic Data Interchange
FDBS	Föderiertes Datenbanksystem
GPO	globale Produktontologie
Ind	Instanz (engl. individual)
LPO	lokale Produktontologie
MPO	Masterproduktontologie
OWL	Web Ontology Language
PDIL	Produktdatenintegrationsebene (engl. product data integration layer)
PDIS	Produktdatenintegrationssystem
PDMS	Produktdatenmanagementsystem
PROCEED	PROactive Consistency for E/E product Data management
SC	Schemakonzept (engl. schema concept)
SCAA	Schemakonzept-Attributaktion (engl. schema concept attribute action)
SCAR	Schemakonzept-Attributregel (engl. schema concept attribute rule)
STEP	Standard for the Exchange of Product Data
UML	Unified Modeling Language
XML	Extensible Markup Language

Kurzfassung

Moderne Produkte, wie etwa Automobile oder Maschinen, werden infolge der zunehmenden Digitalisierung komplexer. Neben mechanischen Bauteilen umfassen sie zahlreiche mechatronische, elektronische und elektrische Bauteile. Um unterschiedliche Kundenbedürfnisse, länderspezifische Charakteristiken oder gesetzliche Anforderungen bedienen zu können, muss für diese Produkte eine hohe Variabilität ermöglicht werden.

Die Produktentwicklung erfolgt üblicherweise system- und komponentenorientiert und wird mit Methoden des Concurrent Engineering realisiert. Unterschiedliche Anforderungen und Aufgaben der Produktentwickler führen zu einer autonomen, heterogenen IT-Systemlandschaft, die sowohl aus etablierten Informationssystemen, etwa Produktdatenmanagement-Systemen, aber auch aus fachbereichsspezifischen Lösungen besteht. Während zwischen den etablierten Informationssystemen häufig Austauschschnittstellen existieren, erfolgt der Abgleich von Produktdaten aus diesen Systemen mit fachbereichsspezifischen Lösungen häufig manuell oder gar nicht. Zusätzlich ist die IT-Systemlandschaft der Produktentwicklung einem ständigem Wandel unterworfen, so dass Austauschschnittstellen kontinuierlich angepasst und erweitert werden müssen.

Während die unabhängige Entwicklung von Systemen und Komponenten die Entwicklungszeit reduziert, wird es zu verschiedenen Zeitpunkten während der Produktentwicklung notwendig, die autonomen, heterogenen Produktdaten zu synchronisieren. Fehlerhafte und inkonsistente Produktdaten in späten Entwicklungsphasen führen zu erheblichen Kosten, so dass die Kontrolle der Vollständigkeit und Konsistenz von Produktdaten möglichst früh sichergestellt werden sollte, um eine hohe Produktqualität zu gewährleisten.

Gegenstand dieser Arbeit ist das PROactive Consistency for E/E product Data management (PROCEED)-Framework, dass die Integration autonomer, heterogener Produktdaten ermöglicht. PROCEED unterstützt den gesamten Lebenszyklus der Integration von Produktdaten, beginnend mit der initialen Integration, über die Steuerung und Überwachung des Integrationsprozesses sowie die Unterstützung von Schema- und Datenänderungen.

Um die strukturelle Heterogenität von Produktdaten zu überwinden, werden Informationssysteme in sog. Produktontologien abstrahiert. Die Produktontologien werden anschließend mit Hilfe von Abbildungsregeln und -aktionen in eine gemeinsame Sicht überführt. Auf Basis dieses Modells werden Qualitätsmetriken der Integration, wie z.B. die Konsistenz und Vollständigkeit definiert. Zusätzlich wird das dynamische Verhalten bei Änderungen von Schema und Daten der Produktontologien erläutert. Schließlich wir das PROCEED-Rahmenwerk prototypisch realisiert und in einer Fallstudie angewandt.

Vorwort

Die vorliegende Arbeit entstand in Zusammenarbeit zwischen dem Institut für Datenbanken und Informationssysteme der Universität Ulm und der Daimler AG in Böblingen.

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während der Erstellung der vorliegenden Arbeit unterstützt und geprägt haben. Allen voran danke ich Prof. Dr. Manfred Reichert, der die Betreuung der Arbeit auf Seiten der Universität übernommen hat. Als Doktorvater danke ich ihm ganz besonders für sein Engagement und seine Geduld.

Ich danke Prof. Dr. Richard Lenz für die Übernahme des Zweitgutachtens sowie den Mitarbeitern des Institutes für Datenbanken und Informationssysteme, die immer mit Rat und Tat zur Seite standen. Hier sei vor allem Dr. Rüdiger Pryss erwähnt, der mich bereits während des Studiums geformt und gefördert hat. Des weiteren möchte ich mich bei Prof. Dr. Thomas Bauer, Dr. Thomas Ringler und Cord Rodefeld für die inspirierenden Diskussionen bedanken.

Mein ganz persönlicher Dank gilt Dr. Joachim Herbst, der für mich die optimalen organisatorischen Rahmenbedingungen innerhalb der Daimler AG geschaffen hat. Sein unermüdlicher Einsatz und seine breites Netzwerk innerhalb der Daimler AG haben mir einen tiefen Einblick in die Produktentwicklung ermöglicht. Vor allem die unzähligen Diskussionen, die immer von tiefem fachlichen Wissen als auch einer Portion Pragmatismus geprägt waren, werden mir noch lange in Erinnerung bleiben.

Zu guter Letzt danke ich meiner Familie und meinen Freunden für ihre Geduld und Verständnis. Ganz besonders danke ich meiner Freundin Fantina, die mir während der gesamten Zeit immer den Rücken frei gehalten hat. Ohne sie wäre die Arbeit nicht möglich gewesen.

Ulm, im Oktober 2018

Teil I

Problembeschreibung und -analyse

1. Einleitung

1.1. Hintergrund

Kontinuierliche Produktinnovationen und stetig steigende Kundenanforderungen führen zu immer komplexeren Produkten, während gleichzeitig die Entwicklungszeiten durch den zunehmenden Wettbewerb stetig kürzer werden sollen [Con12]. An der Entwicklung eines komplexen Produktes, etwa eines Automobiles [MHHR06], sind heutzutage 500 bis 1000 Entwickler beteiligt, zu denen unter anderem Anforderungsanalysten, Softwareentwickler, Zulieferer oder Testspezialisten gehören. Zur Erledigung ihrer Entwicklungsaufgaben verwenden Entwickler autonome, heterogene Informationssysteme[1], die jeweils für spezifische Anwendungsfälle maßgeschneidert sind. Folglich werden die verschiedenen Perspektiven auf Produktdaten (z.B. Anforderungen, Software, geometrische Modelle) in spezifischen, oftmals heterogenen Datenmodellen gespeichert.

Die Produktentwicklung erfolgt in der Regel durch die Zusammenarbeit simultan arbeitender Teams, während der Entwicklungsprozess linear verläuft [CRS+13]. Dadurch werden die einzelnen Produktdaten zu verschiedenen Zeitpunkten erstellt. Um unterschiedliche Kundenwünsche sowie länderspezifische und gesetzliche Anforderungen bedienen zu können, ist eine hohe Produktvariabilität prävalent.

Trotz der autonomen, heterogenen Speicherung von Produktdaten wird es während der Entwicklung notwendig, diese zu unterschiedlichen Zeitpunkten in eine vollständige und konsistente Form zu integrieren. Da die Komponenten komplexer Produkte von einander abhängen können (z.B. vernetzte Steuergeräte in einem Fahrzeug) und teilweise durch Entwickler anderer Hersteller realisiert werden, können Fehler während der Entwicklung auftreten, etwa nach Änderungen an Spezifikationen, die aufgrund fehlender Schnittstellen nicht automatisch an nachfolgende Informationssysteme kommuniziert wurden. Um diesen Problemen entgegen zu wirken, werden in Entwicklungsprojekten sog. Meilensteine definiert, an denen z.B. die Konsistenz von Produktdatenaspekten überprüft und abgesichert wird.

Da Informationssysteme zur Unterstützung spezifischer Aufgaben entwickelt werden, kann es vorkommen, dass die Produktdaten mittels unterschiedlicher Variabilitäts- und Versionierungsmechanismen dokumentiert werden. Darüber hinaus bieten die Datenmodelle der Informationssysteme große Freiheitsgrade in der Speicherung von Produktdaten. So bieten einige Informationssysteme z.B, Freitextfelder an, die beliebige, unstrukturierte Daten enthalten können, wohingegen andere Informationssysteme strenge Einschränkungen für Datentypen und die Inhalte von Attributen besitzen. Die Integration solcher strukturierter und unstrukturierter Daten ist sehr komplex und fehleranfällig.

[1] Wenn im weiteren Verlauf der Arbeit von Informationssystemen gesprochen wird, sind im engeren Sinne Anwendungssysteme gemeint [HMN15].

1.2. Problemstellung

Abbildung 1.1 zeigt die Komplexität eines Produktes für mehrere Ebenen. Ein Produkt besteht aus unterschiedlichen Bauteilen, ein Automobil etwa aus mechanischen (z.B. Karosserie) und elektrischen Bauteilen (z.B. Sensoren, Steuergeräte und Aktuatoren). Um Länder-, Markt- und Kunden-Anforderungen zu bedienen, sind einige Bauteile optional, während für andere von ihnen Varianten existieren [ATW+15, HBR10, Hal10]. Das heißt, es lassen sich verschiedene Produktvarianten konfigurieren. Die Auswahl von optionalen und variablen Bauteilen wird als *Produktkonfiguration* bezeichnet [Ste12].

Ein Beispiel für ein komplexes Produkt ist ein Automobil: Neben tausenden mechanischen Bauteilen umfasst ein modernes Fahrzeug mehr als 100 Steuergeräte, Sensoren und Aktuatoren [MHR06, MHR07, MHR08]. Durch die vielen variablen Merkmale sind theoretisch mehrere Millionen Fahrzeugkonfigurationen möglich. Insgesamt hat sich die Anzahl der Produktvarianten in den letzten 15 Jahren mehr als verdoppelt [Con12]. Ähnliches gilt in Bezug auf moderne Produktionsanlagen und Cyber-physische Systeme [HPS+18, KPSR18].

Für die verschiedenen Ebenen in Abbildung 1.1 existieren unterschiedliche Probleme. Der Fokus dieser Arbeit liegt auf den Ebenen *Produktdaten*, *Informationssysteme* und *Prozesse*. Bei der Produktkonfiguration existiert in der Regel kein durchgängiges Featuremodell zur Beschreibung möglicher Konfigurationen eines Produktes entlang der verschiedenen Informationssysteme. So verwenden letztere proprietäre Modelle, die teils nur implizit dokumentiert sind. Produktdaten sind, wie bereits erwähnt, über autonome, heterogene Informationssysteme verteilt. Eine explizite Dokumentation zusammengehöriger Produktdaten über die Grenzen von Informationssystemen, hinweg ist nicht vorhanden. Durch die heterogenen, konzeptionellen Datenmodelle der einzelnen Systeme werden daher Variabilität und Versionierung von Produktdaten meist unterschiedlich gehandhabt. Die Qualität der dokumentierten Aspekte hängt von der Entwicklungsphase ab und ist in den frühen Phasen oft durch Fehler und unvollständige Informationen geprägt. Neben der Autonomie und Heterogenität der Informationssysteme erschweren technische Aspekte , wie etwa fehlende Schnittstellen, ebenso wie soziokulturelle Aspekte (z.B. verschiedene Verantwortlichkeiten und Interessen der beteiligten Entwickler, Weitergabe impliziten Wissens) die Integration verschiedener Produktdaten und Informationssysteme. Nicht zuletzt stellt die fehlende System- und Prozessorientierung der Informationssysteme [RW12, BBR11], inklusive der Abstimmung von Änderungen untereinander, sowie die mangelnde IT-Unterstützung von Entwicklungsprozessen [Gra16, GOR12] eine weitere Hürde für die Integration dar.

Produktdaten sind alle Dokumente und elektronisch gespeicherten Entwicklungsergebnisse, die während der Entwicklung von Bauteilen anfallen. Sie dienen der lückenlosen Nachvollziehbarkeit der Entwicklung und müssen auf Grund rechtlicher Vorschriften, etwa der Produkthaftung, aufbewahrt werden. Produktdaten umfassen unterschiedliche Aspekte, wie zum Beispiel Anforderungsspezifikationen, Schaltpläne, Leitungssatzdiagramme, geometrische CAD-Modelle oder Software. Häufig werden diese Aspekte in unterschiedlichen Informationssystemen von Entwicklern zu verschiedenen Zeitpunkten dokumentiert. Der hohe Freiheitsgrad der Informationssysteme führt dazu, dass Attribute einzelner Aspekte (z.B. die Bezeichnung) wahlfrei und ohne Berücksichtigung einheitlicher Vorgaben gespeichert werden. So ist es in der Praxis üblich, dass für ein und dieselbe Komponente unterschiedliche Bezeichnungen in den verschiedenen Informationssystemen verwendet werden. Dadurch ist eine automatische Zuordnung zusammengehörender Aspekte nicht ohne weiteres möglich.

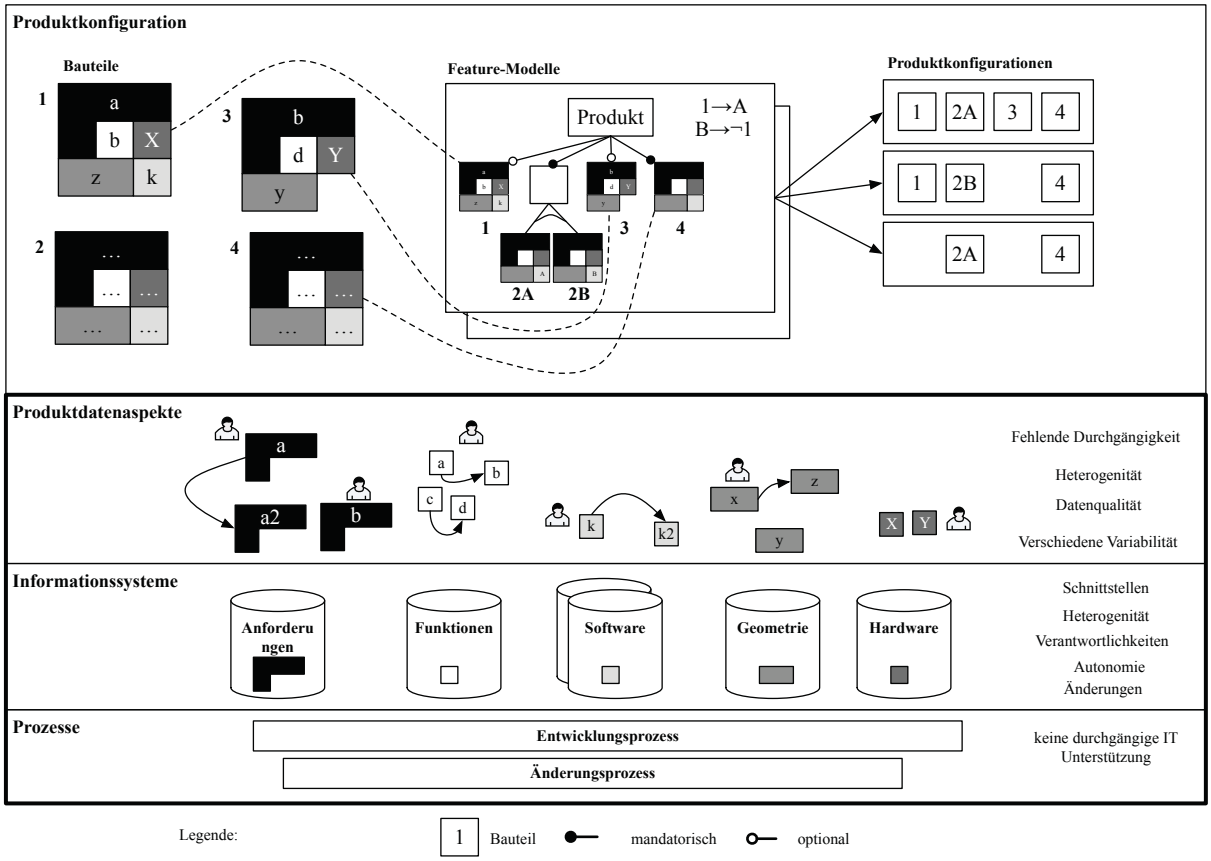


Abbildung 1.1.: Komplexität der Produktentwicklung

Beispiel 1 (Bauteil). Abbildung 1.2 zeigt exemplarisch verschiedene Artefakte eines einzelnen Türsteuergerätes für ein Automobil, die in unterschiedlichen Informationssystemen dokumentiert sein können. Ein Türsteuergerät ist für unterschiedliche Sicherheits- und Komfortfunktionen zuständig und bedient z.B. Aktuatoren für die Spiegelverstellung, den Blinker, das Türschloss oder den Fensterhebermotor. Für die unterschiedlichen Aspekte existieren mehrere Versionen, die zu verschiedenen Zeitpunkten dokumentiert worden sind. Die erwähnte Variabilität von Bauteilen wird in den Informationssystemen unterschiedlich realisiert. Während eine Anforderungsspezifikation alle möglichen Varianten eines Türsteuergerätes beschreibt, werden für die unterschiedlichen Varianten eigene Funktionsmodelle erstellt, die wiederum in Softwaremodellen realisiert werden. Schließlich existieren drei unterschiedliche geometrische Modelle der Türsteuergerätegehäuse, die von unterschiedlichen Herstellern angeliefert werden [Kat15]. Diese unterschiedlichen Methoden zur zeitlichen Dokumentation und Variabilität von Produktdaten erschweren die Integration erheblich.

Während der Entwicklung eines Produktes ist es aufgrund der simultanen Entwicklung [Beu02] essentiell, dass Produktdaten regelmäßig auf verschiedene Eigenschaften, wie etwa Konsistenz und Korrektheit, untersucht werden, um Fehler möglichst früh zu erkennen und beseitigen [RTW15]. Um solche *Anwendungsfälle (AF)* zu realisieren, müssen Produktdaten aus hetero-

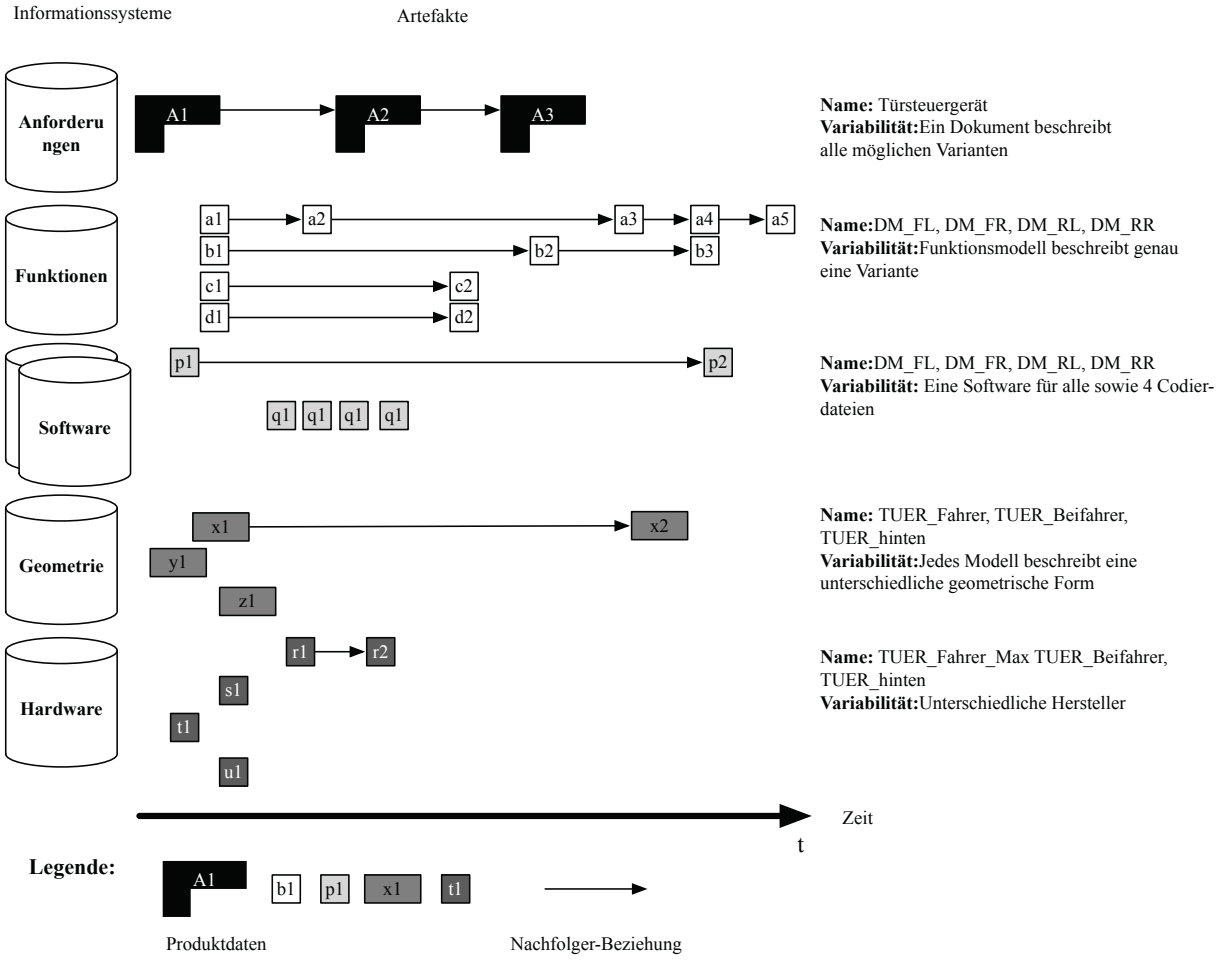


Abbildung 1.2.: Produktdaten während der Entwicklung

genen, autonomen Informationssystemen integriert werden.

Abbildung 1.3 zeigt für das dargestellte Beispiel zwei Anwendungsfälle. Im *Anwendungsfall 1* sollen Anforderungen und Funktionsmodelle für Steuergeräte abgeglichen werden. Dazu müssen zu den einzelnen Anforderungsspezifikationen in den jeweiligen Versionen die zugehörigen Funktionsmodelle in den entsprechenden Versionen ermittelt werden. Darüber hinaus müssen Beziehungen zwischen unterschiedlichen Varianten bestimmt werden. Im speziellen Fall bedeutet dies, dass zu allen Anforderungsspezifikationen alle realisierenden Varianten der Funktionsmodelle bestimmt werden müssen.

Im *Anwendungsfall 2* sollen allen aktuellen Aspekte eines Bauteils integriert werden, damit eine holistische Sicht geschaffen wird. Dazu müssen Produktdaten aus allen Informationssystemen integriert werden. Hier kommt erschwerend hinzu, dass mehr als zwei Informationssysteme integriert werden müssen und somit ein *„gemeinsamer Nenner“* für Variabilitäts- und Versionierungsmechanismen gefunden werden muss.

Heutzutage sind solche Anwendungsfälle schwer zu realisieren, da Beziehungen zwischen zusammengehörigen Produktdaten fehlen. In der Praxis wird die Integration dieser Produktdaten manuell durchgeführt, was sowohl zeitaufwändig als auch fehleranfällig ist. Die Beziehungen

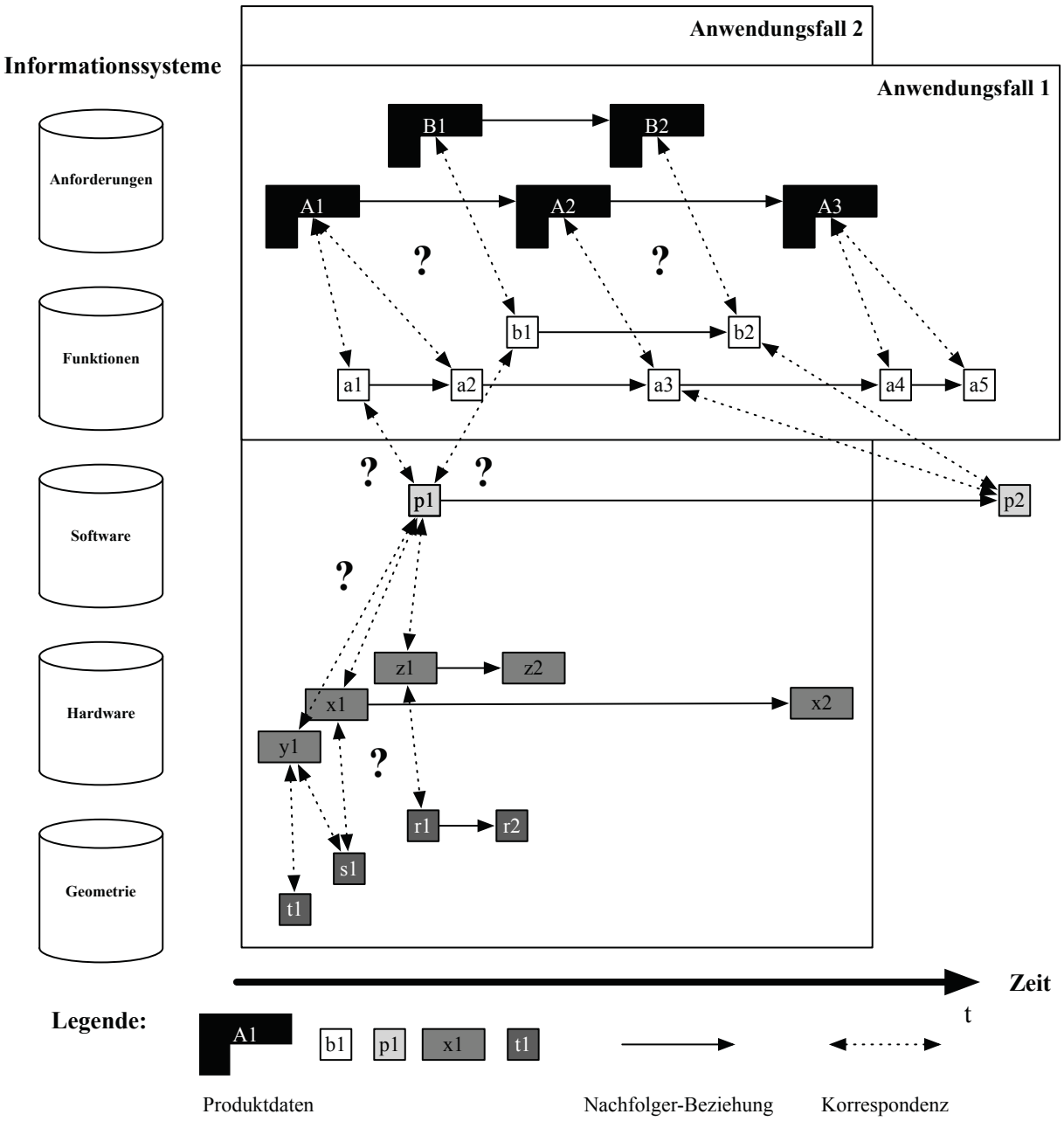


Abbildung 1.3.: Anwendungsfälle für Produktdaten

zwischen zusammengehörenden Aspekten sind, wenn überhaupt, nur implizit vorhanden. Bei der Entwicklung komplexer Produkte sind für die Entwicklung von Bauteilen unterschiedliche Mitarbeiter verantwortlich. Änderungen an Bauteilen können Auswirkungen auf weitere haben und müssen folglich koordiniert werden, wodurch eine manuelle Integration erschwert wird. In einigen Fällen existieren Kopplungen zwischen Applikationen. In der Regel handelt es sich hierbei um den Abgleich von Daten aus zwei Informationssystemen, eine ganzheitliche Integration von mehreren Produktdaten findet jedoch nicht statt.

Neben diesen semantischen und technischen Problemen kann die Qualität der dokumentierten Produktdaten sehr unterschiedlich sein. In frühen Entwicklungsphasen werden oft Entwicklungsergebnisse vorheriger Produkte wiederverwendet und angepasst. Deshalb ist eine vollständige Integration aller Produktdaten zu allen Zeitpunkten nicht möglich bzw. der Aufwand würde in keinem Verhältnis zum Nutzen stehen. Vielmehr ist eine bedarfsgerechte Integration notwendig. Dass heißt, dass nur diejenigen Produktdaten integriert werden sollten, die zur Realisierung eines Anwendungsfalls benötigt werden.
Ziel der Integration von Produktdaten aus heterogenen, autonomen Informationssystemen ist die Unterstützung einer Vielzahl und Vielfalt an Anwendungsfällen. Folglich muss eine Integration zu einem bestimmten Zeitpunkt vollständig und konsistent erfolgen. Damit diese Eigenschaften erreicht werden können, muss der Integrationsprozess überwacht werden, wofür wiederum Kennzahlen benötigt werden, welche die gewünschten Integrationseigenschaften messen.
Da Informationssysteme i.A. autonom sind, erfolgen Datenmodelländerungen unabhängig voneinander, was bei der Integration von Produktdaten berücksichtigt werden muss. Durch Datenmodelländerungen darf bereits existierendes Integrationswissen nicht unbrauchbar werden. Mit Integrationswissen sind vor allem Beziehungen zwischen Produktdaten gemeint.

1.3. Forschungsfragen

Um ein Rahmenwerk zu entwickeln, welches die vorangehend genannten Probleme adressiert, müssen verschiedene Ziele berücksichtigt werden (siehe Tabelle 1.1).

Nr.	**Ziel**
Z1	Integration heterogener und autonomer Produktdaten zur Realisierung von Anwendungsfällen
Z2	Berücksichtigung von Änderungen auf Schema- und Datenebene
Z3	Überwachung und Kontrolle des Integrationsprozesses

Tabelle 1.1.: Forschungsziele

Zunächst muss untersucht werden, wie Produktdaten aus autonomen und heterogenen Informationssystemen mit verschiedenen Mechanismen zur Dokumentation von Variabilität und Versionierung integriert werden können (Z1). Ein Beispiel für Variabilitätsmechanismen in der Automobilindustrie stellen Anforderungsspezifikationen und Funktionsmodelle für Steuergeräte dar (siehe Abbildung 1.4). Während eine Anforderungsspezifikation für ein Steuergerät alle möglichen Varianten beschreibt, müssen ggf. für jede dieser Varianten eigene Funktionsmodelle erstellt werden. Da die Beziehungen zwischen Anforderungsspezifikationen und Funktionsmodellen häufig nicht explizit dokumentiert werden, sondern lediglich implizites Wissen der Entwickler darstellen, kann es vorkommen, dass Änderungen an Anforderungsspezifikationen nicht in entsprechende Funktionsmodelle übernommen werden.
Sind Produktdaten einmal integriert, muss untersucht werden, wie sich unterschiedliche Änderungen sowohl an den Schemata der autonomen Informationssysteme als auch an dessen Daten auf die bereits bestehende Integration auswirken (Z2). Um beim vorherigen Beispiel zu bleiben, werden nach einer initialen Integration von Anforderungsspezifikationen und Funktionsmodellen Änderungen an einigen Spezifikationen vorgenommen. Das heißt, dass neue Versionen in einem

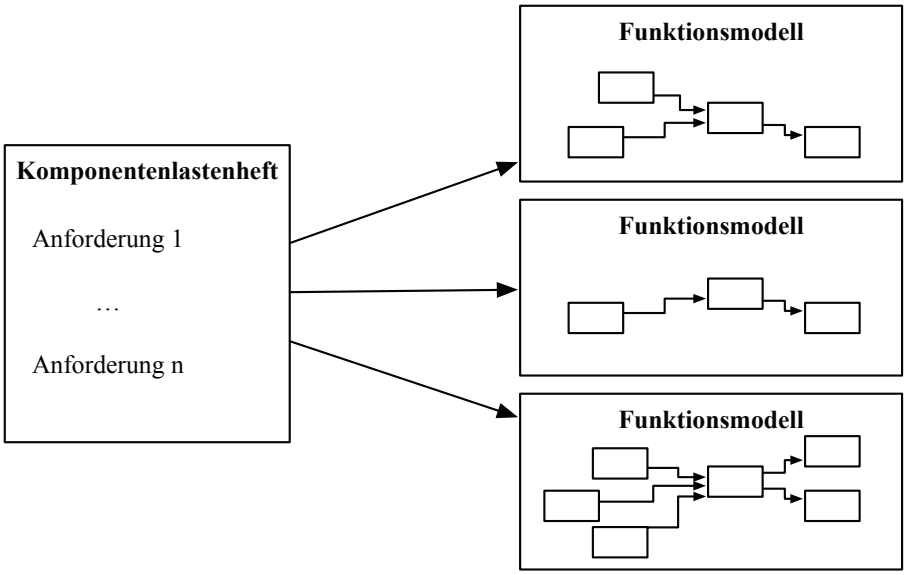


Abbildung 1.4.: Variabilität von Funktionsmodellen

Informationssystem vorhanden sind. Anschließend muss überprüft werden, welche Schritte notwendig sind, um auf diese neuen Produktdaten zu reagieren.
Da die Integration der Produktdaten nur teil-automatisiert erfolgen kann, muss der manuelle Integrationsprozess überwacht werden. Dazu muss untersucht werden, welche Eigenschaften der Integration geeignet sind. Schließlich muss der Integrationsprozess überwacht werden (Z3).
Aus diesen Forschungszielen leiten wir folgende Forschungsfragen ab:

- **Forschungsfrage 1:** Wie lassen sich komplexe Produktdaten aus heterogenen, autonomen Informationssystemen in Entwicklungsumgebungen integrieren, die unterschiedliche Konzepte zur Repräsentation von Variabilität und Versionierung aufweisen?
  - Welche Ansätze und Architekturen existieren zur Integration heterogener, autonomer Informationssysteme? Was sind die Vor- und Nachteile der einzelnen Ansätze?
  - Wie sieht eine Methodik zur Integration heterogener Produktdaten aus?
  - Welche Standards existieren für den Austausch von Produktdaten? Welche Vor- und Nachteile bieten diese?
  - Wie lassen sich die Beziehungen zwischen zusammengehörigen Aspekten beschreiben und auswerten?
  - Welche Ansätze zur Datenintegration bietet Konzepte zur Unterstützung manueller Integrationsaufgaben?
- **Forschungsfrage 2:** Wie lässt sich der Prozess der Integration überwachen?
  - Welche Eigenschaften lassen sich bei der Integration bestimmen und wie sieht ein Monitoringprozess aus?
  - Wie lassen sich diese Eigenschaften zu definierten Zeitpunkten garantieren?

- **Forschungsfrage 3:** Wie beeinflussen strukturelle und inhaltliche Änderungen bereits integrierte Produktdaten?
  - Wie lassen sich Änderungen kompensieren und welche Schritte sind notwendig, um Vollständigkeit und Konsistenz wiederherzustellen?
  - Existieren Änderungen, die bisherige Integrationsinformationen unbrauchbar machen?
- **Forschungsfrage 4:** Mit Hilfe welcher Technologien lässt sich die Integration heterogener Produktdaten umsetzen?
  - Welche Vorraussetzungen muss eine IT-Landschaft besitzen, um eine Integration zu ermöglichen?
  - Wie lassen sich Integrationsprozesse in eine bestehende IT-Landschaft einbinden und welche Aufgaben und Rollen werden benötigt?

1.4. Beitrag

Das im Rahmen der vorliegenden Dissertation entwickelte PROCEED-Rahmenwerk adressiert alle der in Abschnitt 1.3 hergeleiteten Fragestellungen. Der Hauptbeitrag der Arbeit liegt in der Bereitstellung von Konzepten, Techniken und Methoden zur Integration autonomer, heterogener Produktdaten über deren gesamten Lebenszyklus hinweg. Dieser Lebenszyklus reicht von der Abstraktion konzeptioneller Datenmodelle aus autonomen, heterogenen Informationssystemen in *lokale Produktontologien* (engl. local product ontologies, LPOs), über die Definition von *Integrationsregeln* und *-aktionen*, bis hin zur Unterstützung von Änderungen bereits integrierter Produktdaten.
Es werden unterschiedliche Qualitätskriterien für integrierte Produktdaten eingeführt und ein allgemeiner Integrationsprozess, inklusive allen beteiligten Aktivitäten und Rollen, definiert. Zusätzlich werden visuelle Darstellungskonzepte zur Reduktion der Komplexität bei manuellen Integrationsaufgaben erläutert. Die dargestellten Konzepte und Methoden sind generisch und damit domänenunabhängig. Sie lassen sich auf beliebige Produkte (z.B. Fahrzeug, Flugzeug, Maschine, Software) anwenden. Schließlich werden die entwickelten Konzepte prototypisch implementiert und anhand einer Fallstudie praktisch evaluiert.

1.5. Forschungsmethodik

Der Prozess der angewandten Forschung unterteilt sich in folgende vier Aktivitäten:

1. Problemanalyse
2. Anforderungsanalyse
3. Lösungsentwurf
4. Evaluation

Da es das Ziel der Arbeit ist, ein Realweltproblem zu lösen, wurde in der vorliegenden Arbeit die Forschungsmethodik aus Abbildung 1.5 angewandt, sie basiert auf dem *Design Science* Paradigma [HMPR04]. Zunächst erfolgt die Problemanalyse, danach werden Anforderungen hergeleitet

(siehe Kapitel 3) und Lösungen entworfen (siehe Kapitel 5, 6, 7 und 8). Schließlich werden letztere implementiert (siehe Kapitel 9) und evaluiert (siehe Kapitel 10). Im Detail wird das Problem anhand der Domäne der Automobilindustrie und im Speziellen für die Entwicklung von E/E-Komponenten [MHR06, MHR07] beschrieben.
Erkenntnisse der Domäne fließen in die Anforderungsanalyse genau so ein wie eine Literaturstudie zur Integration von Produktdaten. Auf Basis dieser Anforderungen werden Lösungen entworfen und durch Prototypen evaluiert. Erkenntnisse der prototypischen Evaluierung wiederum fließen in den Lösungsentwurf ein und führen somit zu einer Rückkopplung zwischen den beiden Phasen.

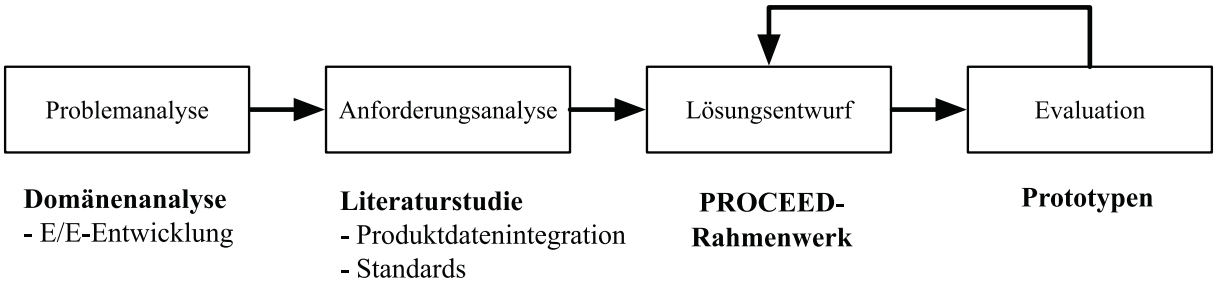


Abbildung 1.5.: Forschungsmethodik

1.6. Aufbau

Die Arbeit gliedert sich in vier Teile: Der erste Teil umfasst die Problembeschreibung und -analyse. Dazu zählen sowohl die Grundlagen, die für das weitere Verständnis der Arbeit erforderlich sind, als auch die notwendige Anforderungsanalyse. Diese werden anschließend mit verwandten Arbeiten abgeglichen, um die Relevanz der Arbeit zu unterstreichen.
Der zweite Teil stellt das PROCEED-Rahmenwerk vor, welches die Lösung für die Handhabung der in Teil I identifizierten Probleme bietet. In Kapitel 6 wird die Integration von Produktdaten aus autonomen, heterogenen Informationssystemen erläutert. Anschließend werden verschiedene Eigenschaften zur Qualität (z.B. Vollständigkeit und Konsistenz) der Integration beschrieben (siehe Kapitel 7), bevor in Kapitel 8 Änderungsoperationen vorgestellt werden. Schließlich werden die prototypische Implementierung der Konzepte in Kapitel 9 erläutert und deren praktische Anwendung in Kapitel 10 erörtert.
Im dritten Teil der Arbeit werden die zuvor dargestellten Konzepte anhand unterschiedlicher Fallstudien validiert. Im letzten und vierten Teil werden die wesentlichen Erkenntnisse zusammengefasst und ein Ausblick gegeben.

2. Grundlagen

Dieses Kapitel definiert grundlegende Begriffe, die für das weitere Verständnis der Arbeit notwendig sind. Zunächst werden allgemein gültige Begriffe zu Daten und Informationen gegeben, bevor Begriffe und Prozesse der Produktentwicklung in der Automobilindustrie erläutert werden. Anschließend werden Aspekte der Integration von Informationen sowie deren Qualität beschrieben.

2.1. Daten und Informationen

Um die Integration von Produktdaten zu definieren, müssen zunächst die Begriffe *Daten* und *Information* erläutert werden. Eine bekannte Definition bietet die DIKW[1]-Hierarchie nach Ackoff [Ack89][2]. Ackhoff strukturiert die Begriffe *Daten*, *Informationen*, *Wissen* und *Weisheit* als Pyramide, bei der Daten als Fundament für Informationen dienen.

Definition 1 (Daten). Daten sind Fakten oder Beobachtungen über Gegenstände der physischen Welt, etwa Dinge oder Menge von Dingen. Ohne einen Kontext oder eine Interpretation besitzen Daten keine Bedeutung.

Definition 2 (Information). Informationen sind Daten, die einen Mehrwert für das Verständnis eines Subjektes besitzen. Folglich übertragen Daten entsprechende Informationen von einem Sender zu einem Empfänger.

Um Daten und Informationen maschinenlesbar zu machen, müssen *Dinge* der physischen Welt mit Hilfe von *Konzepten* beschrieben werden, die durch *Symbole* repräsentiert werden (siehe Abbildung 2.1).

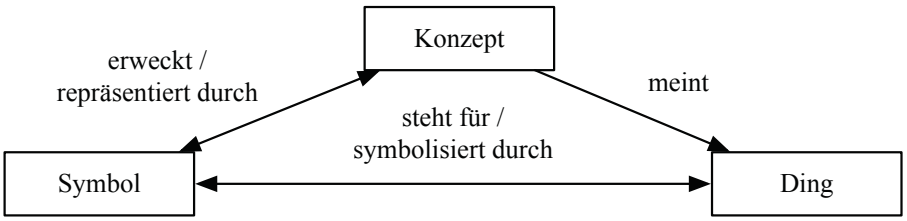


Abbildung 2.1.: Zusammenhänge zwischen Konzept, Symbol und Ding. Semiotisches Dreieck nach [ORMC23]

[1] Data, Information, Knowledge, Wisdom
[2] Eine genaue Untersuchung zu den existierenden Definitionen ist in [Row07] zu finden.

Definition 3 (Konzeptualisierung). Konzeptualisierung ist das Ermitteln und Beschreiben aller notwendigen Konzepte einer Domäne, um Dinge und Mengen von Dingen der physischen Welt mit Hilfe von Informationssystemen zu verarbeiten.

Eine Konzeptualisierung bildet somit die Grundlage für die Erstellung aller Informationssysteme, egal ob das zugrundeliegende semantische Datenmodell auf einem relationalen bzw. objektorientierten Datenmodell oder einer Ontologie basiert. Für letztere existiert keine einheitliche Definition, die am meisten verwendete stammt von [Gru93] (siehe Definition 4).

Definition 4 (Ontologie). Eine Ontologie ist eine explizite Spezifikation einer geteilten Konzeptualisierung.

2.1.1. Evolution vs. Versionierung

Da Dinge der Realwelt nicht statisch sind, sondern ständigen Änderungen [Len03] unterliegen, müssen Konzepte und dementsprechend auch Daten und Informationen angepasst werden. Je nach dem jeweils zugrunde liegendem semantischen Datenmodell eines Informationssystems muss zwischen *Schemaevolution* und *Ontologieevolution* unterschieden werden.
Nach [JDB+98] unterstützt eine Datenbank die *Evolution*, wenn nach Änderungen des Schemas keine bestehenden Daten verloren gehen. Weiter unterstützt eine Datenbank *Versionierung*, wenn verschiedene Schemata genutzt werden können, um Daten abfragen zu können. Die entsprechenden Begriffe werden in der Literatur für prozessorientierte Informationssysteme entsprechend erweitert [RRD04a, RRD04b].

Schemaevolution und -versionierung

„Es wird zwischen *Schemaevolution* und *Schemaversionierung* unterschieden. Schemaevolution ist die Fähigkeit das Schema einer mit Daten gefüllten Datenbank zu ändern, ohne dass dabei Daten verloren gehen“ [Rod95]. Schema-Versionierung bedeutet, auf alle Daten (alte und neue) über verschiedene Schnittstellen zugreifen zu können.

2.2. Produktentwicklung

Zentraler Prozess eines produzierenden Unternehmens ist der Produktentwicklungsprozess. Dieser definiert alle Aktivitäten und Rollen, die notwendig sind, um Produkte zu entwickeln. Im Folgenden werden die Grundlagen der Produktentwicklung am Beispiel der Automobilindustrie erläutert; sie sind für das weitere Verständnis der Arbeit erforderlich sind. Die grundlegenden Prozesse und Informationssysteme sind in anderen Domänen (etwa der Luftfahrtindustrie) ähnlich, so dass die gewonnen Erkenntnisse der Arbeit generisch auf andere Domänen übertragen werden können.

2.2.1. Produkt und Produktdaten

Ein *Produkt* ist ein physisches Erzeugnis und stellt das Ergebnis eines Entwicklungsprozesses dar, an dem eine Vielzahl unterschiedlicher Personengruppen beteiligt sind (z.B. Entwickler, Projektmanager, Tester, Lieferanten). Für die Entwicklung komplexer Produkte, wie etwa Automobilen

oder Flugzeugen, hat sich eine System- und Komponentenorientierte Entwicklung durchgesetzt. Zunächst werden die *Funktionen* eines Produktes spezifiziert und anschließend auf verschiedene, untereinander vernetzte *Systeme* verteilt. Letztere wiederum werden durch das Zusammenspiel unterschiedlicher *Komponenten*, auch Bauteile genannt, nebenläufig realisiert. In diesem Zusammenhang wird auch von *Concurrent Engineering* gesprochen [Beu02, PMR93, YB03].
[SAS05] beschreibt Produktdaten als diejenigen Daten, die von der Konzeption des Produktes bis zu dessen Fertigung benötigt werden. Zu ihnen zählen *Computer-Aided Design* (CAD) Daten, *Computer-Aided Manufacturing* (CAM) Daten, *Computer-Aided Engineering* (CAE) Daten, *Product Data Management* (PDM) und weitere Daten.

Definition 5 (Produktdaten). Produktdaten umfassen alle Dokumente, Erzeugnisse, Modelle und Protokolle, die während der Entwicklung eines Produktes anfallen können. Sie sind die Grundlage der Produktion und müssen auf Grund rechtlicher Rahmenbedingungen (z.B. Produkthaftung) dokumentiert werden.

Typischerweise besteht ein komplexes Produkt aus mehreren tausend Bauteilen. Um die Entwicklungszeit von Produkten zu reduzieren, werden diese Bauteile nebenläufig durch unterschiedliche Verantwortliche entwickelt. Um die unterschiedlichen Bauteile zuordnen zu können, besitzen sie eindeutige *Bauteilnummern*. Dabei erfolgt die Entwicklung von Bauteilen in der Regel durch unterschiedliche Zusammenarbeitsmodelle unter Beteiligung einer Vielzahl von Zulieferern [Kat15].

Beispiel 2 (Produktdaten). Ein Beispiel für ein System eines Fahrzeuges ist eine Autotür. Dieses System realisiert verschiedene Funktionen (z.B. die Spiegelverstellung oder Fensterbetätigung) mit Hilfe unterschiedlicher Komponenten. Abbildung 2.2 zeigt schematisch den Aufbau einer Autotür, die neben mechanischen Komponenten (z.B. Innen- und Außenverkleidung, Scharniere) aus mechatronischen Komponenten (z.B. Außenspiegel, Türschließung und Schalterblock zur Bedienung des Fensters und des Außenspiegels) besteht. Schließlich gibt es rein elektrische und elektronische Komponenten, etwa den Fensterhebermotor, den Lautsprecher oder das Türsteuergerät, welches für die Ansteuerung der verschiedenen Aktuatoren verantwortlich ist.

Ein Produkt durchläuft während seiner Entwicklung einen definierten *Produktlebenszyklus*, in dessen Verlauf, zu den einzelnen Komponenten verschiedene *Produktdaten* anfallen. Letztere sind in der Regel über eine Vielzahl autonomer, heterogener *Informationssysteme* verteilt [Sta11]. Zu den Produktdaten einer Komponente zählen sowohl *Modelle* (z.B. Systemmodelle, Simulationsmodelle), E-CAD Modelle[3] (z.B. Leitungssätze, Steuergeräte, Sensoren, Aktuatoren, Schaltpläne), M-CAD[4] Modelle (z.B. Bauteilgeometrien, Konstruktionszeichnungen), als auch *Daten* (z.B. Anforderungs- und Testspezifikationen).
Die verschiedenen *Produktdaten* eines Produktes werden üblicherweise durch viele Beteiligte zu unterschiedlichen Zeitpunkten erstellt, genutzt und geändert. Entwickler beschreiben ein Bauteil aus speziellen Sichten (z.B. Anforderungsspezifikation, geometrische Modellierung, Softwareentwicklung oder Test), die jeweils durch spezifische Informationssysteme unterstützt werden sollen.

[3] Elektronisches CAD
[4] Mechanisches CAD

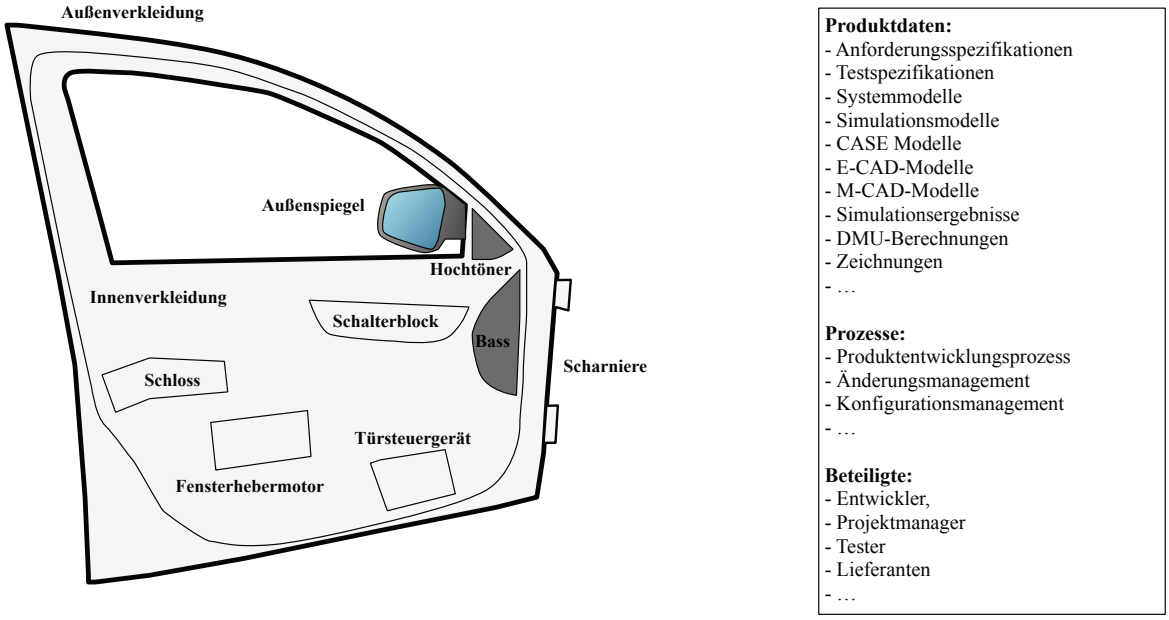


Abbildung 2.2.: Geometrie einer Autotür

Definition 6 (Produktdaten). Produktdaten beschreiben einen Ausschnitt an Produktdaten aus einem fachlichen Bereich. Diese Daten können in einem oder mehreren Informationssystemen verwaltet werden. Innerhalb dieser Systeme können gleiche Aspekte in unterschiedlichen Datenstrukturen repräsentiert sein.

Beispiel 3 (Produktdaten eines Automobils). Abbildung 2.3 zeigt die Verteilung von Bauteilaspekten der Produktentwicklung eines großen deutschen Automobilherstellers.

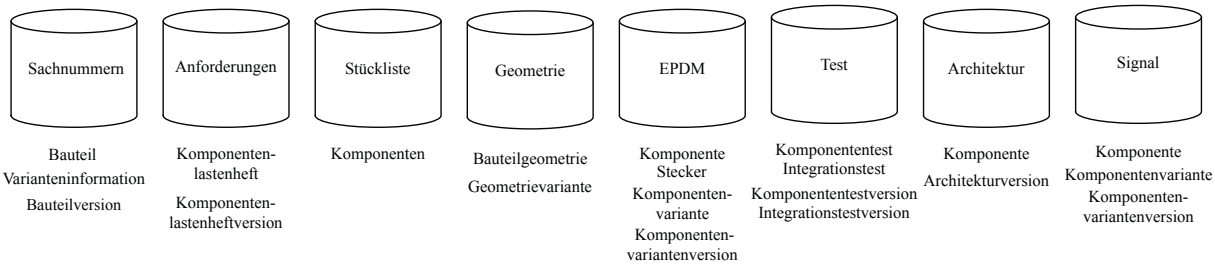


Abbildung 2.3.: Verteilung von Produktdaten über Informationssysteme

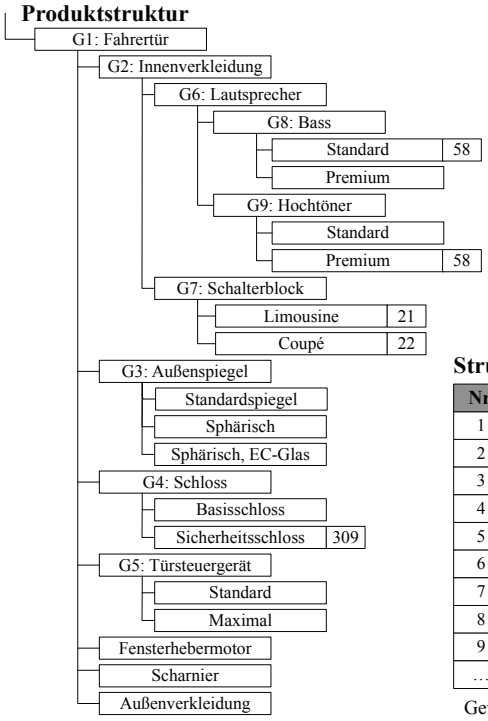
2.2.2. Produktstruktur und -stücklisten

Die strukturierte Anordnung von Bauteilen wird als *Produktstruktur* bezeichnet und spiegelt die Beziehungen und Abhängigkeiten von Bauteilen und Zusammenbauten[5] untereinander wider.

[5]Zusammenbauten sind Bauteile, die sich aus mehreren Bauteilen zusammensetzen und in der Regel von Zulieferern bezogen werden. Daher besitzen sie in der Regel nur eine Bauteilnummer.

Eine spezielle Produktstruktur der Produktentwicklung ist die sog. *(Struktur-)Stückliste*. [6] Sie bildet die hierarchische Struktur ab, die ein Produkt von oben nach unten in Bauteile und Zusammenbauten zerlegt. Eine Stückliste ist notwendig, um ein spezifisches Produkt herzustellen. Sie ist somit für die Bestellung und Bauteilebedarfsermittlung essentiell. In der Regel werden für unterschiedliche Aufgaben während der Entwicklung unterschiedliche Produktstrukturen verwendet [SI08].

Beispiel 4 (Produktstruktur). In Abbildung 2.4 sind ein exemplarischer Auszug der Produktstruktur sowie eine Strukturstückliste für die zuvor beschriebene Autotür dargestellt. In der Produktstruktur auf der linken Seite werden Bauteile in Gruppen (z.B. `Innenverkleidung`, `Spiegel`) zusammengefasst. Innerhalb einer Gruppe können mehrere ähnliche Bauteile vorhanden sein. In der Strukturstückliste auf der rechten Seite ist aus der Produktstruktur eine spezifische Autotür *konfiguriert*. Das heißt, es sind sowohl die benötigten Mengen einzelner Bauteile als auch Gruppen bestimmt. Für alle Bauteile, für die mehrere Alternativen vorhanden sind, ist genau eine ausgewählt.



Coderegeln

Code	Beschreibung
21	Limousine
22	Coupé
58	Premium Soundsystem
309	erhöhter Diebstahlschutz

Strukturstückliste (konfiguriert)

Nr.	Ebene	Gruppe / Teil	Code	Menge
1	1	Fahrertür	;	1
2	2	Innenverkleidung	;	1
3	3	Lautsprecher	;	1
4	4	Bass	;	1
5	5	Premium	58;	1
6	4	Hochtöner	;	2
7	5	Premium	58;	1
8	2	Schalterblock	;	1
9	3	Limousine	21;	1
…	…	…	…	…

Gewählte Codes: Limousine, Premium Soundsystem

Abbildung 2.4.: Beispiel einer Produktstruktur sowie Strukturstückliste

2.2.3. Systemlandschaft und Prozesse

In Abbildung 2.5 ist eine generische IT-Systemlandschaft für Automotive Engineering Dienste dargestellt. Einen ähnlichen Aufbau beschreiben [TRS15, Kat15]. Gemeinsam ist beide Land-

[6] Darüberhinaus existieren weitere Stücklisten wie z.B. die Strukturstückliste, die Variantenstückliste oder die Verwendungsstückliste, welche jedoch für das weitere Verständnis der Arbeit nicht notwendig sind.

schaften, dass während der verschiedenen Phasen der Entwicklung unterschiedliche Informationssysteme verwendet werden.

Beispiel 5 (IT-Entwicklungslandschaft). Der Lebenszyklus der Entwicklung ist in Abbildung 2.5 in `Spezifikation`, `Geometrie`, `E/E und Software`, `Simulation und Berechnung` sowie `Test` unterteilt. Die dargestellte Systemlandschaft stellt ein Idealbild dar, Abweichungen sind möglich. Der Austausch von Produktdaten zwischen den verschiedenen Informationssystemen erfolgt über ein `Produktdaten Backbone`. Zur Unterstützung der Entwicklung existiert eine Vielzahl an Prozessen [BHR05, BRB05, Mül09], wovon ein Teil durch Informationssysteme unterstützt werden, während andere manuell gelebt werden. Neben dem `Projekt-` und `Stammdatenmanagement` sind vor allem das `Stücklisten-` und `Änderungsmanagement` essentiell. Da die Entwicklung von Komponenten und gesamten Systemen häufig in Zusammenarbeit mit Lieferanten geschieht, ist die Integration von Lieferantendaten eine weitere Herausforderung.

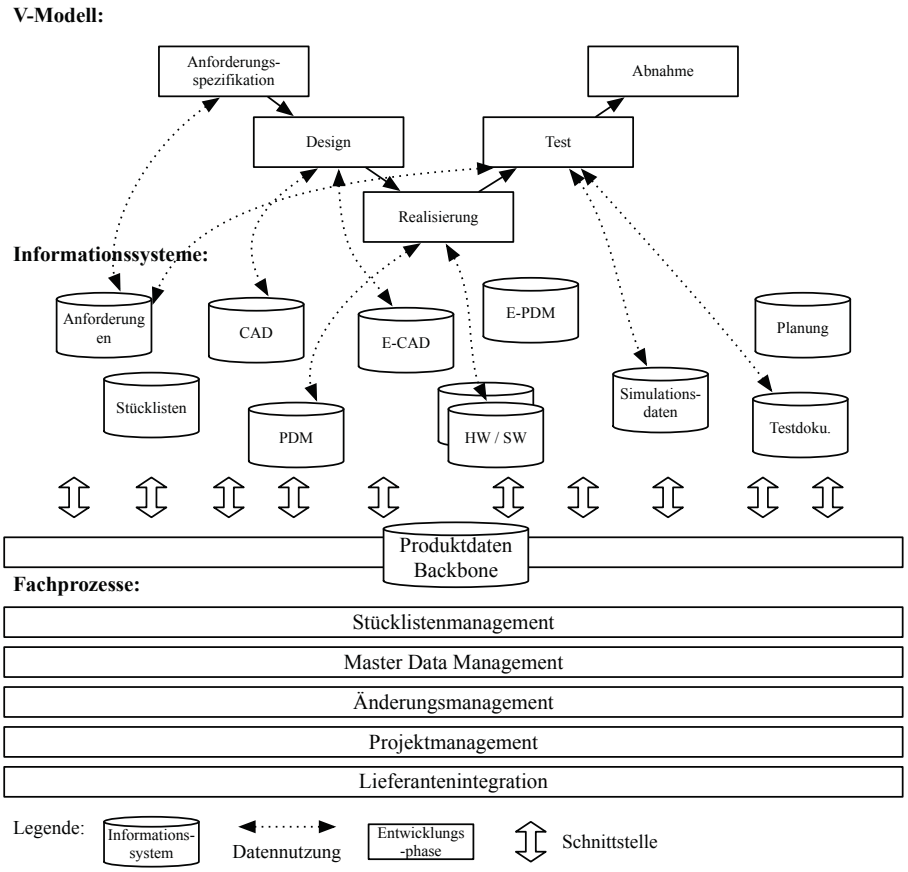


Abbildung 2.5.: Generische Systemlandschaft der Produktentwicklung im Automobilbereich (angelehnt an [Kat15])

[Web14] detailliert weitere Prozesse der Produktentwicklung, zum Beispiel die virtuelle Produktentwicklung oder das Qualitätsmanagement, die im weiteren nicht ausgeführt werden.

Produktentwicklung

Der *Produktentwicklungsprozess* ist durch folgende Merkmale gekennzeichnet: Systeme und Komponenten werden in unterschiedlichen Zusammenarbeitsmodellen zwischen Herstellern und Zulieferern entwickelt. Der Entwicklungsprozess verläuft nach dem V-Modell [BR05], der sich grob in die fünf Phasen `Anforderungsspezifikation`, `Design`, `Realisierung`, `Test` und `Abnahme` einteilen lässt (vgl. Abbildung 2.5). Neben den zuvor genannten Prozessen wie Konfigurations- und Änderungsmanagement [BRB05, BRB06] ist die Produktentwicklung in unterschiedliche Fachprozesse unterteilt. Beispiele für Fachprozesse sind etwa die Entwicklung von elektrischen und elektronischen Komponenten, die virtuelle Produktentwicklung, die mechanische Konstruktion, die Motorentwicklung und die Antriebsstrangentwicklung. Entlang dieser Fachprozesse sind *Meilensteine* definiert, die vordefinierte Entwicklungsstände zu einem festen Zeitpunkt repräsentieren. Die Synchronisation der Fachprozesse erfolgt über sog. *Quality Gates*[7]. An diesen Punkten sind eindeutige Qualitätskriterien festgelegt, die erfüllt sein müssen, damit die Entwicklung in die nächste Entwicklungsphase übergehen kann [PNSD07]. Während Quality Gates zeitlich flexibel sind, besitzen Meilenstein feste Deadlines, was dazu führt, dass diese Deadlines nicht immer eingehalten werden [PS14].

Beispiel 6 (Meilenstein). In Abbildung 2.6 existieren mehrere Meilensteine für unterschiedliche Fachprozesse. Beispielsweise müssen in der mechanischen Konstruktion zunächst alle geometrischen Modelle und Zeichnungen freigegeben werden, bevor eine Kollisionsuntersuchung durch digitale Mock-Ups (DMU)[8]durchgeführt werden kann. Weitere Beispiele für Meilensteine sind die Absicherung der E/E-Architektur oder Hardware-in-the-Loop (HiL)[9].

Beispiel 7 (Quality Gate). In Abbildung 2.6 sind drei Quality Gates dargestellt, d.h. *Lastenheft*, *Realisierung* und *Prototyp*. Sie teilen den Entwicklungsprozess in unterschiedliche Phasen ein. Der Übergang von einer in die nächste Phase geschieht nur, wenn die zuvor definierten Qualitätskriterien des Quality Gates erfüllt sind. Beispielsweise kann die Prototypen-Phase erst dann begonnen werden, wenn z.B. Geometrien und Konstruktionszeichnungen für alle Komponenten freigegeben sind.

[7] Das System wird auch als *Stage-Gate-System* bezeichnet [Coo90]. Weitere Synonyme für Gates bzw. Quality Gates sind Gateways, Synchropunkte, Q-Checks, Moments of Truth [PS14].

[8] DMU ist die digitale Absicherung etwa zur Bestimmung von Kollisionen von Bauteilen durch geometrische Modelle.

[9] HiL wird in späten Entwicklungsphasen eingesetzt, um das Verhalten zwischen mehreren Bauteilen durch physische Prototypen zu simulieren. Umgebungsvariablen eines Fahrzeuges wie etwa die aktuelle Geschwindigkeit oder Lenkwinkel werden durch Echtzeitrechner simuliert.

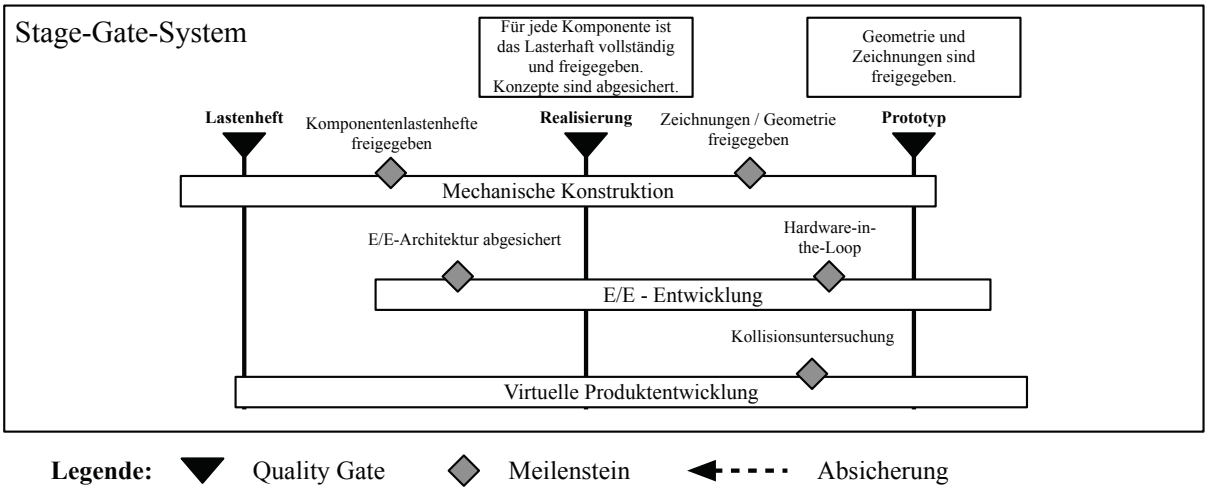


Abbildung 2.6.: Stage-Gate-Modell am Beispiel von Fachprozessen

Konfigurationsmanagement

Wie anhand der Produktstruktur der Autotür bereits zu erkennen ist, besitzen Produkte *variable* und *optionale* Bauteile. Die Gründe für variable und optionale Bauteile sind vielfältig: Einerseits möchten Hersteller ihren Kunden die Möglichkeit bieten, ein Produkt nach Vorlieben und Wünschen zu wählen, andererseits existieren in verschiedenen Ländern unterschiedliche Rechtsgrundlagen, so dass Bauteile für diese unterschiedlichen Gegebenheiten entwickelt werden müssen. So existieren in der Europäischen Union und in den Vereinigten Staaten verschiedene Grenzwerte für Emissionen von Kraftfahrzeugen.
Merkmale eines Produktes, die zu Variabilität führen, sind zum Beispiel geometrische Abmessungen, physikalische Eigenschaften, Funktionalitäten, Form, Oberflächen, Farben oder Materialien. Neben dieser *externen Variabilität*, also durch den Kunden wahrnehmbare Unterschiede, existiert *interne Variabilität*. Beispiele für letztere sind verschiedene Zulieferer für gleiche Bauteile in unterschiedlichen Produktionsstätten.
Die Verwaltung von Produktmerkmalen wird als *Konfigurationsmanagement* bezeichnet und definiert, welche Bauteile obligatorisch bzw. optional sind. Eine Möglichkeit, Produktmerkmale darzustellen, sind Feature-Diagramme [KCH+90].

Beispiel 8 (Feature-Diagramm). Abbildung 2.7 zeigt ein Feature-Diagramm für die zuvor beschriebene Autotür. Das Diagramm umfasst ingesamt 20 *Features* sowie mehrere *Constraints*. Die Features `anklappbar`, `beheizt` und `erhöhter Diebstahlschutz` sind optional, während für `Türsteuergerät`, `Form`, `Glas`, `Lautsprecher` und `Design` alternative Varianten existieren.

Durch die dargestellten Features und Constraints lassen sich theoretisch 104 verschiedene Fahrertüren aufbauen. Die Auswahl variabler und optionaler Bauteile einer Produktstruktur wird als *Produktkonfiguration* bezeichnet. Das damit verbundene *Konfigurationsmanagement* stellt sicher, dass nur sinnvolle, dass heißt baubare bzw. wirtschaftlich sinnvoll Produktkonfigurationen gewählt werden können. In der Praxis wird die Anzahl möglicher Konfigurationen auf eine

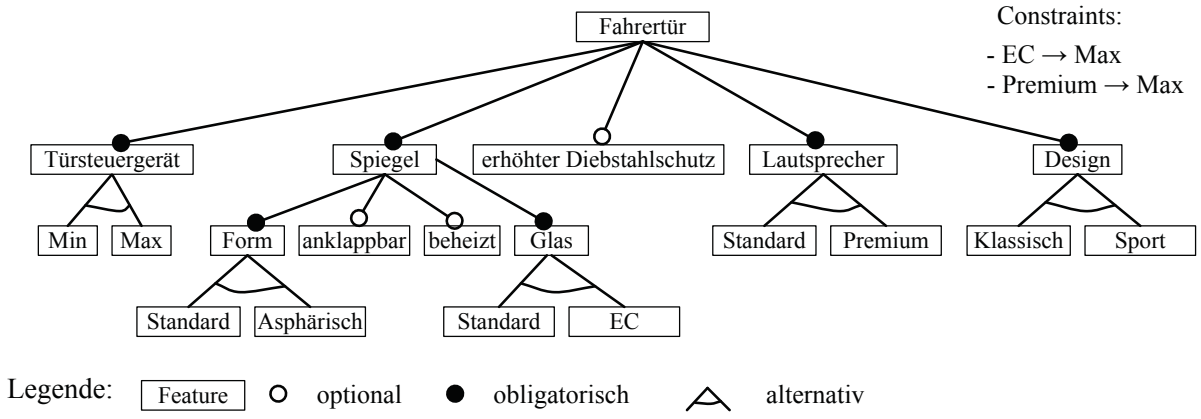


Abbildung 2.7.: Feature-Diagramm einer Autotür

wirtschaftliche Anzahl eingeschränkt, um die Komplexität während der Produktentwicklung zu reduzieren.
Es existieren unterschiedliche Methoden, um variable und optionale Bauteile in der Produktstruktur abzubilden [Nob07]. Eine häufig verwendete Methode, variable Merkmale abzubilden, besteht darin, diese mit Hilfe von Codes[10] zu verschlüsseln[11]. Variable und optionale Bauteile in der Produktstruktur der Autotür in Abbildung 2.4 sind mit Codes versehen. So sind etwa die Bauteile, die zum `Premium Soundsystem` gehören, mit dem Code `58` versehen. Während der Konfiguration einer Produktinstanz werden mit Hilfe logischer Operationen die gewählten Codes ausgewertet und so alle benötigten Bauteile ermittelt, die für den Aufbau erforderlich sind.

Änderungsmanagement

Wie bereits in Abschnitt 2.1.1 erwähnt, unterliegen Daten kontinuierlichen Änderungen. Integraler Bestandteil der Produktentwicklung ist das Änderungsmanagement, welches *Änderungsvorhaben* koordiniert und dadurch eine lückenlose Dokumentation von Bauteiländerungen ermöglicht [Bob08]. Nach [Vi09] sind die folgenden Ursachen für Änderungen an Bauteilen relevant:

- Gesetzliche Änderungen (z.B. Abgasvorschriften im Automobilbereich),
- Änderungen der Markt- und Konkurrenzbedingungen,
- Interne Verbesserungen in Entwicklung, Planung oder Produktion,
- Qualitäts- und Sicherheitsprobleme,
- Ausnutzung zusätzlicher Optimierungspotenziale

In Abbildung 2.8 ist ein vereinfachter Änderungsprozess (Engineering-Change-Prozess) illustriert, der aus vier Schritten besteht. Zunächst wird ein Problem beschrieben bzw. ein Änderungsgrund identifiziert. Anschließend werden mögliche Lösungen für das Problem gesucht und dokumentiert. Ein Entscheidungsgremium stimmt auf Basis dieser Informationen über die Änderung

[10] Weitere Synonyme in der Literatur sind etwa *Ausstattungen*, *Merkmale*, *Schlüsselelemente* und *Optionen*
[11] Diese wurden bereits in der Strukturliste und der Produktstruktur in Abschnitt 2.2.2 verwendet, vgl. Abbildung fig:productstructure

ab. Wird diese genehmigt, erfolgt die Umsetzung. In der Literatur lassen sich noch weitere Änderungsprozesse finden, die weitere Aktivitäten wie etwa die Umsetzungsüberwachung berücksichtigen [Sta11, RBRB06].

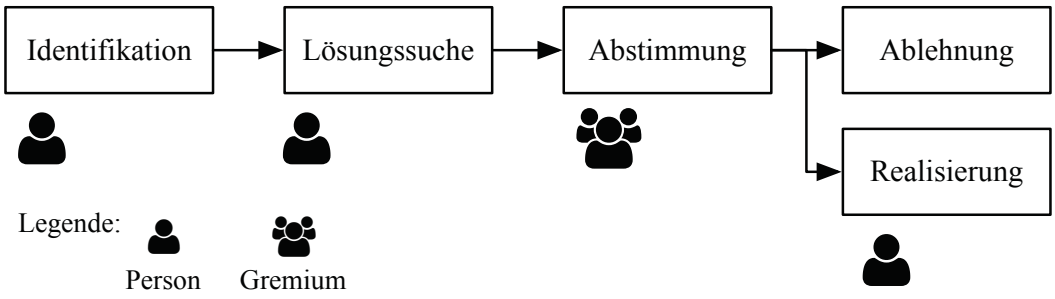


Abbildung 2.8.: Beispiel für einen Engineering-Change-Prozess

2.3. Integration

Die vorliegende Arbeit fokussiert auf die Integration heterogener Produktdaten aus autonomen Informationssystemen. Das Forschungsfeld der Integration reicht bis in die Anfänge der Datenverarbeitung zurück und ist auch weiter ein aktiver Untersuchungsbereich. Im Folgenden werden die grundlegenden Prinzipien erläutert.
Der Begriff *Integration* beschreibt das Zusammenführen von Artefakten, die semantisch zusammengehören, jedoch über heterogene Quellen (z.B. Informationssysteme) verteilt sind. Nach [LN06] ergeben sich drei Grundprobleme, die bei der Integration von Informationssystemen berücksichtigt werden müssen: *Autonomie*, *Verteilung* und *Heterogenität*.
Für die Integration heterogener Informationssysteme sind im Laufe der Zeit verschiedenen Techniken entstanden. Dazu zählen das *Schema Matching*, die *Duplikaterkennung* und die *verteilte Anfrageverarbeitung*. Im Folgenden werden die grundlegende Terminologie eingeführt und anschließend die Integration in verschiedene Kategorien klassifiziert. Allgemeine Begriffe wie *Integration* und *Korrespondenzen* beziehen sich im weiteren Verlauf der Arbeit auf die Integration autonomer, heterogener Produktdaten.

2.3.1. Terminologie

Wie bereits erläutert, kommen während der Produktentwicklung unterschiedliche Informationssysteme zum Einsatz, die wie folgt definiert sind:

Definition 7 (Informationssystem). Ein *Informationssystem* ist ein Softwaresystem zur Ausführung betrieblicher Abläufe, welches die Erfassung, Verarbeitung, Speicherung und Übertragung von Informationen automatisiert. Wir unterscheiden zwischen einem *globalen Informationssystem*, welches eine einheitliche Sicht auf eine Vielzahl autonomer, heterogener Informationssysteme durch Integration bietet. Letztere werden als *lokale Informationssysteme* bezeichnet.

Bei der Produktentwicklung dienen Informationssysteme der Unterstützung spezifischer Entwicklungsaufgaben. Darüber hinaus müssen Anwendungsfälle unterstützt werden, die Produktdaten aus mehreren Informationssystemen benötigen.

Definition 8 (Anwendungsfall). In Rahmen der Produktentwicklung bezeichnet ein *Anwendungsfall (AF)* die Nutzung von integrierten Produktdaten aus heterogenen Informationssystemen zur Realisierung fachlicher Aufgaben. Beispiele für solche Aufgaben sind:

- Suche / Reports,
- Dokumentationsgenerierung,
- Navigation / Visualisierung,
- Konsistenz-, Vollständigkeits- und Plausibilitätsüberprüfung und
- Berechnungen.

Beispiel 9 (Anwendungsfall). Abbildung 2.9 zeigt einen Anwendungsfall für die Konsistenzüberprüfung von Anforderungsspezifikationen für mechatronische Komponenten mit den entsprechenden Funktionsmodellen. Ziel der Überprüfung ist es festzustellen, ob für alle Komponenten entsprechende Funktionsmodelle vorhanden sind. Da die Komponenten durch unterschiedliche Zulieferer erstellt werden, können heterogene Informationssysteme zur Modellierung zum Einsatz kommen. Nach der Integration ist es möglich, Änderungen sowohl an den Anforderungsspezifikationen als auch den Funktionsmodellen nachzuvollziehen. Dazu werden mit Hilfe der Korrespondenzen zwischen den Artefakten zusammengehörige Dokumente ermittelt, die dann von Fachanwendern gesichtet werden können.

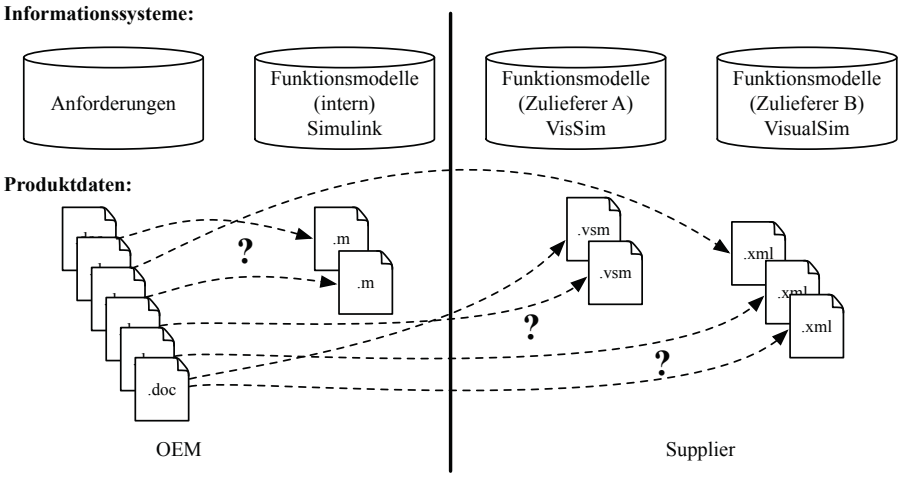


Abbildung 2.9.: Anwendungsfall

Die Realisierung von Anwendungsfällen wird durch die Produktdatenintegration ermöglicht, die folgendermaßen definiert ist:

Definition 9 (Produktdatenintegration). *Produktdatenintegration* ist der Prozess der Integration autonomer, heterogener Produktdaten und dessen unterschiedliche Produktdaten aus lokalen Informationssystemen in eine zentrale Wissensbasis. Die Verwaltung dieser Wissensbasis geschieht durch ein globales Informationssystem, auch *Produktdatenintegrationssystem (PDIS)* genannt, welches die notwendigen Funktionalitäten bereitstellt, um Produktdaten zu integrieren, auf Änderungen zu reagieren und Abfragen integrierter Produktdaten zu ermöglichen.

Wie erwähnt, müssen sowohl die Schemata als auch die Daten heterogener Informationssysteme integriert werden. Auf Grund der verschiedenen Datenmodelle, die bei der Realisierung lokaler Informationssysteme zum Einsatz kommen, müssen die verschiedenen Begriffe vereinheitlicht werden. Dazu definieren wir Schemakonzepte (engl. schema concepts, SC) und Instanzen (engl. individuals, Ind) folgendermaßen:

Definition 10 (Schemakonzept). Ein *Schemakonzept (engl. schema concept, SC)* modelliert in einem *semantischen Datenmodell* (z.B. ER-/UML-Diagramm) eine Menge von Dingen der physischen Welt durch die Definition von *Attributen* (vgl. Definition 3). Zwischen Schemakonzepten können Beziehungen definiert werden. Abhängig vom Datenmodell existieren weitere Synonyme, wie z.B. Klasse, Entitätstyp, Relation, Typ oder Konzept. Das Schema des PDIS wird im weiteren als *globales Schema* bezeichnet, während Schemata von Informationssystemen *lokale Schemata* heißen.

Konkrete Realweltobjekte werden im weiteren Verlauf der Arbeit als Instanzen bezeichnet. Sie sind folgt definiert:

Definition 11 (Instanz). Ein Gegenstand einer Menge mit den gleichen Attributen (Schemakonzept) wird als *Instanz (engl. individual, Ind)* bezeichnet. Jede Instanz kann Ausprägungen der Attribute in Form von *Attributwerten* haben. Je nach Datenmodell werden die Synonyme Individuum, Ausprägung, Objektinstanz, Datensatz, Tupel, Zeile, Entität oder Objekt verwendet.

Ziel der Produktdatenintegration ist es, semantisch zusammengehörige Produktdaten zu identifizieren. Deren Beziehungen werden als *Korrespondenzen* bezeichnet und sind wie folgt definiert:

Definition 12 (Korrespondenz). Eine *Korrespondenz (engl. correspondence, Cor)* ist eine gerichtete Beziehung zwischen zwei oder mehreren Artefakten (Schemakonzept, Instanz, Attribut, Attributwert). Die Beziehung bedeutet, dass die korrespondieren Artefakte in der Realwelt semantisch zusammengehören, sie bedeutet jedoch nicht automatisch deren Äquivalenz. Zwischen zwei Artefakten können mehrere Korrespondenzen existieren. Korrespondenzen können mit einem numerischen Maß versehen sein, welches den Grad der Ähnlichkeit zwischen zwei Artefakten beschreibt. Korrespondenzen zwischen Schemakonzepten werden *Schemakonzept-Korrespondenz* und die zwischen zwei Instanzen als *Instanz-Korrespondenz* bezeichnet.

Der Prozess zum Auffinden von Korrespondenzen wird als *Matching-Prozess* bezeichnet, die Menge der gefundenen Korrespondenzen zwischen zwei Informationssystemen wiederum nennt man *Mapping*:

Definition 13 (Matching). *Matching* bezeichnet den Prozess, Korrespondenzen zwischen Artefakten (Schemakonzepte, Schemaattribute, Instanzen) zu finden. In der Regel soll dieser Prozess soweit wie möglich automatisch ablaufen. In der Praxis werden jedoch manuelles Eingreifen bzw. Korrigieren von Korrespondenzen notwendig.

Definition 14 (Mapping). Das Ergebnis des Matching ist das *Mapping*, welches alle ermittelten gerichteten Korrespondenzen zwischen Artefakten zweier unterschiedlicher Informationssysteme enthält. Das Mapping kann anhand unterschiedlicher Qualitätsdimensionen untersucht werden, wodurch auch die Qualität der Integration bestimmt werden kann.

Beispiel 10 beinhaltet alle zuvor definierten Begriffe.

Beispiel 10 (Terminologie). In Abbildung 2.10 werden die verschiedenen Begriffe an Hand eines Beispiels illustriert. Das konzeptionelle Datenmodell des *lokalen Informationssystem A* basiert auf dem relationalen Datenmodell und ist als ER-Diagramm illustriert, das Datenmodell des *lokalen Informationssystems B* hingegen wird mit Hilfe eines UML-Klassendiagramms dargestellt. Im Zuge der Integration wurden Korrespondenzen zwischen dem Schemakonzept *Entity X* und *Class B* sowie dem *Concept* aus dem *globalen Informationssystem* ermittelt. Weiter wurden auch Korrespondenzen zwischen Instanzen der Schemakonzepte bestimmt.

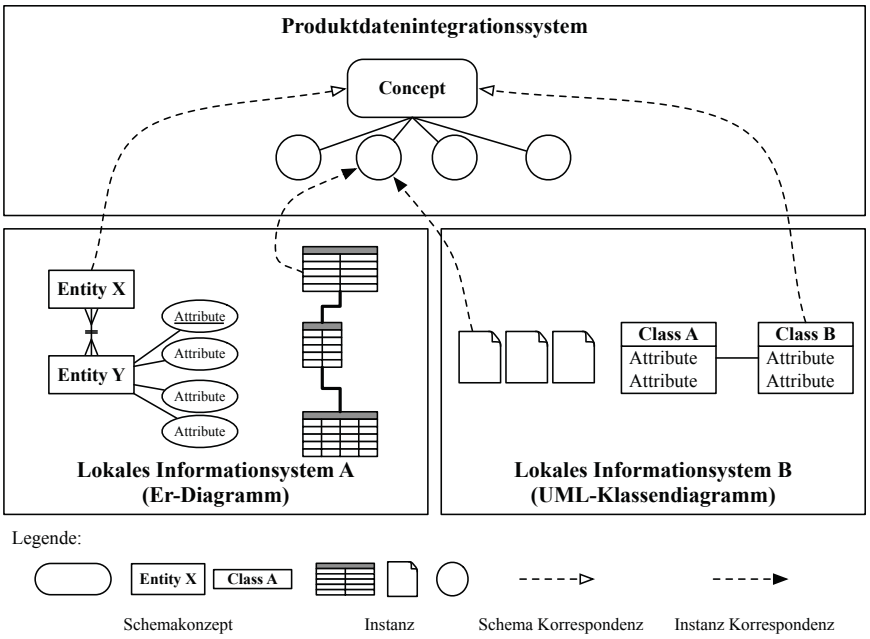


Abbildung 2.10.: Terminologiebeispiel

2.3.2. Klassifikation

[LN06] definiert die *Informationsintegration* als *„... die korrekte, vollständige und effiziente Zusammenführung von Daten und Inhalt verschiedener, heterogener Quellen zu einer einheitlichen und strukturierten Informationsmenge zur effektiven Interpretation durch Nutzer und Anwendungen.“* Während in diesem Fall die Informationsintegration nur die Integration von Daten umfasst, definieren andere Autoren weitere Ebenen der Integration [LBK07]. [Was90] führt fünf Ebenen der Tool-Integration ein. Die *Plattform-Integration* ermöglicht Interoperabilität verschiedener Systeme, in dem diese auf einer virtuellen Operationsschicht laufen. Mit der *Präsentationsintegration* wird ein einheitliches und konsistentes ”Look and Feel”geschaffen. Für die Datenintegration schlägt [Was90] ein gemeinsames Repository vor, welches den Datenaustausch zwischen Systemen ermöglicht. Mit *Kontrollintegration* wird die Fähigkeit eines Systems bezeichnet, sich über Ereignisse zu benachrichtigen und Funktionen anderer Applikationen auszulösen[12]. Schließlich sorgt die *Prozessintegration* für einen einheitlichen Prozess zur Steuerung der Aktivitäten über die unterschiedlichen Systemgrenzen hinweg. Abbildung 2.11 verdeutlicht die unterschiedlichen Ebenen der Tool-Integration. Die verschiedenen Ebenen müssen jeweils unter dem Aspekt technischer, struktureller und semantischer Heterogenität betrachtet werden.

Abbildung 2.11.: Verschiedene Ebenen der Integration

Im weiteren Verlauf der Arbeit wird auf die Schema- und Datenintegration fokussiert, während die anderen Ebenen nicht im Fokus stehen.

2.3.3. Integrationsprozess

Für die Schema- und Datenintegration definiert [BN09] folgende generischen Integrationsprozess (siehe Abbildung 2.12): Zunächst wird ein Mapping (siehe Definition 14) zwischen den zu integrierenden Datenquellen definiert. Anschließend erfolgt die *Duplikaterkennung* der Datensätze, bei dem Artefakte, welche die selben Realweltobjekte repräsentieren, einander zugeordnet werden. Danach erfolgt die *Datenfusion*, das heißt das Zusammenführen der Attributwerte in eine einheitliche Form. Schließlich können die integrierten Daten durch Applikationen genutzt werden. Zu jeder Phase existiert eine Vielzahl an wissenschaftlichen Ansätzen, die in Kapitel 4 ausführlich diskutiert werden.

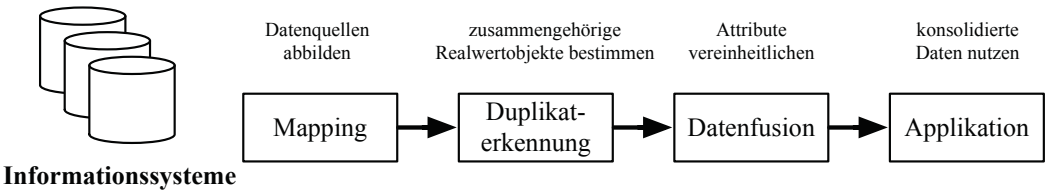


Abbildung 2.12.: Generischer Integrationsprozess

[12]In anderen Arbeiten wird an dieser Stelle auch von Funktionsintegration gesprochen.

2.3.4. Architekturen

Integrationsansätze für Schemata und Daten unterscheiden sich in der Art, wie die Integration der Daten erfolgt. Hier muss zwischen *virtuellen* und *materialisierenden* Ansätzen unterschieden werden. Abbildung 2.13 zeigt die zwei grundlegenden Architekturen für die Integration heterogener Datenquellen [DHI12].

- Bei Ansätzen der ersten Kategorie (siehe Abbildung 2.13 ⓐ) werden Daten aus autonomen, heterogenen Informationssystemen in ein gemeinsames, zentrales Informationssystem kopiert und dort integriert. Anfragen an das zentrale Informationssystem werden nur auf den integrierten Daten ausgeführt.
- Ansätze der zweiten Kategorie (siehe 2.13 ⓑ) kopieren keine Daten, sondern belassen diese im ursprünglichen Informationssystem. Anfragen werden an ein gemeinsames Schema über ein *Integrationssystem* gestellt; anschließend zerlegt diese die Anfrage und erzeugt Abfragen an die einzelnen heterogenen Informationssysteme. Letztere verarbeiten die Abfragen lokal und schicken die Ergebnisse an das Integrationssystem zurück. Dieses ist schließlich für die Zusammenführung bzw. Integration der Daten verantwortlich.

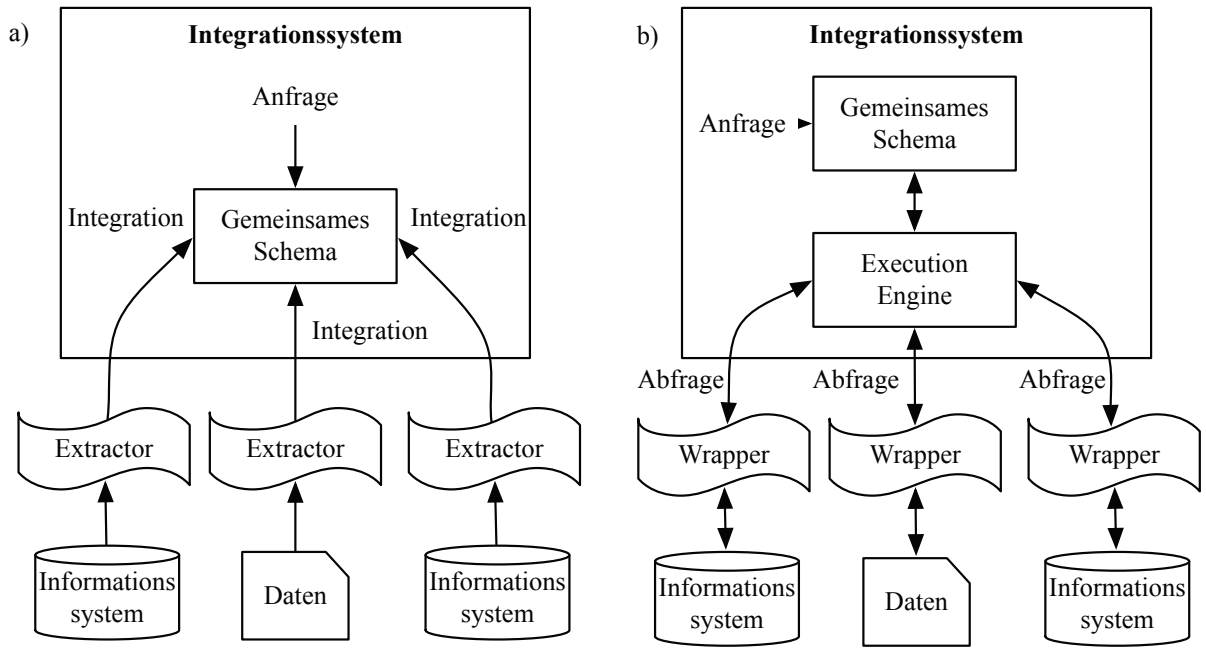


Abbildung 2.13.: Integrationsarchitekturen nach [DHI12]

Die beiden Ansätze haben ihre Vor- und Nachteile. Da bei materialisierten Ansätzen, neue Datensätze erst integriert werden müssen und virtualisierte Ansätze nur mit Quelldaten arbeiten, ist die Aktualität bei Ansätzen der zweiten Kategorie besser. Auf der anderen Seite müssen die Anfragen bei virtuellen Ansätzen zunächst durch lokale Informationssysteme verarbeitet werden, wodurch es zu einer Verzögerung bei Anfragen kommt. Materialisierte Ansätze besitzen gegenüber den virtuellen Ansätzen einige Vorteile. So sind die Daten *vollständig* integriert und können vorher um Fehler bereinigt werden, wodurch die Informationsqualität höher sein kann

als bei virtuellen Ansätzen. Anfrageplanung und -bearbeitung können bei virtuellen Ansätzen sehr komplex werden und die Vollständigkeit der Integration kann auf Grund der höheren Autonomie nicht garantiert werden. Fällt ein Informationssystem aus oder kann eine Anfrage des Integrationssystems nicht verarbeiten, fehlen diese Informationen in der Integration.

Ist die Datenqualität der heterogenen Informationssysteme sehr unterschiedlich, können die Integrationssysteme in beiden Architekturen die Duplikaterkennung und Datenfusion nicht vollständig automatisieren. In diesem Fall ist manuelle Unterstützung durch Endanwender in beiden Ansätzen erforderlich.

2.3.5. Integrationsstrategien

Sobald mehrere Quellen integriert werden sollen, gibt es verschiedene Strategien, die in Abbildung 2.14 illustriert sind. Die erste Möglichkeit besteht darin, alle vorhandenen Quellen gleichzeitig zu integrieren (siehe Abbildung 2.14 a), d.h. zusammengehörige semantische Konzepte aller Schemata gleichzeitig zu betrachten. Bei sehr vielen Quellen mit einer Vielzahl an Konzepten kann diese Vorgehensweise einen Integrationsverantwortlichen mit zu vielen Informationen überfordern. Auf der anderen Seite kann die Betrachtung aller Schemata auf einmal die Möglichkeit bieten, Gemeinsamkeiten zwischen verschiedenen Schemata zu identifizieren. Eine weitere Möglichkeit ist die binäre Integration von Quellen (siehe Abbildung 2.14 b). Dazu werden jeweils zwei Quellen zu einem Zwischenschema integriert, und zwei Zwischenschemata wiederum zu einem neuen Schema. Dieses Problem wiederholt sich solange, bis nur noch ein Schema übrig bleibt. Der Vorteil ist, dass wesentlich weniger Konzepte auf Gemeinsamkeiten hin analysiert werden und Integrationsverantwortliche mit weniger Komplexität zurecht kommen müssen.

Da nicht jedes Schema mit jedem anderen Schema direkt verglichen wird, kann es vorkommen, dass Zusammenhänge zwischen Schemakonzepten unterschiedlicher Quellen übersehen werden können. Eine abgewandelte Form der binären Integration ist die gewichtete Integration (siehe Abbildung 2.14 c). Dabei wird ein Schema identifiziert, welches am wichtigsten für die Integration ist, da es z.B. die meisten Konzepte einer Domäne enthält. Bei diesem Vorgehen wird dieses Schema schrittweise mit einem weiteren Schema zu Zwischenschemata integriert. Letzteres wird dann mit dem nächsten Schema zum nächsten Zwischenschema integriert. Dieser Vorgang wird solange wiederholt, bis alle Schemata integriert sind.

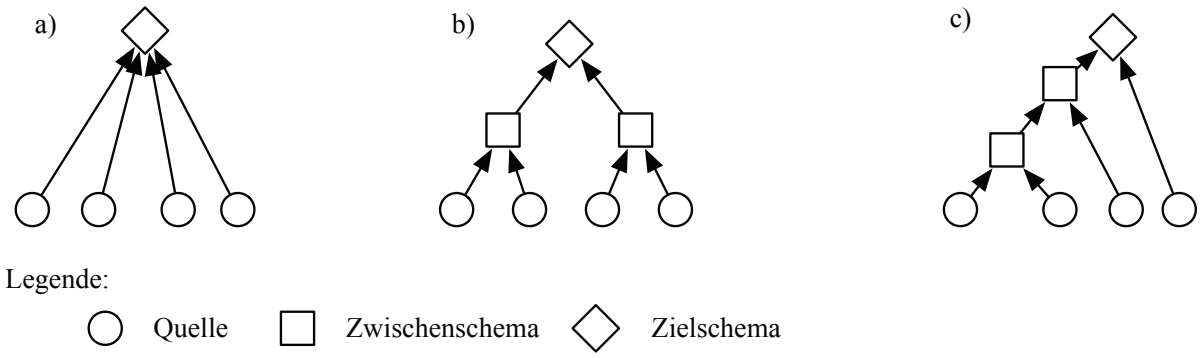


Abbildung 2.14.: Integrationsstrategien

2.3.6. Integrationskonflikte

Häufigstes Problem bei der Integration von Informationssystemen sind unterschiedliche Arten von Heterogenität. *„Heterogenität herrscht, wenn sich zwei miteinander verbundene Informationssysteme syntaktisch, strukturell oder inhaltlich unterscheiden.“* [LN06]. Die Gründe für Heterogenität sind vielfältig und durch unterschiedliche Faktoren bedingt: Informationssysteme basieren auf unterschiedlichen technischen Plattformen (etwa .NET, J2EE), wodurch *technische Heterogenität* entsteht. Zu letzterer gehören auch die Verwendung unterschiedlicher Kommunikationsprotokolle (z.B. CORBA, Web Services) oder verschiedene Schnittstellen. Darüber hinaus werden Informationssysteme durch unterschiedliche Datenmodelle (relational, objektorientiert oder hierarchisch) realisiert. Innerhalb dieser können sowohl *strukturelle* als auch *semantische* Heterogenität auftreten. Sowohl bei der virtuellen als auch bei der materialisierten Integration führen diese Arten von Heterogenität zu unterschiedliche Konflikten. Diese können sowohl auf der Schema- als auch Instanzebene auftreten.

Schemakonflikte

Schemakonflikte betreffen die erste Phase des generischen Integrationsprozesses (siehe Abschnitt 2.3.3). In diesem Schritt werden Abbildungen zwischen Konzepten heterogener Datenmodelle definiert. Zu den Schemakonflikten zählen alle Heterogenitätsprobleme, die bei der Abbildung von Schemakonzepten aus unterschiedlichen Quellen resultieren können. Im Folgenden werden die wichtigsten Konflikte kurz erläutert.

Beispiel 11 (Schemakonflikte). In Abbildung 2.15 sind die konzeptionellen Schemata zweier Informationssysteme illustriert, die überlappende Konzepte besitzen. Im Speziellen dokumentieren beide Schemata die Bauteilaspekte eines Produktes. Während *Informationssystem A* alle Bauteile (mechanische, elekromechanische, elektronische und elektrische) dokumentiert, werden in *Informationssystem B* lediglich elektromechanische, elektrische und elektronische Bauteile verwaltet.
Neben der technischen Heterogenität beider Informationssysteme[a] treten unterschiedliche Schemakonflikte bei der Integration beider Schemata auf. Der Bauteilname im relationalen Schema `A` ist *synonym* mit dem Bezeichner (`label`) von `Component` in `B`. Im Speziellen ist `Component` eine Teilmenge von `Bauteil`.
Während der Herstellername eines Bauteils in `A` mittels einer Entität modelliert ist, wird dieser in Schema `B` durch ein Attribut realisiert. Das Bauteilgewicht wird in `A` in Gramm angegeben, während die Einheit Unzen in Schema `B` verwendet wird. Auf der anderen Seite wird der Verantwortliche eines Bauteils in `A` durch zwei Attribute (`RespName`, `RespFirst`) in `B` modelliert.

[a] Informationssystem A basiert auf dem relationalen Schema und wird mit einem Datenbankmanagementsystem realisiert während Informationssystem B in UML modelliert ist und Instanzen in Dateien persistiert.

Das Mapping (siehe Abschnitt 2.3.1) zwischen `A` und `B` ist in Tabelle 2.1 informell dargestellt. Während die Schemakonzepte aus `A` und `B` aufeinander abgebildet werden konnten, ist keine der Korrespondenzen zwischen `A` und `B` ausreichend, um deren Instanzen einander eindeutig zuordnen zu lassen. Einzige Möglichkeit in diesem Fall wäre die Definition einer Mapping-Tabelle zwischen der Bauteilnummer aus `A` und der ID von `Component` aus `B`.

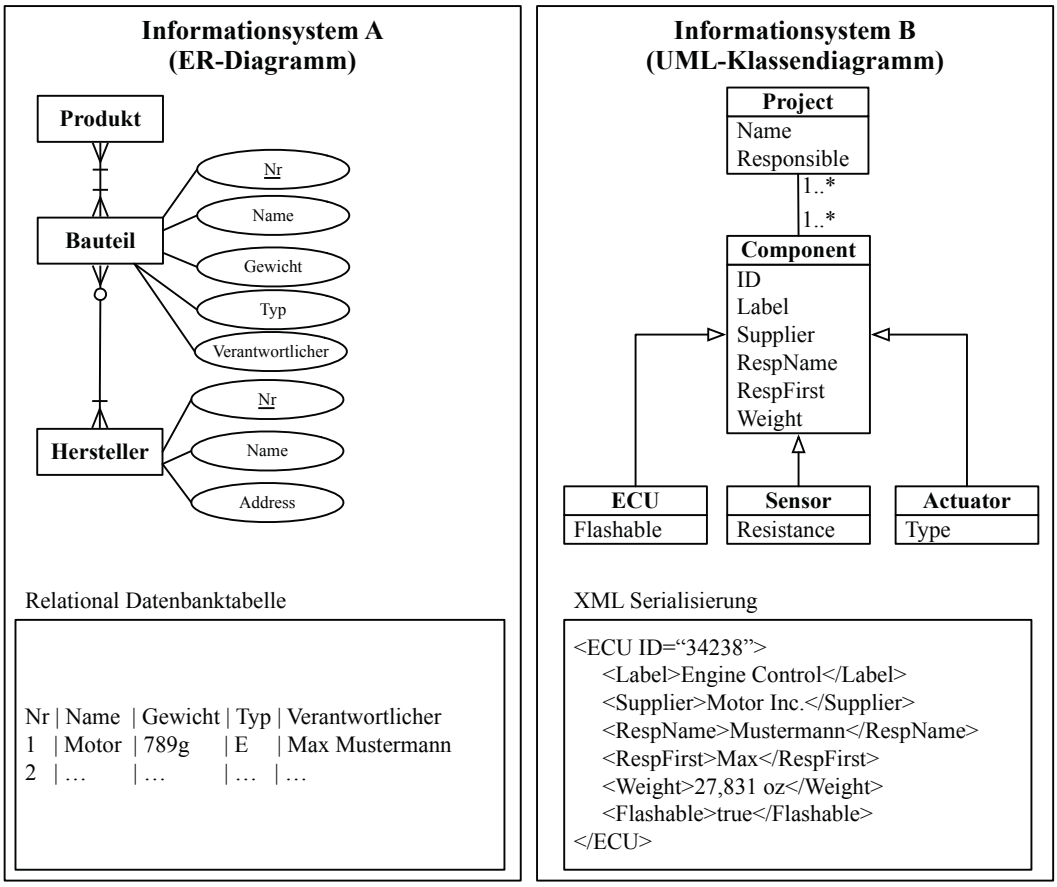


Abbildung 2.15.: Beispiele für strukturelle und semantische Heterogenität

Im obigen Beispiel treten eine Vielzahl an Schemakonflikten auf. So ist die Benennung zwischen synonymen Konzepten und Attributen unterschiedlich. Weiter werden gleiche Sachverhalte unterschiedlich modelliert. So ist der Herstellername in `A` ein Attribut eines eigenen Konzeptes während dieser in `B` lediglich als Attribut desselben Konzeptes modelliert wird.
Neben den bereits erläuterten Integrationskonflikten existieren noch weitere. So können Tabellen mit gleichem Namen unterschiedliche semantische Inhalte besitzen (die Namen werden in diesem Fall als *Homonyme* bezeichnet). Auch wenn zwei Tabellen die gleiche Bezeichnung haben und den selben Inhalt beschreiben, können Strukturprobleme auftreten. Attribute die in einem Konzept dokumentiert sind, können im synonymen Konzept fehlen oder implizit modelliert sein. Schließlich können die Integritätsbedingungen von Konzepten in den Quellen unterschiedlich

Informationssystem A	**Beziehung**	**Informationssystem B**
Bauteil	beinhaltet	Component
Bauteil.Name	gleich	Component.RespFirst + Component.RespName
Bauteil.Gewicht	gleich	Component.Weight * 0.035274
Hersteller.Name	gleich	Component.Supplier

Tabelle 2.1.: Mapping zwischen `Informationssystem A` und `B`

sein. Für eine detaillierte Diskussion der weiteren Integrationskonflikte bei der Integration von relationalen Datenmodellen wird auf [KS91] verwiesen.

Datenkonflikte

Neben den Konflikten auf der Schemaebene treten auch Konflikte bei der Integration von Instanzen auf. Daten können in unterschiedlichen Einheiten repräsentiert sein (z.B. Gewicht, Länge, Zeit, Währungen). Müssen Einheiten untereinander konvertiert werden, können diese durch Genauigkeitsverluste verfälscht werden. Häufiges Beispiel für die Repräsentation von gleichen Sachverhalten sind Adressdaten [LN06], die unterschiedlich geschrieben werden (z.B. Karlstrasse, Karl Str. und Karl Strasse). Weiter können Daten falsch dokumentiert sein, etwa wenn sie manuell übertragen wurden (z.B. durch Tippfehler). Schließlich tritt Heterogenität zwischen Datensätzen auf, wenn diese zu unterschiedlichen Zeitpunkten aktualisiert werden bzw. Aktualisierungen nicht an nachfolgende Informationssysteme propagiert werden.
[BS06] definiert zwei Typen von Datenkonflikten: Ein *key*-Konflikt tritt bei der Integration von zwei Relationen auf, wenn die Attributwerte von Instanzen in beiden Quellen identisch sind, sich jedoch dessen Primärschlüssel unterscheiden. Existieren im umgekehrten Fall von Instanzen mit gleichen Primärschlüssel und unterschiedlichen Werten für weitere Attribute, handelt es sich um *attribute*-Konflikte. Während letztere relativ einfach zu identifizieren sind, müssen im ersten Fall die Attributwerte aller Instanzen aus beiden Relationen miteinander verglichen werden, um möglich Konflikte zu identifizieren.

Beispiel 12 (Datenkonflikte). Abbildung 2.16 zeigt ein Beispiel für Attribut- und Schlüsselkonflikte. `Tabelle A` und `Tabelle B` sollen integriert werden, und es wird angenommen, dass keine schematischen Konflikte zwischen den beiden Tabellen existieren. Für die Instanzen mit der `Nr. 268` tritt ein Konflikt für die korrespondierenden Attribute `Verantwortlicher` und `Responsible` auf, da in beiden Tabellen verschiedene Werte vorhanden sind. Des weiteren korrespondieren der Eintrag mit der `Nr 921` aus `Tabelle A` zum Eintrag mit der `Nr 931` aus `Tabelle B`, da sämtliche Attribute identische Werte besitzen. Da beide Instanzen unterschiedliche Primärschlüssel besitzen, liegt in diesem Fall ein Primärschlüsselkonflikt (Key-Konflikt) vor.

Semantische Konflikte

Semantische Heterogenität ist ein Begriff, der in vielen Disziplinen unterschiedlich verwendet wird [Col97]. Nach [Ver97] umfasst diese Form alle Unterschiede in Bedeutung, Interpretation und Art der Nutzung. Nach [LN06] liegt semantische Heterogenität vor, *„wenn unterschiedliche Elemente des Datenmodells verwendet werden, um denselben Sachverhalt zu modellieren.“*. Als Beispiel geben die Autoren das relationale Datenmodell an, bei dem derselbe Sachverhalt entweder durch eine Relation, ein Attribut oder einen Attributwert modelliert werden kann.

Beispiel 13 (Semantische Heterogenität). Abbildung 2.17 zeigt ein Beispiel für semantische Heterogenität. Die Entitäten in a), b) und c) beschreiben den selben Sachverhalt (den Typ einer Komponente), jedoch mit unterschiedlichen Mitteln des relationalen Datenmodells. In a) wird für jeden Typ eine eigene Entität modelliert, während in b) die vorhan-

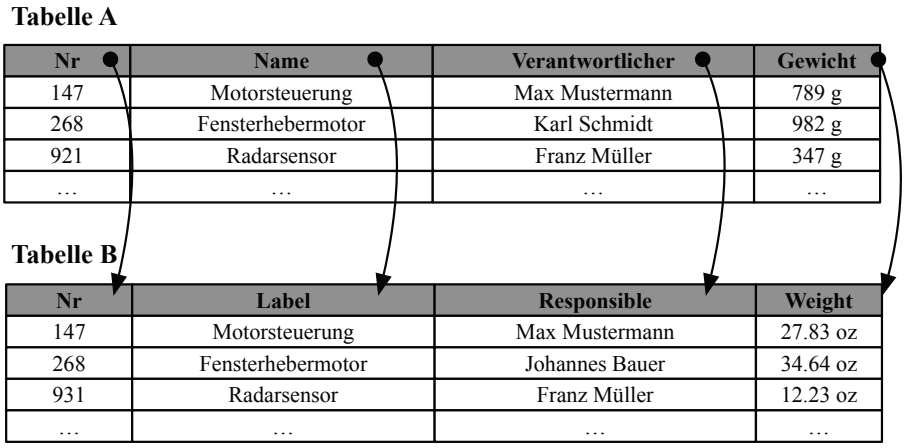


Abbildung 2.16.: Beispiele für Attribut- und Schlüsselkonflikte

denen Typen als Attribut einer Entität ausgedrückt werden. In c) wird der Typ durch einen Attributwert realisiert.
Ein weiteres Beispiel für semantische Heterogenität wurde bereits im obigen Beispiel (siehe Abbildung 2.15) angesprochen. Die Begriffe `Bauteil` und `Component` stehen synonym für einander. Jedoch ist ihre Bedeutung unterschiedlich. Während der erste Begriff alle Bauteile eines Produktes beschreibt, sind mit dem zweiten Begriff nur diejenigen Bauteile gemeint, die einen E/E-Anteil besitzen.

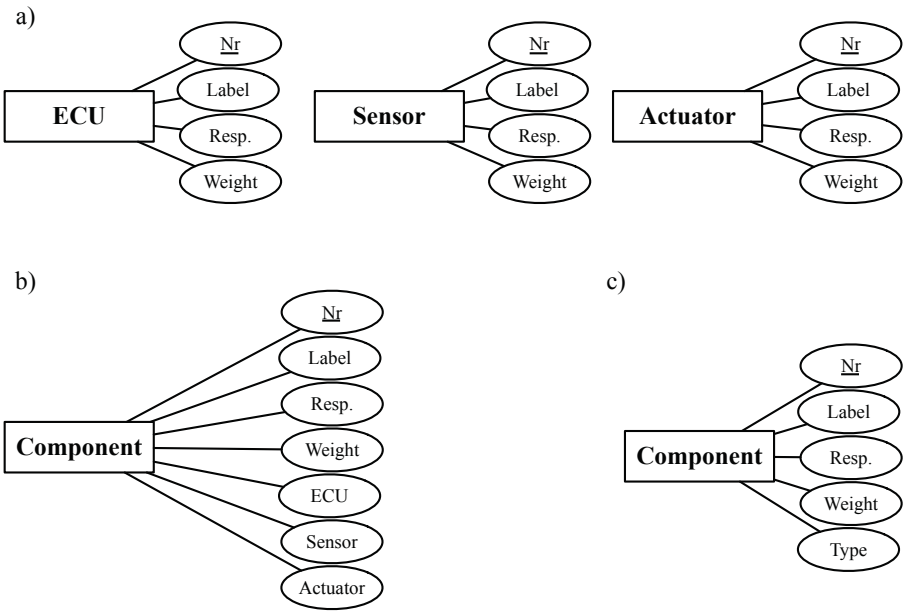


Abbildung 2.17.: Beispiele für semantische Heterogenität (relationales Datenmodell)

2.4. Informationsqualität

Nachdem die grundlegenden Integrationsbegriffe erläutert wurden, werden nun fundamentale Begriffe zur Beschreibung und Bestimmung von Informationsqualität beschrieben. Es existiert kein einheitlicher Begriff für die Qualität von Daten und Informationen. Neben der *„fitness for use“* nach [JG99], stammt die verbreiteste Definition aus der ISO 9000-Normreihe und ist sehr abstrakt gehalten. Sie bezeichnet die Qualität von Daten, als den Grad den eine Menge an Charakteristiken gegebene Anforderungen erfüllt [Hoy09].
Bei der Kategorisierung von Qualitätskriteren erfolgt eine Unterscheidung in die verschiedenen Ebenen, die bei der Integration von Produktdaten relevant sind. Dazu zählen neben der Schemakonzept- und Instanzebene auch die Ebene der Korrespondenzen zwischen Schamkonzepten sowie zwischen Instanzen.

2.4.1. Schemakonzeptqualität

[Red97] definiert folgende Dimensionen für die Qualität eines konzeptionellen Schemas: *Korrektheit* bezieht sich sowohl auf das Modell, dass heißt, ob ein Datenmodell den syntaktischen Vorgaben entspricht, als auch auf die korrekte Abbildung von Anforderungen der Realität im Modell. Im Falle des relationalen Datenmodells sind damit unter anderem Kardinalitäten und Integritätsbedingungen gemeint. Ferner ist ein Schema *minimal*, wenn alle Anforderungen im Modell repräsentiert sind und keine Elemente entfernt werden können, ohne dass dadurch Anforderungen unberücksichtigt bleiben. Die *Vollständigkeit* misst den Grad, den ein konzeptuelles Schema für spezifische Anforderungen erfüllt. Die *Relevanz* bestimmt die Anzahl an unnötigen konzeptuellen Elementen im Modell. Die visuelle Repräsentation eines konzeptionellen Schemas wird als *lesbar* bezeichnet, wenn bestimmte ästhetische Eigenschaften, wie etwa Kompaktheit der Darstellung, gegeben sind. Schließlich sind für das relationale Datenmodell unterschiedliche Normalformen auf Basis funktionaler Abhängigkeiten zwischen Relationen definiert. Diese können ebenfalls zur Betrachtung der Schemaqualität in Betracht gezogen werden.

Definition 15 (Schemakonzeptqualität). Schemaqualität beschreibt den Grad der Erfüll-ung von gegebenen Charakteristika, die ein Schema zu einem Zeitpunkt besitzt.

2.4.2. Instanzqualität

Instanz- bzw. Datenqualität ist ein aktives Forschungsgebiet, in dem eine Vielzahl wissenschaftlicher Arbeiten veröffentlich wurden. Es existieren unterschiedliche Klassifikationssysteme und Terminologien, und die Datenqualitätseigenschaften werden von den Autoren unterschiedlich definiert [WS96, BS06, KCH+03, WZL06, ORH05].
Mit der Normreihe ISO 8000 [Ben09] entsteht ein neuer Standard für die Qualität von Stammdaten, der sowohl Datenqualitätskriterien (Part 110: Master data: Exchange of characteristic data: Syntax, semantic encoding, and conformance to data specification) als auch Prozesse für das Datenqualitätsmanagement (ISO 8000:150–A frameworkfor Data Quality Management) definiert. Eine detaillierte Betrachtung des Standards erfolgt in Abschnitt 4.2.5.
Qualitätsdimensionen für Instanzen beziehen sich auf Fehler einzelner oder mehrerer Attributwerte. Beispiel für einen solchen Fehler ist z.B. ein Schreibfehler eines Namens oder einer Adresse, der entweder bei der manuellen Eingabe durch einen Endanwender enstanden ist oder auf Fehler

bei der automatischen Texterkennung zurückzuführen ist. [LN06] beschreibt eine Liste weiterer Fehler, wie z.B. fehlende Attributwerte, falsche Werte, falsche Referenzen oder eingebettete Werte. In [BS06] werden unter anderem die Dimensionen *Richtigkeit*, *Vollständigkeit*, *Konsistenz* und *Aktualität* erläutert.

Definition 16 (Instanzqualität). Instanzqualität beschreibt den Grad der Erfüllung von Charakteristika, die eine Menge an Instanzen zu einem Zeitpunkt besitzt.

2.4.3. Produktdatenqualität

[SAS05] geht einen Schritt weiter und definiert Produktdatenqualität als Maß der Genauigkeit (engl. accuracy) und Angemessenheit (engl. appropriateness) von Produktdaten, kombiniert mit der Aktualität mit der diese Daten allen benötigten Personen zur Verfügung gestellt werden. Qualität bezieht sich hier vor allem auf die Qualität von geometrischen Modellen.
Entscheidend für die Integration heterogener Produktdaten ist deren Datenqualität. In der Realität ist die Qualität von Produktdaten und Daten im Allgemeinen sehr unterschiedlich und von verschiedenen Faktoren abhängig [HS98]. So nimmt die Datenqualität von Produktdaten in der Regel mit zunehmender Reife zu, während sie zu Beginn der Entwicklung viele Informationen fehlen. Weitere Datenqualitätsprobleme des Concurrent Engineerings werden in [BLL10] erläutert. So können Attribute von Instanzen fehlen, falsche Werte besitzen oder falsch dokumentiert sein. Die Gründe dieser Probleme sind vielfältig [LN06]. Bei der automatischen Erfassung, z.B. durch OCR-Software, oder falsche manuelle Eingaben können fehlerhafte Werte entstehen. Werden Datenbestände nicht kontinuierlich aktualisiert, altern diese und spiegeln nicht mehr die Realität wieder. Bei der Transformation unterschiedlicher Formate und Bereiche können Informationen verloren gehen. Neben Datenqualitätsproblemen, die in einzelnen Datenquellen auftreten können, können bei der Integration weitere Probleme auftreten [LN06].

Definition 17 (Produktdatenqualität). Produktdatenqualität bezeichnet das Verhältnis von zuvor definierten Charakteristika (z.B. Vollständigkeit, Genauigkeit, Konsistenz) mit definierten Zielerreichungswerten im Verhältnis zur tatsächlichen Erfüllung der Charakteristika zu einem bestimmten Zeitpunkt. Charakteristika für Produktdaten sind abhängig vom Produktdatenaspekt und die Zielerreichungswerte von der Reife eines Produktes.

2.4.4. Integrationsqualität

Bisher wurden nur Dimensionen und Fehler einzelner Informationssysteme betrachtet. Werden Schematkonzepte und Instanzen integriert, ergeben sich weitere Fehler und Qualitätsdimensionen. Wie diskutiert, treten bei der Integration heterogener Schemata strukturelle, schematische und semantische Probleme auf. Werden Instanzen mit den gleichen Attributen integriert, können Duplikate bei der Integration entstehen. Werden gemeinsame Attribute geteilt, können unterschiedliche Attributkonflikte auftreten. Attribute können widersprüchliche Werte besitzen, unterschiedliche Einheiten oder Genauigkeiten aufweisen oder verschieden repräsentiert sein.
Während die genannten Qualitäten Charakteristiken betrachtet, die für einzelne Informationssystem isoliert betrachtet werden, fokussiert die Integrationsqualität auf Charakteristiken (etwa Vollständigkeit, Konsistenz), die für die Bewertung der Produktdatenintegration von mehreren Informationssystemen notwendig sind.

Definition 18 (Integrationsqualität). Integrationsqualität bezeichnet das Verhältnis von zuvor definierten Charakteristika (z.B. Vollständigkeit, Genauigkeit, Konsistenz) mit definierten Zielerreichungswerten im Verhältnis zur tatsächlichen Erfüllung der Charakteristika zu einem bestimmten Zeitpunkt. Charakteristika für die Integrationsqualität betrachten das Verhältnis von mehreren zu integrierenden Informationssystemen.

3. Anforderungsanalyse

Dieses Kapitel leitet relevante Anforderungen zur Realisierung eines PDIS her. Zunächst präsentiert Abschnitt 3.1 eine Literaturstudie zu den Herausforderungen bei der Integration von Produktdaten. Anschließend werden in Abschnitt 3.2 die IT-Landschaft für die Entwicklung von E/E-Komponenten bei einem Automobilhersteller analysiert und anhand zweier Fallstudien entsprechende Herausforderungen und Anforderungen abgeleitet. In Abschnitt 4.2 werden Standards im Umfeld der Produktentwicklung analysiert. Schließlich fasst Abschnitt 3.3 die Anforderungen zusammen.

3.1. Literaturstudie

In Tabelle 3.1 sind die fünf zu untersuchenden Forschungsfragen aus Abschnitt 1.3 mit Stichwörtern aufgelistet, die für die Ermittlung relevanter wissenschaftlicher Literatur verwendet wurden. Diese Stichwörter wurden anschließend dazu genutzt, potenzielle wissenschaftliche Literatur in folgenden Suchmaschinen zu identifizieren:

- Google Scholar (`http://scholar.google.com`)
- SpringerLink (`http://link.springer.com`)
- IEEE xplore (`http://ieeexplore.ieee.org/Xplore`)
- ACM Digital Library (`http://dl.acm.org/`)

Die Suchergebnisse wurden anschließend gesichtet und die gewonnen Erkenntnisse zusammengefasst. Danach wurden relevante Anforderungen abgeleitet. Im Folgenden werden die Erkenntnisse der Literaturstudie zusammengefasst.
Insbesondere [Sta11] listet eine Reihe von Problemen auf, die im Zusammenhang von Produktdaten auftreten. Die Heterogenität von Bezeichnern, Produktstrukturen, und Repräsentation sowie unterschiedliche Abstraktionsstufen werden als Herausforderungen für die Integration von Produktdaten genannt. Zusätzlich erschweren die mangelnde Datenqualität (inkorrekte und inkonsistente Daten, Überlappungen, Duplikate) sowie implizites Wissen die Integration zusätzlich.
[VSW15] kommt zu folgenden Erkenntnissen: Um komplexe System zu realisieren, werden Verantwortliche aus unterschiedlichen Domänen (Hardware, Software, Service) benötigt, die während der Entwicklung dynamische und teils widersprüchliche Anforderungen produzieren. Entwicklungsaufgaben werden mit Hilfe von Concurrent Engineering realisiert, zunehmends verschiebt sich die Entwicklung in Richtung Digital Mock-Up (DMU). Insgesamt ist ein Trend zu lokalen, lose gekoppelten Modellen in föderierten Umgebungen zu beobachten.
Nicht nur die Zunahme an Produktdaten insgesamt, sondern auch die wachsende Komplexität infolge von Vernetzung komplexer Komponenten/Bauteile sowie ein fehlender Systemgedanke fördern Heterogenität. Neben diesen technischen Probleme besitzt die Integration zusätzlich eine

Nr.	Forschungsfrage	Stichworte
1	Wie lassen sich komplexe Produktdaten aus heterogenen und autonomen Informationssystemen in anspruchsvollen Entwicklungsumgebungen integrieren, die unterschiedliche Konzepte zur Repräsentation von Variabilität und Versionierung aufweisen?	„product data“ „integration“ „challenges“
2	Wie lässt sich die Qualität (z.B. Vollständigkeit und Konsistenz) der Integration bestimmen?	„integration“ „quality“
3	Wie beeinflussen strukturelle und inhaltliche Änderungen bereits integrierte Produktdaten?	„integration“ „changes“
4	Mit Hilfe welcher Technologie lässt sich die Integration der heterogenen Produktdaten umsetzen?	„integration“ „architecture“
5	Wie lässt sich die Wirtschaftlichkeit der Produktintegration bestimmen?	„integration“ „investment“

Tabelle 3.1.: Forschungsfragen für die Anforderungsanalyse

soziale Komponente, etwa die Bereitschaft implizites Wissen mit anderen zu teilen. Schließlich erfolgt der Austausch von Produktdaten auf Basis einer Vielzahl komplexer Standards (z.B. STEP[1], AUTOSAR[2], JT[3]).

Daraus leiten die Autoren folgende Anforderungen ab: Die Qualität von Produktdaten und deren Integration müssen kontinuierlich überwacht werden. Um Datenmodellheterogenität zu überwinden und Produktdaten unterschiedlicher Hersteller integrieren zu können, werden ein standardisiertes Lieferantenintegrationsportal sowie Investitionen in entsprechende Prozesse benötigt. Schließlich sind „natürliche Benutzerschnittstellen“ für die Interaktion mit Produktdaten notwendig.

Auch [TRS15] sieht die zunehmende Funktionalität und Komplexität als Herausforderungen in der Produktentwicklung. Während für das Stücklisten- und Produktdatenmanagement teils homogene Architekturen entlang des gesamten Entwicklungsprozesses etabliert sind, werden Daten in einer heterogenen Systemlandschaft gespeichert. Vor allem im *Team Data Management (TDM)* ist eine hohe Heterogenität mit bis zu 1000 unterschiedlichen Applikationen vorhanden. Die große Anzahl an heterogenen Systemen entsteht, um neue Technologien und Entwicklungen zu unterstützen. Während die Integration zu definierten Synchronisationspunkten erfolgt, existiert für Methoden, Systeme und Datenmodelle kein unternehmensweites Schema. Zusammengefasst kommen die Autoren zu folgenden Anforderungen: Neben der Integration von Benutzeroberflächen muss vor allem eine konsistente Produktstruktur etabliert werden. Insgesamt muss die Integration von Daten und Prozessen sowohl auf technischer als auch organisatorischer Ebene erfolgen. Technisch sind hierfür offene und standardisierte Schnittstellen erforderlich.

[Kat15] sieht zusätzlich eine Reduktion der Entwicklungsdauer zwischen 30 und 50 Prozent in den letzten 30 Jahren. Auch haben sich in den letzten Jahren die Zusammenarbeitsmodelle deutlich geändert. Nicht nur, dass immer mehr Entwicklungsaufgaben durch Zulieferer übernommen werden, auch Kooperationen zwischen großen Herstellern rücken vermehrt in den Fokus. Dadurch muss der Austausch zwischen unterschiedlichen Produktdatenmanagementsystemen (PDMS) er-

[1] Standard for the Exchange of Product data
[2] AUTomotive Open System ARchitecture
[3] Jupiter Tessellation

möglicht werden. Neben etablierten Systemen, etwa dem Stücklistenmanagement, welches bis zu 500 Schnittstellen zu anderen Systemen haben kann, besteht die IT-Landschaft im Automobilbereich aus einer Vielzahl gewachsener proprietärer Applikationen, kleinen speziellen und selbstentwickelten Applikationen sowie großen Standardapplikationen. In diesem Zusammenhang ist eine Konsolidierung oder Änderung der Systemlandschaft schwierig, da die Nutzung dieser Systeme integraler Bestandteil des spezifischen Entwicklungswissens eines Unternehmens sei. Als Lösung des Problems wird eine Basis-IT-Infrastruktur für die Entwicklung von Systemen vorgestellt, die neben Dokumenten-, Konfigurations-, Versions-, Release- sowie Regeln- und Rechtemanagement vor allem aus einem zentralen Integrationsbus besteht. Für den Autor ergeben sich folgende Herausforderungen im Zusammenhang mit Produktdaten: Aufgrund der zunehmenden Bedeutung von Zulieferern ist die Integration von deren Informationssysteme von entscheidender Bedeutung. Da Änderungen der IT-Landschaft aufwändig sind, muss diese Schritt für Schritt, also bedarfsgerecht, geschehen (hier wird auf eine Service-orientierte Architektur gesetzt, bei der die IT-Landschaft modularisiert aber nicht umstrukturiert werden soll). Dazu werden Standards wie STEP AP242 (siehe Abschnitt 4.2.3) und JT[4] eingesetzt. In diesem Zusammenhang wird der *Code of PLM Openness* hervorgehoben. Zuletzt werden semantische Netzwerke als Möglichkeit diskutiert, um die Nachvollziehbarkeit zwischen verbundenen Objekten zu erreichen.
Die skizzierten Erkenntnisse decken sich mit denen aus [CRS+13], dass CAD-Systeme nicht mehr ausreichend für die Dokumentation von Produktdaten seien. Die fehlende Repräsentation für Funktionen und das Verhalten von Komponenten verhindert eine Integration aller Produktdaten. Vor allem Analysen integrierter Produktdaten, etwa Impact-Analysen[5], lassen sich nicht vollautomatisch durchführen. Die Autoren leiten folgende Anforderungen ab: Interagierende Informationssysteme, die Wissen der Produktentwicklung speichern, müssen integriert werden. Dabei soll ein Paradigmenwechsel erfolgen, weg vom reinen Austausch von Produktdaten hin zu einem Austausch von Produktinformationen über unterschiedliche Disziplinen und Domänen hinweg entlang aller Phasen eines Produktes. Dazu ist die Entwicklung einer umfassenden Repräsentation von Produktentwicklungswissen erforderlich. Die Autoren empfehlen eine Integration der bereits vorhanden Systeme mit wissensbasierten Systemen, etwa auf Basis von Ontologien. Schließlich sollen diese Systeme nachhaltig sein, da die Einführung neuer Informationssysteme mit hohen Kosten und langen Übergangszeiten verbunden ist.
[RTW15] erläutern Herausforderungen im Zusammenhang mit der Integration von Variabilität während der Produktentwicklung. Variabilität tritt in jeder Phase der Produktentwicklung auf und wird oftmals in jeder Phase unterschiedlich modelliert. Nach Ansicht der Autoren fokussieren aktuelle Forschungsprojekte vor allem auf die Reduktion der externen, also durch den Kunden wahrnehmbare, Komplexität, während auf der anderen Seite die interne Variabilität durch Standardisierung und Modularisierung reduziert werden soll. Daraus ergeben sich folgende Anforderungen für das Variabilitätsmanagement bei der Produktentwicklung: Durch die zunehmende Komplexität muss das Variantenmanagement durch formale Logiken abgebildet werden. Da in den verschiedenen Phasen der Produktentwicklung unterschiedliche Modelle zum Einsatz kommen, wird ein „overall composed variability model“ benötigt, welches sehr groß und komplex werden kann. Für eine komplette Produktlinie kann ein solches Modell bis zu 200.000 Regeln enthalten.

[4] Jupiter Tesselation - ISO-Standard zur Speicherung von 3D-Daten.
[5] Damit sind Analysen gemeint, die Auswirkungen von potenziellen Änderungen im Voraus bestimmen, um letztere besser bewerten zu können.

[PNSD07] beschreibt das Qualitätsmanagement in der Produktentwicklung. Demnach haben die zunehmende Produktkomplexität und die kürzeren Entwicklungszeiten dazu geführt, dass die Qualität der Produktentwicklung nachgelassen hat. Der Autor sieht die Beherrschung der Komplexität von Hard- und Software als eine der Kernaufgaben der Automobilbranche an. Dazu sei die Überwachung des Produktentstehungsprozesses essentiell. Die Vollständigkeit und Richtigkeit seien vor allem wichtig für die Detaillierungsphase und müssen folglich effizient, d.h. möglichst automatisch, erfolgen.

3.2. Explorative Untersuchung

Neben der Sichtung der wissenschaftlichen Literatur wurde die Systemlandschaft für die Entwicklung von E/E-Bauteilen eines großen deutschen Automobilherstellers untersucht [THR13b, THR13a]. In diesem Kontext wurden Anwendungsfälle betrachtet, welche die Integration von Produktdaten aus autonomen, heterogenen Informationssystemen erfordern.

3.2.1. Anwendungsfall 1: Integration elektrischer und geometrischer Komponenten

Im ersten untersuchten Anwendungsfall wird die Integration von Produktdaten (mechanisch und elektrisch) zweier Informationssysteme untersucht.
Elektrische Komponenten eines Fahrzeuges können sowohl Sensoren als auch Aktuatoren seien. Ein Sensor (z.B. Temperatursensor) liefert die Eingabeinformationen für Steuergeräte (etwa das Motorsteuergerät), die wiederum Aktuatoren (z.B. Fensterhebermotor) ansteuern. Der Informationsaustausch zwischen Sensoren, Aktuatoren und Steuergeräten kann sowohl analog als auch digital erfolgen.
Im Informationssystem `E-PDM` werden elektrische Komponenten und deren Varianten verwaltet, während im Informationssystem `PDM` geometrische Modelle dieser Komponenten dokumentiert werden.
Auf Grund unterschiedlicher Ausstattungsmerkmale (siehe Abschnitt 2.2.3) und Gesetzgebungen existieren diese Komponenten in unterschiedlichen Varianten. Diese Variabilität kann sowohl auf elektrischer Ebene (verschiedene Software, Bauteile) als auch auf physischer Ebene (Geometrie) vorliegen. Während letztere zu verschiedenen Geometriemodellen im System `PDM` führen, werden erstere in `E-PDM` dokumentiert.
Der zu realisierende Anwendungsfall ist wie folgt: Für jede E/E-Komponente in `E-PDM` soll überprüft werden, ob ein entsprechendes geometrisches Modell in `PDM` vorhanden ist. Dabei sollen unterschiedliche Varianten berücksichtigt werden.
In Abbildung 3.1 sind Ausschnitte der Datenmodelle sowie Beispielinstanzen beider Informationssysteme dargestellt. `E-PDM` verwaltet für ein `Fahrzeugmodell` dessen `Komponenten`, die wiederum in unterschiedlichen `Komponentenvarianten` und `Komponentenvariantenversionen` existieren können. Auf der anderen Seite verwaltet `PDM` mehrere `Bauteile` sowie zugehörige `Bauteilversionen` in unterschiedlichen `Projekten`. Betrachtet man die Beispielinstanzen, so fällt auf, dass in den beiden Informationssystemen die Varianten nach unterschiedlichen Kriterien gespeichert werden. Während in `E-PDM` zu einer elektrischen Komponente die verschiedenen Varianten explizit dokumentiert sind, so ist die Variabilität in `PDM` implizit in der Bezeichnung der Bauteile dokumentiert. Weiter fällt auf, dass beide Systeme eine unterschiedliche Anzahl an elektrischen Komponenten bzw. Bauteilen besitzen. Dies liegt zum einen an der zuvor erwähnten impliziten Speicherung der Variabilität in den Bauteilen, zum anderen an der Tatsache, dass

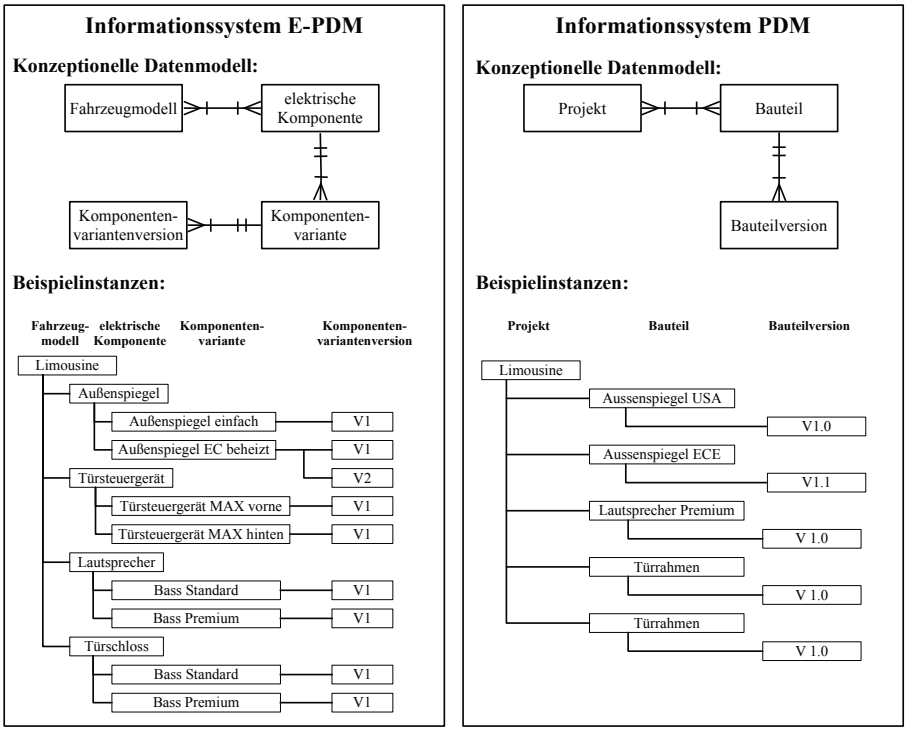


Abbildung 3.1.: Datenmodellausschnitte und Beispielinstanzen

beide Informationssysteme unterschiedliche Semantiken für die Begriffe elektrische Komponente und Bauteil besitzen. Ein Bauteil kann sowohl eine elektrische als auch eine rein mechanische Komponente sein.

Nicht nur die Dokumentation von Variabilität in beiden Systemen unterscheidet sich, auch zeitliche Änderungen der Entwicklungsstände elektrischer Komponenten und Bauteile besitzen verschiedene Semantiken. Während in PDM Änderungen von Geometriemodellen linear dokumentiert werden, können in E-PDM komplexe Vor- und Nachfolgebeziehungen definiert werden.

Bei der Realisierung des Anwendungsfalles ergeben sich somit folgende Herausforderungen: Beide Informationssysteme verwenden unterschiedliche Bezeichner für die Benennungen von Bauteilen bzw. Komponenten. Zusätzlich sind die Daten in unterschiedlichen Produktstrukturen gespeichert. Dies betrifft vor allem die unterschiedlichen Konzepte für Variabilität und Versionierung. Während in PDM Bauteile im Sinne einer Stückliste hierarchisch strukturiert sind, werden die unterschiedlichen elektrischen Varianten von Komponenten in E-PDM zusammengefasst. Der zu integrierende gemeinsame Nenner zwischen beiden Informationssystemen ist die elektrische Komponente, die eine Teilmenge der Bauteile ist.

Folglich kann die Integration solch heterogener Produktdaten nicht vollständig automatisiert werden. Vielmehr muss die Integration und im Speziellen die Zuordnung von semantisch zusammengehörigen Daten mittels manueller Interaktionen durch Endbenutzer unterstützt werden.

Aus dem illustrierten Anwendungsfall ergeben sich folgende Anforderungen an ein PDIS: Die Integration von Produktdaten muss unterschiedliche Konzepte für Variabilität und Versionierung von Produktdaten berücksichtigen. Da nicht alle Korrespondenzen zwischen Artefakten automatisch ermittelt werden können, muss ein PDIS Möglichkeiten bereitstellen, Anwender bei der

Integration zu unterstützen. Schließlich muss es möglich sein, die Integration von Produktdaten nur für Ausschnitte von Datenmodellen und eine Auswahl der Instanzen vorzunehmen, da sich semantische Konzepte zwischen Informationssystemen unterscheiden.

3.2.2. Anwendungsfall 2: Integration der vollständigen Produktentwicklung

Während im Anwendungsfall 1 lediglich Schemata und Daten zweier Informationssysteme integriert wurden, werden nachfolgend die Herausforderungen und Anforderungen erläutert, die sich ergeben, wenn die unterschiedlichen Aspekte von Bauteilen während der Produktentwicklung vollständig integriert werden sollen. Zu den Aspekten zählen sowohl die Anforderungsspezifikationen, Konstruktionszeichnungen, geometrische Modelle, Softwaremodelle und Quelldateien, Hardwareinformationen, Testspezifikationen und Testprotokolle.
Abbildung 3.2 zeigt exemplarisch einen Ausschnitt der Systemlandschaft der Produktentwicklung – eine detaillierte Beschreibung der Basis-IT-Infrastruktur findet sich in [Kat15]. Im CAD-Bereich werden überwiegend angepasste COTS[6]-Systeme eingesetzt, während viele weitere Informationssysteme für Kernaufgaben der Produktentwicklung (z.B. Stücklisten- und Änderungsmanagement) Eigenentwicklungen sind. Neben diesen werden in den übrigen Bereichen häufig Office-Anwendungen und eigene Applikationen eingesetzt, die oft informelle oder undokumentierte Informationsmodelle besitzen, um Produktanforderungen, Funktionen und Verhalten zu beschreiben.

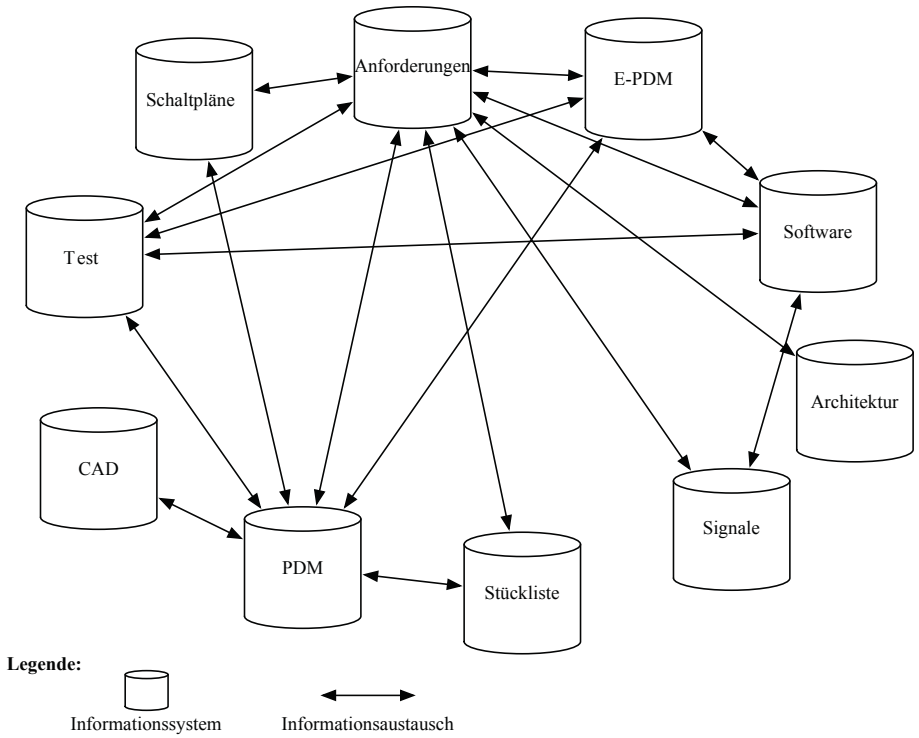


Abbildung 3.2.: Systemübersicht der Produktentwicklung

In Tabelle 3.2 sind die unterschiedlichen Aspekte und Artefakte von Bauteilen aufgelistet, die

[6]Commercial off-the-shelf

Name	Aspekt	Artefakte
Requirements Engineering	Anforderungen	Systemlastenhefte, Komponentenlastenhefte
Architektur	Entwurf	Technologiedokumentation
PDM	3D-Geometrie	Volumenmodelle, Simulationen, FEA
PDM	2D-Zeichnungen	Konstruktionszeichnungen
E-PDM	Hardware	Stecker, Pins, Interne Beschaltungen
Stückliste	Produktstruktur	Konfigurationen
Software	Funktionen	Funktionsmodelle
Software	Verbindungen	Kommunikationsmatrizen, Signale
Software	Source-Code	Quelldateien, Testdateien,
Testing	Test	Testspezifikationen und -protokolle

Tabelle 3.2.: Unterschiedliche Aspekte und Artefakte von Bauteilen

in den einzelnen Informationssysteme während der Produktentwicklung üblicherweise dokumentiert werden. Während einige Informationssysteme genau einen Aspekt von Bauteilen abdecken (z.B. Anforderungen), werden in anderen Systemen (z.B. PDM-System) eine Vielzahl an Aspekten dokumentiert.
Wie im Anwendungsfall 1 verwenden die Informationssysteme eigene Mechanismen zur Bezeichnung, Variabilität und Versionierung von Aspekten. Während zuvor nur zwei Bauteilmengen miteinander verglichen werden mussten, müssen nun eine Vielzahl an Mengen abgeglichen werden. Daraus ergeben sich folgende Anforderungen: Auf Grund der Vielzahl an Informationssystemen und der benötigten manuellen Interaktionen bei der Integration ist der kontinuierliche Abgleich der Bauteilmengen aller Informationssysteme zu allen Zeitpunkten nicht wirtschaftlich. Die Integration muss daher nach Bedarf erfolgen. Während Änderungen an Produktdaten sowie an den beiden Informationssystemen im ersten Anwendungsfall noch überschaubar waren, existiert im zweiten Anwendungsfall ein komplexes Abhängigkeitsgeflecht zwischen den einzelnen Informationssystemen. Änderungen an einem Informationssystem können sich auf eine Vielzahl weiterer Informationssysteme auswirken. Ein PDIS muss somit die Auswirkungen von Änderungen an Produktdatenn sowie den zugrundeliegenden Informationssystemen unterstützen sowie die möglichen Auswirkungen vor der eigentlichen Realisierung bestimmen können.
Grundsätzlich existieren zwei Ansätze, um Schemakonzepte und Instanzen heterogener Informationssysteme zu integrieren. Entweder wird ein Schema mit Schemakonzepten definiert, auf welches Schemakonzepte existierender Informationssysteme abgebildet werden (häufig *Top-Down-Integration* genannt), oder es wird versucht gemeinsame existierende Schemakonzepte zu vereinheitlichen (*Bottom-Up-Integration*). Auf Grund der Vielzahl an zu integrierenden Informationssystemen mit jeweils einer Menge an Schemakonzepten und Instanzen, muss ein PDIS unterschiedliche Integrationsmethodiken unterstützen.

3.3. Anforderungen

Dieser Abschnitt fasst die resultierenden Anforderungen an ein Konzept zur Integration von Produktdaten in komplexen Entwicklungsprozessen zusammen.
Während der Produktentwicklung arbeiten unterschiedliche Personen an einem Produkt und

führen dabei verschiedene, teils nebenläufige, Aktivitäten aus. Um letztere möglichst effektiv und effizient durchführen zu können, nutzen die Entwickler autonome, heterogene Informationsmodelle, mit Datenstrukturen, die auf ihre Bedürfnisse zugeschnitten sind [Sta11]. Die Ablösung dieser komplexen IT-Systemlandschaft ist weder praktisch möglich noch ist sie sinnvoll. Folglich müssen Produktdaten integriert werden. Auf Grund der heterogenen Datenstrukturen müssen bei der Integration die verschiedenen Mechanismen für die Modellierung von Variabilität und Versionierung von Produktdaten besonders berücksichtigt werden.
Durch die große Anzahl an Informationssystemen mit jeweils einer Vielzahl an Datenmodellkonzepten ist eine vollständige Integration all dieser Konzepte weder sinnvoll noch praktikabel. So muss die Integration bedarfsgerecht erfolgen. Zusammenfassend ergibt dies die erste Anforderung für die Realisierung eines Produktdatenintegrationssystems.

Anforderung A1 (Bedarfsgerechte, Anwendungsfall-getriebene Integration). Es muss möglich sein, beliebige Teilmengen von Produktdaten bedarfsgerecht sowie unter Berücksichtigung unterschiedlicher Mechanismen für Variabilität und Versionierung zu integrieren. Die Integration muss Produktdaten sowohl aus unterschiedlichen Repräsentationsformen (technische Heterogenität) als auch unterschiedliche Konzeptionalisierungen (semantische Heterogenität) unterstützen. Abhängigkeiten zwischen Schemakonzepten der heterogenen Informationssystemen müssen explizit dokumentiert werden. Neben Schemakonzepten müssen außerdem die Instanzen von Schemakonzepten integriert werden. Die Identität von Produktdatenn soll möglichst automatisch ermittelt werden. Die Integration dient der Unterstützung zuvor definierter Anwendungsfälle.

Die Datenqualität der Produktdaten variiert stark und korreliert zu den einzelnen Entwicklungsphasen. In frühen Entwicklungsphasen sind Informationen zu einzelnen Komponenten nur teilweise verfügbar oder ändern sich regelmäßig. Darüber hinaus werden in einigen Informationssystemen die Dokumentationsmethodiken definiert, ihre Einhaltung jedoch nicht konsequent überprüft. Dies trifft zum Beispiel für die Bezeichnung von Artefakten (z.B. Komponenten), zu. Komplexe Produkte können aus mehreren tausend Komponenten bestehen, folglich kann die Integration nicht vollständig automatisiert werden [Hei95]. Vielmehr muss ein Integrationsprozess durch Benutzer überwacht und unterstützt werden. Zu dessen Aufgaben zählt es unter anderem, Artefakte, die durch automatische Algorithmen nicht zugeordnet werden konnten, zu überprüfen und gegebenenfalls Korrespondenzen manuell zu erstellen. Insgesamt ergibt dies die nächste Anforderung an ein PDIS.

Anforderung A2 (Unterstützung manueller Integrationsaufgaben). Ein PDIS muss Benutzer bei unterschiedlichen Aufgaben im Integrationsprozess unterstützen. Dazu zählt unter anderem die Identifikation von Artefakten, für die automatische Algorithmen keine Korrespondenzen identifizieren konnte. Vielmehr muss ein PDIS Mechanismen bereitstellen, die es ermöglichen komplexe Produktdaten und dessen Beziehungen darzustellen.

Hauptziel der Integration von Produktdaten ist die Realisierung domänenspezifischer Anwendungsfälle, wie z.B. Vollständigkeits- und Korrektheitsanalysen. Diese können in unterschiedlichen Phasen der Produktentwicklung (z.B. Anforderungsspezifikation, Modellierung, Implementierung/Realisierung, Test, Integration) durchgeführt werden. Ergebnisse einer Integration müssen somit für solche Anwendungsfälle zugänglich gemacht werden. Die dritte Anforderung an ein PDIS ist demnach:

Anforderung A3 (Nutzung integrierter Produktdaten). Zur Realisierung domänenspezifischer Anwendungsfälle müssen Fachanwender in der Lage sein, beliebige Abfragen über integrierte Produktdaten stellen zu können. Ein PDIS muss eine Möglichkeit bereitstellen, diese nutzen zu können.

Schemata von Informationssystemen können sehr viele Schemakonzepte umfassen, deren manuelle Integration mit sehr viel Aufwand verbunden ist. Auch die Integration von Instanzen dieser Schemata kann nicht voll-automatisiert werden, auch hier ist eine Interaktion mit Endanwendern erforderlich. Daher benötigt der Integrationsprozess einen gewissen Zeitraum bis alle erforderlichen Schemakonzepte und Instanzen integriert sind. Zur Realisierung eines Anwendungsfalles, bei dem Produktdaten aus unterschiedlichen autonomen und heterogenen Informationssystemen benötigt werden, wird in der Regel ein Zeitpunkt definiert, ab dem die integrierten Daten verwendet werden sollen und keine weitere Integration mehr statt finden darf. Zu diesem Zeitpunkt sollen alle vorhandenen Produktdaten integriert sein, d.h. zusammengehörige Produktdaten sind mittels Korrespondenzen einander zugeordnet. Deshalb muss der Integrationsprozess überwacht werden, damit die Einhaltung von Nutzungszeitpunkten garantiert werden kann.

Anforderung A4 (Überwachung und Kontrolle des Integrationsprozesses). Ein PDIS muss für die Integration und dessen Fortschrittskontrolle entsprechende Prozesse mit Aufgaben und Rollen definieren. Dabei müssen Prozesse und Rollen bereits existierende Unternehmensprozesse und Aufgaben berücksichtigen.

Informationssysteme entwickeln sich kontinuierlich weiter [Len03, RRD03]. Änderungen an Informationssystemen in der Produktentwicklung sind die Regel und sind Anwender- und Kundengetrieben. In einigen Bereichen existiert eine abgestimmte Release-Planung für kritische Systeme, also solche, die für die direkte Produktion erforderlich sind. Darüber hinaus existieren jedoch eine Vielzahl an bereichsspezifischen Informationssystemen, die autonom entwickelt werden. Hierbei handelt es sich um spezielle Applikationen, die durch Fachanwender als Unterstützung der täglich anfallenden Arbeit entwickelt worden sind. Dies können sowohl Tabellenkalkulationen mit Berechnungen als auch kleine Datenbanken oder ähnliches sein. Diese Systeme entstehen, da Funktionalitäten in existierenden Informationssystemen fehlen oder der Aufwand der Anpassung dem Nutzen nicht gegenüber steht.
Änderungen an einem Schema eines Informationssystems können Auswirkungen auf Schemata anderer integrierter Informationssysteme haben. Wenn Produktdaten bereits integriert worden sind, dürfen Schemaänderungen nicht dazu führen, dass diese integrierten Ergebnisse verloren gehen bzw. unbrauchbar werden [CKR14]. Änderungen von Instanzen sind kontinuierlich, ein Löschen findet nur in Ausnahmen statt. Vielmehr dient die Speicherung aller Entwicklungsergebnisse der lückenlosen Dokumentation der Entwicklung. Für jede Änderung von Instanzen und Schemata müssen entsprechende Operationen und Prozesse entwickelt werden, die eine konsistente Integration sicherstellen bzw. wiederherstellen. Üblicherweise existieren in Unternehmen bereits eine Vielzahl an Prozessen, so etwa auch für die Abwicklung von Änderungen. Deshalb müssen bei der Integration bereits existierende Änderungsprozesse im Unternehmen berücksichtigt werden.

Anforderung A5 (Unterstützung von Änderungen). Ein PDIS muss auf Änderungen von Schemata und Instanzen reagieren und Änderungen dieser unterstützen, ohne dass bereits bestehendes Integrationswissen verloren geht. Im Speziellen bedeutet dies, dass Auswirkungen von Änderungen (sowohl auf Schema- als auch Instanzebene) vor der eigentlichen Realisierung bestimmt werden können müssen. Bei der Realisierung von Änderungen sollen diese möglichst vollautomatisch durchgeführt werden.

Wie erwähnt, verwenden die zu integrierenden Informationssysteme unterschiedliche Konzepte zur Modellierung der Variabilität und Versionen von Produktdaten. Für die Realisierung von Anwendungsfällen können unterschiedliche Granularitäten von Produktdatenvarianten und -versionen benötigt werden. Die Realisierung der Produktintegration kann auf unterschiedliche Arten erfolgen. In der Literatur wird häufig zwischen *top-down*- und *bottom-up*-Ansätzen unterschieden. Bei ersteren werden die zu integrierenden Konzepte in einem zentralen, integrierten Schema vorgegeben und existierende Konzepte von Informationssystemen auf diese abgebildet. Im anderen Fall werden die existierenden Schemata von Informationssystemen auf gemeinsame Konzepte analysiert, die integriert werden können. Hieraus folgt die letzte Anforderung an ein PDIS:

Anforderung A6 (Integrationsmethodiken). Um unterschiedliche Vorgehensweisen bei der Realisierung von Anwendungsfällen zu ermöglichen, muss ein PDIS eine klar definierte Methodik besitzen. Diese muss sowohl eine top-down- als auch bottom-up-Integration unterstützen. Weiter muss diese Methodik die Möglichkeit bieten, verschiedene Varianten- und Versionsmechanismen unterstützen.

In Tabelle 3.3 sind alle Anforderungen auf einen Blick dargestellt.

Nr	Anforderung	Literaturstudie	AF 1	AF 2
A1	Integration	[Sta11]	x	x
A1.1	bedarfsgerecht			x
A1.2	Varianten / Versionen		x	x
A2	Integrationsunterstützung	[TRS15]	x	x
A3	Nutzung		x	x
A4	Monitoring	[VSW15, PNSD07]	x	x
A5	Change Management			x
A6	Methodiken			x

Tabelle 3.3.: Anforderungen auf einen Blick

4. Stand der Technik

Die in Kapitel 3 ermittelten Anforderungen werden nun mit verwandten Ansätzen abgeglichen. Zunächst werden Arbeiten untersucht, die sich der Integration heterogener Informationen widmen. Anschließend werden Standards untersucht, die verschiedene Aspekte der Integration adressieren. Zuletzt werden die behandelten Ansätze mit den erhobenen Anforderungen abgeglichen.

4.1. Heterogenität, Autonomie, Integration

In diesem Abschnitt werden unterschiedliche Architekturen für die Integration heterogener, autonomer Informationssysteme beschrieben. Zunächst werden föderierte Datenbanksysteme [SL90], Data-Warehouses [CD97], Peer-Daten-Management-Systeme [HRZ+08] und Ansätze zur Tool Integration [KS06] diskutiert. Anschließend werden verwandte Arbeiten zu den verschiedenen Schritten des Integrationsprozesses (vgl. Abschnitt 2.3.3) erläutert.

4.1.1. Föderierte Datenbanksysteme

Bei föderierten Datenbanksystemen [SL90] handelt es sich um virtuell integrierende Systeme. Dass heißt, es findet keine physische Integration der Daten aus heterogenen, autonomen Informationssystemen statt. Vor allem die Autonomie der bestehenden Informationssysteme steht im Vordergrund föderierter Systeme. Es gibt verschiedene Referenzarchitekturen, die kurz erläutert werden. Detaillierte Informationen zu den einzelnen Ansätzen finden sich in [Con97].

Referenzarchitekturen

Import-/Exportschema ist die älteste Architektur föderierter Datenbanksysteme, bei der die Verhandlung zwischen Datenbanksystemen im Vordergrund steht. Demnach besitzt jedes DBMS ein *privates Schema*, welches alle Daten beschreibt, die dieses DBMS verwaltet. Im sog. *Exportschema* wird der Teil beschrieben, der anderen DBMS zur Verfügung gestellt wird. Auf der anderen Seite besitzt jedes DBMS ein *Importschema*, das die genutzten Schemakonzepte anderer Exportschemata umfasst.
Diese Architektur hat zum Ziel, eine möglichst hohe Autonomie der lokalen Systeme zu bewahren. Für die Integration sind sowohl Benutzer als auch lokale Administratoren der Systeme verantwortlich. Der Zugriff auf die Föderation erfolgt über die lokalen Systeme.
Die **Multidatenbank-Architektur** kommt ohne ein globales Schema aus. Dabei muss ein Benutzer mittels einer geeigneten Anfrage- und Datenmanipulationssprache die Möglichkeit haben, auf die heterogenen Datenbanken zugreifen zu können. Die Architektur besteht aus einem *physischen Schema*, das die interne Struktur der Daten beschreibt, sowie einem *internen logisches Schema*, welches das konzeptionelle Schema darstellt. Das *konzeptionelle Schema* ist ein Ausschnitt des internen logischen Schemas bzw. stellt dessen Inhalt in einer abweichenden Struktur dar. Mit einem *externen Schema* kann sich ein Benutzer in der gewünschten Sprache ein Schema aus dem vorhandenen konzeptionellen Schemata zusammenstellen und darauf

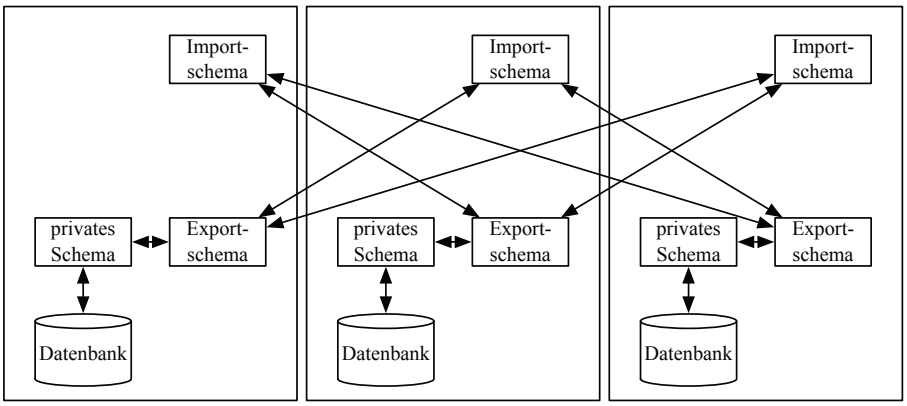


Abbildung 4.1.: Import-/Export-Architektur

arbeiten. Abhängigkeiten zwischen konzeptionellen Schemata, etwa datenbankübergreifende Integritätsbedingungen oder Redundanzen, können im *Abhängigkeitsschema* dokumentiert werden.

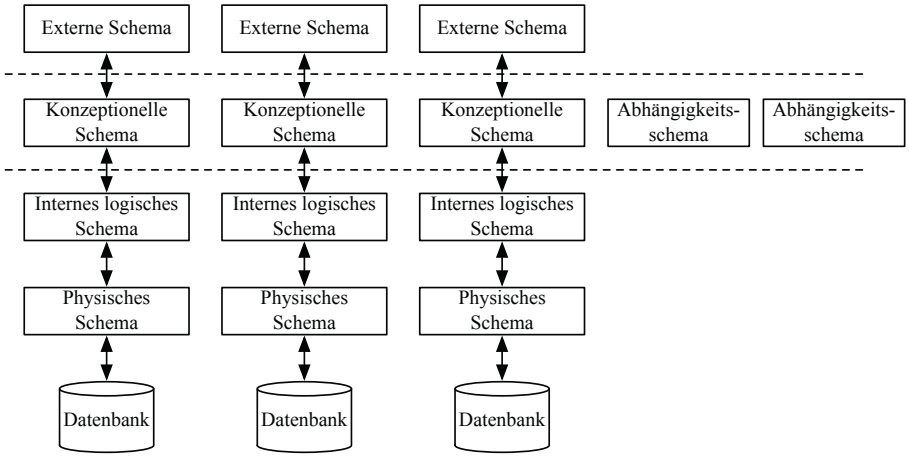


Abbildung 4.2.: Multidatenbank-Architektur

Bei der Multidatenbank-Architektur wird eine Multidatenbanksprache benötigt, welche die Funktionen für die Integration bereitstellt. Ein Vorteil dieser Architektur besteht darin, dass durch die Abhängigkeitsschemata datenbankübergreifende Integritätsbedingungen definiert werden können. Anders als bei der Import-/Export-Architektur ist der Benutzer alleine für die Integration verantwortlich. Der Zugriff auf die Föderation erfolgt über eine globale Schnittstelle.
Die **5-Ebenen-Schema-Architektur** besitzt, im Gegensatz zu den beiden vorangehend behandelten Architekturen, ein übergeordnetes föderiertes Schema [SL90]. In Abbildung 4.3 sind die fünf Schemaebenen der Architektur illustriert. Das *lokale Schema* ist das konzeptionelle Schema einer Datenbank, das alle Daten eines Systems umfasst. Das lokale Schema kann in unterschiedlichen Datenmodellen vorliegen. Das *Komponentenschema* wiederum beseitigt die Datenmodellheterogenität der lokalen Schemata. Einen Ausschnitt des Komponentenschemas, das der Föderation zur Verfügung gestellt werden soll, wird im *Exportschema* definiert. Das *föderierte Schema* fast alle Exportschemata zusammen und basiert in der Regel auf einem globalen (d.h. kanonischen) Datenmodell. Schließlich können mit *Exportschemata* unterschiedliche

Ausschnitte des föderierten Schemas definiert werden. Anders als in den beiden zuvor diskutierten Architekturen erfolgt der Zugriff auf die Föderation über ein globales System bzw. Anfragen gegen dessen Schema. Die Integration wird durch einen globalen Administrator durchgeführt, wohingegen die Datenbankadministratoren der lokalen Systeme für die Erstellung der Exportschemata zuständig sind.

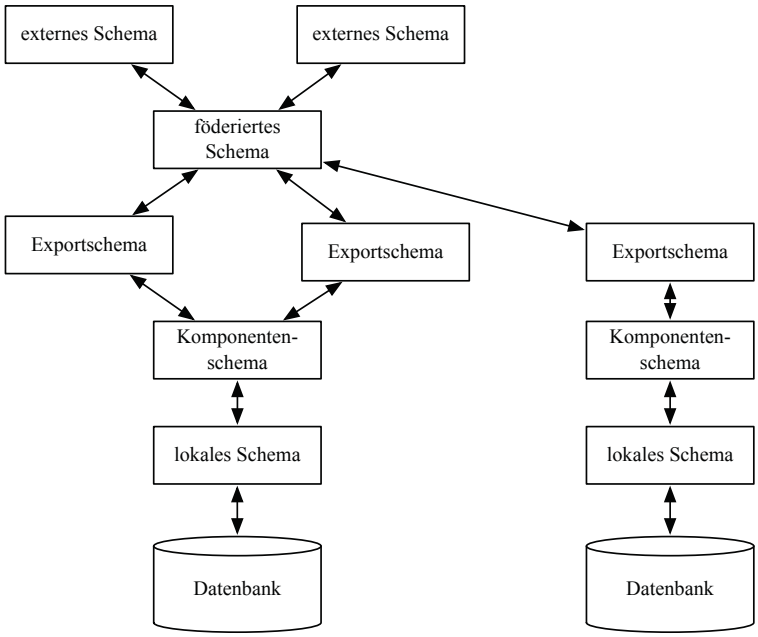


Abbildung 4.3.: 5-Ebenen-Architektur föderierter Datenbanksysteme (nach [SL90])

Integrationsmethoden

Bisher wurden unterschiedliche Architekturen für föderierte Integrationssysteme vorgestellt. Im weiteren Verlauf betrachten wir die 5-Ebenen-Schema-Architektur und diskutieren wie die Integration von Exportschemata in ein föderiertes Schema realisiert werden kann. Dazu werden unterschiedliche Methoden diskutiert.

- **Zusicherungsbasierte Integration:** Mit Hilfe von Zusicherungen werden diejenigen Bestandteile zwischen Schemata beschrieben, die einander entsprechen oder in einer bestimmten Beziehung zu einander stehen. Dadurch sollen strukturelle Konflikte im Integrationsprozess automatisch aufgelöst werden. Die Zusicherungen werden unabhängig von einem Datenmodell in einem *generischen Datenmodell* beschrieben. Der Ansatz unterscheidet unterschiedliche Schemakonzept- und Attributzusicherungsarten. Dazu zählen Äquivalenz, Inklusion, Überlappung und Disjunktheit von Schemakonzepten und Attributen. Korrespondenzbeziehungen lassen sich auch für Pfade innerhalb der Schemata definieren, analog existieren hier die Pfadäquivalenz, Pfadinklusion und Pfadüberlappung sowie der Pfadausschluss. Auf Basis der Zusicherungen zwischen zwei Schemata erfolgt der Integrationsprozess über binäre *Integrationsregeln*. Treten Konflikte im Integrationsprozess auf, wird im integrierten Schema das Element der beiden Schemata verwendet, welches weniger Einschränkungen besitzt.

Die Integrationsregeln werden nur für Äquivalenzzusicherungen beschrieben. Inklusion, Überlappung und Ausschluss werden nicht näher untersucht. Auch die Integration von komplexen Attributen wird nicht betrachtet.

- **Upward Inheritance:** Hierbei handelt es sich um eine Integrationsmethode für objektorientierte Datenmodelle. Vor allem die Betrachtung von Generalisierungs- und Spezialisierungshierarchien spielt bei diesen Datenmodellen eine wesentliche Rolle.

  Im Speziellen müssen Vererbungshierarchien aus lokalen Schemata auch im integrierten Schema repräsentiert sein. Folglich muss zu jedem Schemakonzept aus dem lokalen Schema ein äquivalentes Schemakonzept im integrierten Schema vorhanden sein. Zwei Schemakonzepte aus unterschiedlichen lokalen Schemata können nur dann in dasselbe Schemakonzept im globalem Schema integriert werden, wenn beide die selbe Semantik besitzen. Überlappen sich die Menge der Instanzen beider Schemakonzepte, wird für jedes Schemakonzept ein explizites Schemakonzept im globalem Schema angelegt. Handelt es sich um eine Teilmengenbeziehung, besteht eine Hierarchie im globalen Schema zwischen den beiden Schemakonzepten. Anderenfalls muss ein gemeinsames Schemakonzept im globalen Schema erzeugt werden, von dem beide Schemakonzepte abgeleitet werden (Aufwärtsvererbung, engl. *upward inheritance*).

  Durch die Forderung, dass bei Hierarchien für jedes Schemakonzept eines lokalen Schemas ein äquivalentes Konzept im globalen Schema repräsentiert sein muss, sind Umstrukturierungen im globalen Schema ausgeschlossen.

- **Generisches Integrationsmodell (GIM):** Die zu integrierenden Schemata werden gemeinsam mit Hilfe der sog. GIM-Notation dargestellt. Diese bildet die Grundlage für die anschließende Integration. Besonderheiten dieser Methode sind die Unabhängigkeit von konkreten Datenmodellen sowie die flexible Behandlung von Generalisierungs- und Spezialisierungshierarchien. Zusätzlich erlaubt der GIM-Ansatz, im Gegensatz zu anderen objektorientierten Integrationsmethoden, die Umstrukturierung lokaler Beziehungen im globalen Schema. Allerdings werden Aggregationsbeziehungen sowie 1-n- und n-m-Beziehungen nicht unterstützt.

[Con97] merkt an, dass alle bisherigen Ansätze ein relativ einfaches Datenmodell als Grundlage für die Integration verwenden und sieht darin eine konzeptionelle Beschränkung, merkt gleichzeitig aber an, dass in den meisten bestehenden Datenbanksystemen nur ein Bruchteil der Modellierungsmöglichkeiten angewendet wird.

Integration durch Sichten

Eine weitere Methode zur Integration von Exportschemata in ein föderiertes Schema ist die Definition von Sichten zwischen Exportschemata.
In Abschnitt 2.3.4 wurden bereits die zwei grundlegenden Architekturen zur Integration von Informationssystemen erläutert.
Die bisherigen Methoden haben lediglich die Integration lokaler Schemata beschrieben, jedoch nicht adressiert wie die eigentliche Integration von Instanzen vonstatten geht. Des weiteren fokussierten alle bisher beschriebenen Ansätze auf der Integration bereits bestehender Schemata, um ein globales Schema zu erzeugen. Dieses Vorgehen wird in der Literatur als *Bottom-Up*-Ansatz bezeichnet. Eine weitere Möglichkeit besteht darin, ein globales Schema zu spezifizieren

und anschließend Abbildungen auf die lokalen Schemata zu definieren. Diese Vorgehen wird als *Top-Down*-Ansatz bezeichnet. Abbildung 4.4 veranschaulicht beide Ansätze. Beim Bottom-Up-Ansatz werden die lokalen Schemata zunächst in Komponentenschemata übersetzt, die mit einer gemeinsamen Modellierungssprache spezifiziert sind. Anschließend werden Exportschemata spezifiziert, die diejenigen Schemakonzepte enthalten, die dem föderierten Schema zur Verfügung gestellt werden. Anschließend erfolgt die Integration der Konzepte der Exportschemata in ein gemeinsames integriertes, föderiertes Schema.
Der Beginn des Top-Down-Ansatzes ist äquivalent zu dem des Bottom-Up-Ansatzes, indem lokale Schemata auf Komponentenschemata und anschließend auf Exportschemata abgebildet werden. Anschließend erfolgt jedoch keine Integration der existierenden Schemakonzepte der Exportschemata, sondern es wird ein globales Schema definiert. Erst dann werden Abbildungen von Schemakonzepten des globalen Schemas auf korrespondierende Schemakonzepte der Exportschemata definiert.

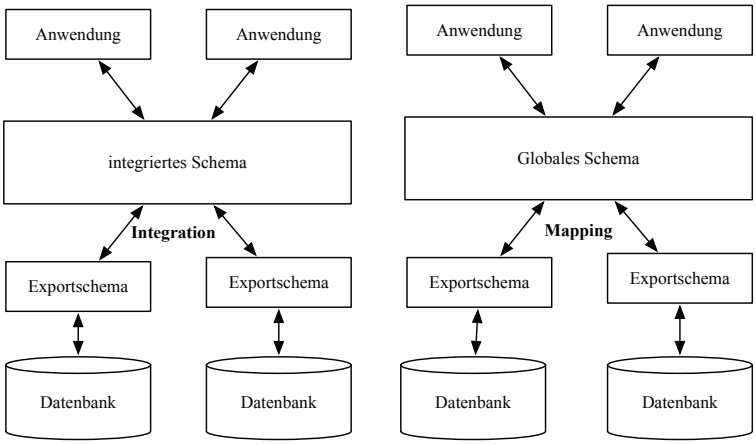


Abbildung 4.4.: Bottom-Up- und Top-Down-Ansatz

Die Integration im Bottom-Up-Ansatz sowie die Abbildungen im Top-Down-Ansatz lassen sich mit Hilfe von Sichten im relationalen Datenmodell realisieren. Für die Definition dieser Sichten gibt es drei Methodiken, die im Folgenden erläutert werden.

Local-as-View (LaV). Die Spezifikation von Abbildungen zwischen Exportschemata und föderiertem Schema im Top-Down-Entwurf mit Hilfe von Sichten wird in der Literatur als *Local-as-View (LaV)* bezeichnet [Len02]. Im Speziellen werden für ein gegebenes föderiertes Schema die Abbildungen von Exportschemata als Sichten auf das föderierte Schema definiert.

Beispiel 14 (LaV). Abbildung 4.5 illustriert ein Beispiel, in dem ein globales Schema definiert wurde und existierende Exportschemata als Sichten auf dieses Schema definiert werden. Das föderierte Schema besteht aus der Relation `Komponente` mit den Attributen `Nr`, `Label`, `Verantwortlicher`, `Lieferant` und `Typ`. Das Exportschema von `Informationssystem A` besteht aus der Relation `Bauteil`, während `Informationssystem B` die Relation `Component` beinhaltet.
Die Abbildungen zwischen föderiertem Schema und den beiden Exportschemata lassen sich

durch folgende Sichten spezifizieren.

```
CREATE VIEW Bauteil AS
SELECT K.Nr, K.Label AS Name, K.Lieferant, K.Verantwortlicher,
K.Typ FROM Komponente AS K

CREATE VIEW Component AS
SELECT K.Nr AS Id, K.Label AS Name, K.Lieferant AS Supplier,
K.Verantwortlicher AS Responsible FROM Komponente AS K
```

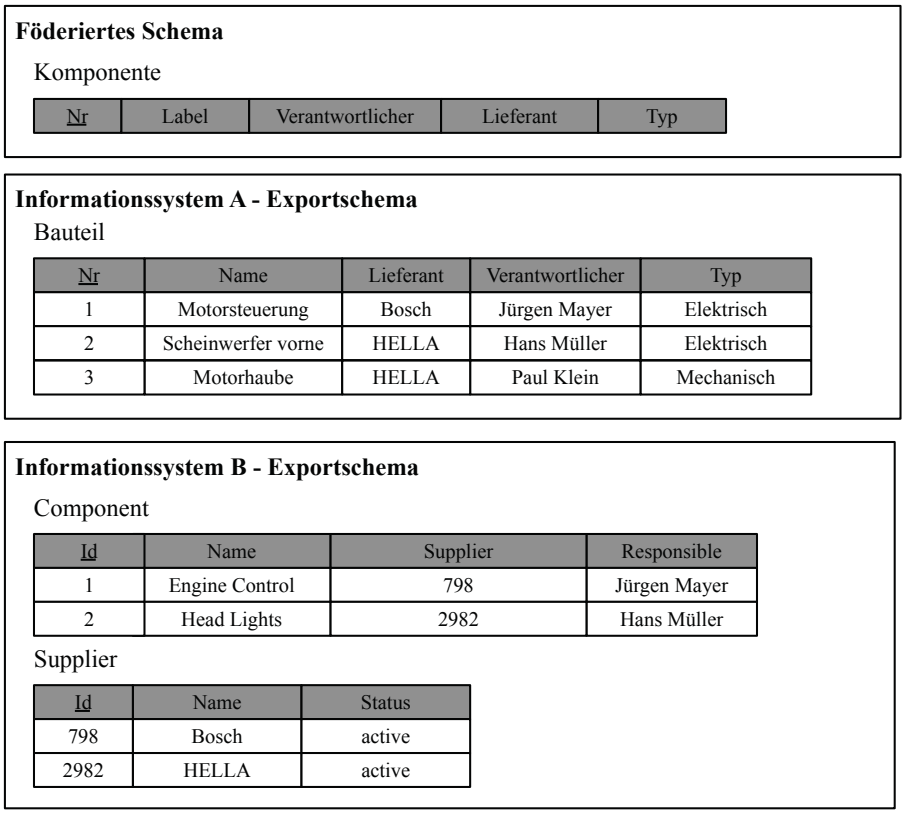


Abbildung 4.5.: Beispiel für LaV

Während in diesem relativ einfachen Beispiel alle Schemakonzepte der Exportschemata durch Sichten auf das föderierte Schema definiert werden können, gibt es einige Einschränkungen die mit der LaV-Methode nicht modelliert werden können. Existieren Beziehungen zwischen Schemakonzepten in einem Exportschema, die keine Entsprechung im föderierten Schema besitzt, kann diese nicht abgebildet werden. Ist ein Schemakonzept eines Exportschemas genereller als ein korrespondierendes globales Schemakonzept, kann dies nicht in einer Sicht ausgedrückt werden. Wie zuvor erwähnt, sind Föderierte Datenbanken virtuell integrierende Systeme, dass heißt die Instanzen verbleiben in den Informationssystemen, ohne dass ihre Daten in das föderierte Schema kopiert werden. Folglich müssen alle Anfragen an das föderierte Schema übersetzt und zerlegt werden in Teilanfragen an die einzelnen Exportschemata.

Grundidee des LaV-Ansatzes ist es, die einzelnen Sichten so zu einer Anfrage zu kombinieren, dass deren Ergebnis einen Teil der Anfrage (oder die ganze Anfrage) beantworten. Das Gesamtergebnis ist dann die Vereinigung der Ergebnisse mehrerer Anfragen.
Diese sog. Anfrageplanung kann sehr komplex werden, und es existiert hierfür eine Vielzahl spezieller Algorithmen [Hal01].

Global-as-View (GaV). Analog zum LaV-Ansatz wird die Spezifikation von Abbildungen zwischen Exportschema und föderiertem Schema im Bottom-Up-Ansatz mit Hilfe von Sichten in der Literatur als *Global-as-View (GaV)* bezeichnet [Len02]. Dass heißt, das föderierte Schema wird als Sicht auf die Exportschemata definiert. Folglich können nur Schemakonzepte und Attribute im föderierten Schema vorhanden sein, die auch in den Exportschemata spezifiziert worden sind.

Beispiel 15 (GaV).
Abbildung 4.6 illustriert ein Beispiel zur GaV-Methode. Die drei Exportschemata der Informationssysteme `A`, `B` und `C` sollen zu einem föderierten Schema integriert werden. Jedes Schema besitzt ein Schemakonzept (`Anforderungen`, `Test` und `CASE`); die Schemakonzepte wiederum können zu einer Sicht aggregiert werden:

```
CREATE VIEW Produkt AS
SELECT A.Komponente AS Name, A.Verantwortlicher, A.Anforderun-
gen, A.Variante, C.Version, T.Test FROM Anforderungen AS A
INNER JOIN CASE AS C ON A.Komponente = C.Label INNER JOIN Test
AS T ON A.Komponente = T.Component
```

Im Gegensatz zum LaV-Ansatz ist die Anfrageplanung beim GaV-Ansatz deutlich einfacher, da bereits in der Sichtdefinition festlegt ist, welche Elemente aus dem föderierten Schema welchen Elementen aus den Exportschemata entsprechen. Andererseits lässt sich die GaV-Methode nur realisieren, wenn die Datenqualität der Instanzen sehr gut ist, insbesondere dürfen Primärschlüsselattribute keine Fehler (z.B. ungültige Verweise, Mehrfachvergabe) enthalten. Des Weiteren geht die GaV-Methode davon aus, dass Instanzen, welche dieselben Dinge in der Realwelt darstellen, auch gleiche Bezeichner bzw. Identifikationsmerkmale besitzen (etwa eine global gültige Nummer oder Id).
Lokale Schemata können Attribute besitzen, die nicht in der globalen Sicht vorkommen müssen. Alle Relationen der globalen Sicht müssen Relationen aus lokalen Schemata sein oder konjunktive Anfragen dieser Relationen sein. Ist ein Schemakonzept im föderierten Schema genereller als das korrespondierende Konzept im Exportschema, kann dies nicht in der Sicht ausgedrückt werden. Treten Änderungen an lokalen Schemata auf, muss die Sicht des globalen Schemas neu erstellt werden. Die Integration zusammengehöriger Instanzen ist auf die verfügbaren Befehle der SQL-Spezifikation (z.B. UNION- und JOIN-Operator) beschränkt.

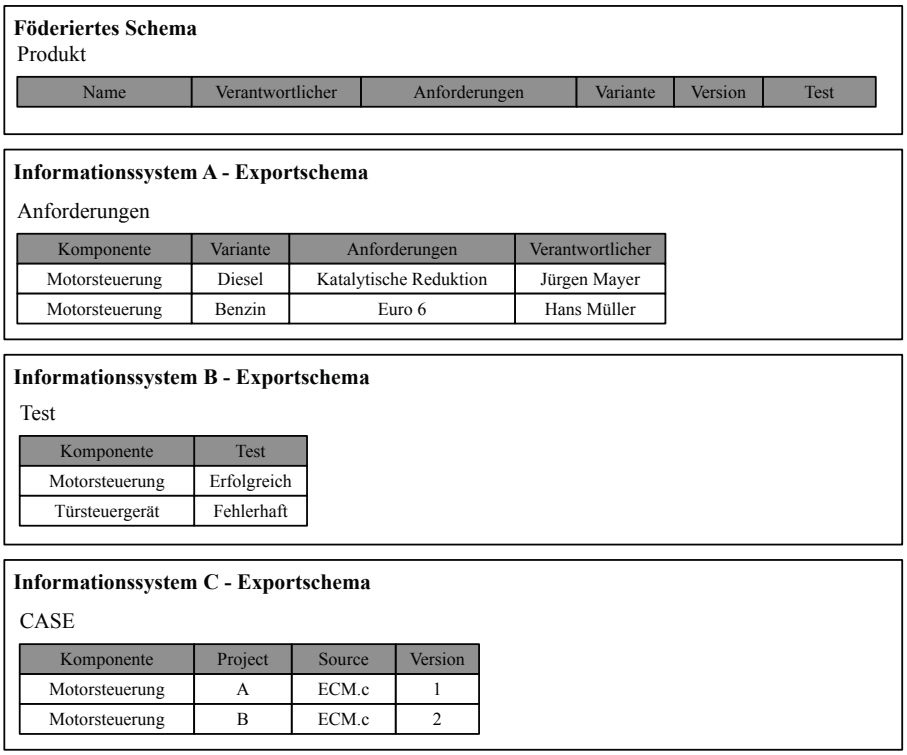


Abbildung 4.6.: Beispiel für GaV-Methode

Global-Local-as-View (GLaV). Die *Global-Local-as-View (GLaV)*-Methode ist eine Kombination aus den beiden vorherigen. Während bei der LaV-Methode das globale Schema eine Sicht auf lokale Schemata darstellt, bilden bei der GaV-Methode lokale Schemata die Sichten auf das globale Schema. Bei GLaV werden Sichten auf das globale Schema als Sichten der lokalen Schemata definiert. Die Komplexität für die Anfragebearbeitung ist die selbe wie für den LaV-Ansatz.

Abgleich der Anforderungen

Im Folgenden werden die Anforderungen aus Abschnitt 3.3 mit den bisherigen Erkenntnissen zu föderierten Datenbanksystemen abgeglichen.

Anforderung 1 (Bedarfsgerechte Integration, Berücksichtigung von Variabilität und Versionierung) Alle existierenden Ansätze für föderierte Datenbanken gehen von guter Datenqualität bzw. dem Vorhandensein global eindeutiger Identifikationsmerkmale aus. Diese sind in der Praxis häufig nicht gegeben. Aufgrund der hohen Komplexität bei der Anfrageverarbeitung verwenden, bis auf wenige Ausnahmen, alle wissenschaftlichen Ansätze die GaV-Methode. Bei der Integration mit Hilfe der GaV-Methode sind nur eingeschränkte Integrationsfunktionen vorhanden, diese beschränken sich auf JOIN- und UNION-Operationen sowie definierte SQL-Funktionen für relationale Datenbanken. Die meisten wissenschaftlichen Ansätze sind überwiegend Methodenbeschreibungen, prototypische Implementierungen der Konzepte fehlen dagegen meist und können daher nicht getestet werden.

Mit Hilfe föderierter Datenbanksysteme lässt sich eine Integration bedarfsgerecht durchführen, indem nur diejenigen Schemakonzepte in Exportschemata abgebildet werden, die auch tatsächlich integriert werden sollen. Es existieren keine Ansätze, die sich mit der expliziten Integration von Konzepten für Variabilität und Versionierung im Kontext föderierter Datenbanksysteme auseinandersetzen.

Anforderung 2 (Benutzerunterstützung) Es gibt keine Unterstützung von Integrationsaufgaben, etwa durch visuelle Editoren. Je nach verwendeter Architektur (Import-Export, Multidatenbank, 5-Ebenen-Schema) sind unterschiedliche Personengruppen für die Erstellung von Modellen sowie den Abbildungen zwischen ihnen verantwortlich. Jedoch werden die genauen Aufgaben dieser Gruppen nicht näher spezifiziert. Ein Konzept zur Unterstützung von Integrationsaufgaben, etwa die grafische Aufbereitung komplexer Arbeitsschritte, existiert nicht.

Anforderung 3 (Nutzung) Anfragen an das föderierte Schema sind mit der GaV-Methode zwar einfach, bleiben aber auf vorhandene Schemakonzepte im föderierten Schema beschränkt. Die Anfrageplanung mit der LaV-Methode wiederum kann sehr komplex werden.

Anforderung 4 (Monitoring) Der Integrationsprozess als solches wird nicht überwacht bzw. dessen Qualität nicht gemessen. Vielmehr wird aufgrund der Annahme einer guten Datenqualität von einer vollständig automatischen Integration ausgegangen.

Anforderung 5 (Change Management) Bei Änderungen bzw. der Integration neuer Informationssysteme muss beim GaV-Ansatz die gesamte Sicht angepasst werden. Da Integrationswissen hart in die Sichten kodiert ist, kann sich die Anpassung bei einem großen föderierten Schema aufwendig gestalten. Beim LaV-Ansatz muss bei Änderungen die Anfrageplanung angepasst werden. Während Änderungen von Schemata dazu führen, dass solche Sichten oder die Anfragebearbeitung angepasst werden müssen, sind das Einfügen, Ändern und Löschen von Instanzen in föderierten Datenbanken möglich.

Anforderung 6 (Methodiken) Durch GaV, LaV und GLaV werden unterschiedlichen Methoden für die Erstellung einer Föderation bereitgestellt. Explizite Methoden für die Abbildung von Variabilitäts- und Versionierungsmechnismen existieren dagegen nicht.

4.1.2. Data Warehouse Systeme

Virtuell integrierende Systeme, wie föderierte Datenbanken, weisen eine Vielzahl an Nachteilen auf, etwa die Dauer der Anfragebearbeitung, die Verfügbarkeit von Informationssystemen oder die Komplexität der Anfragebearbeitung betreffend.

Multidimensionale Datenmodellierung

Data Warehouses (DWHs) führen eine *materialisierte Integration* durch, d.h. Daten aus lokalen Informationssystemen werden in ein globales System kopiert, bereinigt und integriert [CD97]. DWHs kommen in verschiedenen Anwendungsfällen zum Einsatz. Dazu zählen vor allem Anwendungen, bei denen die Analyse großer Datenbestände (z.B. Verkaufszahlen von Produkten oder

Korrelationen zwischen verkauften Produkten und Käufergruppen) im Fokus stehen. Folglich unterscheidet sich die Modellierung von Daten grundlegend zu klassischen Informationssystemen, die auf dem relationalen Datenmodell basieren. Diese Modellierung wird als *multidimensionale Datenmodellierung* bezeichnet, bei der *Fakten* entlang verschiedener *Dimensionen* modelliert werden [KR11]. Beispiele für Fakten sind Geschäftsmetriken wie etwa Verkäufe oder Telefonate in einem Service-Unternehmen. Zu den Dimensionen zählen oftmals Zeitpunkte, Regionen oder Lieferanten. Dimensionen lassen sich in der Regel hierarchisch anordnen (z.B. Jahr-Monat-Woche-Tag). Die grafische Darstellung von Fakten kann bei drei Dimensionen als Würfel erfolgen.
Abbildung 4.7 zeigt die grundlegende Architektur eines DWH. Zunächst wird definiert, welche Datenquellen integriert und welche Anwendungsfälle unterstützt werden sollen. Anschließend werden Schemata und Sichten des DWH sowie dessen physisches Schema definiert, bevor schließlich die Datenquellen über Schnittstellen eingebunden werden. Diese werden mit Hilfe eines zuvor definierten Extract-Transform-Load (ETL) Prozesses in ein DWH kopiert, bereinigt (Fehler beseitigt, Daten in einheitliche Formate überführt) und integriert. Dazu werden Skripte für die Datenextraktion, -transformation, -bereitstellung und -aktualisierung erstellt. Ein Integrationsprozess ist immer anwendungsspezifisch, dass heißt der ETL-Prozess definiert nur ein abstraktes Ablaufschema, dessen einzelne Aktivitäten angepasst werden müssen. Ein wichtiger Schritt ist die Definition eines *Refresh-Plans*, der spezifiziert wann Aktualisierungen von operationalen Datenbanken in das DWH übernommen werden sollen. Da letztere in der Regel mehrere Terrabyte umfassen können, ist eine kontinuierliche Aktualisierung nicht praktikabel. Vielmehr geschehen Aktualisierungen nach Bedarf oder festgelegten Zeitpunkten. Das DWH wird mit Hilfe des zuvor festgelegten Schemas sowie der Sichten durch die Skripte befüllt und kann anschließend für unterschiedliche Analysemöglichkeiten aufbereitet werden.

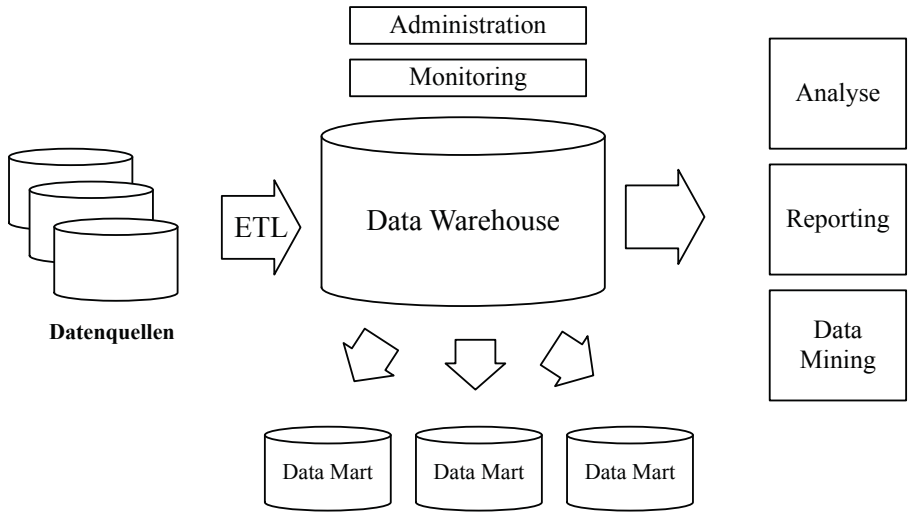


Abbildung 4.7.: Architektur eines DWH, in Anlehnung an [CD97]

Star- und Snowflake-Schema

Die Abbildung multidimensionaler Daten auf eine relationale Datenbank kann mit dem *Star-* oder dem *Snowflake-Schema* geschehen (siehe Abbildung 4.8). In beiden Fällen ist der Fakt ein

`Verkauf`. Im Star-Schema wird jede Dimension in einer eigenen Tabelle gespeichert, im Fall von a) sind dies `Produkt`, `Zeitpunkt` und `Filiale`. Im Fall des Snowflake-Schemas in b) erhält jede Klassifikationsstufe eine eigene Tabelle. Während das Star-Schema weniger Tabellen und damit weniger JOIN-Operationen bei Abfragen benötigt, werden Daten redundant (d.h. nicht normalisiert) gespeichert. Anders sieht es im Snowflake-Schema aus, wo zwar die Redundanz vermieden wird, sich Anfragen aber komplexer gestalten.

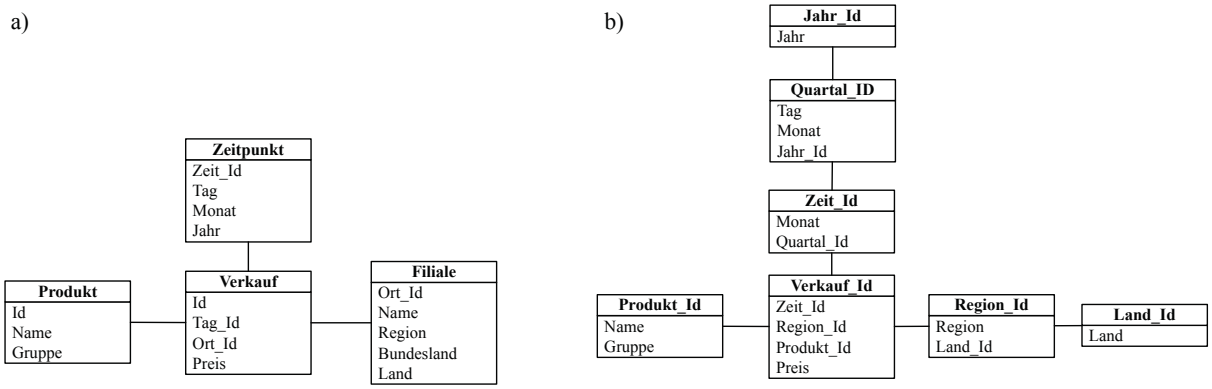


Abbildung 4.8.: Beispiel eines multidimensionalen Datenmodells im Star-Schema

Ein multidimensionales Datenmodell lässt sich mit Hilfe eines Würfels visualisieren, falls die Anzahl der Dimensionen drei entspricht. Sind mehr Dimensionen vorhanden, handelt es sich um einen Hyperwürfel, der jedoch nicht ohne weiteres grafisch dargestellt werden kann. Auf einem Würfel sind für DWHs spezielle Operationen definiert, welche die Navigation im bzw. Transformation des Würfels erlauben. Zu diesen Operationen zählen u.a. *roll-up*, *drill-down*, *slice-and-dice* und *pivot*.

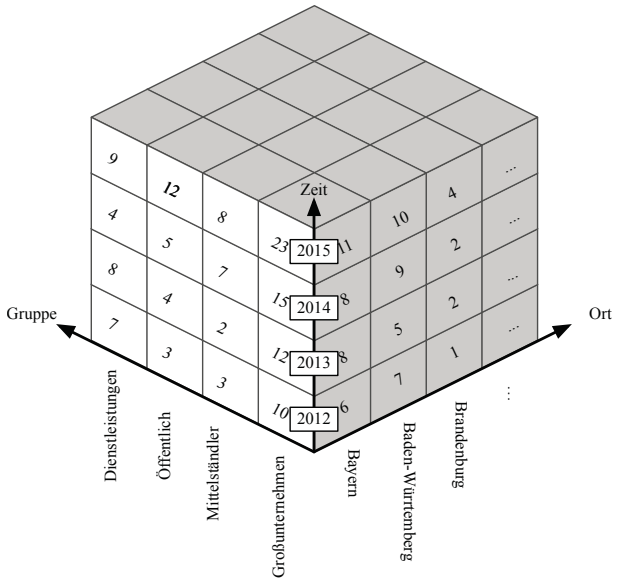


Abbildung 4.9.: Würfeldarstellung eines DWH-Schemas

Da das Schema des DWH eine Vielzahl an Fakten entlang verschiedener Dimensionen speichert, werden anwendungsspezifische Ausschnitte dieses Schemas als sog. *Data Marts* definiert. Abbildung 4.9 zeigt einen Würfel für die visuelle Darstellung des Star-Schemas aus Abbildung 4.8 a.

Bottom-Up- und Top-Down-Entwurf

Methodisch lässt sich ein Data Warehouse auf verschiedene Arten realisieren. [Inm05] schlägt dazu einen Top-Down-Ansatz vor. Dabei wird zunächst ein normalisiertes Datenmodell erstellt, bevor die Data Marts definiert werden, die dann Daten für spezifische Geschäftsbereiche oder Abteilungen bereitstellen. Dem entgegen steht der Bottom-Up-Ansatz von [KR11], in dem zunächst die Data Marts erstellt werden. Diese werden anschließend kombiniert, um das Datenmodell des Data Warehouse zu erhalten. Beide Ansätze haben ihre Vor- und Nachteile, die Anwendung des jeweiligen Ansatzes hängt vom Einsatzbereich ab. Der in [Inm05] skizzierte Ansatz ist sehr zeitaufwändig und mit hohen initialen Kosten verbunden, da die Erstellung des DWH durch Spezialisten erfolgen muss. Dafür ist die Verwaltung eines bestehenden DWH einfach. Auf der anderen Seite ist der Ansatz von [KR11] mit geringeren initialen Kosten verbunden und kann schneller realisiert werden, da auch Generalisten die Integrationsaufgaben erledigen können. Dagegen ist die Verwaltung eines bestehenden DWH erheblich aufwändiger als mit dem ersten Ansatz.

Abgleich der Anforderungen

Die Erkenntnisse zu Data Warehouses werden nun mit den Anforderungen aus Kapitel 3.3 abgeglichen.

Anforderung 1 (Bedarfsgerechte Integration, Berücksichtigung von Variabilität und Versionierung) Während der Fokus eines DWH auf der Entscheidungsunterstützung für Unternehmen durch die Integration von Fakten (z.B. Verkaufszahlen) entlang verschiedener Dimensionen liegt, ist eine Modellierung von Produktdaten durch eine multidimensionale Modellierung nicht möglich. Wichtiges Merkmal von Daten für DWHs ist die Fähigkeit der Aggregation, welche für Produktdaten nur bedingt gegeben ist. DWHs bieten die Möglichkeit, Daten bedarfsgerecht zu integrieren, jedoch existieren wie bei den föderierten Datenbanksystemen (FDBS) keine expliziten Konzepte für die Integration von Variabilitäts- und Versionierungskonzepten. Weiter sind die Integrationsskripte im ETL-Prozess anwendungsspezifisch.

Anforderung 2 (Benutzerunterstützung) Durch die kommerzielle Verbreitung von Data Warehouse Systemen, besitzen diese unterschiedliche Funktionsumfänge. Dazu zählen auch Funktionen zur Unterstützung der ETL-Prozesse. Für dessen Definition existiert eine Vielzahl an Software-Systemen, die u.a. eine grafische Modellierung der ETL-Prozesse erlauben.

Anforderung 3 (Nutzung) Durch die multidimensionale Modellierung sind die Möglichkeiten für Abfragen begrenzt. Die Anfragen an integrierte Produktdaten sind allgemeiner als eine Einschränkung auf Fakten und Dimensionen, wodurch sich ein DWH nicht als Realisierung für ein PDIS eignet.

Anforderung 4 (Monitoring) Genau wie bei der Benutzerunterstützung hängen die Funktionalitäten für das Monitoring eines DWH vom eingesetzten Produkt ab. *IBM InfoSphere Information Server* etwa ist eine Komponente für die Integration von Daten für ein DWH. Sie ermöglicht zudem die Definition von Monitoring-Prozessen [BBK+13]. Es lassen sich Qualitätsziele für Daten definieren, ebenso wie ein entsprechender Prozess, der diese Qualitätsziele überprüft. Da es sich lediglich um ein System zur Realisierung von ETL handelt, wird der Integrationsprozess an sich nicht überwacht.

Anforderung 5 (Change Management) Probleme, die durch Änderungen eines DWH entstehen, werden in der Literatur als *View Maintenance Problem* bezeichnet. Die meisten Ansätze befassen sich mit der Fragestellung, wie materialisierte Sichten konsistent mit den zugrundeliegenden Datenquellen gehalten werden können. [Bel02] beschreibt, wie strukturelle Änderungen in einem DWH unterstützt werden können. Dabei werden z.B. materialisierte Sichten dynamisch angepasst, und es wird versucht, soweit sinnvoll und möglich, alte Sichten wiederzuverwenden. Darüber hinaus können Änderungen am Schema des Data Warehouse unabhängig von den Schemata der integrierten Systeme durchgeführt werden. Schließlich beschreibt [PVSV07] einen Ansatz, mit dem *What-If-Analysen* für Änderungen an Schema und Instanzen von Datenquellen durchgeführt werden können. Darüber hinaus werden die Auswirkungen für ETL-Prozesse bestimmt.

Anforderung 6 (Methodiken) Analog zu föderierten Datenbanken existieren Bottom-Up- und Top-Down-Ansätze, die unterschiedliche Szenarien unterstützen. Jedoch fehlen bei den Data Warehouse Systemen eine explizite Unterstützung für die Modellierung von Variabilitäts- und Versionierungskonzepten.

4.1.3. Peer-Daten-Management-Systeme

Basierend auf den Prinzipien von Peer-to-Peer-Systemen sind die Peer-Daten-Management-Systeme entstanden [HRZ+08]. Zunächst werden Grundlagen erläutert, bevor die Anfragebearbeitung detailliert wird. Schließlich werden die Anforderungen abgeglichen.

Grundlagen

Grundidee ist, dass Anfragen an alle Informationssysteme (Peer) gestellt werden können, die Teil einer Integration sind. Damit zählen Peer-Daten-Management-Systeme zu den virtuell integrierenden Systemen. Ein Peer versucht Anfragen mit Hilfe seiner eigenen Daten sowie den Daten weiterer Informationssysteme (Peers) zu beantworten. Ziele eines Peer-Daten-Management-Systems sind die Autonomie des Peers sowie geringe Initial- und Wartungskosten, ohne Einbindung eines zentralen Koordinators [RS13].
Abbildung 4.10 illustriert ein Peer-Daten-Management-System. Zwischen den verschiedenen Peers existieren unterschiedliche Beziehungen zwischen deren Schemata. Während zwischen `BOM`, `PDM` und `Requirements` jeweils bidirektionale Mappings bestehen, sind alle anderen Abbildungen unidirektional. Ein Mapping zwischen den Schemata zweier Peers definiert Äquivalenzen zwischen denselben Schemakonzepten sowie notwendige Transformationen. Ein Peer fungiert sowohl als Datenquelle als auch Mediator, in dem er Anfragen an weitere Peers weiterleitet. Mappings

zwischen Schemata der Peers können sowohl mittels GaV-, LaV- als auch $GLaV$-Ansatz definiert werden (vgl. Abschnitt 4.1.1).

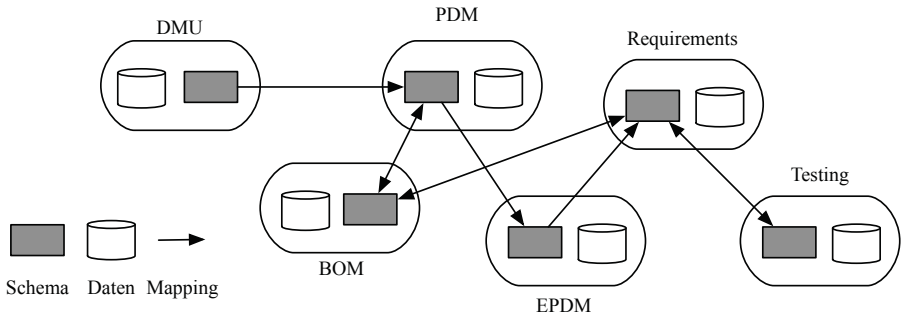


Abbildung 4.10.: Peer-Daten-Management

Anfragebearbeitung

Angenommen die Peers in Abbildung 4.10 besitzen alle ein gemeinsames Schemakonzept, welche untereinander entsprechend der Pfeile mittels Mappings abgebildet sind. Dann könnte eine Anfrage etwa an den Peer `BOM` im Netzwerk wie folgt abgearbeitet werden: Die Anfrage soll alle Attribute des gemeinsamen Schemakonzeptes aus allen Peers zurückgeben, so dass tatsächlich alle Peers abgefragt werden müssen. Abbildung 4.11 zeigt zwei sogenannte *rule-goal-trees*, also Anfragebäume an das Peer-Netzwerk, die die Anfrage erfüllen.

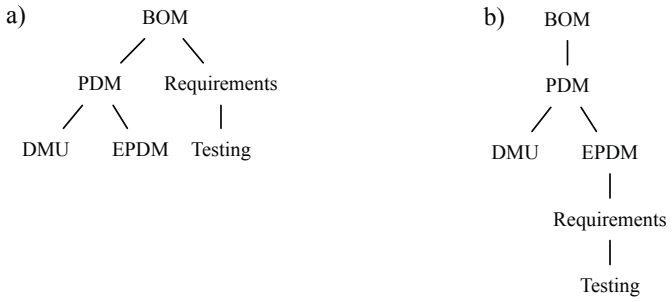


Abbildung 4.11.: Beispiele für Rule-Goal-Trees

Abgleich der Anforderungen

Die Erkenntnisse von Peer-Daten-Management-Systeme werden nun mit den Anforderungen aus Kapitel 3.3 abgeglichen.

Anforderung 1 (Bedarfsgerechte Integration, Berücksichtigung von Variabilität und Versionierung) Da es sich bei Peer-Daten-Management-Systemen um virtuell integrierende Systeme handelt, ist die Integration ähnlich zu bewerten wie bei föderierten Datenbanksystemen. Es wird angenommen, dass die Datenqualität gut ist und ähnliche Schemakonzepte in den Schemata der Peers vorhanden sind. Durch die fehlende Materialisierung müssen Inkonsistenzen bereits

während der Anfragebearbeitung vollständig aufgelöst werden. Es gibt einige Konzepte, die sich mit Konsistenzproblemen während der Integration befassen. Dabei wird zwischen *lokaler Inkonsistenz* und *P2P-Inkonsistenz* unterschieden. Im ersten Fall sind Daten im lokalen Speicher eines Peers inkonsistent. Im zweiten Fall ist jeder Peer lokal konsistent, aber es gibt Widersprüche in den Daten zwischen den Peers, oder ein Peer importiert inkonsistente Daten eines anderen. Lokale Inkonsistenz eines Peers führt automatisch dazu, dass das System unbrauchbar ist, da es global inkonsistent ist. Um diesem Problem zu begegnen, lassen sich in einigen Systemen lokal inkonsistente Peers von Abfragen ausschließen. Bei P2P-Inkonsistenz wird versucht, durch Algorithmen die Inkonsistenzen automatisch aufzulösen. Bestehende Ansätze gehen davon aus, dass alle Peers die gleiche Domäne teilen, d.h. dass diese gleiche Elemente für die Repräsentation von Realweltobjekten verwenden. Wie bereits in der Anforderungsanalyse erläutert, spiegel dies nicht die Realität in der Produktentwicklung wider. Wie alle bisher diskutierten Ansätze bieten auch Peer-Daten-Management-Systeme keine expliziten Konzepte für die Integration unterschiedlicher Variabilitäts- und Versionierungskonzepte.

Anforderung 2 (Benutzerunterstützung) Da es sich um virtuell integrierende Systeme handelt, wird davon ausgegangen, dass die Integration von Daten im Zuge der Anfragebearbeitung vollautomatisch und ohne Benutzerinteraktion geschieht. Für das Mapping zwischen Peers sind keine expliziten Unterstützungskonzepte bekannt.

Anforderung 3 (Nutzung) Da die Abbildungen zwischen Schemata der Peers mit Hilfe von LaV, GaV und GLaV erfolgen, sind die Aussagen zur Nutzung integrierter Daten ähnlich wie bei föderierten Datenbanksystemen. Durch die dezentrale Architektur kommt zusätzlich Komplexität für die Anfragebearbeitung in folge der Ermittlung der *Rule-Goal-Trees* hinzu.

Anforderung 4 (Monitoring) Durch die dezentrale Architektur findet keine zentrale Überwachung der Integration statt. Wie erwähnt, gehen Peer-Daten-Management-Systeme davon aus, dass die Integration vollstän-dig und konsistent in der Anfragebearbeitung erfolgt.

Anforderung 5 (Change Management) Bisher gibt es wenige Konzepte zur Unterstützung von Änderungen auf Schemaebene. Einige Ansätze propagieren Änderungen lokaler Datenquellen an ihre direkten Nachbarn, was jedoch zu vielen Problemen führen kann [RS13]. Da Änderungen sowohl an Schemakonzepten als auch an Instanzen eine zentrale Eigenschaft von Produktentwicklungsprozessen darstellen, ist die fehlende Unterstützung von Änderungsoperationen ein Ausschlusskriterium für die Verwendung von Peer-Daten-Management-Systemen zur Integration von Produktdaten.

Anforderung 6 (Methodiken) Da die Abbildungen zwischen den Schemakonzepten eines Peers zu weiteren Peers entweder durch LaV- oder GaV-Abbildungen erfolgen können, sind, wie bei föderierten Datenbanksystemen, zwei Methodiken für die Modellierung von Schemakorrespondenzen möglich. Wie bei allen bisherigen Ansätzen sind auch hier keine expliziten Konzepte für die Modellierung unterschiedlicher Variabilitäts- und Versionierungskonzepte vorhanden.

4.1.4. Tool Integration

Bereits in Abschnitt 2.3.2 wurde diskutiert, dass eine Tool-Integration deutlich mehr umfasst, als nur Schema- und Datenintegration. Im weiteren Verlauf sollen aber nur die Bereiche der Schema- und Datenintegration ausgewählter Ansätze untersucht werden. Die wichtigste Gruppe von Ansätzen basiert auf sog. *Triple-Graph-Grammatiken* (TGG) [KS06]. Die Integration von Informationssystemen geschieht paarweise auf Metamodellebene. Im Speziellen werden mit Hilfe von Korrespondenzen, die in einem eigenen Metamodell definiert werden, Elemente der zu integrierenden Metamodelle bidirektional aufeinander abbildet.
Abbildung 4.12 illustriert die Metamodelle zweier Informationssysteme aus der Softwareentwicklung für eingebettete Echtzeitsysteme in der Automobilindustrie sowie das Abbildungsmodell zwischen beiden (siehe Abbildung 4.12b). In Abbildung 4.12a ist das Metamodell zur Spezifikation von Anforderungen an eine Komponente dargestellt (`RequirementsDocument`). `Feature` können zu Gruppen zusammengefasst werden (`FeatureGroup`) sowie Beziehungen (`Dependency`) zu anderen Features haben. In Abbildung 4.12c ist zudem das Metamodell für die Modellierung von UML-Use-Case-Diagrammen illustriert. Anwendungsfälle (`UseCaseDocument`) sind Diagramme, die in `Packages` zusammengefasst werden und die Beziehungen (`Relationship`) zu anderen Anwendungsfällen haben können. In Abbildung 4.12b ist zudem die Abbildung der Metamodellelemente spezifiziert, in der semantisch zusammengehörige Schemakonzepte aufeinander bidirektional abgebildet sind. Demnach gehört eine `FeatureGroup` zu einem `Package`, ein `Feature` zu einem `UseCase` sowie eine `Dependency` zu einer `Relationship`.

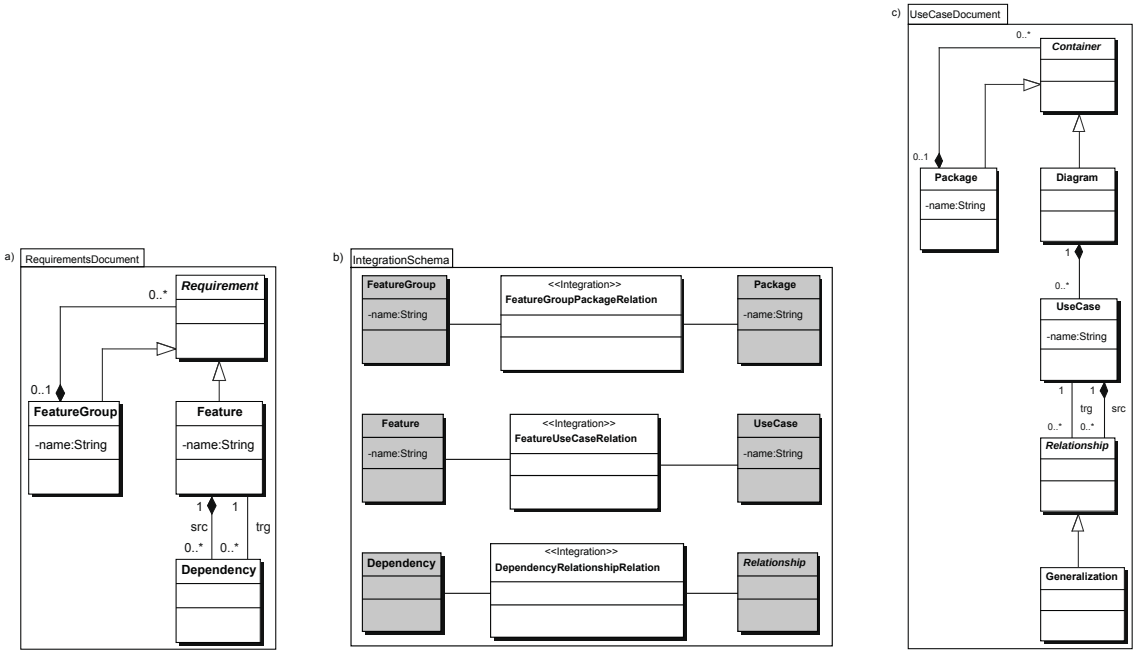


Abbildung 4.12.: Metamodelle zweier Informationssysteme sowie dessen Abbildung [KS06]. Creative Commons Lizenz Attribution-NonCommercial-NoDerivs 3.0 Unported (CC BY-NC-ND 3.0), `https://creativecommons.org/licenses/by-nc-nd/3.0/`

TGGs erlauben es, bidirektionale Transformationen zwischen Paaren von Graphen zu definieren, die über einen *Korrespondenzgraphen* verbunden werden. Analog werden die Metamodelle

in Graphgrammatiken [Roz97] überführt. Die Transformation erfolgt durch sog. *Graph Rewriting Rules*. Jede Regel besteht aus einer rechten und linken Seite. Letztere ist ein zu findenes Muster innerhalb des Graphen, welches durch ein anderes auf der rechten Seite spezifiziertes Muster ersetzt wird. Die Triple-Graph Grammatiken erlauben die Spezifikation von Korrespondenzen auf Basis von Attributen der Schemakonzepte. Abbildung 4.13 zeigt unterschiedliche Regeln, mit der die Metamodelle aus Abbildung 4.12 konsistent gehalten werden. Die erste Regel `createFeatureGroupAndPackage` in Abbildung 4.12a ist für die Erstellung eines neuen Modells zuständig, in dem sowohl eine `FeatureGroup` im Modell für Anforderungen angelegt wird und gleichzeitig ein `Package` im Model für Use Cases. Diese beiden Modellelemente werden miteinander verbunden und besitzen den selben Namen. Die zweite Regel in Abbildung 4.12b fügt einer bestehenden `FeatureGroup` sowie einem zugehörigem `Package` eine neue `FeatureGroup` mit einem korrespondierendem `Package` hinzu.

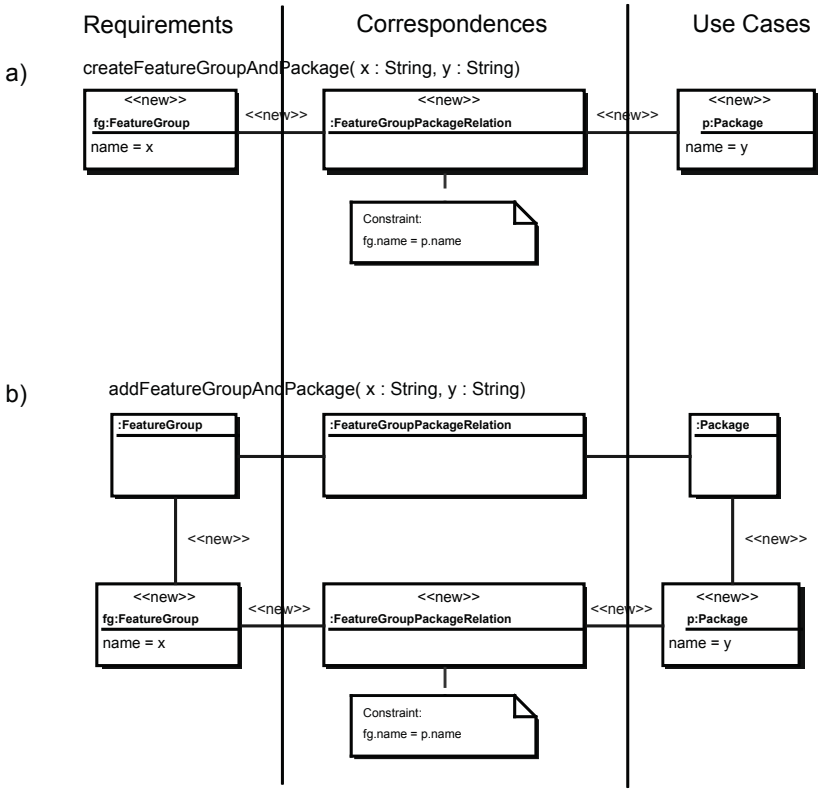


Abbildung 4.13.: Graph rewriting rules [KS06]. Creative Commons Lizenz Attribution-NonCommercial-NoDerivs 3.0 Unported (CC BY-NC-ND 3.0), `https://creativecommons.org/licenses/by-nc-nd/3.0/`

Abgleich der Anforderungen

Die Erkenntnisse zur Tool Integration werden nun mit den Anforderungen aus Kapitel 3.3 abgeglichen.

Anforderung 1 (Bedarfsgerechte Integration, Berücksichtigung von Variabilität und Versionierung) Fokus der Tool-Integrationsansätze mittels TGGs ist die Sicherstellung der Konsistenz zwischen Modellen. Die Integration erfolgt dabei paarweise auf Metamodell-Ebene. Bei großer Anzahl zu integrierender Systeme, wird die Zahl der benötigten Abbildungen zwischen den Informationssystemen sehr groß. Eine vollständige Kopplung aller n Systeme untereinander würde $n*(n-1)/2$ TGGs benötigen. Bei 10 Systemen sind dies bereits 45 potenzielle Schnittstellen. Bisher werden in den Abbildungsmetamodellen nur einfache Regeln für die Beschreibung von Beziehungen zwischen Attributen unterstützt. [KS06] führt außerdem an, dass die vollständige Konsistenz der Modelle durch TGGs zu jedem Zeitpunkt in der Praxis utopistisch ist, da sich alle Dokumente und Entwicklungsmodelle gleichzeitig weiterentwickeln müssten.

Anforderung 2 (Benutzerunterstützung) Sowohl mit MOFLON [AKRS06] als auch FUJABA [BGN+04] existieren zwei Ansätze, die grafische Benutzeroberflächen für die Definition von TGGs sowie Graph Rewriting Rules bieten. Die Ansätze gehen von einer hohen Datenqualität aus, etwa dass Modellelemente in den zu integrierenden Informationssystemen gleich benannt sind und sich somit eindeutig identifizieren lassen. Diese Annahme kann für die Produktentwicklung jedoch aufgrund von *Concurrent Engineering* nicht getroffen werden. Folglich wird auch keine grafische Unterstützung für die Auflösung von Konsistenzproblemen auf Modellebene (Instanzen) angeboten. Schließlich ist es schwierig, einen Überblick über das integrierte Metamodell zu behalten, wenn die zu integrierenden Metamodelle komplex sind. In diesem Fall können die Graph Rewriting Rules derart komplex werden, dass sie nur noch von Experten verwaltet werden können.

Anforderung 3 (Nutzung) Der Nutzen von Tool-Integrationsansätzen beschränkt sich auf die Konsistenzsicherung von Modellen zwischen Informationssystemen. Die Integration der Metamodelle erfolgt nur paarweise, und ein globales Integrationsmodell fehlt in diesem Fall.

Anforderung 4 (Monitoring) Es ist keine Arbeit bekannt, die sich mit dem Monitoring des Prozesses einer Tool-Integration beschäftigt. Durch die Definition von Graph Rewriting Rules wird angenommen, dass Modelle stets konsistent sind und eine manuelle Integration nicht erforderlich ist.

Anforderung 5 (Change Management) Ziel der Tool-Integration ist die Konsistenzsicherung von Modellen zwischen verschiedenen Informationssystemen. D.h. Änderungen von Instanzen bzw. Modellen werden durch die Graph Rewriting Rules an das jeweils korrespondierende Informationssystem propagiert. Anders verhält es sich mit Änderungen der Metamodelle. Bisher ist kein Ansatz bekannt, der sich mit der expliziten Änderung von Metamodellen und den daraus resultierenden Auswirkungen auf eine bestehende Integration befasst. Im Fall einer Metamodelländerung muss die Integration der Schemakonzepte erneut durchgeführt werden. Dazu zählt auch die Erstellung eines Abbildungsmetamodells sowie die Definition von Graph Rewriting Rules.

Anforderung 6 (Methodiken) [KRS08] beschreibt einen Prozess zur Integration von Informationssystemen, die sowohl Anforderungen als auch Use-Case-Diagramme zur Entwicklung

mechatronischer Systeme verwalten. Dazu definieren die Autoren eine Architektur, die aus einem *Integrationsservice*, einem *Modellrepository* und *Tooladaptern* besteht. Der Integrationsservice ist für die Erstellung sog. *Traceability Links* zuständig, die zwischen Schemakonzepten unterschiedlicher Metamodelle definiert werden. Der Service legt diese Informationen im Modellrepository ab und ist darüber hinaus für die Propagation von Änderungen in Modelle der Informationssysteme zuständig. Für jedes Informationssystem, dass integriert werden soll, muss ein Tooladapter implementiert werden. Explizite Konzepte zur Abbildung unterschiedlicher Variabilitäts- und Versionierungskonzepte gibt es nicht.

4.1.5. Weitere Vorgehensweisen

Wie in Abschnitt 2.3 erläutert, ist Heterogenität eines der Hauptprobleme bei der Integration von Informationssystemen. [GOP02, Tho09] schlagen Lösungen für die Interoperabilität heterogener Informationssysteme (vgl. Abschnitt 2.3.6) vor und bewerten die einzelnen Ansätze.

- **Kategorie 1:** Alle bestehenden Informationssysteme werden durch ein System abgelöst, das alle Daten beinhaltet.
- **Kategorie 2:** Zwischen jeweils zwei Informationssystemen werden Datentransformationen realisiert.
- **Kategorie 3:** Daten der Informationssysteme werden in ein neutrales Format überführt.
- **Kategorie 4:** Es folgt eine manuelle Wiedereingabe von Daten.

Auf den ersten Blick erscheint die Vorgabe eines einzelnen Systems als Lösung für den Austausch von Daten vorteilhaft, jedoch gibt es einige Argumente, die gegen dieses Vorgehen sprechen. Innerhalb eines Unternehmens haben Abteilungen unterschiedliche Anforderungen, die möglicherweise nicht alle durch ein einzelnes System abgedeckt werden können. So kann ein einzelnes System nur als Kompromiss angesehen werden. Schwieriger wird es, ein einzelnes System entlang einer Wertschöpfungskette mit verschiedenen Zulieferern zu etablieren. Letztere arbeiten häufig für mehrere OEMs und müssten dem entsprechend für jeden OEM ein System zum Datenaustausch vorhalten, was erhebliche Wartungs- und Schulungskosten nach sich zieht.
In der Produktentwicklung wurde bereits versucht, mit PDM-Systemen ein zentrales System für die Dokumentation von Entwicklungsergebnissen zu etablieren [Sta11]. Trotzdem werden heute viele kleine bereichsabhängige Applikationen eingesetzt und entwickelt, die schneller an Änderungen (Gesetze, Innovationen) angepasst werden können. Schließlich löst ein einziges System nicht das Problem, dass Datenmodelländerungen auftreten können oder neue Applikationen integriert werden müssen. Des weiteren ist es aus praktischer Sicht nicht realisierbar, da bestehende IT-Landschaften in der Produktentwicklung über Jahrzehnte gewachsen und dem entsprechend komplex sind.
Punkt-zu-Punkt Datenaustausch hat den Nachteil, dass die Anzahl der benötigten Transformatoren mit der Anzahl der zu integrierenden Systeme quadratisch steigt.[1] Zusätzlich müssen bei Datenmodelländerungen eines Systems im ungünstigsten Fall alle Transformatoren zu diesem System angepasst werden. Vertreter sind Peer-Daten-Management-Systeme oder Ansätze zur

[1] Sollen Daten zwischen alle Systeme untereinander transformiert werden, so werden maximal $\frac{n*(n-1)}{2}$ Transformationen benötigt.

Tool-Integration. Schließlich existiert die Möglichkeit, sich auf ein gemeinsames neutrales Format zu einigen und dieses zu standardisieren. Auch wenn ein solches Format einen Kompromiss zwischen allen beteiligten Systemen darstellt, bietet es doch die Möglichkeit, Änderungen am gemeinsamen Format koordiniert abzustimmen. Im Abschnitt 4.2 werden Standards betrachtet, die in der Produktentwicklung relevant sind.
Die manuelle Wiedereingabe von Daten ist mit enormen Kosten verbunden und kann bei Fehleingaben weitere Kosten nach sich ziehen. Sie ist somit nur in sehr kleinem Umfang und in Ausnahmefällen eine mögliche Lösung.

4.2. Standards

Eine Möglichkeit zur Überwindung von Heterogenität, ist die Standardisierung des Austauschs von Informationen zwischen unterschiedlichen Unternehmen. Während die Lebenszeit eines Informationssystems nur wenige Jahre betragen kann, müssen Produktdaten oftmals bis zu mehrere Jahrzehnte aufbewahrt werden [Tho09]. Folglich ist es sinnvoll, nachhaltige Standards zu entwickeln, die diesen Anforderungen gerecht werden. Im Umfeld der Produktentwicklung existiert eine Vielzahl solcher Standards, die unterschiedliche Phasen im Entwicklungsprozess abdecken. Dabei lassen sich Standards in verschiedene Kategorien einteilen [SRF+05, RSB+08].
Abbildung 4.14 zeigt eine schematische Übersicht der Zusammenhänge der verschiedenen Standards. Mit Hilfe von *Informationsmodellstandards* lässt sich Wissen konzeptualisieren, d.h. Konzepte der Realwelt müssen mit Hilfe von Artefakten abstrahiert und beschrieben werden. Weiter existieren *Inhaltsstandards*, die Schemakonzepte für verschiedene Anwendungsbereiche spezifizieren. Für die Produktentwicklung etwa sind STEP, SysML und AUTOSAR zu nennen. Während STEP versucht, alle Bereiche der Produktentwicklung zu unterstützen, fokussieren SysML[2] und AUTOSAR auf der Modellierung von Systemen und Komponenten eines Produktes. Modellinstanzen dieser Inhaltsstandards können mittels verschiedener *Informationsaustauschstandards* ausgetauscht werden. Erwähnenswert sind z.B. EDI, XML, SOAP und RDF. Im Folgenden werden die wichtigsten Inhaltsstandards für Produktdaten erläutert und die mit ihnen einhergehenden Herausforderungen diskutiert. Schließlich erfolgt der Abgleich mit den Anforderungen aus Abschnitt 3.3.

4.2.1. Informationsmodellstandards

Informationsmodelle lassen sich mit Hilfe unterschiedlicher Standards modellieren. Zu den am weitesten verbreiteten Standards zählen die *Unified Modeling Language* (UML), das relationale Datenmodell, die *Ontology Web Language* (OWL) und, im Zusammenhang mit Produktdaten zu nennen, *EXPRESS*. Die verschiedenen Standards unterscheiden sich aufgrund der verschiedenen Anwendungsdomänen in ihrer Methodik und Ausdrucksmächtigkeit. Während die UML eine grafische Sprache zur Modellierung unterschiedlicher Aspekte von Softwarekomponenten und -systemen ist, dient OWL zur Beschreibung semantischer Wissensnetze und bietet, über die Modellierung hinaus, die Möglichkeit Schlussfolgerungen (engl. Inferences) aus bestehendem Wissen mit Hilfe von Algorithmen abzuleiten. EXPRESS ist eigens für die ISO 10303[3] entwickelt

[2] Systems Modeling Language ist eine auf UML 2 basierende Modellierungssprache für das Systems Engineering.
[3] ISO Normreihe 10303 „Industrial automation systems and integration - Product data representation and exchange“

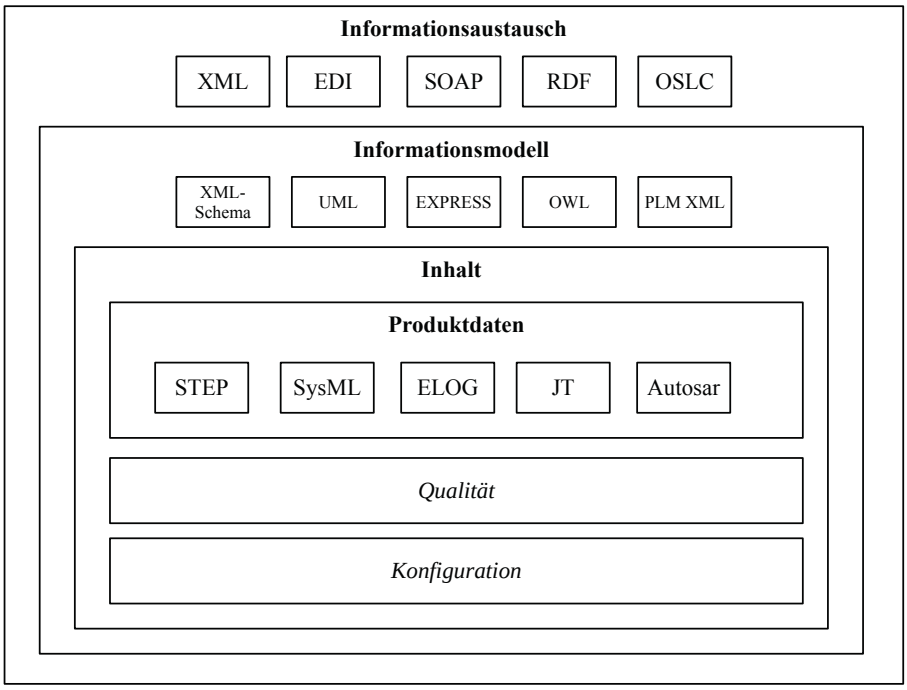


Abbildung 4.14.: Inhaltsstandards im Produktlebenszyklus

worden, um eine einheitliche und systemunabhängige Beschreibungssprache für Produktdatenkonzepte zu bieten. Neben der Beschreibung der Struktur sog. *Entitäten (entities)* mit *Attributen (attributes)* lassen sich komplexe Schreibregeln mit Hilfe arithmetischer, logischer und numerischer Funktionen und Operatoren definieren. Eine logisch komplette Einheit an Entitäten wird als *Schema (schema)* bezeichnet.

4.2.2. Inhaltsaustauschstandards

Für den Austausch sowohl von Datenmodellen als auch deren Instanzen existiert eine Vielzahl an Standards. Zu ihnen zählen *EDI (electronic data interchange)* und *XML (extensible markup language)*. EDI fasst unterschiedliche elektronische Transferverfahren (z.B. ebXML, VDA recommendations) zusammen, die auf XML-Schema basieren. In der Vergangenheit gab es weitere Initiativen, wie etwa *STEPml*[4] oder auch das XML-basiertes Austauschformat für Produktdaten *PDML (Product Data Markup Langauge)*, welche beide ebenfalls auf XML basieren. In den letzten Jahren kommt vermehrt PLM XML[5] zum Einsatz, ein Standard der die Beschreibung von Produktstrukturen durch XML ermöglicht.

4.2.3. Inhaltsstandards

Die Internationale Organisation für Normung (ISO) besitzt ein eigenes technisches Komitee (engl. Technical Committee, TC), das für die Automatisierung und Integration von Systemen zuständig ist. Dieses TC und im Speziellen das Subkomitee 4 sind für den Entwurf und die

[4] A library of XML specifications
[5] PLM XML ist kein internationaler Standard, sondern ein Format der Siemens PLM Software

Aktualisierung einer Vielzahl technischer Standards und Verfahrensstandards für industrielle Daten verantwortlich. Zu den wichtigsten Standards für den Austausch und die Integration von Produktdaten zählen *ISO 8000: Data Quality*, *ISO 10303: Standard for the Exchange of Product Model Data (STEP)*, und *ISO 18876: Integration of Industrial Data for Exchange, Access, and Sharing* (IIDEAS).
Die Ziele dieser Standards sind in allen Fällen ähnlich: Viele OEMs und Zulieferer verwenden unterschiedliche Informationssysteme (z.B. CAD- oder PDM-Systeme) basierend auf unterschiedlichen Datenmodellstrukturen. Dadurch ist der Austausch von Informationen zwischen OEMs und Zulieferern nicht ohne Weiteres möglich, und es können unterschiedliche Probleme auftreten. Hier muss zwischen *Datenaustauschproblemen* (z.B. Flächen und Linien treffen sich nicht) und *Datenqualitätsproblemen* (z.B. doppelte Einträge) unterschieden werden.
Um diese Probleme zu adressieren, begann Mitte der 80er Jahre die Arbeit an der ISO Normreihe 10303 *„Industrial automation systems and integration - Product data representation and exchange“*, informell auch als *„Standard for the Exchange of Product Data (STEP)“* bekannt. Während ISO 10303 verschiedene Standards für den Austausch von Produktdaten umfasst, werden in den ISO 8000 Standards verschiedene Kriterien für die Qualität von Daten definiert. Eng verbunden mit letzterem ist die Normreihe ISO 22745, die eine zentrale Registry für die eindeutige Identifikation von Personen, Organisationen, Orten, Produkten, Services, Prozessen, Regeln und Regularien beschreibt.
Die diversen Standards haben unterschiedliche Annahmen und Ziele und lassen sich daher nur schwer in Einklang bringen. Dieses Problem kann aus ökonomischen Gesichtspunkten nicht durch eine zentrale Stelle oder Organisation gelöst werden, vielmehr ist ein offener partizipativer Prozess notwendig [SRF+05].
Im Folgenden werden technische Standards von ISO 10303 sowie der Verfahrensstandard ISO 8000 näher beleuchtet. Darüber hinaus werden noch weitere Standards nationaler Verbände betrachtet und mit den Anforderungen aus Abschnitt 3.3 abgeglichen.

ISO 10303: Standard for the Exchange of Product Model Data (STEP)

STEP ist eine modulare ISO Normreihe mit dem Ziel, den Austausch von durch Computer-interpretierbaren Produktinformationen über den gesamten Lebenszyklus eines Produktes hinweg zu ermöglichen. Hauptaugenmerk während der Entwicklung des Standards war es vor allem, den Austausch geometrischer Modelle zwischen Herstellern und Zulieferern zu ermöglichen. Dabei stand die Übertragung von Volumenmodellen, die auch für den Aufbau digitaler Prototypen (Digital Mockup) eine wichtige Voraussetzung sind, im Vordergrund [XN09].
Kern der ISO Normreihe sind sog. *Application Protocols (AP)*, die komplexe Datenmodelle für spezifische Produktdatenanwendungen darstellen. In Abbildung 4.15 ist die Architektur von STEP dargestellt. Application Protocols können sich aus verschiedenen *Application Modules (AMs)* zusammensetzen, die verschiedene wiederverwendbare Entitäten enthalten. Grundlage für diese Module bilden die gemeinsamen Ressourcen, die auch als integriertes Produktmodell bezeichnet werden [AT00]. Dieses setzt sich aus den anwendungsunabhängigen Produktmerkmalen, etwa dem Geometriemodell, der Produktstruktur oder dem Präsentationsmodell sowie den anwendungsabhängigen Elementen, etwa Konstruktion oder Finite Element Analyse, zusammen. Erstere werden als *Integrated Generic Resources*, letztere als *Integrated Application Resources* bezeichnet. Schließlich existieren *Application Interpreted Constructs*, die Spezifalisierungen der vorherigen beiden Gruppen darstellen und in mehreren Application Protocols wiederverwendbar

sind. Die Beschreibung aller Ressourcen, Konstrukte und Protokolle basiert auf der Informationsmodellierungssprache EXPRESS, die teilweise grafisch durch EXPRESS-G dargestellt werden kann. Die Implementierung kann durch unterschiedliche Methoden erfolgen. Bspw. existieren Abbildungen auf verschiedene Programmiersprachen (z.B. C, C++ und Java) ebenso besteht die Möglichkeit der Ausgabe als Text- oder XML-Datei.
Der Prozess zur Erstellung und Genehmigung eines neuen AP ist komplex und durchläuft mehrere Phasen, in denen verschiedene Modelle entstehen [XN09]: Zunächst wird ein *Application Activity Model* (AAM) erstellt, das alle Aktivitäten und Datenflüsse für ein zu erstellendes Application Protocol enthält. Es dient als Diskussionsgrundlage nachfolgender Verfeinerungsrunden. Auf Basis dieses Modells wir das sog. *Application Reference Model* (ARM) erstellt, das alle Daten enthält, die für eine spezifische Anwendungsdomäne benötigt werden. Das ARM wird in der eigens für STEP entwickelten Datenmodellierungssprache EXPRESS spezifiziert, damit dieses von Anwendern der Zieldomäne verstanden werden können. Zusätzlich existiert die grafische Modellierung EXPRESS-G, die jedoch nicht die komplette Ausdrucksmächtigkeit von EXPRESS besitzt. [6] Schließlich wird dieses Modell von einem STEP-Experten mit Hilfe gemeinsamer integrierter Ressourcen der STEP-Spezifikationen (engl. generic integrated resources) auf ein *Application Interpreted Model* (AIM) abgebildet.
Um STEP zu modularisieren, wurden über die Zeit unterschiedliche Konstrukte eingeführt: Zunächst wurden *Unit of Functionality* und *Application Interpreted Constructs* (AICs) eingeführt. Diese wurden im Laufe der Zeit durch sog. *Application Modules* (AM) ersetzt. Letztere stellen im Prinzip kleine Application Protocols dar, die wiederverwendet werden können und besitzen wie Applications Protocols interpretierte Modelle, die als *Module Interpreted Model* bezeichnet werden. Application Modules wiederum können weitere Applications Modules referenzieren und somit komplexe Datenmodelle aufbauen. Seit der Initiierung von STEP sind eine Vielzahl von Application Protocols für unterschiedliche Domänen entstanden.
Zu den wichtigsten Application Protocols (APs) der Automobilindustrie zählen:

- *AP 203: Configuration controlled 3D design of mechanical parts and assemblies*,
- *AP 204: Mechanical design using boundary representation*,
- *AP 210: Electronic assembly, interconnect, and packaging design*,
- *AP 212: Electrotechnical design and installation*,
- *AP 214: Core data for automotive mechanical design processes*,
- *AP 242: Managed model based 3D engineering*, wurden 2014 die beiden APs 203 und 214 zusammengeführt.

„Wichtigster Themenbereich in AP 214 ist seit der Initiierung die Produktstruktur und hierbei insbesondere die Handhabung der komplexen Variantenvielfalt, die im Automobilbau besonders ausgeprägt ist. Dies beinhaltet Themen des Produktdatenmanagements wie Spezifikation, Konfiguration, Klassifikation und Gültigkeit, bezieht sich aber auch auf Geometrien und Konstruktionselemente sowie deren graphische Darstellung" [AT00].

[6] EXPRESS bietet neben der Modellierung von Entitäten, Attributen und Beziehungen auch die Möglichkeit komplexe Integritätsbedingungen zu definieren.

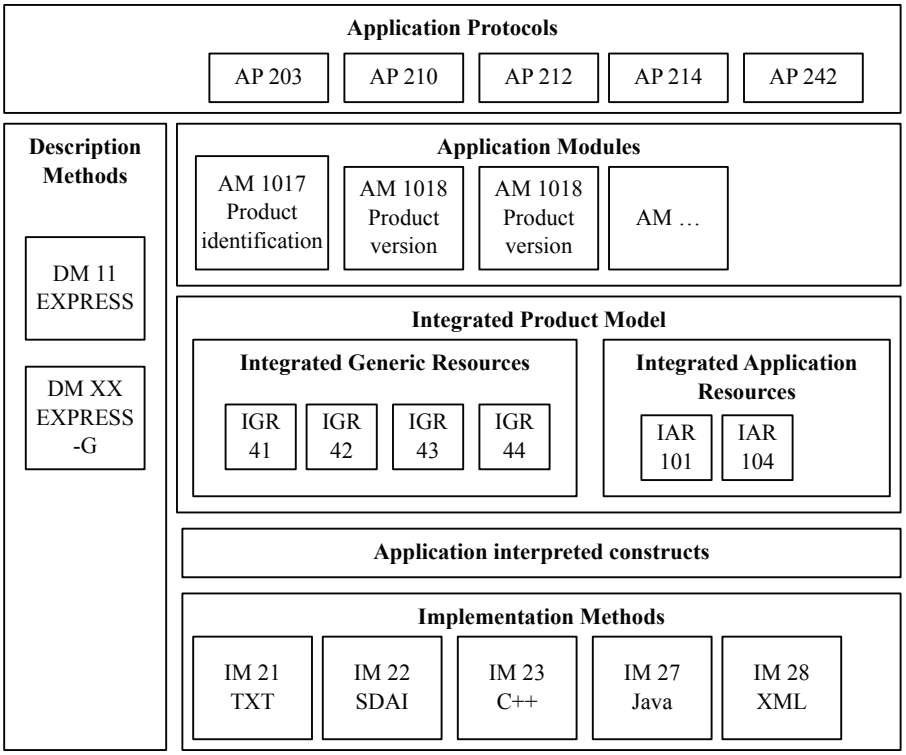


Abbildung 4.15.: . In Anlehnung an *STEP on a Page* `https://web.archive.org/web/20161221123916/http://www.mel.nist.gov/sc5/soap/soapgrf030407.pdf` (abgerufen am 19.11.2017)

In Abbildung 4.16 sind die wesentlichen Entitäten der oben genannten Application Protocols zur Repräsentation von Produktdaten und dessen Konfiguration illustriert. An den einzelnen Entitäten sind jeweils die entsprechenden Application Modules dokumentiert. Application Modules enthalten Entitäten (inklusive Attributen), von denen mehr als 900 in STEP spezifiziert sind. Der Übersicht halber sind die Entitäten als UML-Klassendiagramm anstelle von EXPRESS dargestellt. Zentrale Entität ist das *Product*, welches sowohl ein Bauteil als auch ein Dokument beschreiben kann. Jedes Produkt ist Teil eines Produktkontextes (z.B. „mechanisch“). Variantenbeziehungen zwischen Produkten werden mittels der Entität *alternate_product_relationship* hergestellt. Unterschiedliche Versionen eines Produktes werden in sog. *product_definition_formation* abgelegt. Darüber hinaus können auf einzelne Produktversionen von diesen Sichten (*product_definition*) gebildet werden und Produkte unterschiedlichen Kategorien (z.B. Bauteil, Zusammenbau, Detail oder Standard) zugeordnet werden (*product_related_product_category*). STEP definiert auch die Möglichkeit explizite Konfigurationen zu definieren. Dazu fasst ein *product_concept* mehrere *configuration_item*s zusammen, dessen Ausprägungen Referenzen auf Produktversionen sind (*configuration_design*).

Das integrierte Produktmodell aus Abbildung 4.16 soll als Integrationsplattform für Informationssysteme dienen, die entlang der verschiedenen Phasen der Produktentwicklung eingesetzt werden. Dadurch sollen alle Produktdaten abgebildet und maschinenverarbeitbar sein, wodurch ein „durchgängiger Informationsfluss“ entsteht [AT00]. Beim integrierten Produktmodell handelt es sich um eine allgemeingültige, abstrakte Beschreibung von Produktdaten für einzelne

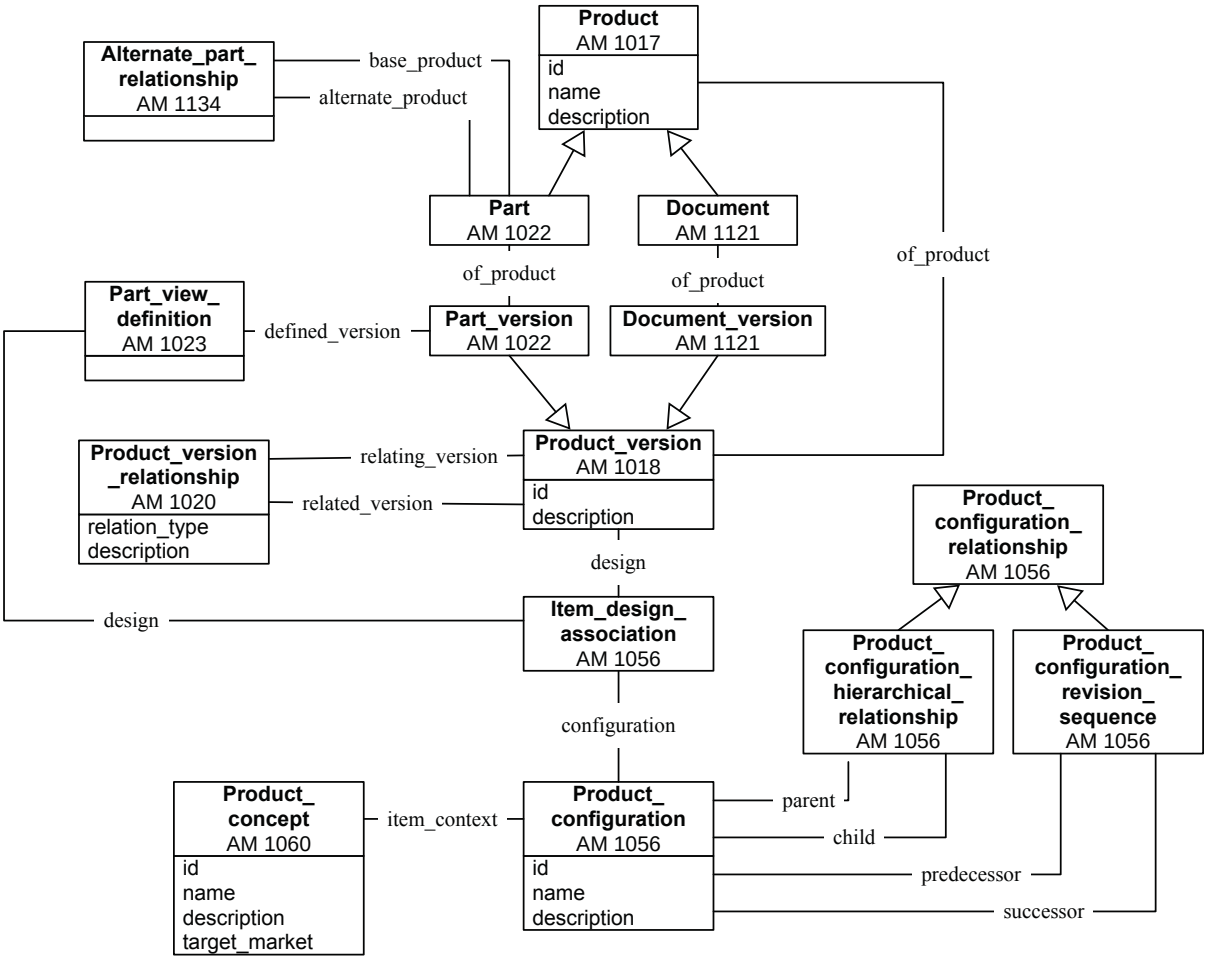


Abbildung 4.16.: Ausschnitt von STEP-Entitäten verschiedener Application Modules

Domänen (z.B. Automobil- oder Elektroindustrie) erfolgt auf Basis dieser eine Spezialisierung des Modells.

Neben dem integrierten Produktmodell existiert das *STEP PDM Schema* [UB02], welches diejenigen Entitäten aus den STEP-Ressourcen enthält, die für das Produktdatenmanagement notwendig sind. Hierbei handelt es sich nicht um einen offiziellen Standard, sondern um eine Initiative der ProSTEP iViP[7] und PDES[8], um Interoperabilität zwischen den verschiedenen Application Protocols zu ermöglichen. Zu den Hauptfunktionalitäten des Schemas zählen die Identifikation, Versionierung, Klassifikation und Strukturierung sowie Eigenschaften von Bauteilen, Projektmanagement und Änderungsprozesse.

[7] ProSTEP iViP ist ein Verein von Mitgliedern aus Industrie und Forschung, der Anforderungen von OEMs und Zulieferern für die Produktentwicklung und -fertigung formuliert und Standards definiert.

[8] PDES, Inc. ist ein internationales Konsortium, das die Entwicklung und Implementierung von Standards für den Austausch von Produktdaten forciert.

ISO 18876: Integration of Industrial Data for Exchange, Access, and Sharing (IIDEAS)

Parallel zu STEP ist ein weiterer ISO Standard entstanden, der ähnliche Ziele verfolgt, jedoch einen breiteren Fokus einnimmt und sich nicht auf Produktdaten beschränkt. Im Mittelpunkt steht das Teilen und Integrieren von Daten anstelle ihres Austausches, wie er bei STEP angestrebt wird. Im Zuge der Entwicklung von ISO 18876 wurde erkannt, dass es für die Integration von Daten notwendig ist, die Identität zusammengehöriger Daten aus heterogenen Quellen zu bestimmen [Wes96]. D.h., bei der Integration sollen diejenigen Teile identifiziert werden, welche das selbe Realweltobjekt repräsentieren.
Der ISO-Standard 18876 definiert sowohl eine Architektur als auch eine Methodik, um Daten aus unterschiedlichen Quellen zu integrieren [ISO03b, ISO03c]. Letztere können durch unterschiedliche Modelle beschrieben und durch heterogene Modellierungssprachen realisiert sein. Darüber hinaus sollen Konflikte zwischen Modellen aufgelöst und Daten, die unterschiedliche kodiert sind, transformiert werden.

Architektur. Die Integration von Datenquellen erfolgt durch sog. *Integrationsmodelle*, die wiederum mit anderen Datenmodellen zu neuen Integrationsmodellen integriert werden können. Ein Datenmodell kann in eines oder mehrere Integrationsmodellen integriert werden. Die Kodierung von Daten und Modellen kann wiederum durch unterschiedliche Formate (z.B. SGML, XML, UML und EXPRESS) geschehen.
Ein Integrationsmodell setzt sich aus primitiven und abgeleiteten Konzepten zusammen. Erstere umfassen *Foundation Concepts*, *General Concepts* und *Specific Concepts*. Beispiele für Foundations Concepts sind Klassifikation, Beziehungen und Komposition. Zu den General Concepts zählen u.a. physikalische Konzepte. Specific Concepts schließlich beziehen sich auf einen eingeschränkten Anwendungsbereich (z.B. Automobilindustrie).
Die Integration von Datenmodellen erfolgt durch die Definition von Abbildungen (*Mappings*). Diese spezifizieren die Transformationen, um Instanzen eines Modells in einem anderen Modell repräsentieren zu können. Dabei werden bestimmte Annahmen getroffen, etwa das Mappings auf Transformationen der Struktur, Terminologie und Kodierung der Datenmodelle beschränkt sind. Transformationen sind immer bidirektional, was jedoch bedeutet, dass die Transformationsrichtungen separat definiert werden müssen.
Ziel des Integrationsprozesses ist es, die selben Informationen eines Daten- oder Integrationmodells mit Hilfe von Transformationen darzustellen, ohne dass bei der Integration die Semantik verloren geht. Als Ergebnis dieses Prozesses resultieren Abbildungen zwischen Datenmodellen von Informationssystemen und Teilmengen des Integrationsmodells. Gegebenenfalls muss letzteres im Zuge der Integration erweitert werden, so dass dieses die Schemakonzepte der Datenmodelle präzise ausdrücken kann.
Der Integrationsprozess lässt sich in drei unterschiedliche Szenarien einteilen. Im ersten Szenario existieren vor Start des Integrationsprozesses sowohl das Integrationsmodell als auch die Datenmodelle der zu integrierenden Informationssysteme. Im zweiten Szenario existieren zwar die Datenmodelle der zu integrierenden Informationssysteme, jedoch noch kein Integrationsmodell. Im dritten Szenario existiert das Integrationsmodell, jedoch noch keine Datenmodelle der zu integrierenden Informationssysteme, d.h. lediglich die Anforderungen an diese Datenmodelle liegen vor.
Im Folgenden wird lediglich das erste Szenario beschrieben, da dieses die anderen Szenarien mit abdeckt. Zunächst werden das Datenmodell eines zu integrierenden Informationssystems analy-

siert und äquivalente Schemakonzepte aus dem Integrationsmodell identifiziert. Gegebenenfalls muss das Integrationsmodell erweitert werden, falls nicht alle Schemakonzepte des zu integrierenden Datenmodells abgebildet werden können. Anschließend wird die Teilmenge des Integrationsmodells ermittelt, welche die Konzepte des zu integrierenden Datenmodells beschreibt. Da Datenmodelle immer in dem Anwendungskontext betrachtet werden müssen, für den sie erstellt wurden, muss dieser, so fern nicht explizit im Modell vorhanden, bei der Integration berücksichtigt werden und ggf. in die Mappingspezifikation mit aufgenommen werden. Als Beispiel kann etwa das Informationssystem `E-PDM` aus Abschnitt 3.2.1 betrachtet werden. Das Schemakonzept `Komponente` beschreibt implizit, dass es sich im Kontext von `E-PDM` um elektrische Komponenten handelt. Dieses wird jedoch nicht aus der Bezeichnung ersichtlich und sollte im Zuge einer Integration explizit in einer Mappingspezifikation dokumentiert werden.

Nun werden für beide Integrationsrichtungen strukturelle und terminologische Transformationen zwischen Datenmodell und Integrationsmodell definiert. Sollten beide unterschiedlich kodiert sein, muss zwischen beiden Sprachen ebenfalls eine Transformation definiert sein. Diese Schritte werden solange wiederholt, bis alle zu integrierenden Informationssysteme abgebildet sind.

Abbildung 4.17 illustriert die ISO 18876 Architektur. Sowohl das Integrationsmodel, die Mapping-Spezifikationen sowie die Informationssysteme besitzen Schemakonzepte und Instanzen, die in Schemata definiert sind. Diese können in verschiedenen Informationsmodellen (z.B. UML, XML-Schema, relationales Datenmodell) spezifiziert sein. Das Integrationsmodell kann entweder ein Modell mit unterschiedlichen Abstraktionsstrufen oder Hierarchien sein, oder es existieren mehrere Integrationsmodelle, die integriert werden. Für jedes zu integrierende Informationssystem ist genau eine Mapping-Spezifikation definiert, die einen Ausschnitt des Integration Model referenziert. Neben strukturellen Transformationen für die Abbildung von Schemakonzepten sowie terminologischen Transformationen für die Abbildung von Instanzen, können Einschränkungen (engl. Constraints) definiert werden. Diese werden dann erforderlich, wenn Informationssysteme, die integriert werden sollen, zueinander in Konflikt stehende Einschränkungen besitzen. Dann wird es notwendig, dass Konzepte im Integrationsmodell allgemeiner definiert werden als in den zu integrierenden Modellen.

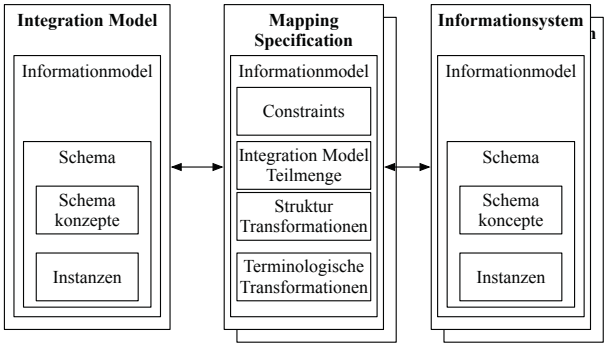


Abbildung 4.17.: In Anlehnung an [ISO03b]

Für die Konsolidierung von Instanzen muss eine gemeinsame, zuverlässige und dauerhafte Identifikation zusammengehöriger Instanzen vorhanden sein. Sollten bestehende Identifikationsmerkmale wiederverwendet werden können, müssen keine zusätzlichen Attribute im Integrationsmodell spezifiziert werden.

Methodik. Die Methodik beschreibt wie Integrationsmodelle erstellt, erweitert und aktualisiert werden. Dabei werden unterschiedliche Szenarien unterstützt, etwa die initiale Integration von zwei oder mehreren Informationssystemen in ein neues Integrationsmodell. Im Folgenden werden die verschiedenen Phasen der Methodik kurz erläutert.
Die erste Phase definiert die *Analyse* der Informationssysteme und die damit verbundene Erstellung des Integrationsmodells. Zunächst muss der Kontext des Informationsmodells festgelegt werden. Steht mehr als ein Integrationsmodell zur Auswahl, so wird jenes gewählt, das die größte Überdeckung mit den zu integrierenden Informationssystemen hat. Anschließend werden die Konzepte der Informationssysteme untersucht und Beziehungen zu Konzepten im Integrationsmodell bestimmt. Hier unterscheidet die Methodik zwischen Synonymen, Homonymen, Identität, Kompatibilität, Inkompatibilität, Komplexität und Partitionierung.
In der zweiten Phase geht es um die Erweiterung eines Integrationsmodells für den Fall dass während der Analysephase Konzepte identifiziert worden sind, für die keine Entsprechung im Integrationsmodell gefunden werden kann. Dazu müssen dem Integrationsmodell neue Konzepte hinzugefügt werden, ohne aber bestehende Konzepte zu ändern. Sollte es nicht möglich sein, Änderungen am Integrationsmodell durchzuführen, welche eine Integration neuer Konzepte ermöglicht, muss ein neues Integrationsmodell erstellt werden. In diesem Fall wird das alte Integrationsmodell zu einem Integrationsmodell, welches ebenfalls in das neue Modell integriert werden muss.
Die nächsten beiden Phase beschreibt die Auswahl derjenigen Teilmenge an Konzepten aus dem Integrationsmodell, welche die Abbildung eines Informationssystems ermöglichen sowie die Integration beider. Damit sind das Modell des zu integrierenden Informationssystems sowie die Teilmenge aus dem Integrationsmodell semantisch äquivalent. Da beide sich in Struktur und Terminologie unterscheiden, müssen diese Unterschiede durch Transformationen aufgelöst werden.

4.2.4. Standards für das Konfigurationsmanagement

Wie in Abschnitt 2.2.3 erläutert, ist das Konfigurationsmanagement einer der wichtigsten Teilbereiche der Produktentwicklung. Im seinem Kontext existieren unterschiedliche Standards und Richtlinien für die Erstellung und Pflege von Konfigurationen. Zu den verbreitetsten zählen der ANSI Standard *EIA-649*[EIA98][9] sowie der ISO Standard *ISO 10007*[10] [ISO03a].
„Configuration management (CM): A management process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design and operational information throughout its life.“ [EIA98]
„Configuration management documents the product's configuration“ „Product configuration information: requirements for product design, realization, verification, operation and support“ ISO 10007
EIA-649 definiert grundlegende Begriffe und Aktivitäten des Konfigurationsmanagements und liefert dazu 50 Prinzipien. Das Konfigurationsmanagement umfasst folgenden Prozessen:

- **Konfigurations- und Planungsmanagement:** Definition des Geltungsbereiches, Integration in Unternehmensprozesse.

[9] National Consensus Standard for Configuration Management
[10] Quality management systems – Guidelines for configuration management

- **Konfigurationsidentifikation:** Definition der Produktstruktur und -attribute, Vergabe eindeutiger Bauteilnummern und Etablierung eines Releaseprozesses.
- **Konfigurationsänderungsmanagement:** Identifikation von variablen Anteilen sowie die Dokumentation aller Änderungen.
- **Konfigurationsstatuskontrolle:** Realisierung von Informationssystemen, Erfassung aller relevanten Konfigurationsmanagementinformationen.
- **Konfigurationsverifikation und -prüfung:** Sicherstellung der Konsistenz von Releaseinformationen, Überprüfung der Leistungserfüllung.
- **Konfigurationsmanagement für digitale Daten:** Anweisungen für die IT-Unterstützung des Konfigurationsmanagement.

Die Inhalte von EIA 649 und ISO 10007 decken sich in großen Teilen, lediglich der Prozess *Konfigurationmanagement für digitale Daten* wird im ISO-Standard nicht erwähnt.

4.2.5. Qualitätsstandards

Auch bzgl. Datenqualität existieren verschiedene Standards, der verbreitete ISO Standard ISO 9001 [ISO08], der die Anforderungen an Qualitätsmanagementsysteme definiert. Mit ISO 8000 [Ben09] ist ein Standard entstanden, der sich mit der Qualität von Daten im Speziellen beschäftigt. Dazu werden neben unterschiedlichen Rollen für das Qualitätsmanage-ment von Daten auch verschiedene Prozesse beschrieben.

ISO 8000: Data Quality

Bei ISO 8000 handelt es sich um eine Normreihe von Standards zur generellen Datenqualität bzw. zur Qualität von Masterdaten. Unter Masterdaten werden zentrale Geschäftsobjekte (etwa Kunden-, Mitarbeiter-, Zulieferer- oder Produktdaten) verstanden, die durch eine Vielzahl von Informationssystemen innerhalb eines Unternehmens verwendet werden [Los09]. Neben grundlegenden Informationen zur Qualität von Daten wird in ISO 8000 ein Rahmenwerk beschrieben, welches das Management von Datenqualität erlaubt.
Unter Datenqualität wird im Standard die Definition der ISO 9000 referenziert: *„Degree to which a set of inherent characteristics fulfils requirements“*. Die erste Normreihe (Teile 0 bis 99) von ISO 8000 beschreibt die grundlegende Terminologie für Datenqualität. Die zweite Normreihe (Teile 100 bis 199) beschäftigt sich mit der Qualität von Masterdaten. Darunter fallen alle Informationen, Unternehmensunabhängig und fundamental für das operative Geschäft sind. Dazu zählen Informationen zu einzelnen Personen, Organisationen, Standorten, Waren, Services und Regularien. Um eine einheitliche Semantik sowie zentrale Beschreibung von Konzepten zu ermöglichen wird, mit der ISO Normreihe 22745 ein zentrales technisches Verzeichnis als Grundlage für das Datenqualitätsmanagement vorausgesetzt. Mit Hilfe eines solchen Verzeichnisses lassen sich Konzepte eindeutig beschreiben und identifizieren. Des weiteren werden in der Normreihe charakteristische Eigenschaften für Datenqualität beschrieben. Dazu zählen die Datenherkunft (*Provenance*), die Vollständigkeit (*Completeness*) sowie die Genauigkeit (*Accuracy*) von Masterdaten.

Rahmenwerk. Neben grundlegenden Definitionen beschreibt die Normreihe ein Rahmenwerk für das Management von Datenqualität. Dieses umfasst drei Rollen und drei Prozesse (siehe Abbildung 4.18). Zu den Rollen zählen der *Data Manager*, *Data Administrator* und *Data Technician*. Diese führen unterschiedliche Aktivitäten in den Prozessen *Data Operations*, *Data Quality Monitoring* und *Data Quality Improvement* aus.

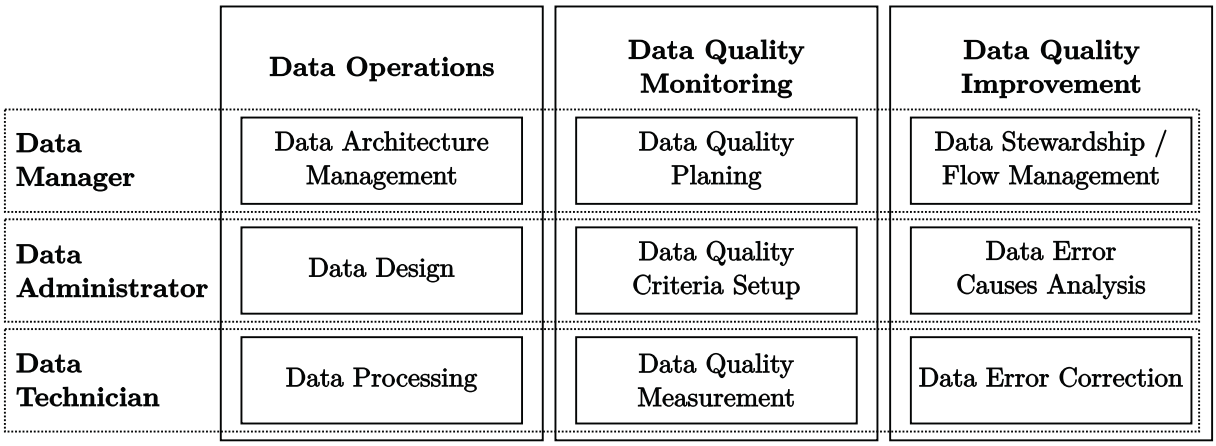


Abbildung 4.18.: ISO 8000-150 Data Quality Management Framework

Der Prozesse *Data Operations* betrachtet Faktoren, welche die Datennutzung und die Datenqualität direkt betreffen:

- *Data Architecture Management* ist für die strategische, unternehmensweite Planung der Datenqualität verantwortlich. Zu den Aktivitäten zählt die Verwaltung unternehmensweiter Datenmodelle, Standards und Schnittstellen sowie deren Koordination zwischen unterschiedlichen Informationssystemen.
- *Data Design* verwaltet Datenstandards und Definitionen, Datenbank-/Systemimplementierungen und deren Konfiguration. Zu den Aufgaben zählen die Entwicklung logischer und physischer Datenmodelle, ebenso wie die Etablierung von Standards und Regeln auf Betriebsebene. Datenadministratoren müssen sensibel für Anforderungen aus dem *Data Architecture Management* sein sowie einen Austausch mit anderen Abteilungen pflegen, um auf Änderungen der Anforderungen reagieren und Systemkonfigurationen anpassen zu können.
- *Data Processing* ist der Prozess auf operativer Ebene, der für die Erzeugung sowie Aktualisierung von Daten verantwortlich ist. Zu den Aufgaben zählt die Überwachung des Datenerstellungs- und Datenänderungsprozesses, sowie die Protokollierung verschiedener Parameter (z.B. Datennutzung, Änderungen).

Der zweite Prozess *Data Quality Monitoring* beurteilt, inwieweit bestimmte Stufen der Datenqualität im Unternehmen erreicht werden. Vor der Aktivität *Data Quality Planning* werden die Ziele des Datenqualitätsmanagement fixiert, die wiederum an den Unternehmenszielen ausgerichtet sein müssen. Im *Data Quality Criteria Setup* werden Methoden und Maße festgelegt, um Datenqualität bestimmen zu können. Im *Data Quality Measurement* wiederum werden diese Methoden verwendet, um die aktuelle Datenqualität im Unternehmen verschiedenen Stufen zuzuordnen.

Der letzte Prozess *Data Quality Improvement* korrigiert gefundene Datenfehler und beseitigt die Gründe für diese. Im *Data Stewardship and Flow Management* werden Datenflüsse und Verantwortlichkeiten analysiert und Datenoperationen verwaltet. In der Aktivität *Data Error Cause Analysis* werden die Gründe für Fehler analysiert, um diese nachhaltig zu eliminieren. Schließlich werden im *Data Error Correction* Daten korrigiert, die nicht vorgegebenen Standards und Kriterien entsprechen.

SASIG - Product Data Quality for the Global Automotive Industry

Auf im Umfeld der Produktentwicklung in der Automobilindustrie hat sich eine Reihe von Standards und Richtlinien für die Qualität von Produktdaten etabliert. In den meisten Fällen handelt es sich um Vorgaben für den Austausch von Geometrien zwischen Herstellern und Lieferanten. [SAS05] stellt Richtlinien für die Qualität von Produktdaten zusammen, die sich auf CAD-Aspekte beschränken. Zunächst werden die Begriffe „Produktdaten“ und „Produktdatenqualität“ definiert. Produktdaten sind demnach alle Daten, die von der Konzeption bis zur Fertigung eines Produktes benötigt werden, etwa CAD-[11], CAM-[12], CAE-[13] und PDM- Daten. Die Richtlinie umfasst bisher nur Qualitätskriterien für geometrische Modelle und Zeichnungen; Kriterien für weitere Bereiche sind geplant.

4.2.6. Abgleich der Anforderungen

Die diskutierten Standards werden nun mit den Anforderungen aus Abschnitt 3.3 abgeglichen.

Anforderung 1 (Bedarfsgerechte Integration, Berücksichtigung von Variabilität und Versionierung)

Mit dem ISO Standard ISO 18876 wird eine grobe Integrationsarchitektur skizziert, jedoch keine detaillierte Spezifikation gegeben. Diese Architektur kann als Grundlage für ein Produktdatenintegrationssystem herangezogen werden. Dabei bleiben jedoch viele Herausforderungen offen. Es wird lediglich gefordert, dass die Identität von Instanzen ermittelt werden kann, um die Instanzen zu integrieren bzw. konsolidieren. Gleiches gilt für die Erstellung der Mappings zwischen Integrationsmodell und den Modellen der zu integrierenden Informationssysteme. Wie die genaue Transformation zwischen den Schemakonzepten aussehen soll, bleibt allerdings offen. Damit sind auch keine expliziten Konzepte für die Integration von Konzepten für den Umgang mit Variabilität und Versionierung vorhanden.
ISO 10303 ist in erster Linie entstanden, um den Austausch von Geometriemodellen zwischen Herstellern und Lieferanten zu ermöglichen. In der Literatur existieren einige Analysen zur Verbreitung und zum praktischen Einsatz. In diesem Kontext beschreibt [GOP02] vier Hauptprobleme, die einen praktischen Einsatz von STEP erschweren:

1. Hohe initiale Investitionskosten für die Entwicklung bzw. Etablierung eines Standards, von dem alle beteiligten Industriepartner profitieren,
2. Marktrisiken, die durch Rivalitäten zwischen CAD-Herstellern und Firmen entstehen, die STEP-Softwarebausteine entwickeln,

[11] Computer-aided design
[12] Computer-aided manufacturing
[13] Computer-aided engineering

3. Technische Risiken, die bei der Entwicklung von STEP-Übersetzern und zugehörigen Tools auftreten können sowie

4. Notwendigkeit einen unvoreingenommen Experten zu etablieren, der die Verhandlungen, Entwicklung und Realisierung von Standards leitet.

Hauptanwender von ISO 10303 sind die Automobilbranche sowie die Luft- und Raumfahrtindustrie [AT00]. Der Fokus liegt auf dem Austausch von Geometriedaten. Damit wird ein großer Teil der modernen Produktentwicklung, etwa die E/E-Entwicklung und deren Prozesse [BHR05, MHR06, MHR07], zu wenig berücksichtigt und somit nicht mehr ausreichend unterstützt.
Auf der einen Seite sind die Application Protocols zu komplex und die Weiterentwicklung der Normierungen langsam. Um auf veränderte Rahmenbedingungen (z.B. neue Technologien oder neue Kundenwünsche) schneller eingehen zu können, wurden die Entwicklungszyklen komplexer Produkte in den letzten Jahrzehnten deutlich verkürzt. Folglich werden Application Protocols durch Unternehmen sehr zögerlich umgesetzt.
In STEP werden allen Bauteile und Dokumente als *Product* dokumentiert. Stammen zwei STEP-Files von unterschiedlichen Herstellern bzw. Zulieferern, ist das Problem der Identität von Bauteilen bzw. Dokumenten mit STEP nicht gelöst. STEP dient lediglich dazu, den Austausch technisch zu ermöglichen. Zwar ist die Theorie des Variabilitätsmanagement in der Industrie gut verstanden, jedoch fehlen durchgängige Ansätze, um ein unternehmensweites Variantenmanagement zu ermöglichen.

Anforderung 2 (Benutzerunterstützung)

Weder ISO 10303 noch ISO 18876 besitzen Konzepte für die Unterstützung von Benutzern bei der Integration von Produktdaten.

Anforderung 3 (Nutzung)

ISO 10303 bietet unterschiedliche Möglichkeiten, um Produktdaten zu kodieren. Dazu zählen neben einer API, die auf relationalen Datenbanken basiert, auch die Unterstützung von verschiedenen Programmiersprachen oder auch die Kodierung als Text oder XML-Dokument.

Anforderung 4 (Monitoring)

Wie bei der Benutzeruntstützung betrachten ISO 10303 und ISO 18876 nicht das Monitoring der Integration. Lediglich ISO 8000 sieht das Monitoring der Datenqualität als essentiell an, beschreibt jedoch keine konkrete Lösung.

Anforderung 5 (Change Management)

ISO 10303 betrachtet Änderungen an Datenmodellen und deren Auswirkungen auf bereits integrierte Produktdaten nicht. ISO 18876 beschreibt verschiedene Szenarien der Integration, worunter auch die Anpassung des Integrationsmodells fällt. Wie erwähnt, handelt es sich bei der Architektur um eine grobe Skizze, explizite Lösungskonzepte werden nicht erläutert.

Anforderung 6 (Methodiken)

ISO 8000 und 18876 definieren zwar Rahmenwerke für die Integration bzw. das Qualitätsmanagement von Daten, jedoch ist deren Beschreibung abstrakt gehalten, ohne bestehende Probleme, etwa dem Umgang mit Variabilität und Versionierung bei der Datenintegration, zu lösen.

4.3. Zusammenfassung

Wie in Tabelle 4.1 illustriert, gibt es eine Lücke bei der Integration von Produktdaten, unter Berücksichtigung von Variabilitäts- und Versionierungkonzepten. Auf Grund der verschiedenen Annahmen, findet nur bei Data Warehouse-Systemen eine Überwachung des Integrationsprozesses statt, während die virtuell integrierenden Ansätze (z.B, Föderierte- und Peer-Daten-Management-Systeme) von guter Datenqualität und fehlenden Integrationskonflikten ausgehen.

Ansatz	**A1**	**A1.1**	**A1.2**	**A2**	**A3**	**A4**	**A5**	**A6**
Föderierte Datenbanksysteme	o	-	-	-	o	-	o	o
Data Warhouse Systeme	+	-	-	+	o	+	+	o
Peer-Data-Management-Systeme	o	-	-	-	o	-	o	o
Tool Integration	o	-	-	o	o	-	o	o
ISO 10303	o	-	-	-	o	-	-	-
ISO 18876	o	-	-	o	o	-	o	o
ISO 8000	-	-	-	o	o	o	o	o

Tabelle 4.1.: Abgleich der Anforderungen + : wird unterstützt o : teilweise unterstützt - : nicht unterstützt

Teil II

Lösungskonzepte

5. PROCEED-Rahmenwerk im Überblick

Wie die Anforderungsanalyse aus Kapitel 4 gezeigt hat, gibt es eine Forschungslücke bzgl. der Integration von Produktdaten, wenn grundlegende Konzepte für den Umgang mit Variabilität und Versionierung mit unterstützt werden sollen. Im Folgenden wird ein Überblick über das im Rahmen der vorliegenden Dissertation entwickelte PROCEED-Rahmenwerk gegeben, welches ein Produktdatenintegrationssystem bietet, dass die erläuterten Probleme und Anforderungen adressiert.
Produktdaten besitzen keine allgemeingültige Definition (vgl. Abschnitt 2.2.3). Jedes Unternehmen verwendet eigene Strukturen, Richtlinien und Terminologien für den Umgang mit Produktdaten, so dass jedes PDMS letztlich einmalig ist. Folglich können zwei Produkthersteller dasselbe PDMS verwenden und dennoch auf Grund der unterschiedlichen Unternehmensvorgaben (Produktstrukturen, Benennungen, Variabilitätsmechanismen, Versionierungsstrategien) keine Produktdaten untereinander austauschen (vgl. Abschnitt 3.1). Gemeinsame Basis für den Austausch von Produktdaten zwischen Herstellern und Zulieferern sind Bauteile, d.h. physisch hergestellte Erzeugnisse.
Auch die in Abschnitt 4.2 betrachteten Standards, allen voran STEP, sehen Bauteile als essentielles Konzept der Produktentwicklung an. Darüber hinaus haben alle Produktentwicklungsstandards gemein, dass es unterschiedliche Varianten dieser Bauteile geben kann. Um deren fortlaufende Entwicklung und die Anforderung nach lückenloser Dokumentation zu gewährleisten, werden Bauteile in der Produktentwicklung versioniert. Dem entsprechend findet die Integration heterogener Produktdaten im PROCEED-Rahmenwerk auf Bauteilebene statt.

5.1. Lebenszysklus der Produktdatenintegration

Das PROCEED-Rahmenwerk wird nachfolgend anhand des Integrationslebenszyklus erläutert (siehe Abbildung 5.1). Ziel der Produktdatenintegration ist die Unterstützung von Anwendungsfällen, die integrierte Produktdaten aus heterogenen Informationssystemen der Produktentwicklung benötigen. Da die Qualität von Produktdaten während der Produktentwicklung unterschiedlich sein kann, erfolgt die Integration materialisiert in einem zentralem Modell. Dadurch wird es möglich, ähnlich wie beim ETL-Prozess für Data Warehouses (siehe Abschnitt 4.1.2), Daten zu bereinigen und korrigieren, bevor sie schließlich integriert werden.
Der Integrationslebenszyklus beginnt mit der *Projektvorbereitung* (①), in der die zu realisierenden Anwendungsfälle definiert und die relevanten Informationssysteme ermittelt werden. Anschließend erfolgt die *manuelle Schema-Integration* (②). Im Speziellen werden Abbildungen zwischen korrespondierenden Schemakonzepten aus den zu integrierenden Informationssystemen und dem PDIS definiert. Diese Abbildungsregeln stellen die Grundlagen für die Integration von Instanzen der Schemakonzepte dar. Dazu ist es notwendig, die Identität von Instanzen zwischen den Informationssystemen und dem PDIS zu definieren. Damit ist gemeint, dass diejenigen Attribute der Schemakonzepte identifiziert werden müssen, die Instanzen eindeutig beschreiben.

Zusätzlich werden Aktionen definiert, die während der Integration durchgeführt werden sollen. Die Auswertung der Abbildungsregeln erfolgt in der *automatischen Datenintegration* (③).
Auf Grund unterschiedlicher Datenqualität kann diese Integration allerdings nicht immer für alle Instanzen korrespondierende Instanzen finden. Daher wird eine *manuelle Datenintegration* notwendig (④). Ist die Integration abgeschlossen, kann die *Datennutzung* erfolgen. Bereits mit der manuellen Schemaintegration beginnt die Überwachung des Integrationsprozesses, wodurch es möglich wird, dessen Fortschritt zu beobachten (⑦). Treten Änderungen an Produktdaten auf (⑥), müssen neue bzw. geänderte Schemakonzepte integriert (⑧) oder aber die automatische Integration für neue bzw. geänderte Instanzen erneut durchgeführt werden (vgl. ⑨).

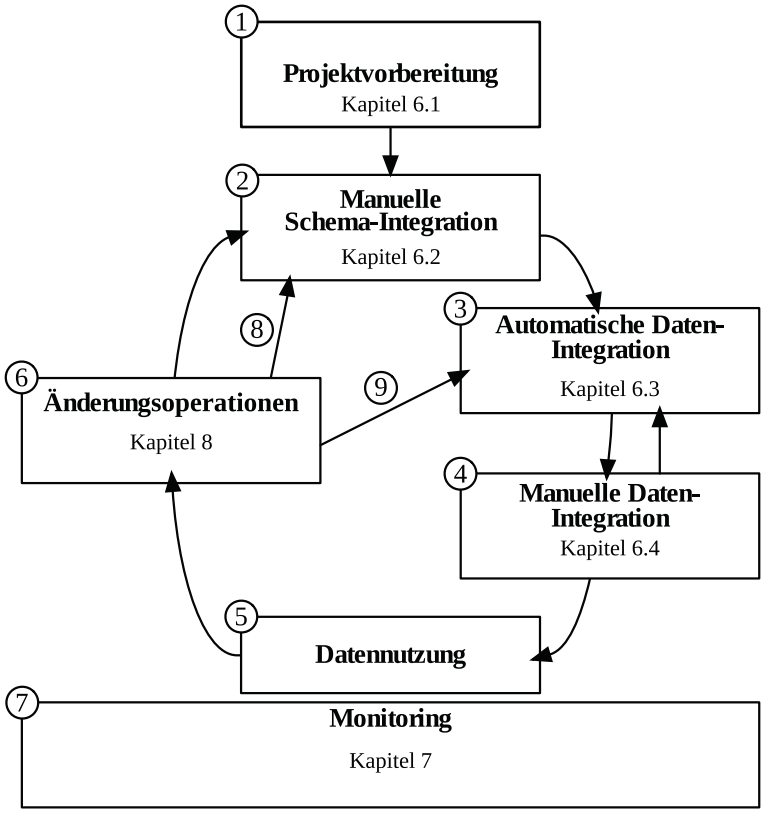


Abbildung 5.1.: Lebenszyklus der Produktdatenintegration

5.2. Integration

Bereits in Abschnitt 2.3.4 wurden die zwei grundlegenden Architekturen zur Integration heterogener Systeme vorgestellt, bei denen zwischen lokalen Informationssystem und einem globalen, integrierenden System unterschieden wird. Auf dieselbe Weise werden im PROCEED-Rahmenwerk nun Produktdaten lokaler Informationssysteme in ein globales Produktdatenintegrationssystem integriert, das eine ganzheitliche Sicht auf unterschiedliche Produktdaten bieten soll. Auf Grund der unterschiedlichen Qualität von Produktdaten aus heterogenen, lokalen Informationssystemen erfolgt deren Integration im PROCEED-Rahmenwerk *materialisiert* in das PDIS. Zur Unterstützung von Anwendungsfällen, die integrierte Produktdaten benötigen, ist ein rein lesender

Datenzugriff ausreichend.
Da eine vollständige Integration aller Datenmodellkonzepte in der Praxis unrealistisch ist bzw. sich zu teuer darstellt, wird eine bedarfsgerechte, d.h. anwendungsfallgetriebene, Integration von Schemakonzepten angestrebt. Dieses Vorgehen wird in der Literatur als *Pay-as-you-Go*-Prinzip bezeichnet [DSDH08].
In PROCEED wird zwischen Schema- und Instanzintegration unterschieden. Während die Schemaintegration vollständig manuell durch Experten durchgeführt wird, ist die Integration von Instanzen zweigeteilt. Zunächst werden auf Grundlage von Abbildungsregeln korrespondierende Instanzen ermittelt. Die Definition von Abbildungsregeln erfolgt dabei auf Attributebene der Schemakonzepte. Für ein Schemakonzept eines lokalen Informationssystems können eine oder mehrere Regeln auf ein Schemakonzept aus dem globalen PDIS definiert werden. Die Menge der Regeln für ein Schemakonzeptpaar aus lokalem Informationssystem und PDIS dient dazu, korrespondierende Instanzen dieser Schemakonzepte zu ermitteln. Neben den Regeln lassen sich weitere Aktionen zwischen Schemakonzeptpaaren definieren, die während der Integration von Instanzen durchgeführt werden sollen. Dies kann z.B. das Kopieren eines Attributwertes aus dem Schemakonzept eines lokalen Informationssystems in das PDIS sein.
Nach der Definition von Abbildungsregeln und -aktionen findet eine automatische Auswertung der Regeln auf die zu integrierenden Instanzen statt. Anschließend erfolgt die Überprüfung der Ergebnisse der automatischen Integration durch Fachanwender. Letztere erstellen gegebenenfalls fehlende Korrespondenzen oder korrigieren fehlerhafte Korrespondenzen.
In der Praxis werden die verschiedenen Perspektiven auf Produktdaten in applikationsspezifischen Informationssystemen gespeichert. Diese basieren auf unterschiedlichen Datenmanagementtechnologien, etwa relationalen Datenbanken, XML-Dokumenten oder Dateien. Um diese technische Heterogenität zu bewältigen, müssen Produktdaten in einer plattformunabhängigen Form abstrahiert werden. Aus diesem Grund werden sie im PROCEED-Rahmenwerk als wiederverwendbare, interoperable und plattformunabhängige Ontologien verwaltet. Abbildung 5.2 gibt einen Überblick über die einzelnen Konzepte des PROCEED-Rahmenwerks. Datenmodellkonzepte aus heterogenen, autonomen und lokalen Informationssystemen werden in sog. *lokale Produktontologien* abstrahiert. Diese dienen dazu, unterschiedliche Arten von Heterogenitäten zu überwinden (siehe Abschnitt 2.3.6). Durch eine einheitliche Struktur lassen sich so strukturelle, technische und semantische Heterogenitäten überwinden.
Da ein komplexes Produkt aus tausenden von Bauteilen bestehen kann, welche wiederum in verschiedenen Varianten und Versionen vorliegen, sollte die Komplexität der Integration verringert werden. Eines der wichtigsten Strukturierungsprinzipien zur Reduktion von Komplexität ist die Hierarchiesierung [Sim96]. Dem entsprechend werden lokale Produktdaten hierarchisch in vier *Produktdatenintegrationsebenen (engl. product data integration layer, PDILs)* unterteilt. *Schemakonzepte (SCs)* aus lokalen Informationssystemen werden entsprechenden PDILs zugeordnet. Über *Schemakonzept-Attributregeln (engl. schema concept attribute rules, SCARs)* der gleichen PDIL aus *lokalen Produktontologien (LPOs)* und der *globalen Produktontologie (GPO)* werden die zuvor erwähnten Abbildungsregeln definiert. Während der automatischen Instanzintegration erfolgt die Auswertung von SCARs für lokale und globale Produktontologien. Im Falle von Übereinstimmungen werden entsprechende *Korrespondenzen* zwischen Instanzen aus lokalen und globalen Schemakonzepten erzeugt.
Die globale Produktontologie stellt die holistische Sicht der einzelnen lokalen Produktontologien dar und dient als zentrale Datenbasis für die zu unterstützenden Anwendungsfälle. Über *Schemakonzept-Attributaktionen (engl. schema concept attribute actions, SCAAs)* wird definiert,

welche Attribute von lokalen Schemakonzepten in korrespondierende Instanzen der globalen Produktontologie integriert werden.

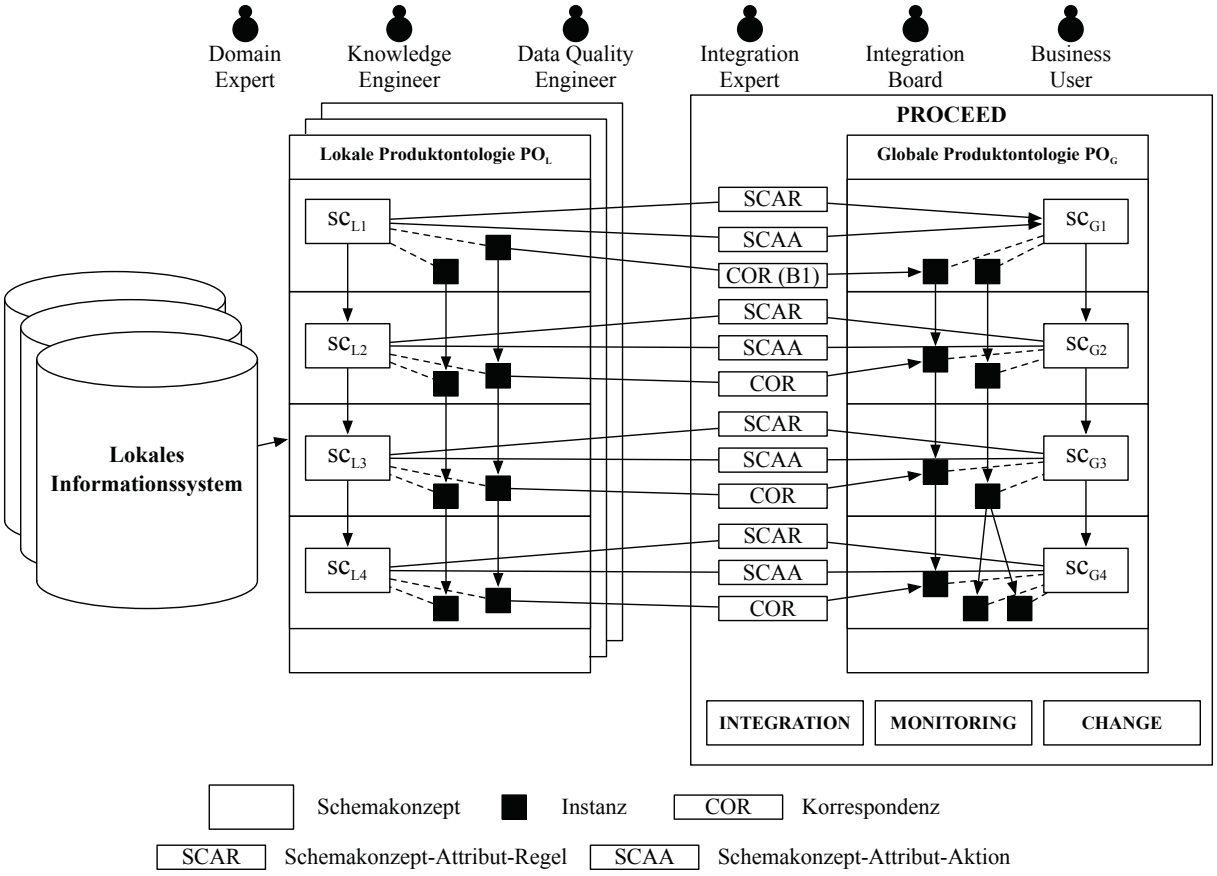


Abbildung 5.2.: Übersicht PROCEED-Rahmenwerk

Für die verschiedenen Schritte im Lebenszyklus der Integration sind unterschiedliche Rollen verantwortlich, die in den folgenden Kapiteln noch im Detail erläutert werden. Im Speziellen werden die Schritte ① bis ④ des Lebenszyklus aus Abbildung 5.1 in Kapitel 6, Schritt ⑦ in Kapitel 7sowie Schritt ⑥ in Kapitel 8 beschrieben.

6. Integration heterogener Produktdaten

Die Integration heterogener Produktdaten aus autonomen, lokalen Informationssystemen umfasst verschiedene Aktivitäten (vgl. Lebenszyklus aus Abschnitt 5.1), die in diesem Kapitel erläutert werden. Dazu zählen die Abbildung lokaler Informationssysteme auf sog. *lokale Produktontologien*, die *manuelle Schemaintegration* sowie die *automatische* und *manuelle Instanzintegration*.

6.1. Projektvorbereitung

Der erste Schritt im Lebenszyklus der Produktdateintegration ist die Projektvorbereitung. Hierzu zählen:

- **Anwendungsfälle definieren:** Zunächst definieren Fachanwender eine Menge relevanter Anwendungsfälle, welche durch die Integration von Produktdaten aus heterogenen, autonomen Informationssystemen unterstützt werden sollen. Dazu spezifizieren Mitglieder der Fachabteilungen, die Beteiligte der Anwendungsfälle sind, zusammen mit Integrationsexperten, die zu integrierenden Schemakonzepte. Während Mitarbeiter der Fachabteilungen in der Regel nur die Schemakonzepte kennen, für die sie auch im verantworteten Informationssystem zuständig sind, verfügen Integrationsexperten über umfassendes Wissen zu den Zusammenhängen mehrerer Informationssysteme.

- **Informationssysteme bestimmen:** Die Anwendungsfälle determinieren die zu integrierenden Informationssysteme. Ein *Integrationsgremium* wird aus Verantwortlichen der betroffenen Informationssysteme, Integrationsexperten sowie den Initiatoren des Anwendungsfalles gebildet. Nachdem dieses Gremium die Anwendungsfälle freigegeben hat, werden *Integrationsteams* gebildet.

- **Integrationsumfang bestimmen (Schemakonzepte, Instanzen):** Da ein Informationssystem typischerweise Dutzende von Schemakonzepten und Tausende von Instanzen enthält, muss der Umfang der Integration möglichst exakt definiert werden. Für die zeitliche und logische Strukturierung existieren verschiedene Mechanismen (z.B. Zusammenfassung von Instanzen nach Projekten oder Plattformen). Diese Projekte können nebenläufig in unterschiedlichen Phasen vorhanden sein, d.h. der Reifegrad von Produktdatenartefakten ist unterschiedlich. Folglich sollten auch nur diejenigen Instanzen integriert werden, die für einen Anwendungsfall definiert worden ist.

- **Integrationszeitplan erstellen:** Wie erwähnt, ist die Produktentwicklung in fest definierte Phasen mit konkreten Zeitintervallen eingeteilt. D.h., Anwendungsfälle müssen ebenfalls in einem festgelegten Zeitraum durchgeführt werden können. Dazu wird analog zum Entwicklungsprozess ein *Integrationszeitplan* durch das Integrationsgremium definiert.

- **Integrationsmethodiken festlegen:** In Abschnitt 2.3.5 wurden verschiedene Integrationsmethodiken vorgestellt. Ebenso lassen sich Produktdaten auf verschiedene Art und Weise integrieren. Mit welcher Methode Schemakonzepte und Instanzen eines Informationssystems in das PDIS integriert werden, wird wieder durch das Integrationsgremium festgelegt.

6.2. Manuelle Schemaintegration

Nach Abschluss der Projektvorbereitung, stehen die zu integrierenden Systeme fest. Domänenexperten modellieren anschließend, zusammen mit den Administratoren der Informationssysteme, lokale Produktontologien. Dabei sind unterschiedliche Fälle zu unterscheiden. Findet eine Integration zum ersten Mal statt und ist noch keine globale Produktontologie vorhanden, muss zuerst eine globale Produktontologie erstellt werden. Die genaue Vorgehensweise hängt dabei von den Entscheidungen aus der Projektvorbereitung ab.

6.2.1. Abbildung von Informationssystemen auf lokale Produktontologien

Während Schemakonzepte der Informationssysteme in der Produktentwicklung durch unterschiedliche konzeptionelle Modellierungssprachen (z.B. relationales oder objektorientiertes Datenmodell) definiert werden, können Instanzen dieser Konzepte technisch verschieden repräsentiert sein (z.B. relationale Datenbank oder Datei-basiert).
Um die sich hier bietende technische Heterogenität zu überwinden, werden Schemakonzepte und Instanzen von Informationssystemen im PROCEED-Rahmenwerk in sog. *Produktontologien* abstrahiert. Diese dienen als Grundlage für die Integration autonomer, heterogener Informationssysteme in der Produktentwicklung. Eine Produktontologie etwa abstrahiert Schemakonzepte und Instanzen von Informationssystemen der Produktentwicklung in eine definierte Struktur. Da die definierten Konzepte für eine spezifische Domäne – im vorliegenden Fall ist dies die Produktentwicklung – gelten, handelt es sich bei Produktontologien somit um Domänenontologien.
Jedes Informationssystem, das für die Umsetzung von Anwendungsfällen integriert werden soll, wird in dieselbe Struktur abstrahiert. Dadurch wird semantische Heterogenität vermieden. In Abschnitt 4.1.1 wurden z.B. föderierte Datenbanksysteme als eine Möglichkeit betrachtet, um heterogene Systeme zu integrieren. Das dort vorgestellte 5-Ebenen-Schema-Modell erläutert die verschiedenen Ebenen, die bei der Integration heterogener Systeme auftreten können. Vergleichbar mit dem Komponenten- und Exportschema aus dem 5-Ebenen-Schema-Modell föderierter Datenabanksysteme stellt eine lokale Produktontologie eine Kombination beider Schema dar. Abbildung 6.1 zeigt das 5-Ebenen-Schema-Modells (vgl. Abb. 6.1a)) sowie die Ebenen im PROCEED-Rahmenwerk (siehe Abb. 6.1b).
Liegen lokale Schemata in verschiedenen Datenmodellen vor, müssen sie im 5-Ebenen-Schema-Modell zuvor in Komponentenschemata überführt werden, bevor sie anschließend integriert werden. Im PROCEED-Rahmenwerk findet darüber hinaus mehr als nur eine Umwandlung in ein zuvor festgelegtes Datenmodell, in diesem Fall Ontologien, statt. Konzepte des lokalen Schemas werden semantischen Ebenen innerhalb der lokalen Produktontologie zugeordnet. Weiter kann implizites Wissen über Schemata explizit dokumentiert werden.
Während im 5-Ebenen-Schema-Modell eine Unterscheidung zwischen Komponenten- und Exportschema erfolgt, entspricht letzteres einer lokalen Produktontologie, d.h. Exportschema und

lokale Produktontologie enthalten nur diejenigen Schemakonzepte und Daten, die von einem Informationssystem für die Integration öffentlich bereitgestellt werden.

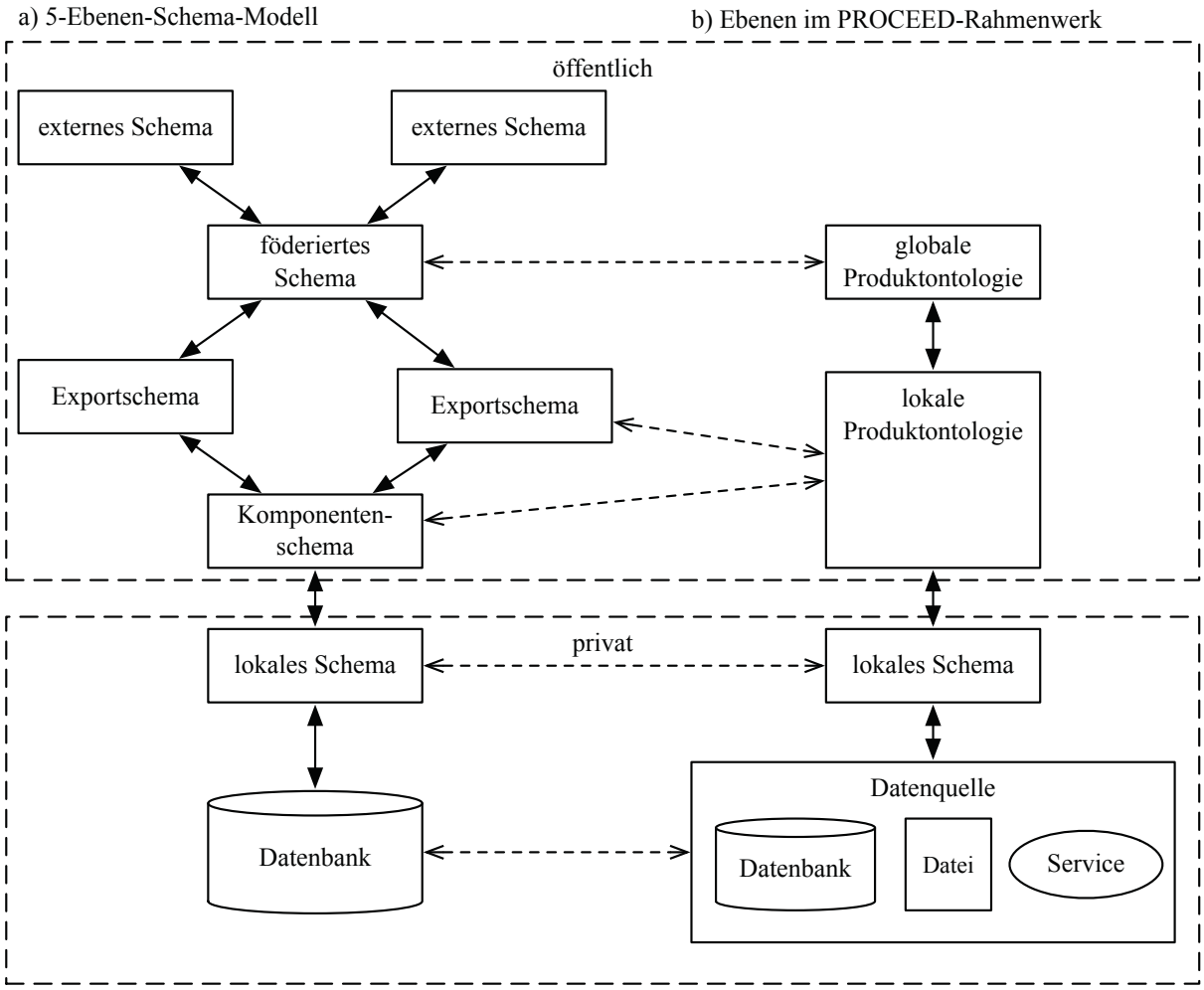


Abbildung 6.1.: Relation zwischen 5-Ebenen-Schema-Architektur FDBS und lokaler Produktontologie

Domänenexperten verfügen häufig über *implizites Wissen* (engl. sog. tacid knowledge [LR07]). Dazu zählen etwa Dokumentationsmethodiken, also Vorgehensweisen und Erfahrungswerte für die Strukturierung und Dokumentation von Produktdaten im Rahmen eines vorgegebenen konzeptionellen Datenmodells.
Ein Beispiel für implizites Wissen, das im Zuge der Anforderungserhebung durch Interviews von Anwendern geschildert wurde, ist wie folgt:
In einem Informationssystem werden Signale dokumentiert, die zwischen E/E-Komponenten ausgetauscht werden. Um den Modellierungs- und Dokumentationsaufwand zu reduzieren, werden E/E-Komponenten, welche sich semantisch im Sinne der Signalübermittlung gleich verhalten, nur einmal dokumentiert. Erfolgt die Weitergabe dieser Produktdaten an andere Informationssysteme, werden weitere E/E-Komponenten durch Kopieren erzeugt. Geschieht dieser Vorgang manuell durch einen Anwender, können Fehler beim Kopieren auftreten oder gar E/E-Komponenten vergessen werden.

Diese Information kann für die Integration nützlich sein und sollte daher explizit dokumentiert werden. Vor allem für die Integration von Produktdaten sind Angaben zur Domäne einer Applikation, zu verwendeten Konzepten für die Modellierung von Variabilität und Versionierung sowie zur Gruppierung von Produktdaten erforderlich. Diese Informationen werden in lokalen Produktontologien als informelle Annotationen zu Schemakonzepten hinterlegt und können bei der späteren Integration in ein PDIS durch Integrationsexperten genutzt werden. Letztere können auf Basis solcher Informationen bei Unklarheiten schneller Rücksprache mit Domänenexperten halten und so den Integrationsprozess beschleunigen. Das Dokumentieren zusätzlicher Informationen, die erst im weiteren Verlauf nützlich werden können, wird in anderen Zusammenhängen auch als *Look-Ahead* bezeichnet [BBR11].
Grundlage für die Produktentwicklung sind Systeme, die aus mehreren interagierenden Komponenten bestehen. Komponenten, im Folgenden auch Bauteile genannt, sind physische Erzeugnisse, die in Stücklisten verwaltet werden und die Grundlage für die Produktion eines Produktes bilden (vgl. Abschnitt 2.2.2). Da während der Produktentwicklung verschiedene Aspekte von Bauteilen dokumentiert werden, ist der gemeinsame Nenner der Informationssysteme das Schemakonzept *Bauteil*, welches in den verschiedenen Systemen unterschiedlich benannt sein kann und dabei oftmals unterschiedliche Semantik besitzt.
Häufig findet eine Vermischung verschiedener Datenstrukturen zu einem Schemakonzept statt, welches in lokalen Informationssystemen als *Bauteil*, Komponente oder ähnliches bezeichnet wird. Spezielle Datenstrukturen in der Produktentwicklung sind z.B. Stücklisten, CAD-Daten, PDM-Daten, Zeichnungsdaten, die *"Bauteile"* zu einem *Produkt* aggregieren. Bei dieser Aggregation spielen sowohl zeitliche als auch logische Aspekte eine Rolle. Ein wichtiger zeitlicher Aspekt der Produktentwicklung ist die *Versionierung*, während ein relevanter logischer Aspekt das *Konfigurationsmanagement* ist. Wie bereits in der Problemanalyse erläutert (vgl. Abschnitt 1.2), führen unterschiedliche Anforderungen und Sichtweisen der verschiedenen Entwickler in der Produktentwicklung dazu, dass sowohl verschiedene Datenstrukturen als auch verschiedene Versionierungs- und Konfigurationsmechanismen verwendet werden.
Um diese Mehrdeutigkeiten aufzulösen, erfolgt die Einordnung von Schemakonzepten lokaler Produktontologien in semantisch eindeutig spezifizierte *Produktdatenintegrationsebenen (engl. product data integration layers) (PDILs)*. Durch die einheitliche Zuordnung von Schemakonzepten lassen sich diese aus verschiedenen lokalen Produktontologien integrieren, ohne dass es zu semantischen Konflikten kommt.
In Anlehnung an das integrierte Produktmodell von ISO 10303 (siehe Abschnitt 4.2.3) sind diese Ebenen in PROCEED hierarchisch aufgebaut. Abbildung 6.2 illustriert den Zusammenhang zwischen den Artefakten aus dem integrierten Produktmodell von STEP und den PDILs von PROCEED.
Schemakonzepte, die einen Bauteilaspekt (z.B. Metadaten, Anforderungen, logistische Informationen) beschreiben, werden der *Object*-Produktdatenintegrationsebenen zugeordnet.
Da ein Produkt aus einer Vielzahl an Bauteilen bestehen kann, werden diese zu Kollektionen zusammengefasst. In der Produktentwicklung werden unterschiedliche Bezeichnungen hierfür verwendet, etwa *Produkt*, *Modell*, *Baureihe* und *Projekt*. Diese strukturierenden Schemakonzepte werden der *Product Data Collection*-Produktdatenintegrationsebene zugeordnet.
Wie bereits im Kontext des integrierten Produktmodells von ISO 10303 erwähnt (vgl. Abschnitt 4.2.3), können Bauteile in unterschiedlichen Bauteilvarianten existieren. Informationssysteme realisieren die Modellierung von Variabilität auf verschiedene Art und Weise. Schemakonzepte aus lokalen Informationssystemen, die der Beschreibung von Bauteilvariabilität dienen, werden

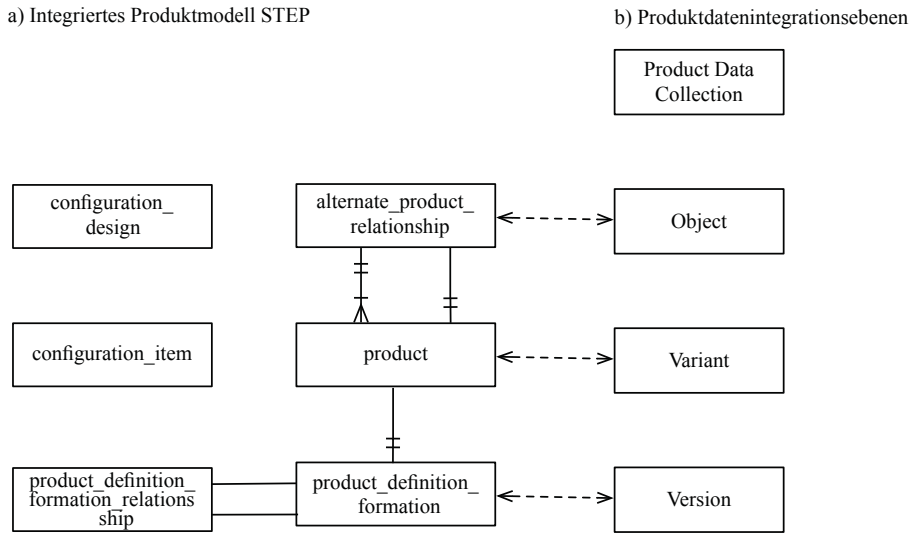


Abbildung 6.2.: Zusammenhang zwischen Artefakten des integrierten Produktmodells von STEP und PROCEED

der *Variant*-Ebene zugeordnet.
Schließlich durchläuft jedes Bauteil eine Entwicklung, in der unterschiedliche Entwicklungsstände anfallen, die zwecks lückenloser Dokumentation der Entwicklungsergebnissen aufbewahrt werden müssen. Schemakonzepte, die der Beschreibung zeitlicher Aspekte von Bauteilen dienen, werden der *Version*-Ebene zugeordnet.
Lokale Informationssysteme müssen nicht für jede Produktdatenintegrationsebene korrespondierende Schemakonzepte besitzen. Wie erwähnt, können Schemakonzepte lokaler Informationssysteme mehrere Strukturierungsmechanismen beinhalten. Weiter kann es vorkommen, dass ein lokales Informationssystem nicht jeden Strukturierungsmechanismus unterstützt.
Auch kann es vorkommen, dass ein Schemakonzept eines lokalen Informationssystems mehrere Produktdatenintegrationsebenen abdeckt. In diesem Fall sollten Transformationen zwischen lokalem Informationssystem und lokaler Produktontologie erfolgen, um eine explizite Trennung von Strukturierungsmechanismen, wie Variabilität und Versionierung, zu erreichen.
Jedes Schemakonzept einer Produktontologie besitzt unterschiedliche Attribute. Dazu zählen sowohl *Metadatenattribute*, wie Namen, Erstellungs-/Änderungszeit, eindeutige Kennungen und Informationen zu Zugriffsrechten, als auch *Inhaltsattribute*, welche die eigentlichen Entwicklungsergebnisse von Bauteilen beinhalten.
Zusammengefasst definieren wir eine Produktontologie informell wie folgt:

Definition 19 (Produktontologie). Eine Produktontologie beschreibt einen oder mehrere Aspekte einer Menge an Bauteilen eines Produktes. Es wird zwischen *lokalen Produktontologien* und integrierender *globaler Produktontologie* unterschieden. Lokale Produktontologien beinhalten diejenigen Schemakonzepte und Instanzen lokaler Informationssysteme der Produktentwicklung, die für die Realisierung von Anwendungsfällen benötigt werden bzw. für eine Integration bereitgestellt werden.
Innerhalb einer lokalen Produktontologie sind Schemakonzepte und ihre Instanzen einer von

vier Produktdatenintegrationsebenen zugeordnet. Schemakonzepte sind auf diesen Ebenen hierarchisch angeordnet. Zwischen einem Schemakonzept einer Ebene und einem zugehörigen Schemakonzept auf der darunterliegenden Ebene besteht eine Aggregationsbeziehung. Die Produktdatenintegrationsebenen heißen *Product Data Collection (PDC)*, *Object (Obj)*, *Variant (Var)* und *Version (Ver)*.

Beispiel 16 (Produktontologie). Abbildung 6.3 illustriert den Zusammenhang zwischen einem exemplarischen lokalen Schema eines Informationssystems und einer korrespondierenden lokalen Produktontologie. Schemakonzepte des lokalen Informationssystems werden auf entsprechende Schemakonzepte in den verschiedenen Produktdatenintegrationsebenen abgebildet. Der Übersicht halber sind keine Attribute dargestellt; sie werden aus dem lokalen Informationssystem den korrespondierenden Metadaten- oder Inhaltsattributen in der lokalen Produktontologie zugeordnet. Schließlich wird implizites Wissen explizit mittels informeller Anmerkungen in der lokalen Produktontologie dokumentiert.

Definition 20 (Produktdatenintegrationsebenen). Produktdatenintegrationsebenen besitzen Konstrukte zur Zuordnung von Schemakonzepten aus lokalen Informationssystemen in Produktontologien. Mit jeder Ebene sind semantische Konzepte der Produktentwicklung verknüpft. Sie ermöglichen es, semantische Heterogenität während der Integration von Produktontologien zu beseitigen. Die Ebenen einer Produktontologie sind hierarchisch aufgebaut und beschreiben Aggregationsbeziehungen.

Beispiel 17 (Produktdatenintegrationsebenen). Das Beispiel aus Abbildung 6.3 besitzt für jede Produktdatenintegrationsebene jeweils ein Schemakonzept. Bei dem lokalen Informationssystem handelt es sich um ein System zur Dokumentation von Anforderungen. Der Produktdatenaspekt der lokalen Produktontologie betrifft die Anforderungen von Komponenten. Entsprechend beschreibt das Schemakonzept *Component Requirement* die Anforderungen für eine Komponente, die zu einem Projekt (*Project Requirements*) zusammengefasst werden. Da *Component Requirement* einen Aspekt beschreibt, ist das Schemakonzept der Produktdatenintegrationsebene *Object* zugeordnet, während *Componente Requirement* ein Schemakonzept der Ebene *Product Data Collection* darstellt. Die Anforderungen einer Komponente beinhalten alle Anforderungen an alle möglichen Komponentenvarianten. Das entsprechende Schemakonzept *Component Requirement Variant* ist der Produktdatenintegrationsebene *Variant* zugeordnet. Zusätzlich wird das zuvor erwähnte implizite Wissen, dass Anforderungsspezifikationen alle Komponentenvarianten beschreiben explizit am Schemakonzept *Component Requirement Variant* dokumentiert. Schließlich existieren Anforderungsdokumente in verschiedenen Versionen, so dass das Schemakonzept *Component Requirement Version* der Ebene *Version* zugeordnet ist.

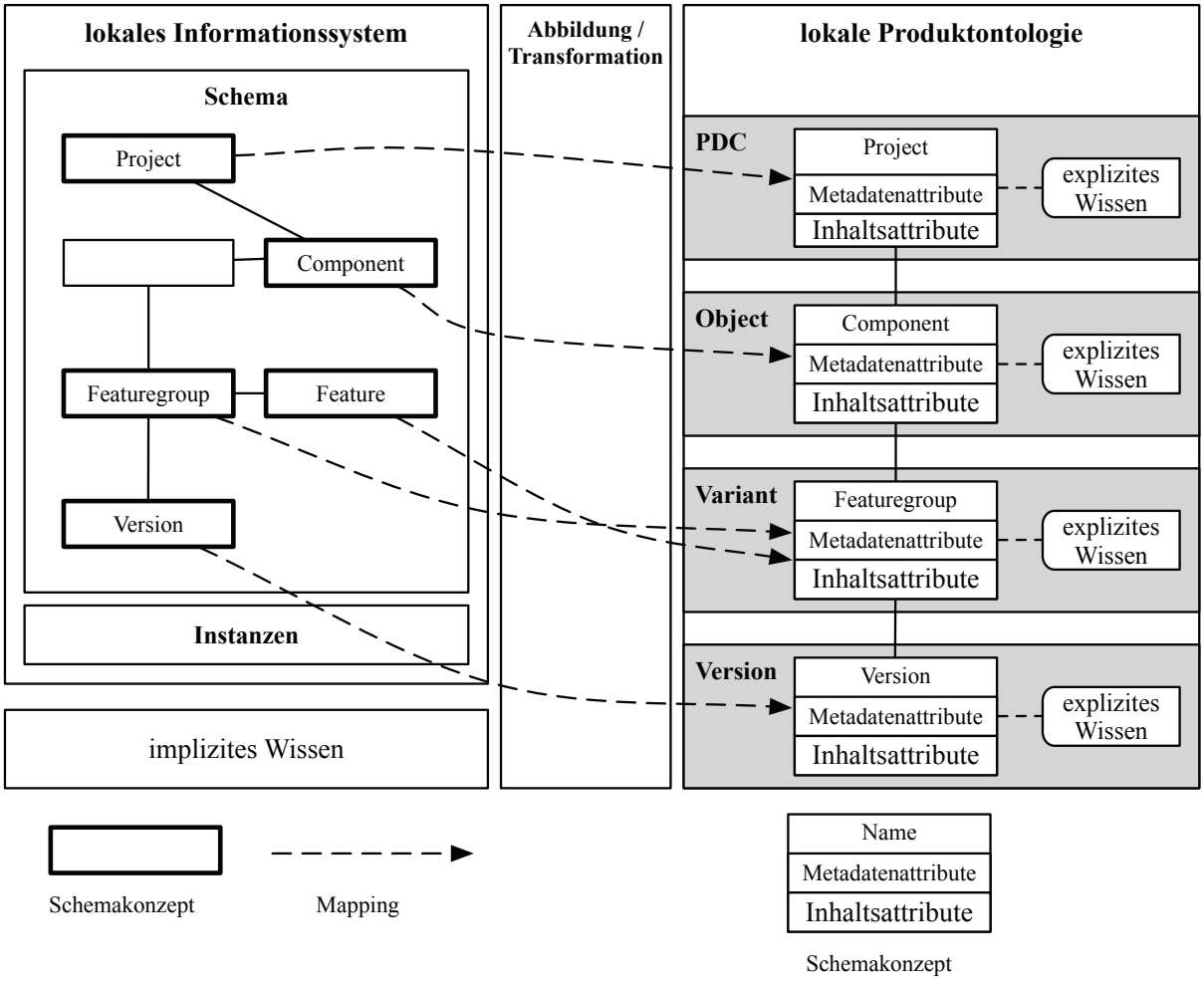


Abbildung 6.3.: Abbildung eines lokalen Informationssystems in eine lokale Produktontologie

Bei der Abbildung von Schemakonzepten aus einem lokalen Informationssysteme in eine lokale Produktontologie wird es ggf. notwendig, Transformationen durchzuführen, falls keine direkten Entsprechungen gefunden werden. Beispielsweise können Konzepte für die Repräsentation von Variabilität über mehrere Schemakonzepte im lokalen Informationssystem verteilt sein. Diese strukturelle Heterogenität muss durch geeignete Transformationen aufgelöst werden. Wie solche Transformationen durchgeführt werden können, hängt von der jeweils zugrundeliegenden Datenmanagementtechnologie des lokalen Informationssystems ab. Im Falle relationaler Datenbanken können Transformationen durch SQL-Abfragen oder -Prozeduren durchgeführt werden, während bei XML-Dateien XQuery oder XSLT zum Einsatz kommen können. Handelt es sich um ein proprietäres Format, kann die Transformation mit Hilfe verfügbarer APIs oder durch Implementierung spezifischer Parser geschehen. Das Vorgehen der Transformation ähnelt dem ETL-Prozess bei Data Warehouses (vgl. Abschnitt 4.1.2).

Die Behandlung von Instanzen aus einem lokalen Informationssystem kann unterschiedlich erfolgen. Entweder werden die Instanzen im Transformationsprozess in die Produktontologie übertragen oder sie werden nicht in lokalen Produktontologien gespeichert (materialisiert), son-

dern im Integrationsprozess über Schnittstellen direkt aus dem Informationssystem abgefragt (virtualisiert).
Beide Vorgehensweisen haben jeweils Vor- und Nachteile. Werden Instanzen in lokale Produktontologien kopiert, können sie eine hilfreiche Informationsquelle für die Abbildung von Schemakonzepten sein. Weiter ist es möglich, Daten während der Transformation in lokale Produktontologien zu bereinigen und damit die Datenqualität zu erhöhen. Nachteilig ist, dass lokale Produktontologien sehr groß werden können und Instanzen aus dem lokalen Informationssystem und der lokalen Produktontologie synchronisiert werden müssen.
Werden Instanzen nicht in lokale Ontologien transferiert, sondern lediglich bei Bedarf von einem lokalen Informationssystem abgerufen werden, bleiben lokale Produktontologien leichtgewichtig. Jedoch fallen dadurch nützliche Informationen für die Integration von Schemakonzepten auch weg. Vorteilhaft ist, dass keine Synchronisation zwischen Instanzen aus lokalem Informationssystem und lokaler Produktontologie notwendig wird. Für die Erstellung lokaler Produktontologien ist eine Zusammenarbeit zwischen *Domänenexperten* und *Datenbankadministratoren* der einzelnen lokalen Informationssysteme erforderlich. Die Modellierung von Schemakonzepten kann durch *Knowledge Engineers* unterstützt werden. Letztere sind Wissensträger, die mit der Modellierung von Ontologien vertraut sind und die Erfahrungen für die Abbildung von Schemakonzepten unterschiedlicher Datenmodellierungssprachen besitzen.
Tabelle 6.1 fasst die wesentlichen Eigenschaften eines lokalen Informationssystems und einer lokalen Produktontologie sowie die Abbildung zwischen ihnen zusammen.

	lokales Informationssystem	**lokale Produktontologie**	**Abbildung**
Sichtbarkeit	privat	öffentlich	öffentlich
Verantwortlichkeit	Datenbankadministrator	Domänenexperte	Datenbankadministrator, Domänenexperte
Datenmodell	beliebig	Ontologie	abhängig vom lokalen Informationssystem

Tabelle 6.1.: Eigenschaften lokaler Informationssysteme und Produktontologien

Im Folgenden werden die einzelnen Produktdatenintegrationsebenen näher erläutert.

Produktdatenintegrationsebene 1: Product Data Collection (PDC)

Die oberste Produktdatenintegrationsebene umfasst diejenigen Schemakonzepte aus lokalen Informationssystemen, die einen Kontext, d.h. einen Gültigkeitsbereich für Produktdaten, definieren. Der Kontext von Produktdaten in der Produktentwicklung kann in den lokalen Informationssystemen unterschiedlich definiert sein:

- **Produkt:** Ist der Kontext ein Produkt, gelten Produktdaten für ein spezifisches Produkt und seine Varianten.
- **Plattform:** Eine Plattform definiert einen Kontext, der sich über mehrere Produkte erstreckt. Um die Entwicklungskosten für ein Produkt zu reduzieren, ist es im Automobilbereich üblich, dass Produkte auf gemeinsamen Plattformen entwickelt werden. Unter einer Plattform ist eine technische Basis, etwa Teile der Karosserie oder das Fahrwerk, zu

verstehen, die von mehreren Produkten verwendet werden. Dass heißt, ein Produktdatenaspekt für mehrere Produkte wird gleichzeitig entwickelt bzw. kann in mehreren Produkten wiederverwendet werden.

- **Projekt:** Ist ein allgemeiner Begriff und der Kontext kann unterschiedlich definiert sein.

Abbildung 6.4 illustriert schematisch einen Datenmodellausschnitt mit Schemakonzepten und Instanzen zweier lokaler Produktontologien. In Abbildung 6.4a ist der Kontext der Produktdaten durch eine Plattform gegeben, während in Abbildung 6.4b ein Produkt den Kontext bildet.

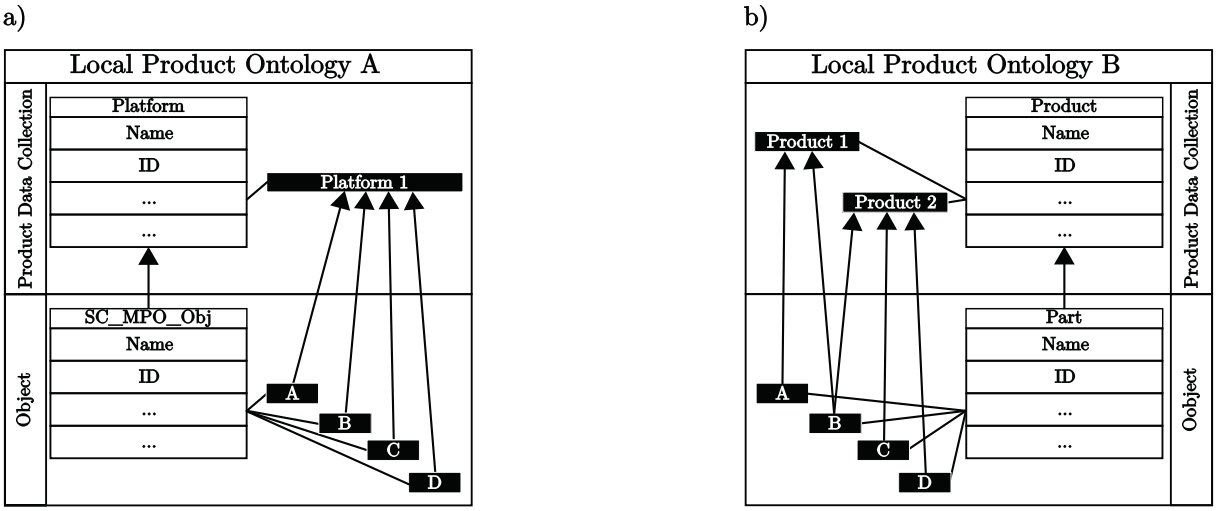


Abbildung 6.4.: Produktdatenintegrationsebene: PDC

Werden die Produktontologien aus Abbildung 6.4a und 6.4b integriert, können ggf. Korrespondenzen zwischen einigen Artefakten aus `Local Product Ontology A` und `Local Product Ontology B` herstellt werden. Jedoch können für einige Artefakte aus `Local Product Ontology A` keine Entsprechungen in `Local Product Ontology` gefunden werden, da `Local Product Ontology A` mehrere Produkte beschreibt, während `Local Product Ontology A` nur ein Produkt umfasst.
Um eine möglichst automatische Integration von Artefakten zu ermöglichen, muss der Integratonsprozess überwacht werden. Dazu sind Metriken, etwa zur Vollständigkeit einer Integration, notwendig. Wir definieren die Vollständigkeit der Integration einer lokalen Produktontologie als den Quotienten der Instanzen, für die entsprechende Instanzen in einer globalen Produktontologie gefunden worden sind zur Menge aller Instanzen einer lokalen Produktontologie. Soll die Vollständigkeit für die Integration von `Local Product Ontology A` in `Local Product Ontology B` bestimmt werden, wird das Ergebnis verfälscht, da für einige Instanzen aus `Local Product Ontology A` nie Entsprechungen in `Local Product Ontology B` gefunden werden können, da es sich um mehrere Produkte handelt.
Folglich wird die Integration von Produktdatenn vereinfacht, wenn die zu integrierenden lokalen Produktontologien den selben Kontext verwenden. Dazu werden im PROCEED-Rahmenwerk der Produktkontext für lokale und globale Produktontologien gewählt. Besitzt ein lokales Informationssystem eine abweichende Aggregation, etwa eine Plattform, bei der Produktdaten aggregiert werden, die für mehrere Produkte gültig sind, muss im ETL-Prozess von einem lokalen In-

formationssystem in eine entsprechende Produktontologie eine Transformation bzw. Selektion stattfinden.

Produktdatenintegrationsebene 2: Object-Integrationsebene

Die zweite Produktdatenintegrationsebene ist die *Object*-Ebene. Produkte setzen sich aus einer Vielzahl an Bauteilen zusammen (vgl. Abschnitt 2.2.1), die entweder atomar oder zusammengesetzt sind. Während atomare Bauteile nicht weiter zerlegt werden, bestehen zusammengesetzte Bauteile aus weiteren Unterbauteilen. Die vorliegende Arbeit fokussiert auf atomare Bauteile, d.h. Bauteile, die eine eindeutige, global gültige Bauteilnummer besitzen und in einer Stückliste referenziert werden. Für jedes dieser Bauteile werden in lokalen Informationssystemen entsprechende Produktdaten verwaltet. Diese Informationssysteme können sowohl für einen spezifischen Produktdatenaspekt dienen oder mehrere Produktdaten abdecken. In der Object-Ebene werden Schemakonzepte für die verschiedenen Produktdaten verwaltet.
Abbildung 6.5 illustriert die obersten zwei Produktdatenintegrationsebenen einer lokalen Produktontologie. Die PDC-Integrationsebene enthält das Schemakonzept *Product*, während in der Object-Integrationsebene zwei Produktdaten als Schemakonzepte *Software* und *Hardware* modelliert sind.

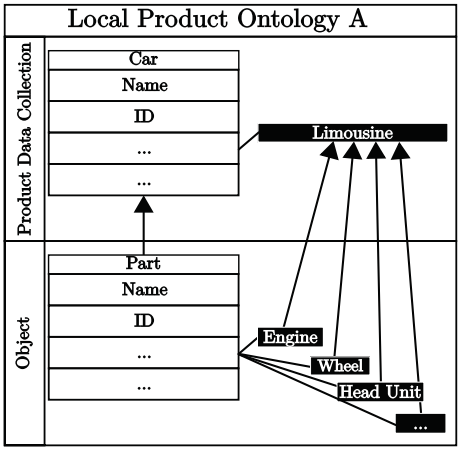


Abbildung 6.5.: Produktdatenintegrationsebene: Object

Für die Integration ist die Dokumentation derjenigen Produktdaten essentiell, die durch ein Schemakonzept in der Object-Integrationsebene repräsentiert sind. Werden für die Dokumentation von Produktdatenn in lokalen Produktontologien global definierte Begriffe verwendet, kann die Integration von Schemakonzepten unterstützt werden. Korrespondierende Schemakonzepte können so automatisch ermittelt werden und als Vorschläge für die manuelle Integration von Schemakonzepten durch Integrations- und Domänenexperten dienen. Dazu wird jedes Schemakonzept der Object-Integrationsebene mit einem Produktdatenaspekt aus einem globalen Repository versehen.

Produktdatenintegrationsebene 3: Variant-Integrationsebene

In der 3. Produktdatenintegrationsebene werden Variabilitätskonzepte für die einzelnen Produktdaten dokumentiert. Produkte werden in unterschiedlichen Varianten angeboten, um un-

terschiedlichen Kundenwünschen (Ausstattungsmerkmale) gerecht zu werden. Darüber hinaus gibt es unterschiedliche gesetzliche Rahmenbedingungen und Regelungen, die länderspezifische Produktvarianten erforderlich machen [HBR08].
Variabilität untergliedert sich in *interne* und *externe* Variabilität. Letztere ist durch den Nutzer eines Produktes erlebbar, etwa durch unterschiedliche Funktionalitäten, Formen und Farben eines Produktes. Interne Variabilität bezeichnet unterschiedliche Varianten, die in Informationssystemen dokumentiert sind, jedoch nicht zu unterschiedlichen Produktvarianten führen. So kann beispielsweise ein Bauteil von mehreren Zulieferern hergestellt werden, die sich im Einkaufspreis unterscheiden, deren Funktionalität aber identisch ist.
Diese unterschiedlichen Arten von Variabilität spiegelt sich in den Produktdatenn der verschiedenen Informationssysteme der Produktentwicklung wider. Auf Grund der unterschiedlichen Anforderungen und Sichtweisen der Entwickler wird Variabilität durch verschiedene Mechanismen realisiert. Im Folgenden werden alternative Mechanismen für die Repräsentation von Produktdatenvariabilität anhand unterschiedlicher Bereiche der Produktentwicklung erläutert.

Requirements Engineering. Am Anfang der Produktentwicklung werden Anforderungen an Bauteile in Anforderungsdokumenten spezifiziert. Diese Dokumente dienen der Auftragsvergabe an externe Lieferanten und stellen die Grundlage der Produktentwicklung dar. Anforderungsdokumente beschreiben funktionale und nichtfunktionale Anforderungen für alle möglichen Bauteilvarianten. Letztere sind physische Erzeugnisse mit eindeutigen *Sachnummern*, die der Produktionsplanung dienen und in Stücklisten verwendet werden (siehe Abschnitt 2.2.2).
Bauteile können sowohl auf Hardware- als auch Software-Ebene eine gewisse Variabilität aufweisen. Physisch kann ein Bauteil, etwa ein Außenspiegel, unterschiedliche Formen und Farben (externe Variabilität) besitzen und aus verschiedenen Materialien bestehen.
Alle gültigen Konfigurationen eines 150-Prozent Modells[1] werden in einem Feature-Modell dokumentiert (vgl. Abschnitt 2.2.3). Eine gültige Variante wird aus dem 150-Prozent Modell durch Konfiguration abgeleitet.

Konstruktion. Für die Konstruktion von Bauteilen sind sowohl zweidimensionale Zeichnungen als auch dreidimensionale Geometrien (Volumenmodelle) notwendig, die üblicherweise mit Hilfe von CAD-Systemen erstellt werden. Ein Teilbereich der Konstruktion ist die *Variantenkonstruktion*, bei der bestehenden Konstruktionen übernommen und Details weggelassen oder ergänzt werden. Eine Möglichkeit besteht darin, Zeichnungen mit Parametern zu erstellen, die variabel sind. Durch das Verändern dieser Parameter lassen sich, gestützt durch CAD-Systeme, unterschiedliche Bauteile generieren bzw. erstellen. Dieses Vorgehen wird auch als *parametrische Konstruktion* bzw. *parametrische Generierung* bezeichnet. D.h. für jede geometrische Bauteilvariante wird eine explizite Zeichnung bzw. ein explizites geometrisches Modell erstellt.
Während für Bauteile, die nur aus mechanischen Teilen bestehen, die unterschiedlichen geometrischen Varianten explizit in einem CAD-System modelliert werden[2], existiert für Bauteile, die auch Software enthalten, meistens nur ein Software- bzw. ein Funktionsmodell, welches alle möglichen Varianten umfasst.

[1] Ein 150-Prozent Modell bezeichnet ein Modell, das alle variablen Anteile inklusive der möglichen Ausprägungen letzter beinhaltet.

[2] Die explizite Modellierung von geometrischen Bauteilvarianten ist notwendig, da diese Grundlagen für DMU-Untersuchungen (Digital Mock-Up) sind. Damit sind rechnergestützte Simulationsverfahren gemeint, mit denen etwa Einbaukollisionen verschiedener Produktkonfigurationen ermittelt werden.

Softwareentwicklung. Bauteile, die Software umfassen (z.B. Steuergeräte), können unterschiedliche Funktionalität realisieren. Häufig wird die Variabilität eines Steuergerätes über Codierparameter realisiert, so dass für unterschiedliche Bauteilvarianten die gleiche Hardware- und Software-Basis verwendet werden kann. Die Konfiguration einer bestimmten Bauteilvariante erfolgt mittels Konfiguration verschiedener Parameter, deren Festlegung das Ausführungsverhalten der Software beeinflusst.
Die Modellierung von Softwarevariabilität lässt sich etwa mit Featuremodellen (vgl. Abschnitt 2.2.3) realisieren. Eine weitere Methode stellt *Decision Modeling* dar. Dabei müssen domänenspezifische Fragen mit Hilfe einer Menge möglicher Antworten/Entscheidungen beantwortet werden. Die Entscheidungen sind mit Variationspunkten verbunden. Jede Entscheidung besitzt eine Beschreibung mit den Auswirkungen, welche die Auswahl einer Entscheidung auf Artefakte haben soll [CGR+12]. Während die Featuremodellierung auf das Verständnis der möglichen Features abzielt, liegt der Fokus beim Decision Modeling auf der Ableitung von Entscheidungen aus Produktkonfigurationen.
Neben der Modellierung von Variabilität sind die unterschiedlichen Möglichkeiten für deren Realisierung ein entscheidendes Merkmal von Variabilitätsmechanismen. Variabilität lässt sich in der Softwareentwicklung zu unterschiedlichen Zeitpunkten realisieren. Zu den verschiedenen Zeitpunkten zählen etwa die *Präkompilierzeit*, die *Kompilierung*, das *Linken* oder das *Deployment* [CBK13]. Auch innerhalb der Softwaremodelle bzw. Entwicklungsartefakte (z.B. Quell-, Binär- und Konfigurationsdateien) lässt sich Variabilität mittels unterschiedlicher Konstrukte realisieren. Beispiele für Mechanismen von Programmiersprachen sind Vererbung, Generatoren, Parameter, Präprozessoranweisungen und Laufzeitvariablen. Diese sind entweder Bestandteil von Programmiersprachen oder werden von Applikationen (z.B. Compiler, Generatoren) realisiert. Eine detaillierte Analyse zur Modellierung und Realisierung von Variabilität findet sich in [CBK13].
Für die Softwareentwicklung von Steuergeräten gibt es zwei verschiedene Herangehensweisen: Modellbasierte und Codebasierte Softwareentwicklung. Bei modellbasierter Softwareentwicklung werden zunächst Modelle definiert, die mit Hilfe von Codegeneratoren dann Softwareartefakte erzeugen (z.B. Schnittstellen). Letztere werden anschließend um Featureimplementierungen durch Softwareentwickler vervollständigt. Beim zweiten Ansatz entfällt die Modellierung und Entwickler realisieren Features ohne zusätzlichen Generierungsschritt.
Eine für die Entwicklung eingebetteter Systeme etablierte Software-Entwicklungsmethode ist AUTOSAR[3], das die Möglichkeit bietet, Softwarevariabilität durch Variationspunkte (*Variation Points*) und Featuremodelle (*Feature Models*) zu beschreiben. Für jeden Variationspunkt einer Systemspezifikation muss in AUTOSAR der Auflösungszeitpunkt definiert werden. Dies kann zur Systemzeit sein, während der Codegenerierung, vor dem Compilieren, während dem Linken oder nach dem Bauen [SWB12].

Stückliste. Wie bereits in Abschnitt 2.2.3 erwähnt, ist die Stückliste das zentrale Informationssystem für die Herstellung von Produkten. In der Stückliste sind alle möglichen bestellbaren Bauteilvarianten dokumentiert, die für die Herstellung alle Produktvarianten notwendig sind. Die Auswahl entsprechender Bauteilvarianten erfolgt mittels Konfiguration.

[3] AUTomotive Open System ARchitecture

Aspekt	Variabilitätsmechanismus
Requirements Engineering	Alle Varianten werden in einem Anforderungsdokument dokumentiert.
Konstruktion	A: 1 Modell für jede Variante angelegt B: Ein Modell enthält alle Varianten Ein Modell für jede geometrische Variante
Softwareentwicklung	Codierparameter, Features-Flags, Vererbung
Test	Siehe Requirements Engineering

Tabelle 6.2.: Variabilitätsmechanismen für verschiedene Produktdaten

Klassifikation unterschiedlicher Variabilitätsmechanismen. Tabelle 6.3 fasst die beschriebenen Mechanismen für die Modellierung und Realisierung von Variabilität in der Produktentwicklung zusammen. Die unterschiedlichen Beispiele verdeutlichen, dass Variabilität von Produktdaten auf unterschiedliche Art und Weise realisiert werden kann. Weiter wird ersichtlich, dass diese unterschiedlichen Methodiken explizit in lokalen Produktontologien in Form von Schemakonzepten dokumentiert werden müssen, um die Integration unterschiedlicher Methoden ermöglichen zu können.
Fasst man die Beispiele zusammen, ergeben sich zwei Variabilitätstypen für Produktdaten:

- **Variabilitätstyp 1:** Ein Artefakt (z.B. Anforderungsdokument, Konstruktionszeichnung, geometrisches Modell, Quellcode) umfasst alle möglichen Varianten. Eine spezifische Variante wird durch Konfiguration abgeleitet. Dazu sind die variablen Bereiche des Artefaktes durch Variabilitätspunkte gekennzeichnet. Dass heißt, die Variabilität im Artefakt ist noch nicht aufgelöst. Variabilitätspunkte können für jeden Produktdatenaspekt unterschiedlich realisiert sein. In der Softwareentwicklung etwa können Feature-Flags [MSR13] eingesetzt werden, um variable Anteile zu realisieren, oder in der Prozessmodellierung dem entsprechend sog. Anpassungspunkte (engl. adjustment parts), für die dann jeweils variable Anteile festgelegt sind [HBR09, ATW+15].
- **Variabilitätstyp 2:** Beim zweiten Typ wird die Variabilität bereits während der Modellierung bzw. Implementierung aufgelöst. Dass heißt, dass für alle möglichen Varianten explizite Artefakte erstellt werden, die keine Variabilitäspunkte mehr aufweisen.

Aus einem Artefakt mit dem Variabilitätytyp 1 lassen sich ohne weiteres Artefakte des zweiten Typs erzeugen. Umgekehrt ist es aufwändiger, aus mehreren Typ-2 Artefakten ein Typ-1 Artefakt zu erstellen. Im Detail müssen zunächst die Variabilitätspunkte ermittelt werden, d.h. diejenigen Bereiche, in denen sich die Artefakte unterscheiden. Anschließend müssen die möglichen Konfigurationen für die Variabilitätspunkte bestimmt werden.
Auf Basis der unterschiedlichen Vorgehensweisen für den Umgang mit Variabilität in der Produktentwicklung, definieren wir folgende Taxonomie für das Variantenmanagement:

Taxonomie 1 (Variantenmanagement).

- **Variantenmanagement:** Umfasst alle Aktivitäten, um unterschiedliche Anforderungen, Eigenschaften und Funktionalitäten von Produkten in den unterschiedlichen Informationssystemen der Produktentwicklung zu verwalten. Hierbei wird zwischen externer und interner Variabilität unterschieden.
- **Variabilitätspunkt:** Stelle in einem Artefakt, bei der eine Auswahl zwischen verschiedenen Features getroffen werden kann. Gültige Kombinationen von Features für Variabilitäspunkte in einem Artefakt werden als **Konfiguration** bezeichnet.
- **Feature:** Funktionalität für einen Variabilitätspunkt, die entweder optional oder obligat ist.
- **Variante:** Konfiguriertes Artefakt, d.h. für alle Variabilitätspunkte wurden gültige Belegungen.
- **Variantenkollektion:** Menge aller Varianten eines Artefaktes.
- **Variationszeitpunkt:** Zeitpunkt an dem aus der Variantenkollektion eine spezifische Variante ausgewählt wird.
- **Variabilitätsmechanismus:** Technische Realisierung von Variabilität eines Artefaktes für einen Produktdatenaspekt.

Informationen zu den einzelnen Variabilitätsmechanismen werden in einer lokalen Produktontologie dokumentiert, um eine anschließende Integration in eine globale Produktontologie zu erleichtern. Durch die Spezifikation der Art von Variabilitätsmechanismus lassen sich Aussagen über semantische Beziehungen integrierter Artefakte treffen. Abbildung 6.6 illustriert eine lokale Produktontologie. Der Variabilitätstyp für das Schemakonzept *Part Variant* in Abbildung 6.6 hat den Variabilitätstyp 1, d.h. jede Instanz (inklusive hinterlegtem Artefakt) des Schemakonzeptes beschreibt genau eine Variante.

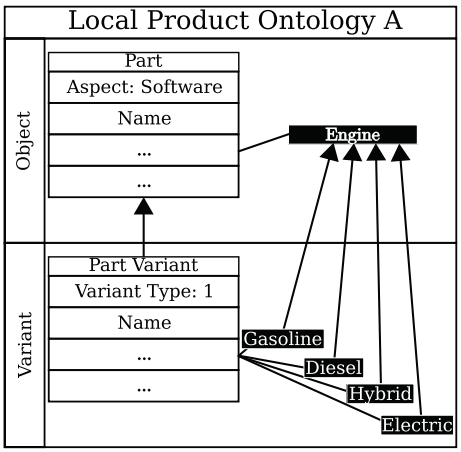


Abbildung 6.6.: Produktdatenintegrationsebene: Variant

Produktdatenintegrationsebene 4: Version-Integrationsebene

Wie in Abschnitt 2.2.3 erläutert, erfolgt die Entwicklung von Produkten system- und komponentenorientiert durch parallel entwickelnde Teams. Elementarer Gedanken der Produktentwicklung ist die sukzessive Verfeinerung des Reifegrades von Bauteilen. D.h. in den frühen Phasen der Entwicklung werden zunächst Grund- und Sicherheitsfunktionen von Bauteilen entwickelt und abgesichert. Im weiteren Verlauf werden die anderen Funktionalitäten realisiert. Darüber hinaus treten meist Änderungen am System- und Komponentendesign auf, etwa zwecks Fehlerbeseitigung, Design-Änderungen infolge technischer Limitationen oder der Umsetzung neu aufkommender Anforderungen oder neuer gesetzlicher Rahmenbedingungen [Hal10]. Durch das komplexe Beziehungsgeflecht zwischen Systemen und Komponenten wirken sich Änderungen an einem System möglicherweise auf weitere Systeme aus. Folglich ist ein globaler Prozess (engl. *change management*) notwendig, der diese Änderungen koordiniert.
Soll ein Bauteil geändert werden, wird ein Änderungsantrag (engl. *change requests*) gestellt und der Änderungsprozess [BRB05] gestartet. Der Antrag dient der Koordination und Nachverfolgung der Änderung und enthält die Änderungsgründe sowie Angaben zum Antragssteller. Ein weiterer Grund für die revisionssichere Dokumentation von Änderungen ist die Einhaltung gesetzlicher Anforderungen (z.B. Produkthaftung). Um die Auswirkungen einer Änderung zu bestimmen, wird letztere von verschiedenen Verantwortlichen des Entwicklungsprozesses bewertet (z.B. Systemverantwortliche, Einkaufs- und Produktionsverantwortliche) und anschließend entweder zur Umsetzung freigegeben oder abgelehnt.
Änderungen spiegeln sich in lokalen Informationssystemen in unterschiedlichen Artefaktversionen von Produktdatenn wider. Obwohl Artefaktversionen von Produktdatenn zu unterschiedlichen Zeitpunkten im Entwicklungsprozess entstehen können, müssen diese über die verschiedenen Informationssystemen hinweg korreliert werden. Nur so kann lückenlos nachvollzogen werden, ob genehmigte Änderungsanträge in den betroffenen Informationssystemen auch technisch umgesetzt worden sind.
Um Versionierung zu realisieren, müssen Mechanismen zur eindeutigen Identifizierung von Änderungen eines Artefaktes verfügbar sein. Dies geschieht in der Regel durch eine eindeutige *Versionsnummer* [EN09]. Versionierte Artefakte stehen in Beziehungen zueinander. Wird ein Artefakt geändert, besitzt die neue Artefaktversion einen Verweis auf seinen Vorgänger und die alte Artefaktversion entsprechend einen Nachfolgerverweis. Eine Artefaktversion kann mehrere Nachfolgerversionen besitzen, die parallel zueinander existieren. Weiter muss eine Unterscheidung zwischen aktuell gültiger und neuester Artefaktversion stattfinden. Artefakte können geändert werden, ohne dass zuvor ein Änderungsantrag erstellt wurde, etwa um die Machbarkeit einer geplanten Änderung zu dokumentieren. Die aktuelle Artefaktversion ist demnach das Resultat des zuletzt freigegebenen Änderungsantrags.
[CW98] beschreiben unterschiedliche Realisierungsarten für Versionierung:

- Bei der *extensionalen Versionierung* werden Entwicklungsstände von Artefakten explizit durch Anwender *„eingecheckt“*.
- Bei der *intensionalen Versionierung* hingegen werden explizite Versionen von Artefakten bei Änderungsanträgen zugeordnet.

Zusammengefasst fällt in den lokalen Informationssystemen zu den verschiedenen Entwicklungsphasen eine Vielzahl an Artefaktversionen an, die zeitlich aufeinander aufbauen. Die Verwaltung

der Änderungen an diesen Artefaktversionen wird als *Versionsverwaltung* bezeichnet. Sie erfolgt in den Informationssystemen mittels unterschiedlicher Mechanismen, wobei sich die konkrete Terminologie je nach Anwendungsdomäne unterscheidet.
Im Allgemeinen muss man zwischen Versionsverwaltung für textbasierte Daten (z.B. Quellcode) und binären Daten (etwa geometrische CAD-Modelle) unterscheiden. Im Folgenden wird die Versionsverwaltung von Artefakten in der Softwareentwicklung mit Ansätzen aus dem CAD-Bereich verglichen.
Eng verbunden mit der Versionsverwaltung von Produktdaten ist das *Release-Management* [MHR06]. Sein Ziel ist es, diejenigen Versionen von Artefakten zu identifizieren, die für die Auslieferung eines Produktes in einem zuvor definierten Reifegrad (z.B. Absicherung, Erprobung, Auslieferung) notwendig sind. Dieser Prozess muss mit dem Konfigurations- und Änderungsmanagement synchronisiert werden (siehe Abschnitt 2.2.3).
In der Softwareentwicklung werden sog. *Versionskontrollsysteme* (englisch *Source Control Management System, SCM*) eingesetzt [Roc75]. [CAD03] vergleicht die verschiedenen Aspekte von SCM- und PDM-Systemen, unter anderem auch die Versionierung sowie das Release-Management von Artefakten: Weit verbreitete Systeme sind Subversion, Git, Mercurial und ClearCase. Die Versionierung von Quellcode und weiteren Artefakten der Softwareentwicklung geschieht auf Dateiebene. Die Dateien werden zentral in einem SCM-Repository gespeichert, das es ermöglicht, den zeitlichen Verlauf der Entwicklung lückenlos zu dokumentieren. Werden Dateien geändert, lässt sich ein Entwicklungsstand festhalten, indem sie in das SCM-Repository *eingecheckt* werden. Dabei werden Metadaten (z.B. Zeit), dokumentiert. Um eine kurz Erläuterung zu den gemachten Änderungen bereitzustellen, wird eine Änderungsinformation vom Verantwortlichen in Prosaform hinterlegt.
Wichtiges Konzept von SCM-Systemen ist das *Branching*. Dieses ermöglicht es, von einem Entwicklungsstand eine Kopie im SCM-Repository anzulegen, die parallel zum Hauptentwicklungsstand existiert. Änderungen an Artefakten dieses Entwicklungszweiges existieren nur dort und beeinflussen den Hauptentwicklungsstrang nicht. *Branches* werden für unterschiedliche Zwecke genutzt. So können neue Funktionalitäten parallel entwickelt und getest werden, ohne dabei die Arbeiten im Hauptentwicklungsstrang zu beeinträchtigen. Eine weitere Möglichkeit besteht darin, Softwareauslieferungen (sog. Softwarereleases) mittels Branches zu erzeugen. D.h., zu einem Zeitpunkt wird ein Branch parallel zum Hauptentwicklungsstrang für eine Softwareauslieferung angelegt. Alle Änderungen, welche die Auslieferung betreffen, werden nach Erstellung des Branches nur noch auf diesem eingecheckt, während die Weiterentwicklung der Software auf dem Hauptentwicklungsstrang geschieht.
Abbildung 6.7 illustriert den Zusammenhang von Hauptentwicklungsstand, Branches und Artefakten. Zunächst ist der Hauptentwicklungsstrang leer, d.h. er enthält keine Artefakte. Bei ① werden die Artefakte `A` und `B` eingecheckt. Anschließend werden in ② Änderungen an `A` sowie ein neues Artefakt `C` in den Hauptentwicklungsstrang eingecheckt. Bei ③ wird ein Branch vom Hauptentwicklungsstrang erzeugt und somit der Inhalt (die Artefakte `A'`, `B` und `C`) kopiert. In ④ werden ein neues Artefakt `D` in den Hauptentwicklungsstrang eingecheckt und das existierende Artefakt `C` entfernt. Parallel werden an ⑤ Änderungen am Artefakt `A'` sowie ein neues Artefakt `E` eingecheckt.

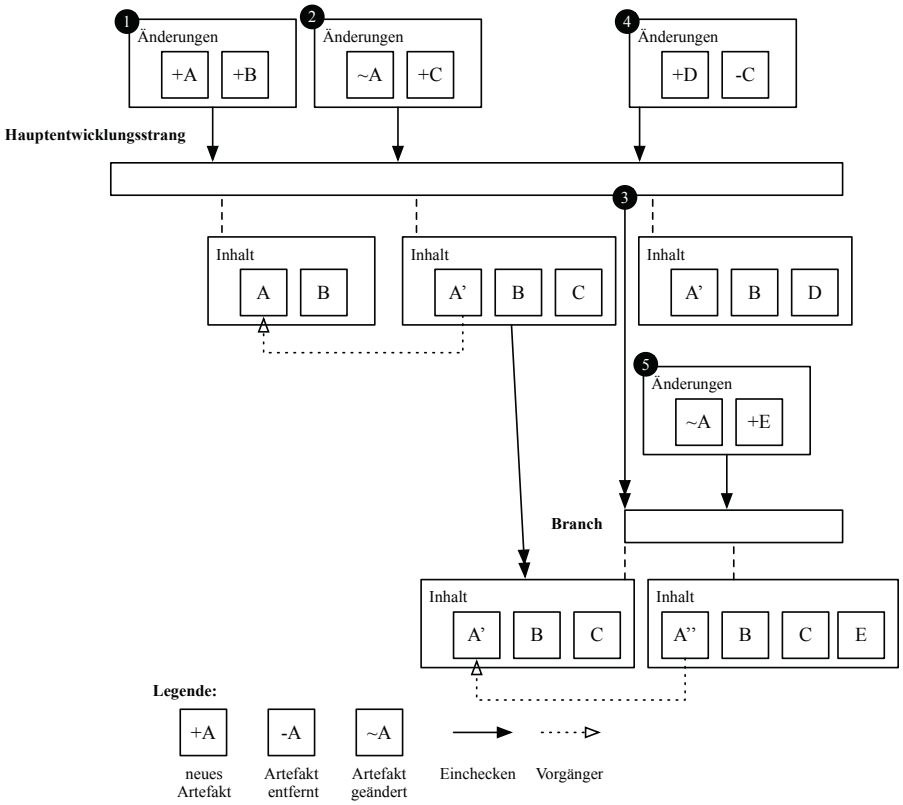


Abbildung 6.7.: Branching in SCM-Systemen

Beispiele für Artefakte in der Softwareentwicklung sind Quelldateien, Konfigurationen und Skripte. Da diese textbasiert sind, lassen sich Änderungen durch Textänderungsoperationen ausdrücken. Das heißt, bei Änderungen von Artefakten wird nicht das gesamte Artefakt, sondern nur die Änderung zum letzen eingecheckten Stand im SCM-System gespeichert. So wird die komplette Änderungshistorie von Artefakten dokumentiert.

Branches können in den Hauptentwicklungsstrang zurückgeführt werden, was als *Merging* bezeichnet wird. Beim Merging wird versucht, den Inhalt von Artefakten auf Textebene automatisch zu integrieren. Ist dies für ein Artefakt nicht möglich, muss der Endbenutzer eingreifen und entscheiden, was bei Integrationskonflikten geschehen soll. Zur Auflösung von Konflikten gibt es zwei Möglichkeiten: Entweder wird das Artefakt aus dem Hauptentwicklungsstrang behalten und Änderungen aus dem Branch verworfen oder umgekehrt. Da das Merging von Artefakten zeilenweise für Textdokumente erfolgt, ist es möglich, dass mehrere Entwickler parallel am selben Artefakt arbeiten können.

In Abbildung 6.8 wird das Mergen eines Branches zurück in den Hauptentwicklungsstrang verdeutlicht. Zum Zeitpunkt ① wird ein Branch des Hauptentwicklungsstranges erzeugt und somit die zu diesem Zeitpunkt vorhandenen Artefakte kopiert. Während bei ② die Änderung des Artefakts `A` sowie ein neues Artefakt `E` im Branch eingecheckt werden, werden im Hauptentwicklungsstrang das Artefakt `C` gelöscht und ein neues Artefakt `D` eingecheckt. Bei ④ erfolgt das Mergen des Branches zurück in den Hauptentwicklungsstrang. Lediglich die Artefakte `A'` des Hauptentwicklungsstranges und `A''` aus dem Branch müssen integriert werden. Sie erzeugen

somit `A'''`, während `C` und `D` in den Hauptentwicklungsstrang kopiert werden und der Branch anschließend gelöscht wird.

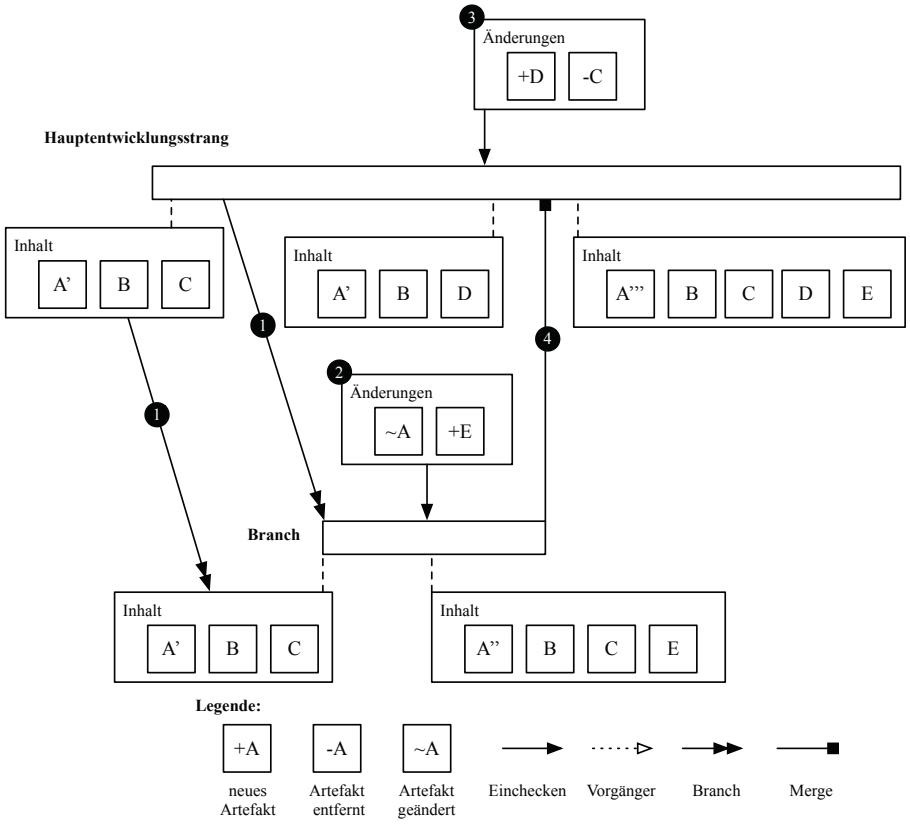


Abbildung 6.8.: Mergen eines Branches zurück in den Hautpentwicklungsstrang

Während SCM-Systeme die Möglichkeit bieten, Entwicklungsstände mit *Tags* zu versehen, die eine semantische Beschreibung darstellen, ist das Release-Management von Software ein nachgelagerter Prozess und nicht Hauptbestandteil eines SCM-Systems.
Während in der Softwareentwicklung überwiegend textbasierte Artefakte verwendet werden, kommen in der Produktentwicklung unterschiedliche Formate (z.B. XML oder Binär-Dateien) zum Einsatz. Einen zentralen Produktdatenaspekt stellen geometrische Modelle dar, die in PDM-Systemen verwaltet werden (siehe Abschnitt 2.2.1). Im Gegensatz zu SCM-Systemen, werden in PDM-Systemen auch die Beziehungen zwischen Artefakten dokumentiert. Auf Grund der unterschiedlichen Formate ist das Mergen von Artefakten, wie bei SCM-Systemen, nicht ohne weiteres möglich. Folglich können an einem Artefakt nicht mehrere Personen gleichzeitig arbeiten. Anders als SCM-Systeme ist das Release-Management zentraler Bestandteil von PDM-Systemen.

Beispiel 18 (PDM Versionierung). Abbildung 6.9a illustriert die Versionierung von Artefakten in einem PDM-System. `Artefakt A` setzt sich aus `Artefakt B` und `Artefakt C` zusammen. Zu letzteren existieren unterschiedliche Versionen (`V1`, `V2` und `V3`).

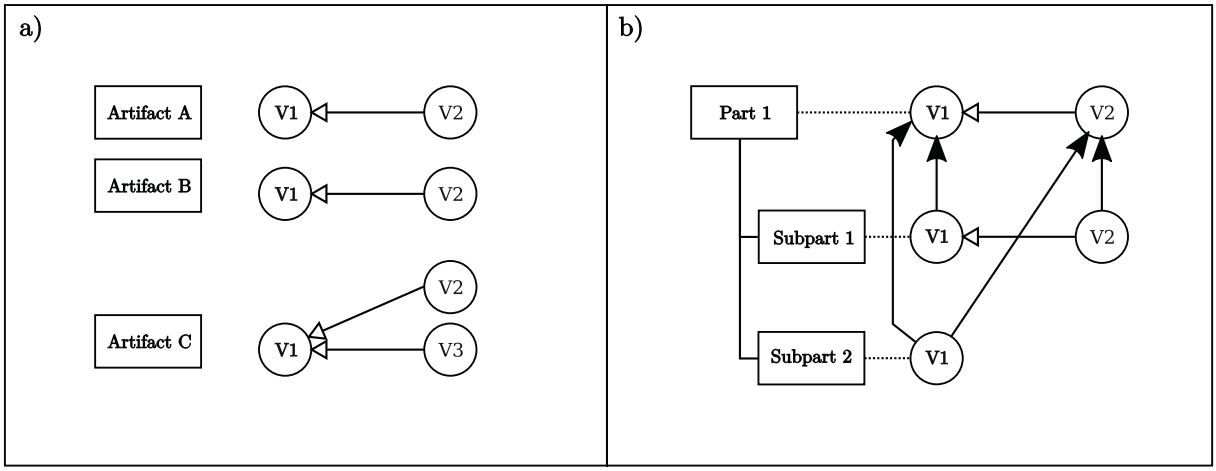


Abbildung 6.9.: Versioning von Artefakten in SCM- und PDM-Systemen

Die Beispiele aus Abbildung 6.9 illustrieren zwei Möglichkeiten, wie die zeitlichen Entwicklungsstände von Produktdatenn gespeichert werden können. Die Entscheidung, wann ein Entwicklungsstand festgehalten werden soll, kann für jedes Informationssystem bzw. jedem Produktdatenaspekt unterschiedlich sein. Wie erwähnt, existieren in der Produktentwicklung übergeordnete Synchronisationspunkte (vgl. Abschnitt 2.2.3). Obwohl Produktdaten in den einzelnen Informationssystemen unterschiedlich versioniert sein können, müssen zu den Synchronisationspunkten entsprechende Entwicklungsstände für eine Integration identifizierbar sein.
Im weiteren Verlauf der Arbeit wird folgende Taxonomie für die Versionierung von Produktdaten verwendet.

Taxonomie 2 (Versionsmanagement).

- **Artefakt:** Dokument, Modell oder Datei von Produktdaten, das Änderungen unterliegt.
- **Version:** Beschreibt den zeitlichen Bezug eines Artefaktes.
- **Versionsnummer:** Eindeutige Nummer zwecks Unterscheidung von Versionen.
- **Änderungsantrag:** Änderungsvorhaben für ein oder mehrere Bauteile. Dient gleichzeitig der Dokumentation von Änderungen, um eine durchgängige Nachvollziehbarkeit zu ermöglichen.
- **Baseline:** Gemeinsamer, akzeptierter Zustand von Artefakten zu einem Zeitpunkt durch alle beteiligten Verantwortlichen eines Produktes. Die Baseline dient als Grundlage für die Definition von Änderungsanträgen. Ein akzeptierter Änderungsantrag ändert den Zustand einer Baseline.
- **Vorgänger** und **Nachfolger** beschreiben die zeitliche Beziehungen zwischen Artefaktversionen. Vorgänger einer Version ist ein Artefakt, das zeitlich vorher angelegt

wurde. Analog ist ein Nachfolger diejenige Version, die zeitlich nach einem Artefakt erstellt wurde.

- Es wird zwischen **aktuell gültiger** und **zuletzt erstellter** Version unterschieden.

Zusammengefasst muss ein lokales Informationssystem folgende Eigenschaften erfüllen, um in eine lokale Produktontologie abgebildet werden zu können:

- Das lokale Informationssystem muss mindestens einen Produktdatenaspekt (z.B. Anforderungen, Zeichnungen) enthalten,
- Bauteile sind zu Kollektionen zusammengefasst.
- Variabilität und Versionierung der Produktdaten müssen dokumentieren werden.

Formalisierung der Produktontologie

Wir definieren eine Produktontologie wie folgt:

Definition 21 (Produktontologie). Eine Produktontologie mit einem eindeutigen Bezeichner x ist ein Tupel $PO_x \equiv (PDIL, SC_x, Ind_x, Attr_x, AttrVal_x, MetaAttr_x, MetaAttrVal_x)$, mit

- $PDIL_x = \{pdc, object, variant, version\}$ ist die Menge der Produktdatenintegrationsebenen,
- $SC_x = \{sc_1, \ldots, sc_n\}$ die Menge der Schemakonzepte,
- $Ind_x = \{ind_1, \ldots, ind_k\}$ die Menge der Individuals der Schemakonzepte,
- $Attr_x = \{attr_1, \ldots, attr_m\}$ die Menge der Attribute von Schemakonzepten,
- $AttrVal_x = \{attrVal_1, \ldots, attrVal_l\}$ die Menge der Attributwerte der Individuals,
- $MetaAttr_x = \{ma_1, \ldots, ma_p\}$ die Menge der Metaattribute für jedes Artefakt der Produktontologie (z.B. Schemakonzept, Instanz, Attribut),
- $MetaAttrVal_x = \{mav_1, \ldots mav_q\}$ die Menge der Werte für Metaattribute

Die Struktur einer Produktontologie ist folgendermaßen definiert:

Definition 22 (Struktur einer Produktontologie). Auf diesen Mengen seien weiter folgende Relationen definiert:

- $(pdil_1, pdil_2) \in isAbove_x = \{(pdc, object), (object, variant), (variant, version)\}$ mit $pdil_1, pdil_2 \in PDIL$ die hierarchische Ordnung der Produktdatenintegrationsebenen,
- $(pdil_1, pdil_2) \in isBelow_X = \{(object, pdc), (variant, object), (version, variant)\}$ mit $pdil_1, pdil_2 \in PDIL$,
- $(pdil, sc) \in hasConcept_x$ mit $pdil \in PDIL$ und $sc \in SC_x$ die Zuordnung von Schemakonzepten zu Produktdatenintegrationsebenen

- $(sc_a, sc_b) \in scAbove_x$ die hierarchische Ordnung zweier Schemakonzepte $sc_a, sc_b \in SC_x$, wobei sc_a in der PDIL über sc_b liegt: $pdil_a, pdil_b \in PDIL_M$ $hasConcept_x(pdil_a, sc_a)$ und $hasConcept_x(pdil_b, sc_b)$ gilt $isAbove(pdil_a, pdil_b)$
- $(sc_a, sc_b) \in scBelow_x$ die hierarchische Ordnung zweier Schemakonzepte $sc_a, sc_b \in SC_x$, wobei sc_a in der PDIL unter sc_b liegt: $pdil_a, pdil_b \in PDIL_M$, $hasConcept_x(pdil_a, sc_a)$ und $hasConcept_x(pdil_b, sc_b)$ gilt $isBelow(pdil_a, pdil_b)$
- $(ind_a, ind_b) \in indAbove$ die hierarchische Ordnung zweier Instanzen $ind_a, ind_b \in Ind_x$, wobei erste zu einem Schemakonzept gehört, welches in einer Integrationsebene überhalb des Schemakonzeptes für ind_b liegt.
- $(ind_a, ind_b) \in indBelow$ die hierarchische Ordnung zweier Instanzen $ind_a, ind_b \in Ind_x$, wobei erste zu einem Schemakonzept gehört, welches in einer Integrationsebene unterhalb des Schemakonzeptes für ind_b liegt.
- $(sc, attr) \in hasAttribute_x$ mit $sc \in SC_x$ und $attr \in Attr_x$ die Zuordnung von Attributen zu Schemakonzepten,
- $(sc, ind) \in hasIndividual_x$ mit $sc \in SC_x$ und $ind \in Ind_x$ die Zuordnung von Individuals zu Schemakonzepten,
- $(ind, attrVal) \in hasValue_x$ mit $ind \in Ind_x$ und $attrVal \in AttrVal_x$ die Zuordnung von Attributwerten zu Individuals,
- $(attrVal, attr) \in references_x$ mit $attrVal \in AttrVal_x$ und $attr \in Attr_x$ die Zuordnung von Attributwerten zu Attributen.
- $(ind_{version1}, ind_{version2}) \in isPredecessor_x$ mit $ind_{version1}, ind_{version2} \in Ind_x^{version}$ die Zuordnung von Vorgängerbeziehungen zwischen Individuals der Version-Ebene,
- $(ind_{version1}, ind_{version2}) \in isSuccessor_x$ mit $ind_{version1}, ind_{version2} \in Ind_x^{version}$ die Zuordnung von Nachfolgerbeziehungen zwischen Individuals der Version-Ebene,
- $(x, ma, mav) \in hasMetaAttrVal_x$ mit $x \in SC_x \bigcup Ind_x \bigcup Attr_x \bigcup AttrVal_x$ und $ma \in MetaAttr_x$ sowie $mav \in MetaAttrVal_x$ die Zuordnung von Metaattributwerten zu Elementen in der Produktontologie.

Definition 23 (Hilfsrelationen). Auf Basis der Relationen werden weiter folgende Hilfsmengen definiert:

- $SC_x^{pdil} = \{sc \in SC_x | \exists (pdil, sc) \in hasConcept_x\}$ mit $pdil \in PDIL$ die Menge der Schemakonzepte einer Produktdatenintegrationsebene,
- $Ind_x^{pdil} = \{ind \in Ind_x | \forall sc \in SC_x \forall ind \in Ind_x : (pdil, sc) \in hasConcept_x \vee (sc, ind) \in hasIndividual_x\}$ mit $pdil \in PDIL$ die Menge der Individuals einer Produktdatenintegrationsebene

Strukturelle Konsistenzbedingungen

Zwischen den Produktdatenintegrationsebenen existieren strukturelle Konsistenzbedingungen, die bei der Modellierung von Schemakonzepten eingehalten werden müssen. Die strukturelle Konsistenz bezieht sich sowohl auf Beziehungen zwischen Schemakonzepten als auch die Instanzen dieser Schamkonzepte.

Definition 24 (Strukturelle Konsistenz einer Produktontologie). Eine lokale Produktontologie ist ein Tupel
$PO_L \equiv (PDIL_L, SC_L, Ind_L, Attr_L, AttrVal_L, MetaAttr_L, MetaAttrVal_L)$ [a] und strukturell konsistent, wenn folgende Bedingungen erfüllt sind:

- **Schemakonsistenzregel 1:** Jede Produktdatenintegrationsebene besitzt mindestens ein Schemakonzept, welche untereinander verbunden sind. Ein Schemakonzept kann nur mit einem Schemakonzept verbunden werden, welches sich auf einer darunter liegenden Produktdatenintegrationsebene befindet:
  $\forall pdil \in PDIL_L \exists sc \in SC_L \exists (phil, sc) \in hasConcept_L$ und $\forall (sc_1, sc_2) \in SCC_{LPO}$ mit $sc_1, sc_2 \in SC_L$ gilt: $\exists pdil_1, pdil_2 \in PDIL_L$ mit $pdil_1 \neq pdil_2$ und
  $(pdil_1, pdil_2) \in isAbove_L$ und $(pdil_2, pdil_1) \in isBelow_L$
  $\exists (pdil_1, sc_1) \in hasConcept_L$, $\exists (pdil_2, sc_2) \in hasConcept_L$
- **Schemakonsistenzregel 2:** Für jedes Schemakonzept der Variant-Ebene ist der Variabilitätstyp definiert:
  $\forall sc \in SC_L^{variant} \exists (sc, variantType) \in hasAttribute_L$
- **Schemakonsistenzregel 3:** Zusätzlich ist der **Versionierungstyp** für jedes Schemakonzept der Version-Ebene definiert:
  $\forall sc \in SC_L^{version} \exists (sc, versionType) \in hasAttribute_L$
- **Instanzkonsistenzregel 1:** Die Instanzen einer globalen Produktontologie sind zwischen den Produktdatenintegrationsebenen analog zu ihren Schemakonzepten über Beziehungen miteinander verknüpft. Das heißt, dass für alle Instanzen außer der PDC-Ebene eine Beziehung zu einer Instanz auf der darüberliegenden Produktdatenintegrationsebene existiert:
- **Instanzkonsistenzregel 2:** Die Instanzen eines Schemakonzeptes auf der *Version*-Ebene sind durch *Vorgänger*- und *Nachfolger*-Beziehungen geordnet:
  $\exists (ind_1, ind_2) \in isPredecessor_L$ mit $ind_1, ind_2 \in Ind_L^{version}$ und
  $\exists (ind_1, ind_2) \in isSuccessor_L$ mit $ind_1, ind_2 \in Ind_L^{version}$.
- **Schemakonsistenzregel 4:** Jedes Schemakonzept der 3. und 4. Integrationsebene ist mit genau einem Schemakonzept auf der darüberliegenden Ebene verbunden (Ausnahme: Schemakonzepte der obersten Ebene):
  $\forall sc_a \in SC_{Var} \cup SC_{Ver}$ mit $l \in PDC(l, sc_a) \in hasConcept_L \exists! sc_b \in SC$ mit $(sc_b, sc_a) \in scAbove_L$
- **Schemakonsistenzregel 5:** Jedes Schemakonzept der 2. und 3. Integrationsebene ist mit genau einem Schemakonzept auf der darunterliegenden Ebene verbunden

$\forall sc_a \in SC_L^{Obj} \cup SC_L^{Var}$ mit $l \in PDC(l, sc_a) \in hasConcept_L \; \exists! sc_b \in SC$ mit $(sc_b, sc_a) \in scBelow_L$

[a] Für die Mengen gilt analog die Beschreibungen aus Definition 21.

Abbildung 6.10 zeigt ein Beispiel für eine lokale Produktontologie aus dem Automobilbereich. Im Einzelnen verwaltet ein Informationssystem geometrische Zeichnungen für elektronische Steuergeräte (engl. electronic control units) (ECUs). Weiter verwendet dieses System spezielle Techniken zur Speicherung von Steuergeräte-varianten und -versionen, die im konzeptuellen Datenmodell des Informationssystems verankert sind. Um die semantische Heterogenität von Datenmodellen zur Speicherung von Produktdatenn zu vereinheitlichen, werden Datenmodellkonzepte über eine einheitliche hierarchische Struktur abstrahiert. Diese Hierarchie besteht aus vier Produktdatenintegrationsebenen.

Beispiel 19 (Lokale Produktontologie). Eine Lokale Produktontologie $PO_L \equiv (PDIL, SC_L, Ind_L, Attr_L, AttrVal_L, MetaAttr_L, MetaAttrVal_L)$ mit

- $PDIL = \{pdc, object, variant, version\}$
- $SC_L = \{Product, Part, PartVariant, PartVersion\}$
- $Ind_L = \{car, engine, diesel, gas, dieselv_1, dieselv_2, gasolinev_1\}$
- $Attr_L = \{Name, VariantType, VersionType\}$
- $scAbove = \{(Product, Part), (Part, PartVariant), (PartVariant, PartVersion)\}$
- $scBelow = \{(Part, Product), (PartVariant, Part), (PartVersion, PartVariant)\}$
- $indAbove = \{(car, engine), (engine, diesel), (engine, gas), (diesel, dieselv_1), (diesel, dieselv_2), (gas, gasolinev_1)\}$
- $indBelow = \{(engine, car), (diesel, engine), (gas, engine), (dieselv_1, diesel), (dieselv_2, diesel), (gasolinev_1, gas)\}$
- $hasConcept = \{(pdc, Product), (object, Part), (variant, PartVariant), (version, PartVersion)\}$
- $hasAttribute = \{(Product, Name), (Part, Name), (PartVariant, Name), (PartVariant, VariantType), (PartVersion, Name), (PartVersion, VersionType)\}$
- $hasIndividual = \{(Product, Car), (Part, engine), (PartVariant, diesel), (PartVariant, gasoline), (diesel, dieselv_1), (diesel, dieselv_2), (gasoline, gasolinev_1)\}$
- $isSuccessor = \{(gasolinev_2, gasolinev_1)\})$
- $isPredecessor = \{(gasolinev_1, gasolinev_2)\})$
- $\{(gasolinev_1,), (gasolinev_2, v_2)\}$

Entsprechend umfasst die oberste Produktdatenintegrationsebene *Product Data Collection* ein-

zelne Produkte und aggregiert so die benötigten Bauteile auf der darunter liegenden *Object*-Ebene. Ein Teil kann in unterschiedlichen Ausführungen entwickelt werden, d.h. unterschiedliche Features unterstützen. Diese werden in der *Variant*-Ebene für jedes Objekt verwaltet. Existiert keine spezifische Variante für ein Objekt, wird nur eine Variante angelegt. Da sich die Entwicklung von Teilen über einen längeren Zeitraum erstreckt, entsteht eine Vielzahl an Entwicklungsständen, die in der *Version*-Ebene verwaltet werden. Da sich aus den Namen unter Umständen nicht automatisch ableiten lässt, in welcher Beziehung einzelne Versionen zueinander stehen, müssen diese Beziehungen, auch *Versionshistorie* genannt, explizit dokumentiert werden. Zusammengefasst ist eine Produktontologie eine Baumstruktur mit fest definierten Hierarchieebenen.

In Abbildung 6.10 ist in jeder Produktdatenintegrationsebene ein Schemakonzept abgebildet: Ein *Product* besitzt mehrere *ECU*, welche wiederum in unterschiedlichen *ECU Variant* existieren können. Jede Steuergerätevariante nimmt im Laufe der Zeit eine Vielzahl an Entwicklungsständen (*ECU Version*) ein. Neben den Schemakonzepten sind Instanzen dargestellt. Das Produkt ist ein Auto (*Car*), das ein Steuergerät zur Regelung des Motors (*Engine*) besitzt, der entweder ein Dieselmotor (*Diesel*) oder ein Benzinmotor (*Gas*) ist. Für die einzelnen Steuergerätevarianten existieren mehrere Entwicklungsstände.

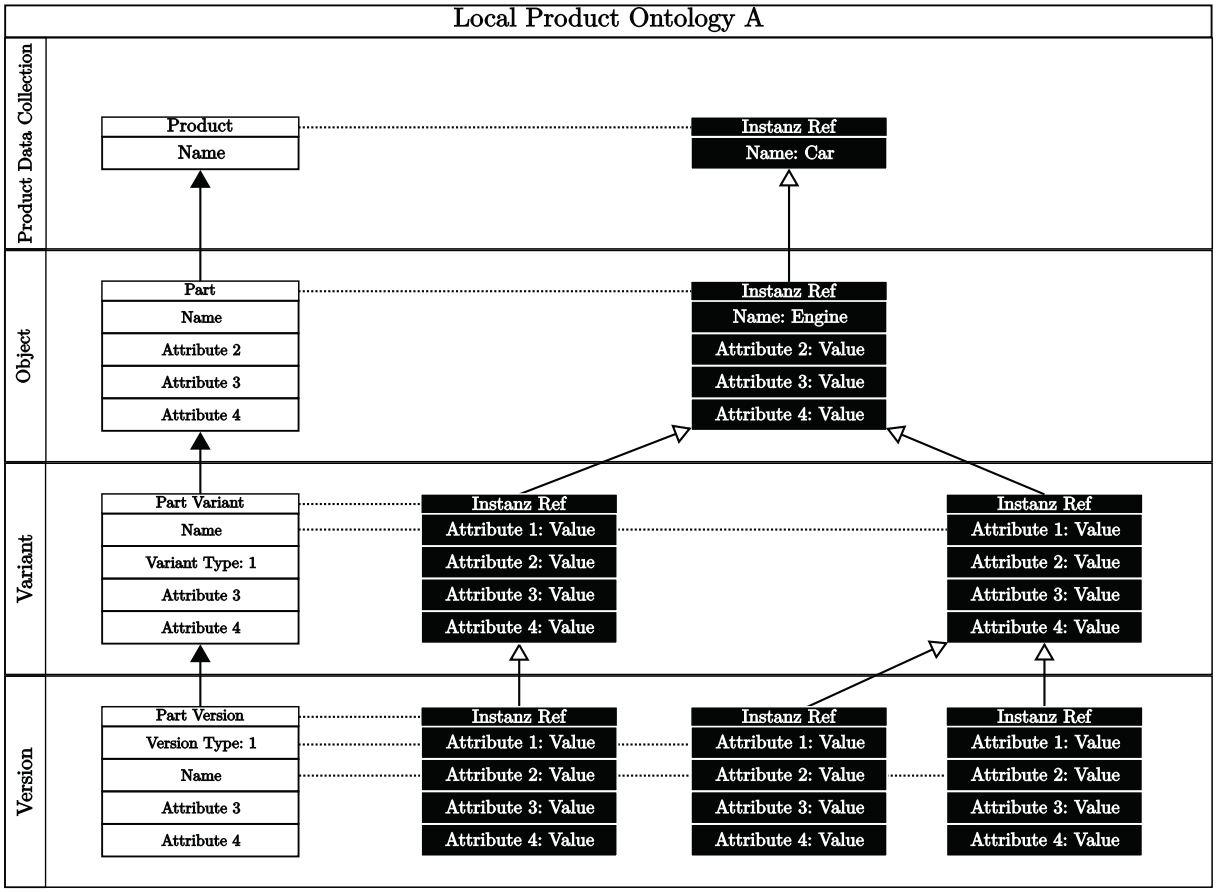


Abbildung 6.10.: Beispiel einer lokalen Produktontologie

6.2.2. Globale Produktontologie

Genau wie die konzeptuellen Datenmodelle der lokalen Informationssysteme in lokale Produktontologien abstrahiert werden, ist das konzeptuelle Datenmodell des PDIS ebenfalls eine Produktontologie. Diese wird als *globale Produktontologie* bezeichnet.
Produkte bestehen in der Regel aus Komponenten / Teilen, die gleichzeitig durch unterschiedliche Abteilungen entwickelt werden. Die globale Produktontologie besitzt, genau wie die lokalen Produktontologien, die gleichen vier Produktdatenintegrationsebenen. Dadurch ist es möglich, Produktdaten für jede Ebene separat zu integrieren. Die größte Herausforderung ergibt sich bei der Integration von Instanzen aus lokalen Produktontologien in die globale Produktontologie. Im Speziellen müssen diejenigen Instanzen zueinander in Beziehung gebracht werden, die sich auf dieselben Realweltobjekte beziehen.

Definition 25 (Globale Produktontologie). Eine globale Produktontologie PO_G dient der Integration lokaler Produktontologien und ist ein Tupel
$PO_G \equiv (PDIL, SC_G, Ind_G, Attr_G, AttrVal_G, MetaAttr_G, MetaAttrVal_G)$ und besitzt die in Definition 24 spezifizierten strukturellen Konsistenzbedingungen.

Die Integration von verschiedenen Produktdatenn kann unterschiedlich in eine globale Produktontologie erfolgen. Entweder wird in einer globalen Produktontologie nur ein Schemakonzept (vgl. Abbildung 6.11) oder für jeden Produktdatenaspekt ein eigenes (siehe Abbildung 6.12) in der Object-Integrationsebene angelegt.
Im ersten Fall wird eine globale Produktontologie durch die Schemakonzepte *Produkt*, *Bauteil*, *Bauteilvariante* und *Bauteilversion* beschrieben. Alle Produktdaten der lokalen Produktontologien werden auf das Schemakonzept *Bauteil* abgebildet. So existiert in der globalen Produktontologie für jedes Bauteil nur eine Instanz, welche alle Produktdaten umfasst.

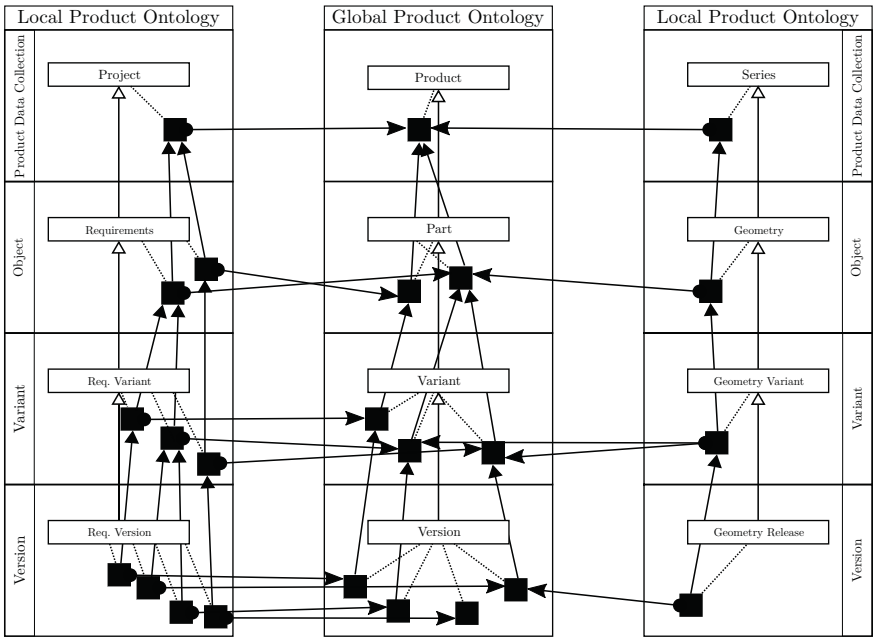


Abbildung 6.11.: Modellierung einer GPO ohne explizite Unterscheidung von Produktdaten

Anders verhält es sich im zweiten Fall, bei dem für jeden Produktdatenaspekt ein eigenes Schemakonzept, inklusive Varianten und Versionen, modelliert wird. In diesem Fall existieren zu jedem Schemakonzept eigene Instanzen, dessen Korrespondenzen dokumentiert werden müssen, um Zusammengehörigkeiten darzustellen. Zwar ist im zweiten Fall auf einen Blick ersichtlich, welche Produktdaten in einer globalen Produktontologie vorhanden sind, jedoch müssen dafür mehr Schemakonzepte modelliert werden. Außerdem entstehen mehr Instanzen in der globalen Produktontologie zwischen denen Korrespondenzen angelegt werden müssen, um einzelne Bauteile zu beschreiben. Die Nachteile des zweiten Ansatzes sind zugleich die Vorteile des ersten. Durch die geringere Anzahl von Schemakonzepten sind weniger Regeln zur Abbildung zwischen lokaler und globaler Produktontologie notwendig.

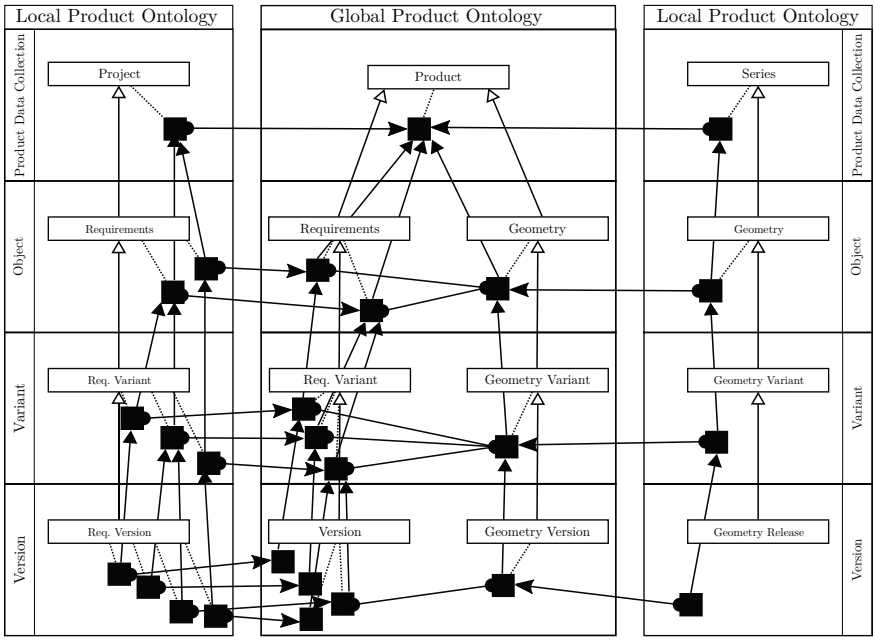


Abbildung 6.12.: Modellierung einer GPO mit expliziter Unterscheidung von Produktdatenn

Der grundlegende Aufbau einer globalen Produktontologie ist in Abbildung 6.13 illustriert. Jede Integrationsebene umfasst ein Schemakonzept (*Product*, *Part*, *Variant* und *Version*). Ein Produkt (engl. *Product*) fasst alle Bauteile (engl. *Part*) zusammen, die zur Herstellung aller möglichen Produktvarianten notwendig sind. Eine Variante (engl. *Variant*) beschreibt eine oder mehrere Bauteilvarianten (je nach Variabilitätstyp, siehe Abschnitt 6.2.1). Für alle Bauteile in allen Varianten existieren mehrere Zeitpunkte (engl. *Releases*) zu denen Entwicklungsergebnisse explizit global für das Produkt dokumentiert werden. Darüber hinaus besitzt jedes Schemakonzept eine Reihe von Metaattributen[4].
Die in der Abbildung gezeigte globale Produktontologie kann um weitere Attribute und Schemakonzepte erweitert werden, falls dies für die Realisierung von Anwendungsfällen notwendig ist.

[4]Metaattribute können etwa aus dem Dublin Core [WKLW98] übernommen werden

Metaattribut	Beschreibung
GUID	Global eindeutige Identifikationsnummer.
Name	Kurznamen für das Schemakonzept.
Description	Beschreibung des Schemakonzepts. Zusätzlich kann explizites Wissen für eine spätere Integration dokumentiert werden.
Last modified	Zeitstempel, an dem zuletzt Änderungen am Schemakonzept vorgenommen wurden.

Tabelle 6.3.: Variabilitätsmechanismen für verschiedene Produktdaten

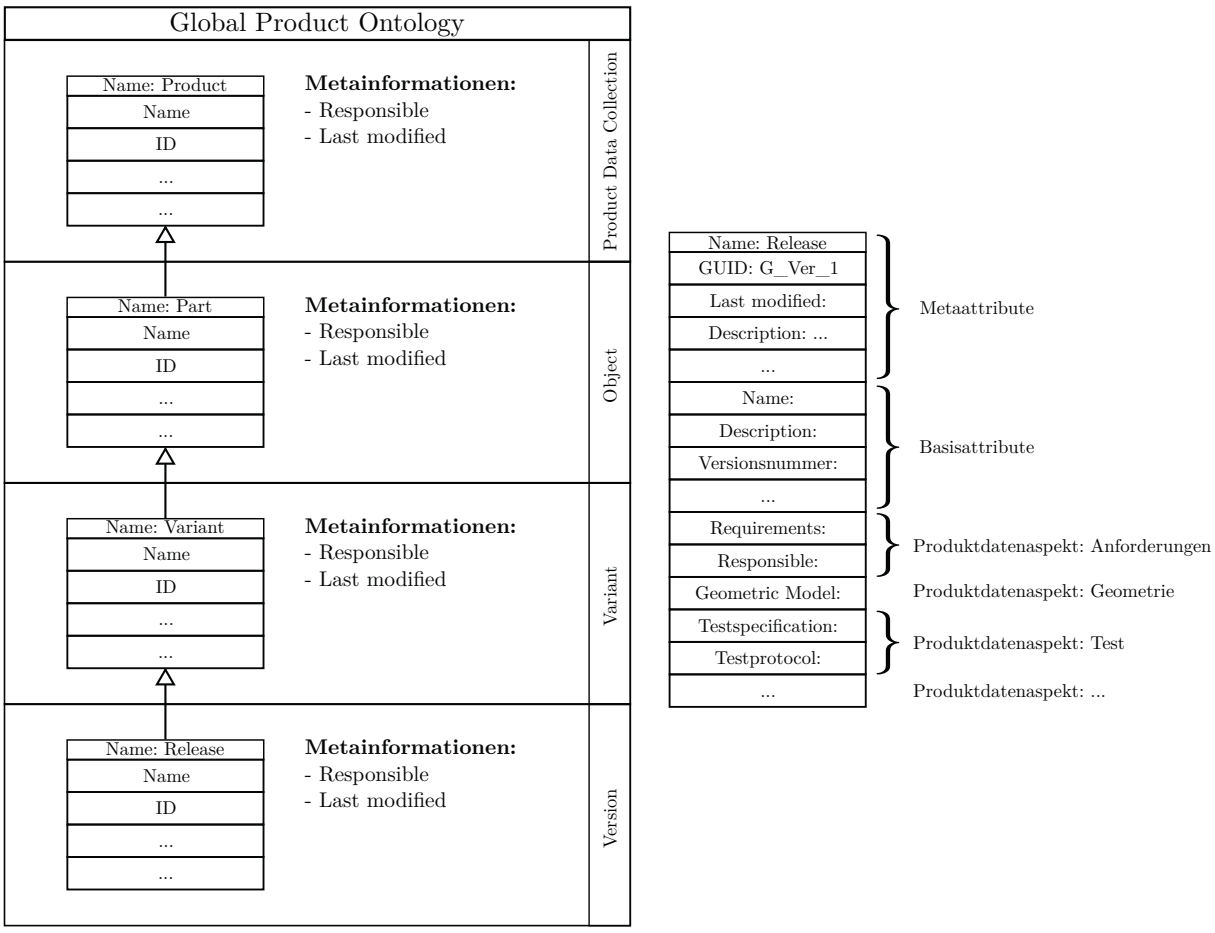


Abbildung 6.13.: Aufbau einer globalen Produktontologie

6.2.3. Abbildung lokaler Produktontologien auf eine globale Produktontologie

Nach der Modellierung lokaler und globaler Produktontologien müssen diese aufeinander abgebildet werden. Dazu werden *Schemakonzept-Attributregeln* definiert. Sie legen fest, wie Korrespondenzen von Instanzen zwischen zwei Schemakonzepten automatisch abgeleitet werden können. Während der automatischen Integration werden für ermittelte Korrespondenzen zwischen Instanzen sog. *Schemakonzept-Attributaktionen* ausgeführt, die Attributwerte von Instanzen aus lokalen Produktontologien in die globale Produktontologie überführen.

Schemakonzept-Attributregeln

Wie erwähnt, sind Informationssysteme der Produktentwicklung meist auf spezifische Anwendungsfälle zugeschnitten. Sie fokussieren in der Regel auf einen bestimmten Produktdatenaspekt. Folglich besteht eine lokale Produktontologie aus wenigen Schemakonzepten, die von Applikations- und Domänenexpterten verwaltet werden. Auf der anderen Seiten können mehrere Tausend Instanzen für ein Produkt vorhanden sein, was vor allem auf die *Version*-Ebene zutrifft. Daher sollte die Integration dieser Instanzen, soweit möglich, automatisiert werden.
Ziel der der globalen Produktontologie ist es, eine holistische Sicht auf alle Produktdaten herzustellen. Folglich müssen Abbildungen zwischen den Schemakonzepten aus lokalen und globaler Produktontologie definiert werden. Ziel ist es, *Korrespondenzen* zwischen Instanzen von Schemakonzepten aus lokalen Produktontologien und globaler Produktontologie automatisch zu ermitteln. Wir definieren entsprechende Korrespondenzen wie folgt:

Definition 26 (Korrespondenz). Eine Korrespondenz beschreibt einen semantischen Zusammenhang zwischen einer Instanz einer lokalen Produktontologie und einer entsprechenden Instanz einer globalen Produktontologie auf der gleichen Produktdatenintegrationsebene. Für eine Instanz aus einer lokalen Produktontologie können Korrespondenzen auf mehrere Instanzen einer globalen Produktontologie vorhanden sein und umgekehrt.
Für eine lokale Produktontologie PO_L und globale Produktontologie PO_G sind Korrespondenzen zwischen Instanzen wie folgt definiert:
$(ind_L, ind_G) \in Correspondence$ mit $ind_L \in Ind_L \wedge ind_G \in Ind_G$

Das Problem, dass Informationen zu Realweltobjekten über unterschiedliche Informationssysteme verteilt sind, ist bereits seit den 1960er-Jahren bekannt. Um zusammengehörige Datensätze (engl. *records*) zu identifizieren, sind unterschiedliche *Record-Linkage*-Techniken [Win99] entstanden. Diese zielen darauf ab, Einträge aus unterschiedlichen Datensätzen zu finden, die ein Realweltobjekt beschreiben. Einige dieser Techniken basieren auf Regeln, die Attribute von Einträgen in Beziehung zueinander stellen.
Dieses Vorgehen lässt sich auch für die Integration im PROCEED-Rahmenwerk anwenden. So werden zwischen Attributen von Schemakonzepten aus lokalen Produktontologien und globaler Produktontologie sog. *Schemakonzept-Attributregeln* (engl. *schema concept attribute rules* (SCAR)) definiert. Diese Regeln werden für jede lokale Produktontologie definiert. D.h., jedes Schemakonzept einer lokalen Produktontologie wird auf genau ein Schemakonzept der selben Produktdatenintegrationsebene in einer globalen Produktontologie abgebildet. Eine Schemakonzept-Attribut-Regel beschreibt, unter welchen Voraussetzungen die Instanzen zweier Schemakonzepte aus lokaler und globaler Produktontologie zueinander korrespondieren, so dass eine Korrespondenz zwischen diesen vermerkt werden kann.

Schemakonzept-Attributregeln setzen sich aus einer oder mehreren *Attribut-Matching-Bedingungen* (engl. *attribute matching condition* (AMC) zusammen. Letztere definieren für Attribute eines lokalen Schemakonzeptes, wie die Korrespondenz zu Attributen eines globalen Schemakonzeptes mit Hilfe einer *Attribut-Matching-Funktion* (engl. *attribute matching function* (AMF)) automatisch bestimmt werden kann.
Sind zum Beispiel die Namenskonventionen zur Benennung von Instanzen ähnlich, können Zeichenkettenmetriken als Attribut-Matching-Funktion definiert werden, um Korrespondenzen zu ermitteln. Häufig teilen Produktdaten der einzelnen Informationssysteme gemeinsame Attribute (z.B. Abkürzungen, eindeutige Nummern), die ebenfalls ausgenutzt werden können, um korrespondierende Instanzen zu ermitteln. Existieren wiederkehrende Muster, etwa für die Benennung von Artefakten in einer Produktontologie, können reguläre Ausdrücke verwendet werden, um Korrespondenzen zwischen lokalen Produktontologien und globaler Produktontologie zu ermitteln. Lassen sich keine der zuvor erwähnten Funktionen definieren, so müssen Abbildungstabellen verwaltet werden, welche eine Liste aus Quell- und Zielattributwerten von Instanzen aus lokaler und globaler Produktonologie darstellen. Attribut-Matching-Funktionen sind wie folgt definiert:

Definition 27 (Attribute-Matching-Funktion). Für eine lokale Produktontologie PO_L und eine globale Produktontologie PO_G liefert eine Attribut-Matching-Funktion für eine Instanz jeweils aus PO_L und PO_G zu einem Attribut jeweils aus $Attr_L$ und $Attr_G$ entweder wahr (engl. true) oder falsch (engl. false). Für eine Attribut-Matching-Funktion kann ein beliebige Anzahl an Parametern definiert sein.
$matchFunction : Ind_L \times Ind_G \times Attr_L \times Attr_G \times Param_1, \times \ldots, \times Param_n \rightarrow \{true, false\}$
Die Menge $(matchFunction_1, \ldots, matchFunction_m) \in MatchFunction$ beschreibt die vorhandenen Attribut-Matching-Funktionen.

Wie erwähnt, werden Attribut-Matching-Funktionen in Attribut-Matching-Bedingungen verwendet, um eine automatische Identifikation von Korrespondenzen zwischen Instanzen auf Basis zweier Attribute aus lokaler und globaler Produktontologie zu beschreiben. Diese Informationen dienen als Grundlage für den in Abschnitt 6.3 vorgestellten Integrationsalgorithmus.

Definition 28 (Attribute-Matching-Bedingung). Für eine lokale Produktontologie PO_L und eine globale Produktontologie PO_G definieren wir eine Attribute-Matching-Bedingung wie folgt:
$(attr_L, attr_G, mf, par) \in AttributeMatchingCondition$ mit
$attr_L \in Attr_L$ und $attr_G \in Attr_G$ und $mf \in MatchingFunction$

Während eine Attribut-Matching-Bedingung den semantischen Zusammenhang zwischen jeweils einem Attribut eines lokalen und globalen Schemakonzeptes definiert, kann es notwendig werden, mehrere Bedingungen zusammenzufassen, um so eine eindeutige Korrespondenz zwischen Instanzen feststellen zu können.
Abbildung 6.14 illustriert einen Ausschnitt einer lokalen Produktontologie mit Instanzen der untersten Produktdatenintegrationsebenen. Betrachtet man die Instanzen in dem Beispiel, so fällt auf, dass ein Attribut nicht ausreicht, um eine Instanz eindeutig zu beschreiben. Würde man nur die Abbildungsregeln der Attribute *Label/ Name* oder *Version/Release* verwenden, würden falsche Korrespondenzen während der Auswertung der SCAR erzeugt werden. Folglich müssen Schemakonzept-Attributregeln mehrere Attribute-Matching-Bedingungen zusammenfassen können.

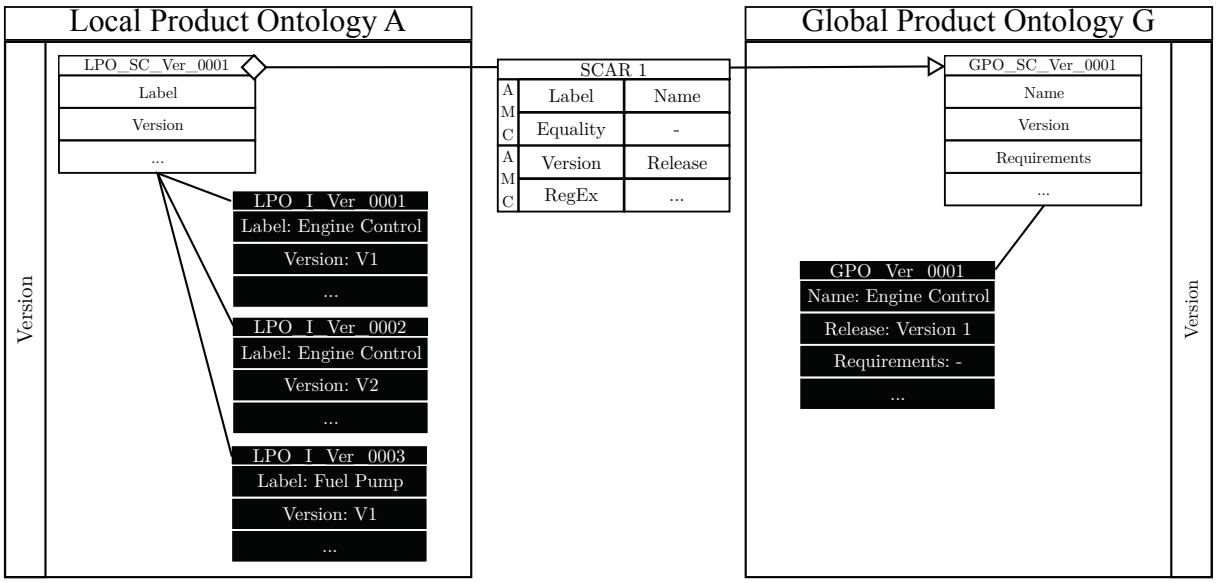


Abbildung 6.14.: SCAR mit mehreren AMC

Die Identität zwischen Instanzen kann durch unterschiedliche Attribute bestimmt werden. Beispielsweise können zwei Instanzen durch eine gemeinsame globale definierte Referenz als Korrespondenzen bestimmt werden. Weiter können beide Instanzen den gleichen, eindeutigen Namen besitzen. In diesem Fall können zwischen einem lokalen und globalen Schemakonzept mehrere SCARs definiert werden.
Zusammenfassend definieren wir Schemakonzept-Attributregeln zwischen Schemakonzepten aus lokaler und globaler Produktontologie wie folgt:

Definition 29 (Schemakonzept-Attributregel (SCAR)). Eine Schemakonzeptattributregel $SCAR := (sc_L, sc_G, AMC)$ ist ein Tupel mit folgenden Eigenschaften:

- sc_L ist ein Schemakonzept aus einer lokalen Produktontologie,
- sc_G ist ein Schemakonzept aus der globalen Produktontologie,
- $\bigwedge_{i \in I} amc_i \in AMC$ ist die Konjunktion von Attribut-Matching-Bedingungen.

Um nachvollziehen zu können, warum eine Korrespondenz zwischen zwei Instanzen besteht, wird während der automatischen Integration für diese eKorrespondenz eine Begründung (engl. *correspondence justification*) hinterlegt. Letztere enthält einen Verweis auf die SCAR, welche für das Zustandekommen der Korrespondenz verantwortlich ist. Werden Korrespondenzen manuell erstellt, kann ebenfalls eine Begründung durch Anwender dokumentiert werden. Wir definieren eine Korrespondenz-Begründung wie folgt:

Definition 30 (Korrespondenz-Begründung). Folgende Funktion liefert die Begründung zu einer Korrespondenz: $getJustification : Correspondence \rightarrow 2^{Justification}$
Eine Begründung ist eine Menge aus geordneten Paaren $(cor, scar) \in Justification$ mit

cor ∈ *Correspondence* und *scar* ∈ *SCAR*. Eine Attribute-Matching-Rule entspricht der Konjunktion einer oder mehreren Attribute-Matching-Conditions.

Je nach Produktdatenintegrationsebene werden unterschiedliche Attribut-Matching-Funktionen benötigt. Für die vier Produktdatenintegrationsebenen werden im Folgenden mögliche Matching-Funktionen erläutert. Da die Informationssysteme der Produktentwicklung typischerweise für spezifische Anwendungsfälle konzipiert worden sind, verwenden diese eigene Konventionen, um Instanzen zu benennen.

Definition 31 (Äquivalenz-Matching-Funktion). Die einfachste Matching-Funktion ist die Äquivalenz von Attributwerten:

$$equivalence : Ind_L \times Ind_L \times Attr_L \times Attr_G \rightarrow \begin{cases} \text{true} \\ \text{false} \end{cases}$$

mit

$$\begin{aligned} equivalence(ind_L, ind_G, attr_L, attr_G) = \{true| \\ & \exists attrVal_L \in AttributeValue_L \wedge \\ & \exists attrVal_G \in AttributeValue_G : \\ & (ind_L, attrVal_L) \in hasValue_L \wedge \\ & (ind_G, attrVal_G) \in hasValue_G \wedge \\ & (attrVal_L, attr_L) \in References_L \wedge \\ & (attrVal_G, attr_G) \in References_G \wedge \\ & attrVal_L = attrVal_G\} \end{aligned}$$

Sind Attributwerte infolge manueller Eingabe in verschiedenen Informationssystemen nicht identisch, sondern enthalten Fehler (z.B. fehlendes Zeichen, Vertauschung von Zeichen), können Ähnlichkeits- und Distanzfunktionen verwendet werden, um Korrespondenzen zwischen Attributen zu bestimmen. Hierzu gehören auch *Edit-Distance*-Funktionen (z.B. *Levenshtein-Abstand*). Die Levenshtein-Matching-Funktion wird verwendet, um Tippfehler zu erkennen, die bei manueller Eingabe entstehen können.

Beispiel 20 (Levenshtein-Matching-Funktion). Der Levensthein-Abstand ist die minimale Anzahl an Änderungsoperationen $op_1 \ldots op_n$ (d.h. Einfügen, Löschen und Ersetzen von Zeichen $z_{11}, \ldots, z_{1n} \in Z$ einer Zeichenkette $w_1 \in W$), um eine Zeichenkette in eine andere Zeichenkette $w_2 = (z_{21}, \ldots, z_{1k}) \in Z$) mit $z_{21}, \ldots, z_{1k} \in Z$) zu überführen.
Die Hilfsfunktion $levenshtein - distance : W \times W \rightarrow \mathbb{N}$ liefert die minimale Anzahl der Änderungs-operationen, die nötig ist, um eine gegebene Zeichenkette in eine andere Zeichenkette zu transformieren. Auf Basis dieser Funktion lässt sich die Ähnlichkeitsfunktion $levenshtein - similarity : W \times W \rightarrow [0, 1]$ wie folgt definieren:

$$levenshtein - similarity(w_1, w_2) = 1 - \frac{levenshtein - distance(w_1, w_2)}{max(|w_1|, |w_2|)}$$

Schließlich definieren wird die Matching-Funktion
$levenshtein : Ind_L \times Ind_G \times Attr_L \times Attr_G \times t \rightarrow \{true, false\}$ für zwei Instanzen mit einer Schwelle t als eine reelle Zahl $r \in [0, 1]$. Die Matching-Funktion ist wahr, wenn die Ähnlichkeitsfunktion größer oder gleich einem Schwellwert t ist:

$$levenshtein(ind_{LPO}, ind_{GPO}, attr_{LPO}, attr_{GPO}, t) = \{true| \\ levenshtein - similarity(ind_{lpo}, ind_{gpo}) \geqslant t\}$$
$$levenshtein(ind_{LPO}, ind_{GPO}, attr_{LPO}, attr_{GPO}, t) = \{false| \\ levenshtein - similarity(ind_{lpo}, ind_{gpo}) < t\}$$

Zwei Zeichenketten korrespondieren zueinander, wenn eine Ähnlichkeitsfunktion einen zuvor definiert Schwellenwert überschreitet bzw. bei einer Distanzfunktion einen Wert unterschreitet. Abbildung 6.15 illustriert ein Beispiel für Instanzen einer lokalen und der globalen Produktontologie, die Tippfehler im Namen bzw. unterschiedliche Kodierungen für Umlaute enthalten. Korrespondenz zwischen diesen Instanzen können durch Verwendung der Levenshtein-Distanz für die Attribute *Name* und *Responsible* bestimmt werden. Im Detail muss dazu eine Schranke für die Ähnlichkeit ermittelt werden. Tabelle 6.4 illustriert den Levenshtein-Abstand bzw. -ähnlichkeit für die Instanzen aus $LocalProductOntologyA$ und $GlobalProductOntologyG$. Die Schranke zur Ermittlung von Korrespondenzen zwischen den beiden Instanzen, ist in diesem Beispiel 0.7142.

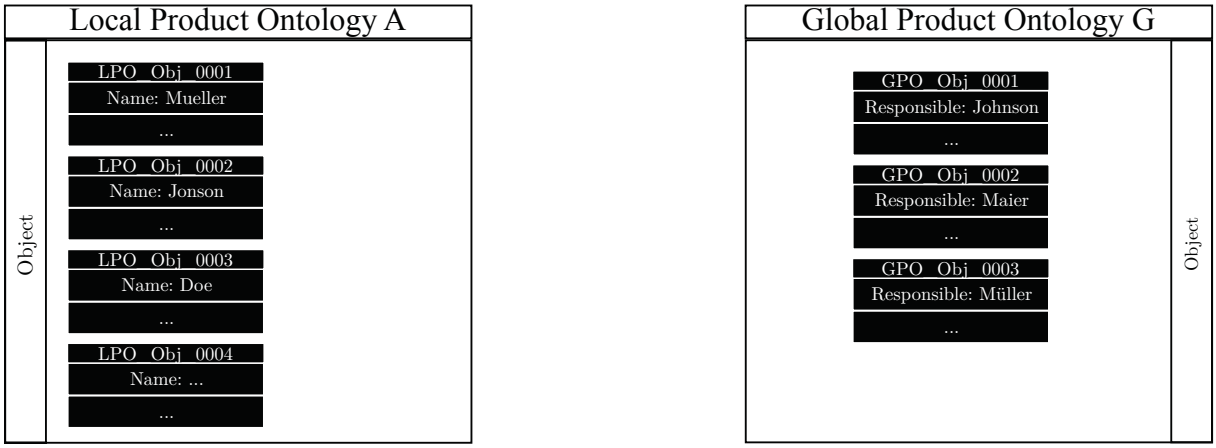


Abbildung 6.15.: Instanzen mit ähnlichen Attributwerten

LPO_A	GPO_G	**Levenshtein-Abstand**	**Levenshtein-Ähnlichkeit**
Mueller	Johnson	7	0.0
Jonson	**Johnson**	**1**	**0.8571**
Doe	Johnson	6	0.1428
Mueller	Maier	4	0.3333
Jonson	Maier	6	0.0
Doe	Maier	4	0.2
Mueller	**Müller**	**2**	**0.7142**
Jonson	Müller	6	0.0
Doe	Müller	5	0.1666

Tabelle 6.4.: Levenshtein-Abstand und -Ähnlichkeit

Bei wiederkehrenden Mustern zwischen den Attributen lokaler und globaler Schemakonzepte lassen sich Korrespondenzen zwischen Instanzen durch reguläre Ausdrücke bestimmen.

Attribut 1	**Attribut 2**
Version 1	V_1
Version 1.1	$V_{1.1}$
Version 2	V_2
Version 3	V_3

Tabelle 6.5.: Unterschiedliche Repräsentation von Attributwerten

Definition 32 (Regulärer-Ausdruck-Matching-Funktion). Durch Gruppierungen und Rückwärtsreferenzen lassen sich wiederkehrende Muster mit Hilfe regulärer Ausdrücke realisieren. Damit Korrespondenzen zwischen Attributwerten aus einer lokalen und der globalen Produktontologie mit Hilfe regulärer Ausdrücke bestimmt werden können, müssen die zu vergleichende Attributwertpaaren konkateniert werden. Um eine eindeutige Trennung zwischen den beiden Attributwerten zu gewährleisten, muss eine Trennsequenz vereinbart werden, die in keinem der beiden Attribute vorkommt.

$$regularExpression : Ind_{LPO} \times Ind_{GPO} \times Attr_{LPO} \times Attr_{GPO} \times RegEx \rightarrow \begin{cases} \text{true} \\ \text{false} \end{cases}$$

wobei $RegEx$ ein regulärer Ausdruck ist

$$regularExpression(ind_{LPO}, ind_{GPO}, regex) = \{true| \\ regex(ind_{LPO}, ind_{GPO}) = true\}$$
$$regularExpression(ind_{LPO}, ind_{GPO}, regex) = \{false| \\ regex(ind_{LPO}, ind_{GPO}) = false\}$$

Neben der Definition regulärer Ausdrücke können auch Funktionen definiert werden, welche Artefakte matchen, die in proprietären Formaten vorliegen. Lassen sich Korrespondenzen zwischen Instanzen weder mittels Identität, Ähnlichkeit, regulärer Ausdrücken oder proprietären Matching-Funktionen von Attributen bestimmen, müssen Abbildungstabellen sukzessive manuell aufgebaut werden.

Definition 33 (Mapping-Tabelle-Matching-Funktion). Eine Mapping-Tabelle enthält manuell gepflegte Korrespondenzbeziehungen zwischen Attributwerten.

$$map : Ind_{LPO} \times Ind_{GPO} \times Attr_{LPO} \times Attr_{GPO} \times Map \rightarrow \begin{cases} \text{true} \\ \text{false} \end{cases}$$

mit $(map_{LPO}, map_{GPO}) \in Map$ wobei $map_{LPO} \in Ind_{LPO}$ und $map_{GPO} \in Ind_{GPO}$ mit

$$map(ind_{LPO}, ind_{GPO}, attr_{LPO}, attr_{GPO}, Map) = \{true| \\ \exists(map_{LPO}, map_{GPO}) \in Map : map_{LPO} = ind_{LPO} \wedge map_{GPO} = ind_{GPO})\}$$
$$map(ind_{LPO}, ind_{GPO}, attr_{LPO}, attr_{GPO}, Map) = \{false| \\ \nexists(map_{LPO}, map_{GPO}) \in Map : map_{LPO} = ind_{LPO} \wedge map_{GPO} = ind_{GPO})\}$$

Produktdaten und deren Attribute werden im Entwicklungsprozess zu unterschiedlichen Zeitpunkten in unterschiedlicher Qualität dokumentiert. Am Anfang der Produktentwicklung werden zunächst nur einige Attribute dokumentiert, so dass es notwendig werden kann, mehr als nur eine Attribut-Matching-Regel zwischen einem lokalen und derem globalen Schemakonzept zu definieren.
In Abbildung 6.16 sind Schemakonzepte und Instanzen einer lokalen Produktontologien sowie einer globalen Produktontologie dargestellt. Der Einfachheit halber werden in diesem Fall nur die obersten beiden Produktdatenintegrationsebenen mit jeweils nur einer Instanz pro Schemakonzept betrachtet. Die lokale Produktontologie verwaltet Anforderungen an Komponenten (*Komponentenlastenheft*) für Fahrzeuge (*Fahrzeuganforderungen*).
Ein Komponentenlastenheft besitzt einen *Komponentennamen* sowie einen *Verantwortlichen*. Fahrzeuganforderungen sind für ein Fahrzeug (*Name*) zusammengefasst.
Die globale Produktontologie beschreibt alle Produktdaten von Komponenten (*Bauteil*) für Fahrzeuge (*Fahrzeug*). Ein Bauteil besitzt einen *Namen*, eine *Nr* und einen *Verantwortlichen*.

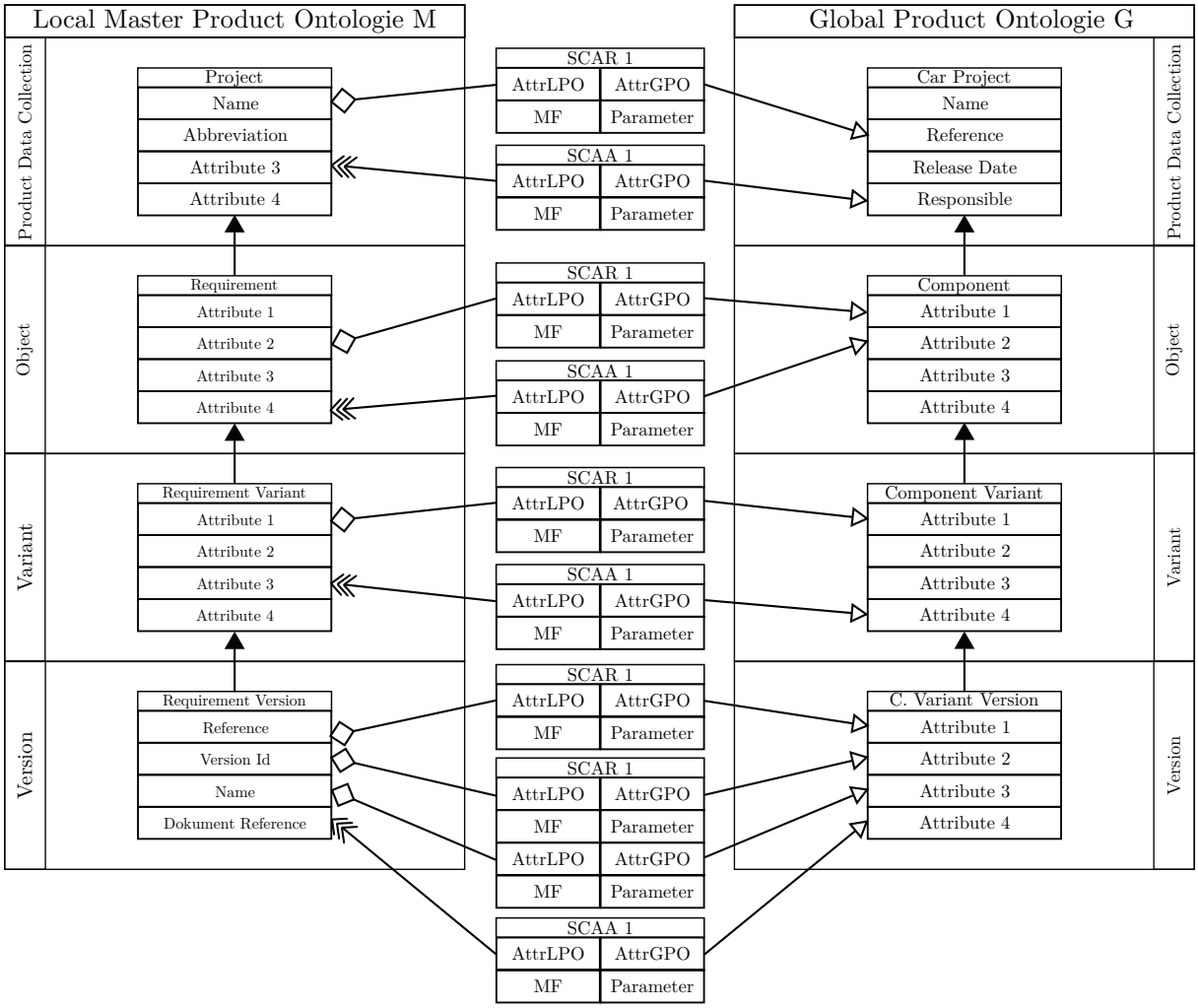


Abbildung 6.16.: Abbildung von Schemakonzepten zwischen lokaler und globaler Produktontologie

Für jedes Schemakonzept der lokalen Produktontologie ist genau eine Schemakonzept-Attributregel in die globale Produktontologie definiert. *SCAR 1* beschreibt die Abbildung von *Fahrzeuganforderungen* auf *Fahrzeuge*. Instanzen beider Schemakonzepte korrespondieren zueinander, genau dann, wenn der *Name* identisch ist. Analog beschreibt *SCAR 2* die Abbildung von *Komponentenlastenheft* auf *Bauteil*. Auch hier ist die Matching-Funktion als Identität der Attribute *Komponentenname* und *Name* definiert.
Nachdem Schemakonzept-Attributregeln für jedes Schemakonzept einer lokalen Produktontologie definiert worden sind, kann letztere in die globale Produktontologie automatisch integriert werden (vgl. Abbildung 6.17). Dabei werden die Regeln, beginnend mit dem Schemakonzept der obersten Produktdatenintegationsebenen für ihrer Instanzen paarweise mit Hilfe der Matching-Funktion ausgewertet. Für jedes Instanzpaar (hier: (L1_1, G_1), (L1_1, G_2), (L1_1, G_3), (L1_2, G_2), (L1_2, G_2), (L2_1, G_3)) aus lokaler und globaler Produktontologie, welche die Matching-Funktion erfüllen, wird eine zugehörige Korrespondenz erzeugt.

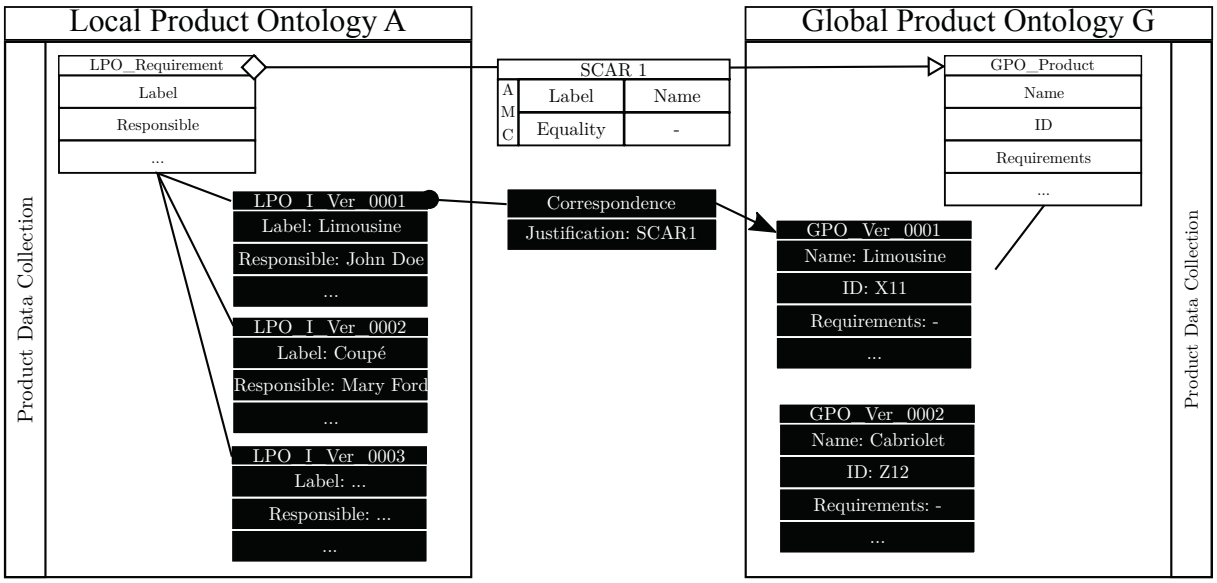


Abbildung 6.17.: Auswertung einer SCAR

Schemakonzept-Attributaktionen

Während eine Schemakonzept-Attributregeln Schemakonzepte aus lokaler und globaler Produktontologie zueinander in Relation stellen, definieren *Schemakonzept-Attributaktionen* (engl. *schema concept attribute action* (SCAA)) diejenigen Attribute aus einer lokalen Produktontologie, die in eine globale Produktontologie integriert werden sollen. Schemakonzept-Attributaktionen werden, ebenso wie SCARs, paarweise für lokale und globale Schemakonzepte definiert. Zwischen einem lokalen und globalen Schemakonzept lassen sich nur dann SCAAs definieren, wenn für sie mindestens eine SCAR definiert ist.

Die globale Produktontologie bietet eine holistische Sicht auf Produktdaten. Dazu können Attribute aus lokalen Produktontologien in die globale Produktontologie integriert bzw. kopiert werden. Attributwerte lokaler Produktontologien werden in die globale Produktontologie integriert, um die Komplexität der Informationen zum Realisieren von Anwendungsfällen zu reduzieren.

Um automatisch ermitteln zu können, wann die Abbildung einer lokalen Produktontologie in die globale Produktontologie abgeschlossen ist, werden die Abbildungsregeln, und besonders die Attribute der Schemakonzepte, betrachtet. Für jedes Schemakonzept muss mindestens eine SCAR vorhanden sein. Damit festgestellt werden kann, ob keine weiteren Regeln und Aktionen benötigt werden, müssen die Integrationsexperten alle nicht benötigten Attribute der Schemakonzepte der lokalen Produktontologie markieren. Nur so kann sichergestellt werden, dass kein Attribut bei der Erstellung der Abbildung übersehen wird.

Schemakonzepte lokaler Produktontologien können mehr Attribute enthalten, als für einen Anwendungsfall notwendig sind. Durch die Definition von SCAAs können die benötigten Attribute der lokalen Produktontologien in der globalen Produktontologie übersichtlich zusammengefasst werden. Daraus resultiert ein schnellerer Zugriff auf die benötigten Informationen. Ohne die Definition von SCAAs müssten mit Hilfe der Korrespondenzen Informationen aus lokalen Produktontologien ermittelt werden. Durch die Korrespondenzen zwischen Instanzen aus lokaler und globaler Produktontologie stellen Attributwerte von Schemakonzepten, die auf Grund von

SCAAs erzeugt werden, redundante Informationen dar. Folglich sind sie optional.
Während der automatischen Integration werden definierte SCAAs für ermittelte Korrespondenzen ausgeführt: Wird für eine Instanz `A` eines Schemakonzeptes SC_A einer lokalen Produktontologie eine korrespondierende Instanz `B` eines Schemakonzeptes SC_B aus der globalen Produktontologie ermittelt, werden alle SCAAs ausgeführt, die zwischen den Schemakonzepten von `A` und `B` definiert sind. Soll in diesem Schritt ein Attributwert von `A` für ein Attribut $Attr_A$ nach $Attr_B$ in `B` kopiert werden, kann ein Integrationskonflikt vorliegen, wenn ein entsprechender Attributwert in `B` vorhanden ist. Dafür kann es unterschiedliche Gründe geben:

- **Fall 1:** Es wurde bereits eine andere lokale Produktontologie integriert, die für dasselbe Schemakonzept SC_B und dasselbe Attribut $Attr_B$ eine SCAA definiert hat. Weiter wurde eine Korrespondenz zwischen einer Instanz dieser lokalen Produktontologie und `B` identifiziert.
- **Fall 2:** Neben `A` sind für andere Instanzen bereits Korrespondenzen derselben lokalen Produktontologie identifiziert und damit auch die selbe SCAA ausgeführt worden.
- **Fall 3:** Der Wert für das Attribut $Attr_B$ in `B` wurde manuell und unabhängig von einer Integration in der globalen Produktontologie dokumentiert.

Unterscheiden sich die Attributwerte, die in die globale Produktontologie kopiert werden sollen, entsteht ein Integrationskonflikt. Folglich muss für eine SCAA, ein Integrationsverhalten angegeben werden, auf dessen Grundlage sich ein solcher Integrationskonflikt auflösen lässt. Zur Lösung von Integrationskonflikten sind verschiedene Vorgehensweisen möglich:

- **Integrationsverhalten 1 (append):** Der bereits festliegende Attributwert der Instanz in der globalen Produktontologie bleibt erhalten und der zu integrierende Attributwert der Instanz aus der lokalen Produktontologie wird in Form einer Liste angehängt.
- **Integrationsverhalten 2 (recent):** Sind zu den Attributwerten in der lokalen und globalen Produktontologie zusätzliche Metainformationen, wie z.B. der letzte Änderungszeitpunkt, vorhanden, kann diese Information verwendet werden, den gültigen Attributwert aus der lokalen und globalen Produktontologie zu ermitteln.
- **Integrationsverhalten 3 (interactive):** Die automatische Integration kann unterbrochen werden, um den Integrationskonflikt manuell aufzulösen.
- **Integrationsverhalten 4 (local):** Der existierende Attributwert wird durch den Wert aus der lokalen Produktontologie überschrieben.
- **Integrationsverhalten 5 (global):** Der existierende Attributwert der globalen Produktontologie wird beibehalten, solange dieser gesetzt ist. Ansonsten wird der Attributwert der lokalen Produktontologie übernommen.

Einerseits können SCAAs verwendet werden, um Attribute von Produktdaten in die globale Produktontologie zu integrieren und somit eine holistische Sicht auf ein Produkt herzustellen. Andererseits kann durch Definition von SCAAs auch die Konsistenz von Attributen, die über mehrere Informationssysteme und damit auch über mehrere lokalen Produktontologien verteilt sind, sichergestellt werden. Dazu werden für ein Attribut der globalen Produktontologie SCAAs auf

Attribute aus mehreren lokalen Produktontologien definiert. Gleichzeitig wird für diese SCAAs das Integrationsverhalten 1 oder 2 gewählt.
Abbildung 6.18 zeigt ein Beispiel für ein solches Szenario. Illustriert sind die Instanzen dreier lokaler Produktontologien und einer globalen Produktontologie der selben Integrationsebene. Ferner seien Schemakonzepte in allen drei lokalen Produktontologien vorhanden, für die jeweils SCAAs definiert sind.[5] Für jede SCAA führt das Integrationsverhalten zur Bildung von Listen für die Werte des Attributs *Name* in das korrespondierende Attribut *Names* für Instanzen in der globalen Produktontologie. Nach Integration einer lokalen Produktontologie in die globale Produktontologie können Inkonsistenzen ermittelt werden, indem alle Instanzen der globalen Produktontologie bestimmt werden, für die das Attribut *Names* eine Liste mit mehreren Einträgen besitzt.

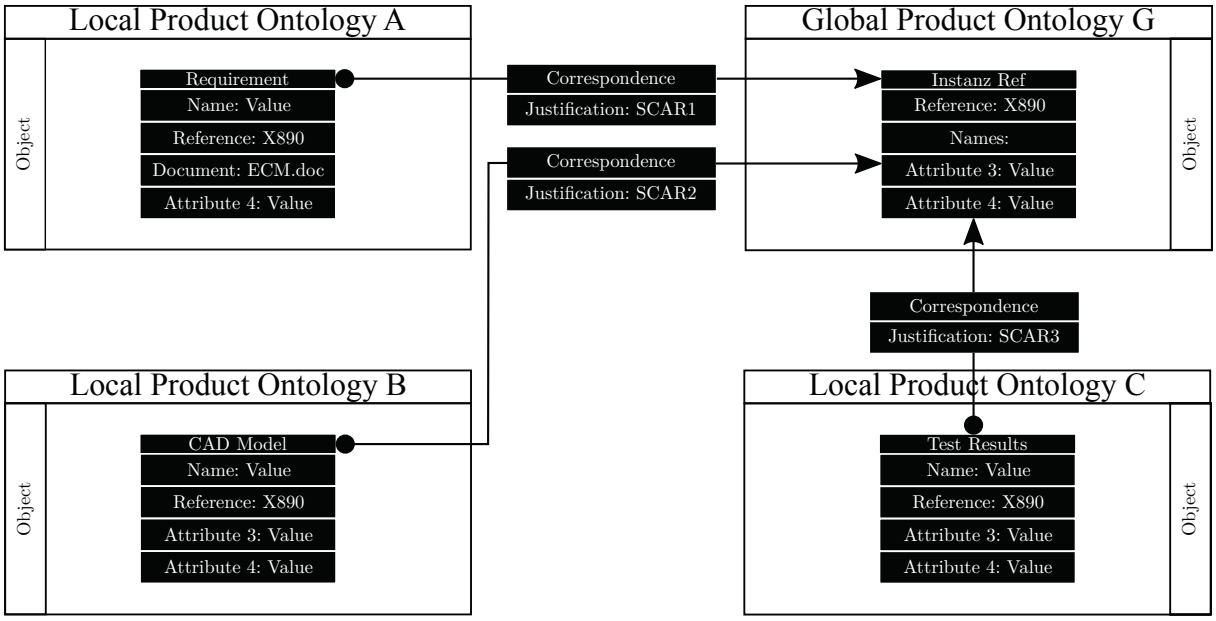


Abbildung 6.18.: Konsistenzüberprüfung durch SCAAs

Ein Beispiel für strukturelle Heterogenität sind Adressinformationen: Vor- und Familienname, Straße, Hausnummer, Postleitzahl und Ort können in einem konzeptuellen Datenmodell sowohl jeweils als separate Attribute oder zusammengesetzt modelliert sein. Unterschiedliche strukturelle Heterogenität muss bei der Abbildung lokaler Informationssysteme in lokale Produktontologien berücksichtigt werden. SCAAs referenzieren nur ein Quell- und ein Zielattribut. Es wäre andererseits möglich Aggregationsfunktionen zu definieren, die mehrere Attribute lokaler Schemakonzepte zu einem Attribut vereinen. Ein Beispiel für eine solche Aggregationsfunktion ist die Konkatenation von Zeichenketten.
Für dasselbe Schemakonzept einer lokalen Produktontologie können mehrere SCAAs definiert sein. Diese dürfen nicht auf dasselbe Attribut in der globalen Produktontologie verweisen, da es sonst zu *Lost-Update*-Problemen kommen kann.

[5] Aus Gründen der Übersichtlichkeit sind diese SCAAs nicht dargestellt.

Definition 34 (Attribute-Aktion). Eine Attribut-Aktion ist ein geordnetes Tripel bestehend aus einem Attribut einer lokalen Produktontologie $attr_L$, einem Attribut einer globalen Produktontologie $attr_G$ sowie einem Integrationsverhalten *behavior*:

$$(attr_L, attr_G, behavior) \in AttributeAction$$

mit

- $attr_L \in Attr_L$ ist ein Attribut eines Schemakonzeptes einer lokalen Produktontologie,
- $attr_G \in Attr_G$ ist ein Attribut eines Schemakonzeptes einer globalen Produktontologie,
- $behavior \in \{append, recent, interactive, local, global\}$ ist das Integrationsverhalten, welches bei Integrationskonflikten angewendet wird.

Zusammengefasst definieren wir Schemakonzept-Attributaktionen wie folgt:

Definition 35 (Schemakonzept-Attributaktion (SCAA)). Für eine lokale Produktontologie PO_L und die globale Produktontologie PO_G sind die Schemakonzept-Attributaktionen $(sc_L, sc_G, aa_1, \ldots, aa_n) \in SCAA$ eine Menge von geordneten Elementen mit $sc_L \in SC_L$ und $sc_G \in SC_G$. Es gelten die folgenden Eigenschaften:

- $sc_L \in SC_L$ ist ein Schemakonzept aus einer lokalen Produktontologie,
- $sc_G \in SC_G$ ist ein Schemakonzept aus der globalen Produktontologie,
- $(aa_1, \ldots, aa_n) \in AttributeAction$ ist die Menge der Attribut-Aktionen.

Analog zu Korrespondenzen muss das Zustandekommen von Attributwerten in der globalen Produktontologie nachvollziehbar sein. So wird für jeden Attributwert einer Instanz der globalen Produktontologie die SCAA dokumentiert (inkl. Zeitpunkt), die für den Wert verantwortlich ist. Da Attributwerte auch manuell gepflegt werden können, wird zwischen automatischen und manuellen Attributwerten unterschieden. Diese *Attributwert-Begründungen* (engl. *attribute value justification*) definieren wir wie folgt:

Definition 36 (Attributwert-Begründung). Für einen Attributwert $attrVal_G$ einer Instanz einer globalen Produktontologie ist die Attributwert-Begründung (engl. *attribute value justification*) eine Menge $AttrValJustification$ von geordneten Paaren, wobei *scaa* eine Schemakonzept-Attributaktion ist:

$$(attrVal_G, scaa) \in AttrValJustification$$

Zusammengefasst beschreibt für eine lokale Produktontologie PO_L und eine globale Produktontologie PO_G die Menge der Schemakonzept-Attributregeln und -aktionen das sog. *Produktontologie-Mapping.*

Definition 37 (Produktontologie-Mapping). Formal ist das Produktontologie-Mapping zwischen einer lokalen und globalen Produktontologie ein Tupel $POMapping_{L\times G} := (PO_L, PO_G, SCAR, SCAA)$ mit:

- PO_L ist eine lokale Produktontologie,
- PO_G ist die globale Produktontologie,
- $SCAR$ ist die Menge der Schemakonzept-Attributregeln für Abbildungen zwischen PO_L und PO_G,
- $SCAA$ ist die Menge der Schemakonzept-Attributaktionen für Abbildungen zwischen PO_L und PO_G.

Die SCARs eines Produktontologie-Mappings können automatisch ausgewertet werden. Im Detail muss für jedes Schemakonzept der lokalen Produktontologie eine Schemakonzept-Attributre-gel in die globale Produktontologie definiert sein:

Definition 38 (Produktontologie-Mapping Vollständigkeit). Ein Produktontologie-Mapping $POMapping_{L\times G} := (PO_L, PO_G, SCAR, SCAA)$ zwischen einer lokalen und globalen Produktontologie ist genau dann vollständig wenn: $\forall sc \in SC_L : \exists(sc, sc_G, AMC) \in SCAR$

Lokale Masterproduktontologie

Nach Darstellung der Regeln und Aktionen zur Abbildung von lokaler Produktontologie auf die globale Produktontologie, wird im Folgenden erläutert, wie Schemakonzepte der globalen Produktontologie modelliert und mit Instanzen versorgt werden können.
Da die globale Produktontologie eine holistische Sicht auf mehrere Produktdaten bietet, muss sie alle benötigten Instanzen für Bauteile umfassen, die zur Realisierung von Anwendungsfällen notwendig sind. Anders ausgedrückt müssen die Instanzen der Schemakonzepte *vollständig* sein. Damit diese Instanzen nicht manuell in der globalen Produktontologie gepflegt werden müssen, ist es sinnvoll, sie aus einem existierenden Informationssystem zu versorgen. Zentrales Informationssystem der Produktentwicklung ist häufig ein PDM-System, das alle Bauteile und deren Beziehungen untereinander verwaltet (siehe Abschnitt 2.2.2). Ein PDM-System kann genutzt werden, um eine globale Produktontologie mit Instanzen zu versorgen. Dazu wird eine lokale Produktontologie erstellt, die als lokale *Masterproduktontologie (MPO)* bezeichnet wird.
Die lokale Masterproduktontologie wird als Referenz für Schemakonzepte in der globalen Produktontologie herangezogen. Die Daten aus dem zugrundeliegenden Informationssystem sind vollständig und konsistent, sie enthalten alle zu integrierenden Instanzen für ein Schemakonzept. Alle Änderungen an Instanzen der Masterproduktontologie werden direkt in die globale Produktontologie übernommen.
Abbildung 6.19 illustriert die 1:1-Abbildung zwischen lokaler Masterproduktontologie und globaler Produktontologie. Jede Integrationsebene enthält genau ein Schemakonzept mit den Attributen *Name* und *ID*, um Instanzen zwischen lokaler und globaler Produktontologie eindeutig zuordnen zu können. Für jedes Schemakonzept sind mindestens eine Schemakonzept-Attributregel und

eine Schemakonzept-Attributaktion definiert. Für jede Instanz der lokalen Masterproduktontologie existiert eine korrespondierende Instanz in der globalen Produktontologie mit jeweiligen Korrespondenzen.
In der Praxis werden unterschiedliche Produktdaten in verschiedenen Informationssystemen dokumentiert. Nachdem mit Hilfe einer lokalen Masterproduktontologie die grundlegenden Schemakonzepte und Instanzen in der globalen Produktontologie abgebildet worden sind, können weitere lokale Produktontologien integriert werden, um so alle notwendigen Produktdaten zu integrieren. Anschließend können Schemakonzepte der globalen Produktontologie um Attribute erweitert werden, die Informationen über verschiedene Produktdaten speichern können, welche bei der Integration durch Schemakonzept-Attributregeln versorgt werden.
In Abbildung 6.19 ist eine lokale Masterproduktontologie abgebildet. Aus Gründen der Übersichtlichkeit, werden nur zwei Integrationsebenen dargestellt. In beiden ist jeweils ein Schemakonzept mit Attributen sowie einer Menge an zugehörigen Instanzen dargestellt. Die Schemakonzepte der beiden Ebenen werden in die globale Produktontologie kopiert und über SCARs aufeinander abgebildet. Im Detail besitzt jedes Schemakonzept ein Attribut, das Instanzen eindeutig beschreibt (z.B. eine global gültige Identifikationsnummer). Für jede Instanz der Schemakonzepte wird in der globalen Produktontologie eine korrespondierende Instanz kopiert und über Korrespondenzen miteinander verbunden. Kommen Instanzen in der lokalen Masterproduktontologie hinzu, werden diese automatisch in die globale Produktontologie kopiert und entsprechende Korrespondenzen angelegt. Analog wird bei der Änderung oder dem Entfernen von Instanzen verfahren.

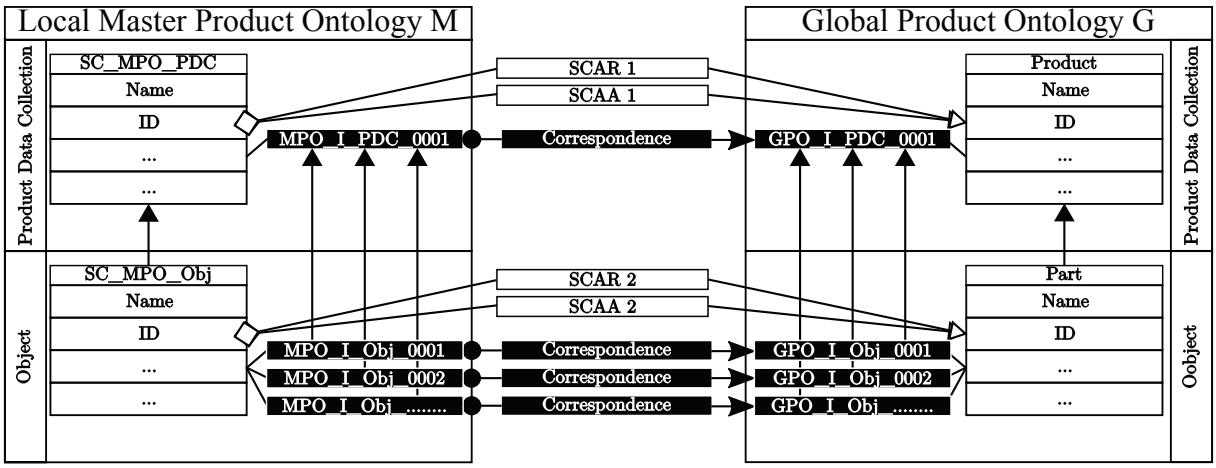


Abbildung 6.19.: Lokale Masterproduktontologie

Wir definieren eine lokale Masterproduktontologie wie folgt:

Definition 39 (Lokale Masterproduktontologie). Für ein Produktontologie-Mapping $POMapping_{M\times G} := (PO_M, PO_G, SCAR, SCAA)$, bestehend aus einer globalen Produktontologie PO_G sowie einer lokalen Masterproduktontologie PO_M, gelten folgende strukturelle Konsistenzbedingungen:

- Jeder Produktdatenintegrationsebene enthält mindestens ein Schemakonzept mit entsprechenden Korrespondenzen $sc_1, sc_2, sc_3, sc_4 \in SC_M$, mit $(sc_1, sc_2), (sc_2, sc_3)$,

$(sc_3, sc_4) \in scAbove_M$

- $((pdc, sc_1), (obj, sc_2), (var, sc_3), (ver, sc_4)) \in hasConcept_M$

- Jedes Schemakonzept ist über eine Schemakonzept-Attributregel in die globale Produktontologie abgebildet $\forall sc \in SC_M \exists (sc, sc_G, AMC) \in SCAR$ mit $sc_G \in SC_G$
- Für jede Instanz von Schemakonzepten, die in die globale Produktontologie abgebildet sind, existiert eine korrespondierende Instanz in der globalen Produktontologie mit einer entsprechenden Korrespondenz:
  $\forall ind_M \in Ind_M$ und $\forall sc \in SC_M$ für die gilt: $\exists (sc, sc_G, AMC) \in SCAR$ und $(sc, ind_M) \in hasIndividual_M$ folgt: $\exists ind_G \in Ind_G$ und $\exists (ind_M, ind_G) \in Correspondence$

Integrationsauschluss

Enthält eine Produktontologie mehr Informationen als für eine Integration erforderlich, können Schemakonzepte und Instanzen von einer Integration ausgeschlossen werden. Umfasst beispielsweise ein PDM-System alle Produkte eines Unternehmens, können zur Realisierung eines Anwendungsfalls die Produkte, und damit Instanzen entsprechender lokaler Produktontologien, von einer Integration ausgeschlossen werden. Der Ausschluss von Schemakonzepten und Instanzen erfolgt manuell durch Integrationsexperten und wird im Produktontologie-Mapping (vgl. Definition 38) gespeichert. Je nach Anwendungsfall können auf diese Weise unterschiedliche Menge definiert werden, die von einer Integration ausgeschlossen werden sollen. Dazu wird die Definition des Produktontologie-Mappings wie folgt erweitert:

Definition 40 (Produktontologie-Mapping mit Ausschluss). Für ein Produktontologie-Mapping $POMapping_{L \times G} := (PO_L, PO_G, SCAR, SCAA)$ können für lokale Produktontologie PO_L und globale Produktontologie PO_G die Mengen SC_X^{Ignore} und Ind_x^{Ignore} definiert werden mit:

- SC_X^{Ignore} ist die Menge von Schemakonzepten,
- Ind_X^{Ignore} ist die Menge der Instanzen der Produktontologie PO_X mit $X \in L, G$, die für eine Integration nicht berücksichtigt werden.

Dadurch ergeben sich folgende Mengen für ein Produktontologie-Mapping, welches den Ausschluss von Schemakonzepten und Instanzen berücksichtigt:
$POMapping_{L \times G}^{Ignore} := (PO_L, PO_G, SCAR, SCAA, SC_L^{Ignore}, Ind_L^{Ignore}, SC_G^{Ignore}, Ind_G^{Ignore})$
mit

- $SC_L^{Ignore} \subset PO_L.SC_L$
- $Ind_L^{Ignore} \subset PO_L.Ind_L$
- $SC_G^{Ignore} \subset PO_G.SC_G$
- $Ind_G^{Ignore} \subset PO_G.Ind_G$

6.2.4. Methodiken zur Erstellung lokaler und globaler Produktontologien

Die Modellierung lokaler und globaler Produktontologien können unabhängig voneinander geschehen. Um die Integration von Schemakonzepten und den entsprechenden Instanzen zu vereinfachen, ist es sinnvoll, die Modellierung untereinander abzustimmen. Darunter fallen z.B. die Benennung von Schemakonzepten und Attributen. Während Variabilitätsmechanismen und Versionierung der lokalen Produktontologien durch die zugrundeliegenden Informationssysteme definiert sind, können diese in der globalen Produktontologie für jedes Schemakonzept der entsprechenden Ebenen frei definiert werden.
Die Modellierung lokaler Produktontologien und der globalen Produktontologie kann auf drei verschiedene Arten erfolgen. Werden Schemakonzepte in beiden definiert, ohne die Datenmodelle der vorhandenen Informationssysteme zu berücksichtigen, sprechen wir im weiteren Verlauf der Arbeit von einer *Top-Down*-Methodik. Im umgekehrten Fall werden zunächst die Datenmodelle existierender Informationssysteme auf Gemeinsamkeiten analysiert (*Bottom-Up*-Methodik). Schließlich können beide Ansätze kombiniert werden und sowohl Schemakonzepte in einer globalen Produktontologie vorgegeben als auch gemeinsame aus den lokalen Informationssystemen abgeleitet werden (*Mixed*-Methodik).

Top-Down-Vorgehen

Bei der Top-Down-Methodik werden zunächst Schemakonzepte der globalen Produktontologie, unabhängig von bestehenden Schemakonzepten, aus den zu integrierenden Informationssystemen definiert (siehe ① in Abbildung 6.20). Dazu werden sowohl Metaattribute, wie zum Beispiel der Name des Schemakonzeptes oder eine Beschreibung modelliert (②). Im Speziellen werden für die Schemakonzepte der Variant- und Version-Ebene die entsprechenden Variabilitäts- und Versionierungstypen dokumentiert (vgl. ③ und ④). Auf Basis dieser globalen Produktontologie werden anschließend die lokalen Produktontologien (siehe ⑤) sowie Abbildungen (⑥) zwischen lokalen Produktontologien und der globalen Produktontologie spezifiziert.
Vorteil des Top-Down-Ansatzes ist es, dass die Begriffe für Schemakonzepte vereinheitlicht werden können, indem bei der Modellierung lokaler Produktontologien jeweils Bezug auf die globale Produktontologie genommen wird. Da die globale Produktontologie zur Realisierung fachlicher Anwendungsfälle dienen soll, sind die benötigten Informationen bereits bekannt. Neben der Benennung von Schemakonzepten kann auch die Granularität von Attributen aus der globalen Produktontologie berücksichtigt werden. Dem gegenüber steht das Problem, dass bei der Modellierung der globalen Produktontologie möglicherweise Attribute aus lokalen Informationssystemen unberücksichtigt bleiben. Unterscheiden sich Schemakonzepte aus lokalen Informationssystemen zu sehr von denen in der globalen Produktontologie, sind komplexe Transformationen notwendig.

Bottom-Up-Vorgehen

Beim Bottom-Up-Ansatz werden die Schemakonzepte der lokalen Produktontologien, unabhängig von der globalen Produktontologie, modelliert (siehe ① in Abbildung 6.21). Dies bedeutet, dass die Bezeichnung von Schemakonzepten beliebig gewählt sein können. Gleiches gilt für die Abbildung von Attributen aus den lokalen Informationssystemen in die lokalen Produktontologien. Möglicherweise ist die gewählte Granularität dieser Attribute nicht passend für eine Integration in eine globale Produktontologie.

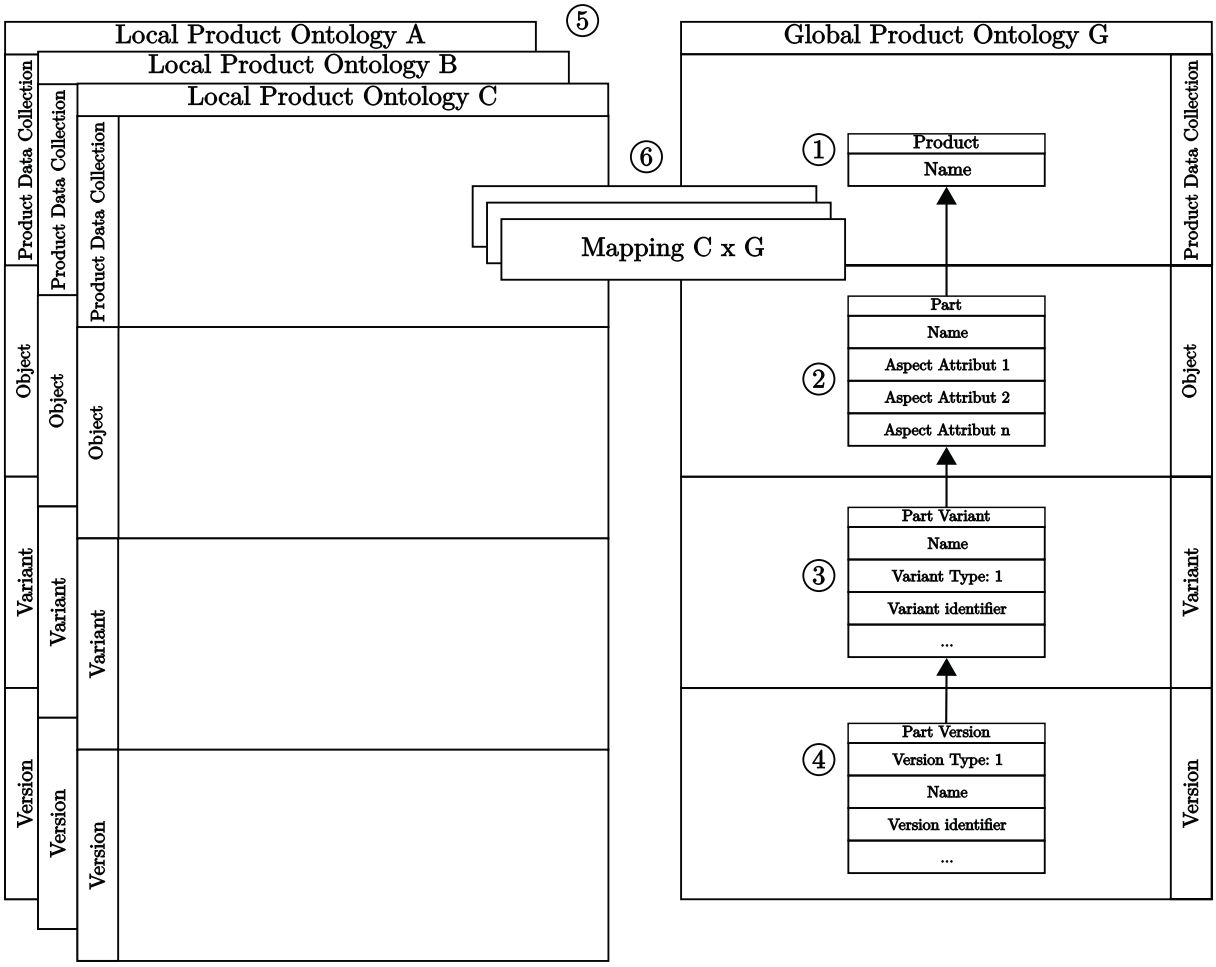


Abbildung 6.20.: Top-Down-Vorgehen

Nachdem die lokalen Produktontologien modelliert sind, werden die Schemakonzepte der einzelnen Produktdatenintegrationsebenen auf Gemeinsamkeiten hin analysiert und entsprechende Schemakonzepte in der globalen Produktontologie definiert (vgl. ② bis ⑤).
Die Vorteile beim Bottom-Up-Vorgehen sind, dass die lokalen Produktontologien unabhängig von einer globalen Produktontologie modelliert werden können und somit nicht von dieser abhängen. Nachteilig ist der Integrationsaufwand, der durch die potenzielle Heterogenität zwischen mehreren lokalen Produktontologien entsteht.

Hybrides Vorgehen

Als dritte Option kommt eine Variante in Betracht, die Top-Down- und Bottom-Up-Vorgehen vereint. Dabei werden essentielle Schemakonzepte und Attribute in einem Top-Down-Vorgehen in der globalen Produktontologie vorgegeben. Anschließend werden die Schemakonzepte lokaler Produktontologien auf diese vorgegebenen Schemakonzepte abgebildet. Schließlich können Schemakonzepte der globalen Produktontologie bei der Analyse lokaler Produktontologien um Attribute erweitert werden (Bottom-Up-Vorgehen).
Dieser Ansatz vereint die Vorteile von Bottom-Up und Top-Down. Abbildung 6.22 zeigt zwei

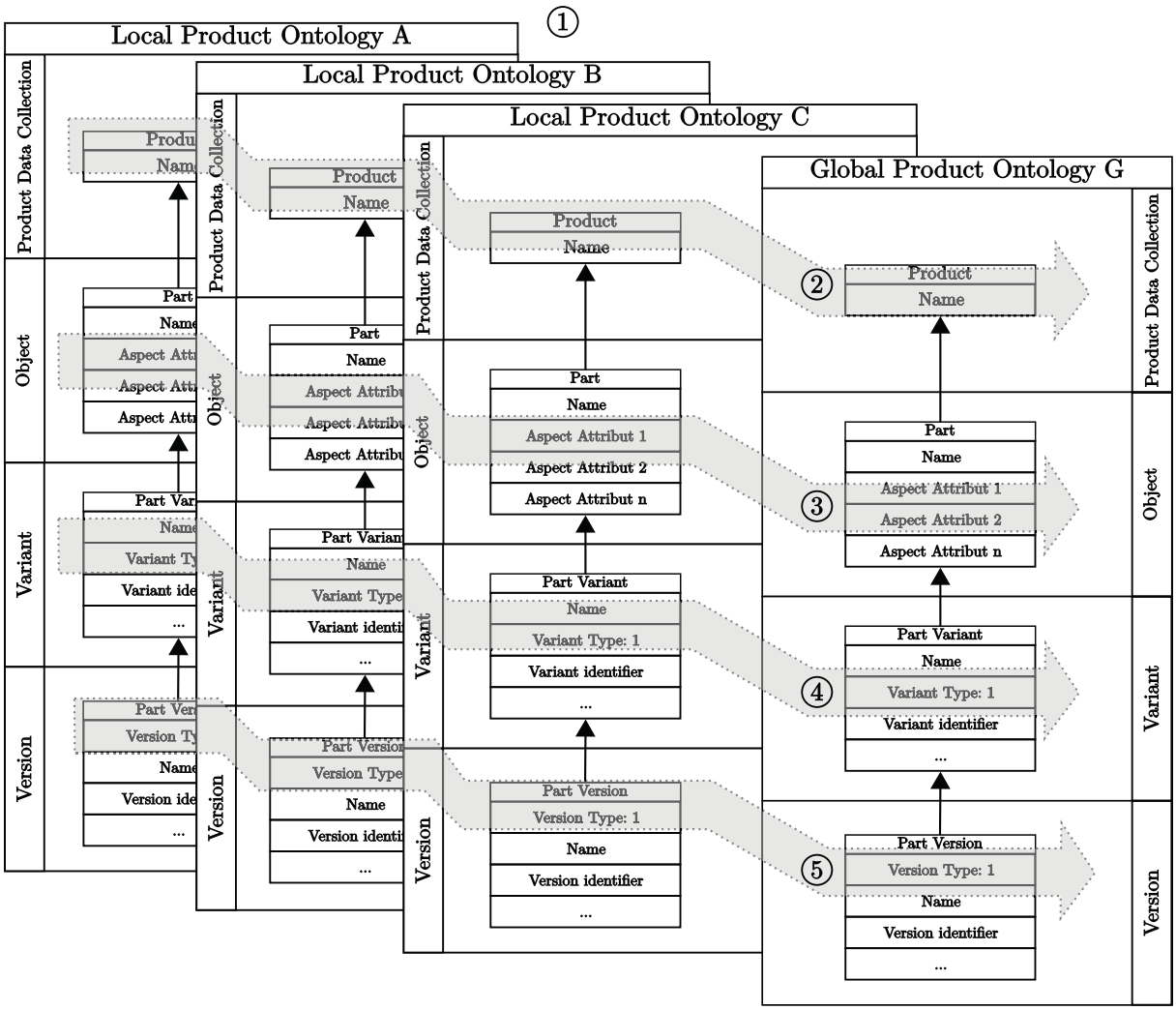


Abbildung 6.21.: Bottom-Up-Vorgehen

lokale Produktontologien A und B, für die einige Schemakonzepte in eine globale Produktontologie G abgebildet worden sind. Zunächst wurden Schemakonzepte in G modelliert (siehe ①), anschließend wurden Produktontologien erstellt (②). Ein Schemakonzept aus B wird in G abgebildet (③), bevor ein weiteres Schemakonzept von B nach G abgebildet wird (④). Dann wird das Schemakonzept in G um ein Attribut aus B erweitert. Da bisher auf das Schemakonzept aus G kein weiteres Schemakonzept einer anderen lokalen Produktontologie abgebildet worden ist, hat diese Änderung der globalen Produktontologie keine weiteren Auswirkungen. Anders verhält es sich im Fall ⑥, bei dem das Schemakonzept aus G um ein weiteres Attribut aus A erweitert wird. Hier müssen alle Schemakonzepte lokaler Produktontologien, die bereits auf dasselbe Schemakonzept aus G abgebildet worden sind, daraufhin untersucht werden, ob sie das neue Attribut ebenfalls verwenden.

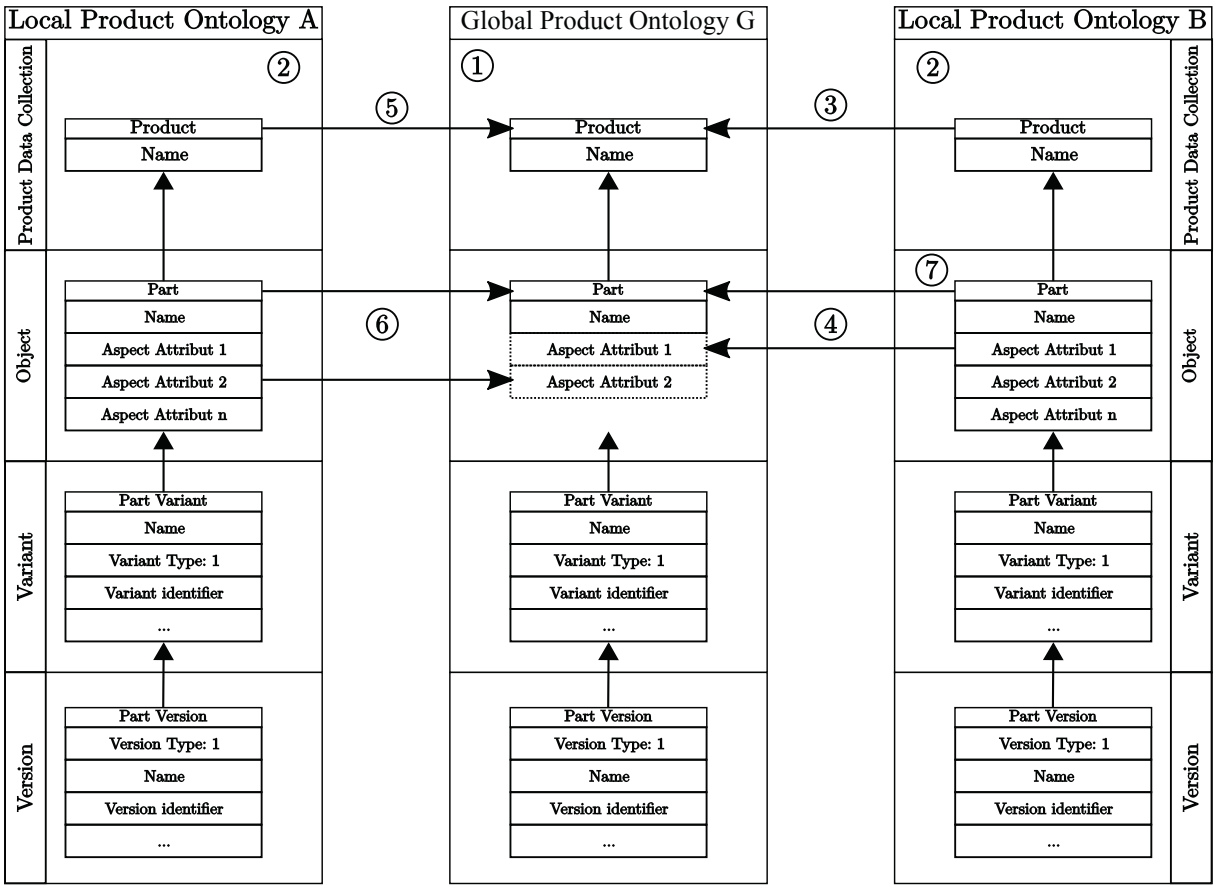


Abbildung 6.22.: Hybrides Vorgehen

6.2.5. Initiales Integrationsprojekt

Im Folgenden wird die Erstellung eines Integrationsprojektes erläutert. Für einen Anwendungsfall werden beispielhaft zwei lokale Informationssysteme in lokale Produktontologien abgebildet sowie in eine globale Produktontologie integriert. Abbildung 6.23 zeigt den Prozess für die initiale Erstellung eines Integrationsprojektes in BPMN-Notation [Whi08]. Ein Integrationsprojekt dient der Realisierung von Anwendungsfällen, die auf integrierten Produktdaten basieren. Der Prozess wird durch das *Integration Board* gestartet. Dessen Mitglieder erstellen gemeinsam einen Projektplan, der die einzelnen Meilensteine und Zeitpunkte enthält. Danach definieren die Mitglieder des Integration Board zusammen mit den Fachanwendern (engl. *Business User*) die zu realisierenden Anwendungsfälle. Sobald diese dokumentiert sind, können Integrationsexperten (engl. *Integration Experts*) mit der Spezifikation der globalen Produktontologie beginnen. Gleichzeitig definieren die Domänenexperten (engl. *Domain Experts*) die Abbildung von lokalen Informationssystemen auf entsprechende lokale Produktontologien. Sobald die Spezifikation abgeschlossen ist, können die Integrationsexperten die Abbildungs- und Integrationsregeln (SCAR und SCAA) definieren. Dann kann die automatische Auswertung der Regeln beginnen. Anschließend erfolgt die manuelle Integration durch die *Komponentenverantwortlichen*, unterstützt durch Datenqualitätsexperten (engl. *Data Quality Engineers*). Wird ein Zeitpunkt erreicht, zu dem Referenzwerte für Integrationsmetriken spezifiziert worden sind, erfolgt eine Begutachtung der

aktuellen Werte durch das Integration Board. Liegen die Werte deutlich unter den definierten Referenzwerten, kann das Integration Board geeignete Gegenmaßnahmen definieren. Ist die Integration vollständig und konsistent, können die Daten der globalen Produktontologie durch die Fachanwender genutzt werden.

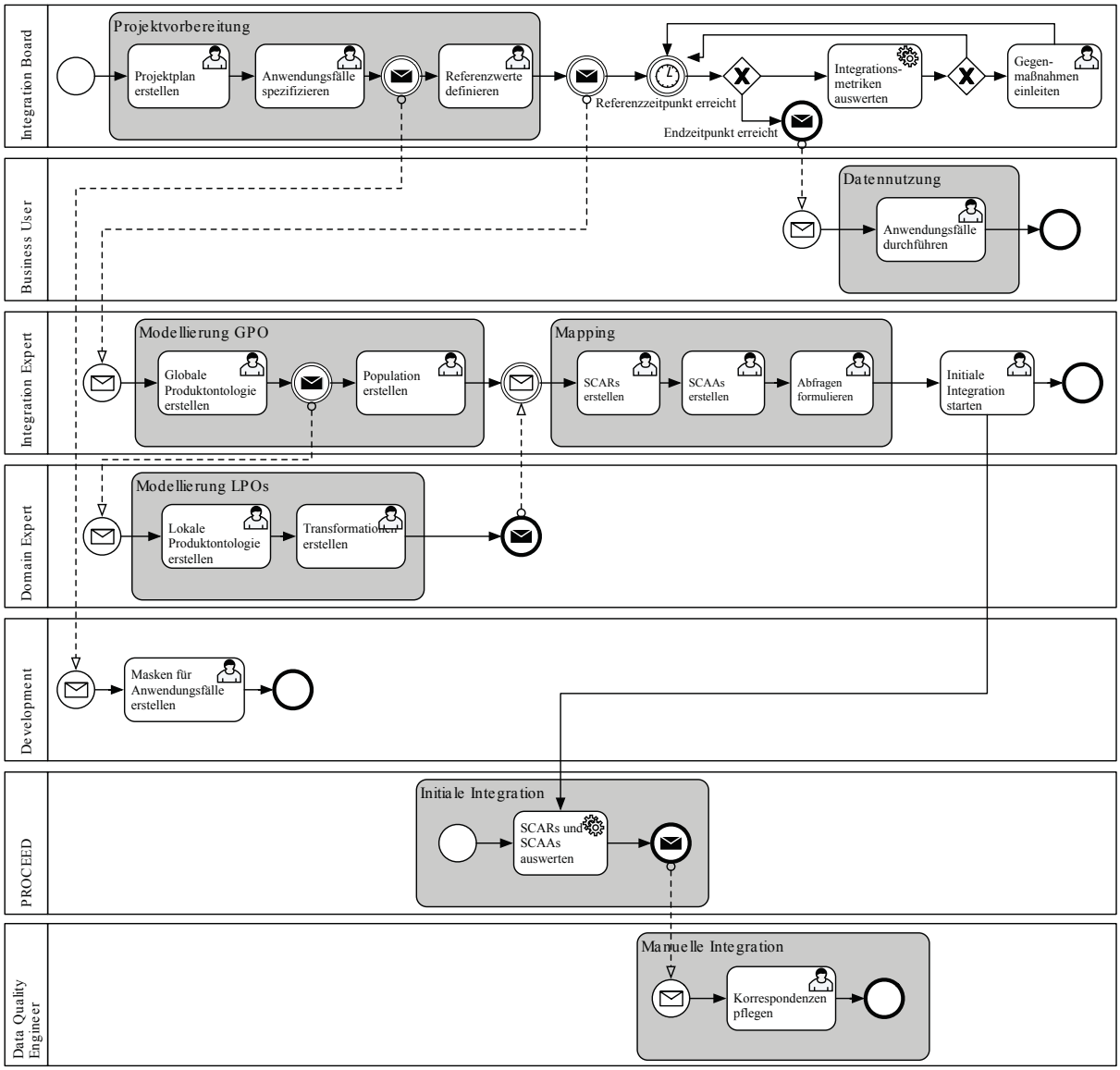


Abbildung 6.23.: BPMN-Prozess einer initialen Produktdatenintegration

Projektvorbereitung

Beispiel 21 (Integrationsprojekt). Ziel des Integrationsprojektes sei der Abgleich von Bauteilen für das Produkt „Limousine 2018“ und dessen Testergebnisse. Dazu sollen die Stammdaten der Bauteile eines Produktes (z.B. Name, eindeutige Bauteilnummer, Test-

ergebnissen) abgeglichen werden. Auf Basis dieser integrierten Produktdaten lassen sich verschiedene Anwendungsfälle definieren:

- Anwendungsfall 1: Erstellung einer Übersicht mit allen Bauteilen, für die keine Testergebnisse vorliegen.
- Anwendungsfall 2: Konsistenzüberprüfung von Attributen der Stückliste und der Testdokumentation.

1. **Projektplan erstellen:** Der Projektplan definiert die grundlegenden Zeitpunkte für das Integrationsprojekt. Dazu zählen die Modellierung aller Produktontologien sowie die Spezifikation des Mappings zwischen lokalen Produktontologien und der globalen Produktontologie. Hinzu kommen die Auswahl von Instanzen und die Festlegung einer Integrationsstrategie. Es wird zwischen *Top-Down-*, *Bottom-Up* und *Hybrid-*Ansatz unterscheiden.

2. **Anwendungsfälle spezifizieren:** Ziel der Integration von Produktdaten ist die Realisierung von Anwendungsfällen. Beispiele für letztere sind die Erstellung von Konsistenzüberprüfungen oder Übersichten.

3. **Systemlandschaftsanalyse:** Nach Festlegung der Integrationsziele und Spezifikation der Anwendungsfälle kann die bestehende IT-Systemlandschaft untersucht werden. Integrationsexperten entscheiden zusammen mit Domänenexperten, welche Informationssysteme für die Realisierung der zuvor spezifizierten Anwendungsfälle notwendig sind. In unserem Beispiel werden die lokalen Informationssysteme A und B identifiziert, wobei A die Stücklisten für Produkte und B die Testergebnisse von Produktbauteilen verwaltet.

Modellierung der globalen Produktontologie

Wie erwähnt, gibt es für die Erstellung der globalen Produktontologie mehrere Möglichkeiten. Entweder werden Schemakonzepte unabhängig von bereits bestehenden Konzepten aus lokalen Produktontologien erstellt und beschrieben (*Top-Down-*Ansatz), oder es werden zunächst die lokalen Produktontologien durch Domänenexperten erstellt und danach vorhandene Schemakonzepte aus lokalen Produktontologien in der globalen Produktontologie wiederverwendet (*Bottom-Up-*Ansatz). Auch ein Mix aus beiden Ansätzen ist möglich: So können Schemakonzepte in der globalen Produktontologie manuell definiert und zusätzlich bestehende Konzepte aus lokalen Produktontologien wiederverwendet werden (*Hybrid-*Ansatz). Da im vorliegenden Beispiel die Anzahl der lokalen Informationssysteme klein ist, wird zunächst die globale Produktontologie unabhängig von lokalen Produktontologien modelliert (Top-Down-Ansatz).
Das Integration Board definiert, zusammen mit den Fachanwendern, auf Grundlage der zuvor ermittelten Anwendungsfälle die benötigten Konzepte in der globalen Produktontologie. Im Speziellen legen sie für die einzelnen Konzepte die benötigte Granularität für Produktvarianten und -versionen fest.
Im Beispiel in Abbildung 6.24 besteht die globale Produktontologie aus jeweils einem Schemakonzept pro Produktdatenintegrationsebene. Das Produkt (*Product*) besitzt einen Namen und eine Beschreibung. Ferner umfasst es mehrere Bauteile (*Part*) mit jeweils eindeutiger Bauteilnummer und Namen (*Partnumber*). Zu einem Bauteil können mehrere Varianten (*Partvariant*) existieren, die explizit als Instanzen verwaltet werden. Der Variantentyp des Schemakonzeptes

ist somit Typ 2 (vgl. Abschnitt 6.2.1). Neben einem Namen der Variante wird ein Attribut verwaltet, das eine Produktvariante eindeutig beschreibt. Zu jeder Bauteilvariante können mehrere Versionen (*Part Version*) existieren. Jede Version umfasst einen Namen und eine eindeutige Versionsnummer (*Version Identifier*). Zusätzlich werden in diesem Schemakonzept die Produktdatenaspekte zusammengeführt. Im Beispiel in Abbildung 6.24 werden für jede Bauteilversion die entsprechenden Testergebnisse integriert (*Testresults*).

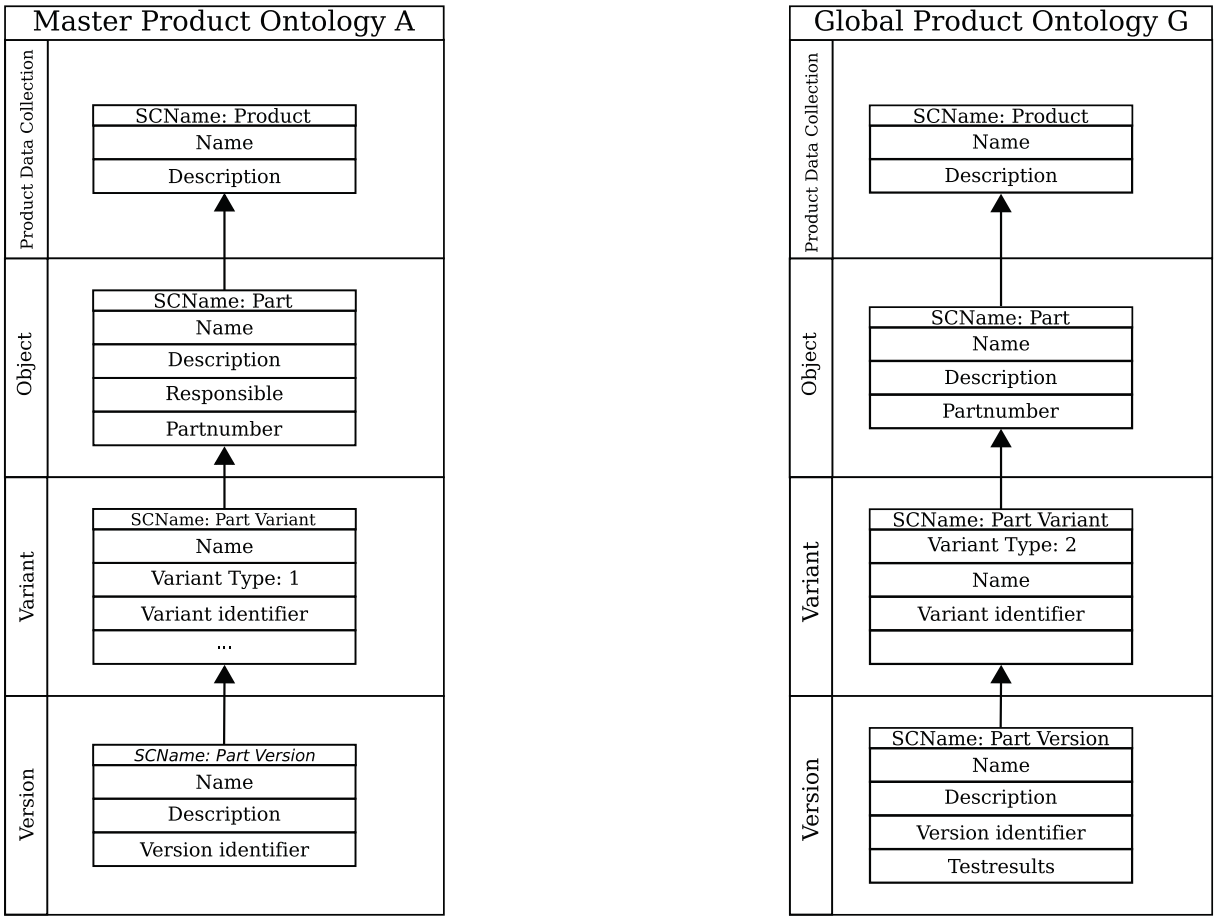


Abbildung 6.24.: Integrationsbeispiel: Globale Produktontologie

Damit Produktdaten aus lokalen Produktontologien in die globale Produktontologie integriert werden können, müssen in der globalen Produktontologie für die einzelnen Konzepte Instanzen vorhanden sein. Diese können entweder manuell von Hand angelegt werden, was sehr zeitaufwändig ist, oder es können Instanzen aus einer bestehenden lokalen Masterproduktontologie als Referenz importiert und gegebenenfalls umbenannt werden. Da die Stückliste in unserem Beispiel alle benötigten Bauteile dokumentiert, wird das lokale Informationssystem A in eine lokale Masterproduktontologie transformiert und anschließend auf die zuvor definierten Schemakonzepte der globalen Produktontologie abgebildet. Das lokale Informationssystem A umfasst Daten für eine Vielzahl von Produkten. Da für die Realisierung der Anwendungsfälle nur die Instanzen eines Produktes benötigt werden, werden auch nur diese in die lokale Produktontologie transformiert.

Modellierung lokaler Produktontologien

Auf Basis der Anwendungsfälle erstellen die Domänenexperten lokale Produktontologien für ihre Informationssysteme. Dazu definieren sie Konzepte auf den einzelnen Integrationsebenen und dokumentieren zusätzlich implizites Wissen zu diesen Konzepten. Dies können etwa Dokumentationsmethodiken, Namensschemata oder Methoden zur Dokumentation von Produktdatenvariabilität und -versionierung sein. Nachdem die Konzepte erstellt wurden, müssen Transformationen auf das Datenmodell des Informationssystems spezifiziert und realisiert werden. Für das Beispiel wird das lokale Informationssystem B in eine lokale Produktontologie abgebildet (siehe Abbildung 6.25). Die Transformation von Datenmodellkonzepten des lokalen Informationssystems in die entsprechende Produktontologie wird von Domänenexperten, unterstützt durch Integrationsexperten, vorgenommen. Für jede Integrationsebene wird ein Schemakonzept mit unterschiedlichen Attributen angelegt. Analog zur lokalen Masterproduktontologie werden nur diejenigen Instanzen in die Ontologie transformiert, die zur Realisierung der zuvor definierten Anwendungsfälle notwendig sind.

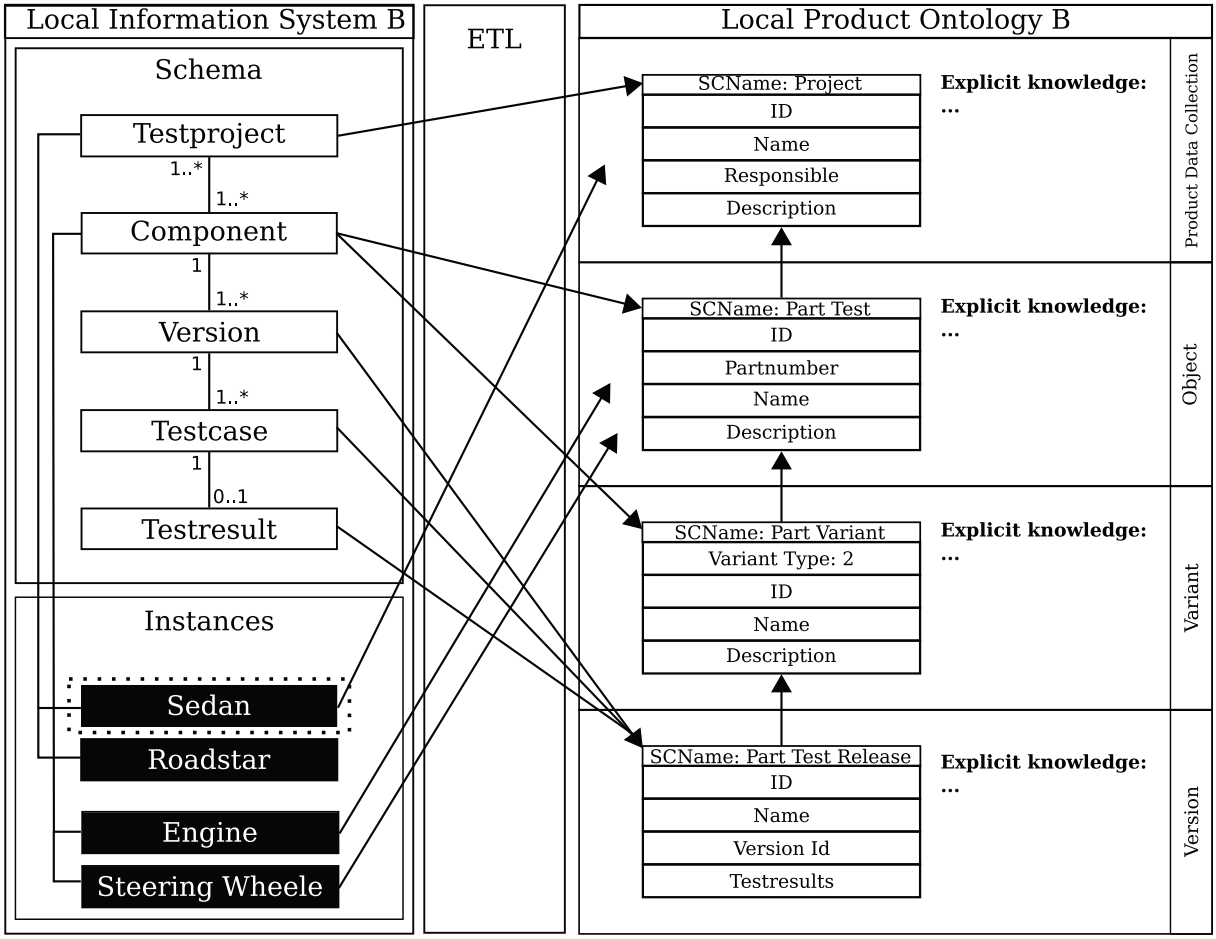


Abbildung 6.25.: Integrationsbeispiel: Abbildung des Informationssystem B in eine lokale Produktontologie

Abbildung 6.25 zeigt die beispielhafte Transformation des lokalen Informationssystems B in eine

entsprechende lokale Produktontologie. Das konzeptuelle Schema des lokalen Informationssystems besteht aus den 5 Konzepten *Testproject*, *Component*, *Version*, *Testcase* und *Testresult*, die auf die Schemakonzepte *Project*, *Test Part*, *Part Variant* und *Part Test Release* abgebildet werden. Exemplarisch sind nur ausschnittsweise Instanzen für die Schemakonzepte des lokalen Informationssystems dargestellt. Während der Transformation werden nur die Instanzen *Sedan* von *Testproject* auf entsprechende Instanzen der lokalen Produktontologie transformiert. Gleiches gilt für alle Instanzen im lokalen Informationssystem, die mit *Sedan* verbunden sind. Wie zu erkennen ist, verfügt das Schema des lokalen Informationssystems über kein explizites Konzept zur Repräsentation von Bauteilvarianten. Vielmehr sind Informationen über Bauteile und Varianten in einem Konzept *Component* zusammengefasst. Als Variabilitätstyp wurde eine explizite Repräsentation von Bauteilvarianten identifiziert.

Mapping

Sind alle Produktontologien spezifiziert, können **Integrationsexperten** Abbildungsregeln (SCARs) und Abbildungsaktionen (SCAAs) zwischen lokalen Produktontologien und globaler Produktontologie definieren. Abbildung 6.26 zeigt eine beispielhafte Schema-Integration der zuvor erläuterten Informationssysteme. Der Übersicht halber werden keine Instanzen der einzelnen Schemakonzepte dargestellt, außerdem werden die Details der Schemakonzept-Attributregeln ausgeblendet.
Zunächst erfolgt die Abbildung zwischen der lokalen Masterproduktontologie A und der globalen Produktontologie G. Dazu ist auf jeder Integrationsebene eine Schemakonzept-Attributregel *SCAR 1* bis *SCAR 4* definiert. Die Schemakonzept-Attributregel *SCAR 4* setzt sich zwei Attributmatching-Bedingungen (*AMC 1* und *AMC 2*) zusammen. Die Schemakonzept-Attributaktion *SCAA 1* bis *SCAA 5* überführen die entsprechende Attributwerte für Instanzen in die globale Produktontologie. Analog werden anschließend Abbildungsregeln für die lokale Produktontologie B in die globale Produktontologie G spezifiziert.
Analog zur lokalen Masterproduktontologie spezifizieren die Schemakonzept-Attributregeln *SCAR 5* bis *SCAR 8* die Abbildung der Schemakonzepte der lokalen Produktontologie B in G.

Rollen

Im Prozess aus Abbildung 6.23 sind unterschiedliche Rollen für das PROCEED-Rahmenwerk definiert, die im folgenden Abschnitt näher beschrieben werden.

- **Domain Expert:** Ein *Domänenexperte* (engl. domain expert) verantwortet ein Informationssystem und kennt die verwendeten Datenmodellkonzepte. Außerdem ist er für die Erstellung lokaler Produktontologien verantwortlich. Dabei erstellt er die notwendigen Transformationen von internen Datenmodellen eines Informationssystems in eine lokale Produktontologie. Schließlich erstellt er zusammen mit Integrationsexperten die Abbildungsregeln zwischen lokaler und globaler Produktontologie.
- **Integration Expert:** Der *Integrationsexperte* (engl. integration expert) kennt die Abhängigkeiten zwischen Schemakonzepten unterschiedlicher Informationssysteme. Er erstellt zusammen mit dem *Integration Board* die Konzepte in der globalen Produktontologie, er erstellt und pflegt die Abbildungs- und Integrationsregeln zwischen lokaler und globaler Produktontologie in Zusammenarbeit mit Domänenexperten. Sie spezifizieren außerdem die fachlichen Abfragen zur Realisierung von Anwendungsfällen.

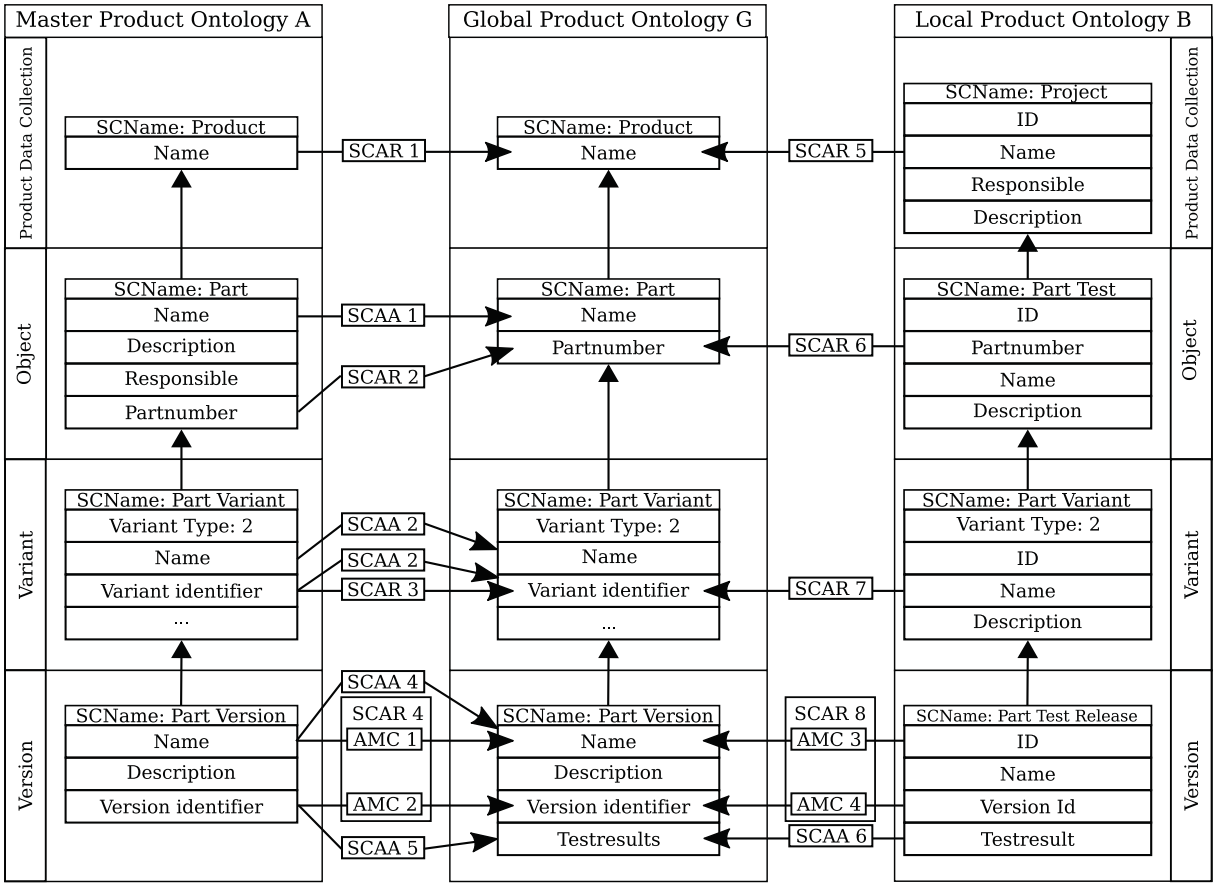


Abbildung 6.26.: Integrationsbeispiel: Ergebnis einer Schema-Integration

- **Knowledge Engineer:** Die Abbildung eines lokalen Informationssystems in eine korrespondierende Produktontologie ist nicht trivial und erfordert sowohl Wissen über das PROCEED-Rahmenwerk im Speziellen als auch die Modellierung von Datenmodellen im Allgemeinen. Entsprechend unterstützt der *Knowledge Engineer* die Domänen- und Integrationsexperten bei der Modellierung von Ontologien. Er ist Spezialist für die Erstellung von Ontologien und kennt deren Strukturen und Konsistenzbedingungen.

- **Data Quality Engineer:** Der *Data Quality Engineer* pflegt Korrespondenzen zwischen lokaler und globaler Produktontologie. Er wird vom System nach der initialen Integration über den Stand der Integration benachrichtigt. Er kann selbstständig Korrespondenzen erstellen, löschen und bearbeiten.

- **Integration- / Data-Quality-Board:** Das Board definiert den zu integrierenden Umfang und die Integrationsmetriken (inkl. Referenzwerten). Zusätzlich definieren das Board die zu erwarteten Elemente der Integration (z.B. Komponenten oder System). Dazu können es etwa ein Master-Informationssystem auswählen, aus dem diese Daten übernommen werden sollen.

- **Software Development:** Nachdem die Anwendungsfälle spezifiziert sind, erstellt die Softwareenwicklungsabteilung (engl. *software development*) entsprechende Masken.

6.3. Automatische Instanzintegration

Nach dem Mapping der lokalen Produktontologien in die globale Produktontologie, können mit Hilfe der Abbildungsregeln geeignete Korrespondenzen ermittelt und Schemakonzept-Attributaktionen ausgeführt werden. Es muss zwischen der initialen Integration von Informationssystemen und dem Änderungsmanage- ment unterschieden werden. Bei der initialen Integration existieren noch keine integrierten Produktdaten bzw. lokalen Produktontologien. Nach der initialen Integration folgt das Änderungs-management. Hierbei werden alle Änderungsfälle betrachtet, die nach einer initialen Integration auftreten können. Die initiale Integration wertet für jede lokale Produktontologie, die in die globale Produktontologie abgebildet ist, die definierten SCARs für die existierenden Instanzen aus und erstellt Korrespondenzen. Für letztere werden ebenfalls vorhandene SCAAs ausgeführt.

6.3.1. Initiale Integration

Die Integration von Instanzen aus den Schemakonzepten lokaler Produktontologien in die globale Produktontologie sollte, soweit wie möglich, automatisiert werden. Da lokale Produktontologien und globale Produktontologie die gleiche Struktur aufweisen, können die SCARs für jede Produktdatenintegrationsebene separat ausgewertet werden. Im Detail erfolgt die Auswertung der SCARs von oben nach unten, d.h. SCARs zwischen Schemakonzepten einer lokalen und der globalen Produktontologie in der *Product-Data-Collection*-Ebene werden zuerst ausgewertet. Danach werden die SCARs auf der *Object*-Ebene ausgewertet. Dieser Prozess wird für die weiteren Produktdatenintegrationsebenen fortgeführt.
Ausgangspunkt einer initialen Integration bilden ein oder mehrere Produktontologie-Mappings (mit oder ohne Ausschluss von Schemakonzepten und Instanzen) $POMapping_{L_1 \times G}, \ldots,$ $POMapping_{L_n \times G}$ lokaler Produktontologien $L_1, \ldots, L_n$ auf eine globale Produktontologie G. Der Übersichtlichkeit halber wird im Folgenden nur die Integration einer bestimmten lokalen Produktontologie in die globale Produktontologie illustriert. Damit die Instanzen einer lokalen Produktontologie in die globale Produktontologie integriert werden können, muss eine Reihe von Eigenschaften erfüllt sein. Erstens müssen beide Produktontologien strukturell konsistent sein (siehe Definition 24). Zweitens muss die Abbildung von Schemakonzepten der lokalen Produktontologien auf die globale Produktontologie vollständig sein (vgl. Definition 38).
Ergebnis der initialen Integration ist eine Menge von Korrespondenzen (inkl. Begründungen) zwischen Instanzen der lokalen Produktontologien $PO_{L_1}, \ldots, PO_{L_n}$ einerseits und Instanzen der globalen Produktontologie PO_G andererseits.

Definition 41 (Initiale Integration). Algorithmus 1 liefert für ein Produkt- ontologie-Mapping $POMapping_{L \times G}$ zwischen einer lokalen Produktontologie PO_L und einer globalen Produktontologie PO_G eine Menge an Korrespondenzen (*Correspondences*) inklusive Begründungen (*Justifications*), Attributwerte mit Begründungen (*AttrValJustification*) und eine Liste mit Integrationsaufgaben (*Tasklist*) zwischen diesen:

$$integrate : POMapping_{L \times G} \rightarrow (Correspondences_{L \times G}, Justifications, AttrValJustification, Tasklist)$$

Algorithmus 1 (Produktontologie-Mapping auswerten)

procedure INTEGRATE($POMapping_{LxG}$)
  $Layer = \{pdc, object, variant, version\}$
  $Correspondences = \varnothing$
  $Justification = \varnothing$
  $AttrValJustification = \varnothing$
  $Tasklist = \varnothing$
  for each $layer \in Layer$ **do**
    for each $sc_L^{cur} \in SC_L^{layer} \backslash SC_L^{Ignore}$ **do**
      $sc_G^{cur} = (sc \in SC_G^{layer} \mid \exists(sc_L^{cur}, sc, AMC) \in SCAR)$
      $Ind_L^{sc_L^{cur},layer} = (ind \in Ind_L \mid (sc_L^{cur}, ind) \in hasIndividual_L \wedge ind \notin Ind_L^{Ignore})$
      $Ind_G^{sc_L^{cur},layer} = (ind \in Ind_G \mid (sc_G^{cur}, ind) \in hasIndividual_G \wedge ind \notin Ind_G^{Ignore})$
      $IndCandidates_L = \varnothing$
      $IndCandidates_G = \varnothing$
      if $\nexists l \in Layers : isAbove(l, layer)$ **then**
        $IndCandidates_L = Ind_L^{sc_L^{cur},layer}$
      else
        for each $ind_L^{cur} \in Ind_L^{sc_L^{cur},layer}$ **do**
          $parentInd_L = (ind \in Ind_L \mid \exists(ind, ind_L^{cur}) \in indAbove_L$
          if $\exists(parentInd_L, x) \in Correspondence$ mit $x \in Ind_G$ **then**
            $IndCandidates_L = IndCandidates_L \bigcup ind_L^{cur}$
          end if
        end for
      end if
      for each $ind_L^{cur} \in IndCandidates_L$ **do**
        if $\nexists l \in Layers : isAbove(l, layer)$ **then**
          $IndCandidates_G = Ind_G^{sc_G^{cur},layer}$
        else
          $parentInd_L = (ind \in Ind_L \mid \exists(ind, ind_L^{cur}) \in indAbove_L$
          $ParentInd_G = (ind \in Ind_G \mid (parentInd_L, ind) \in Correspondence)$
          for each $ind_G^{cur} \in ParentInd_G$ **do**
            $indCandidate_G = (ind \in Ind_G \mid \exists(ind, ind_G^{cur}) \in indBelow$
            $IndCandidates_G = IndCandidates_G \bigcup indCandidate_G$
          end for
        end if
        for each $ind_G^{cur} \in IndCandidates_G$ **do**
          for each $scar^{cur} = (sc_L^{cur}, sc_G^{cur}, AMC) \in SCAR$ **do**
            if $evaluateAMC(ind_L^{cur}, ind_G^{cur}, scar^{cur}.AMC)$ **then**
              $newCor = (ind_L^{cur}, ind_G^{cur}, scar^{cur})$
              $Justification = Justification \bigcup (newCor, scar^{cur})$
              $Correspondences = Correspondences \bigcup newCor$
              $scaaResult = executeSCAA(newCor)$
              $AttrValJustification = AttrValJustification \bigcup$
                $scaaResult.AttrValJustification$

43: $Tasklist = Tasklist \bigcup scaaResult.Tasklist$
44: **end if**
45: **end for**
46: **end for**
47: **end for**
48: **end for**
49: **end for**
50: **return** $Correspondences, Justification, AttrValJustification, Tasklist$
51: **end procedure**

Algorithmus 1 stellt den initialen Integrationsprozess für ein Produktontologie-Mapping dar. Er wertet die Schemakonzept-Attributregeln aus und führt für gefundene Korrespondenzen zugehörige Schemakonzept-Attributaktionen aus. Im Detail erfolgt die Integration der lokalen Produktontologie in die globale Produktontologie für jede Produktdatenintegrationsebene separat (Zeile 2), beginnend mit der *Product-Data-Collection*-Ebene *pdc*. Zunächst sind die Menge der Korrespondenzen, Begründungen und Attributbegründungen sowie die Aufgabenliste leer (Zeilen 3 bis 6).
Die nachfolgenden Schritte werden für jede Produktdatenintegrationsebene ausgeführt (Zeile 7): Für jedes Schemakonzept sc_L^{cur} der lokalen Produktontologie der aktuellen Ebene (Zeile 8) wird das korrespondierende Schemakonzept sc_G^{cur} der globalen Produktontologie über die Schemakonzept-Attributregeln ermittelt (Zeile 9). Anschließend werden für beide Schemakonzepte die zugeordneten Instanzen $Ind_L^{sc_L^{cur}}$ der lokalen Produktontologie (Zeile 10) sowie die Menge der Instanzen $Ind_G^{sc_G^{cur}}$ der globalen Produktontologie (Zeile 11) bestimmt. Instanzen, die von einer Integration ausgeschlossen worden sind (vgl. Abschnitt 6.2.3), werden nicht berücksichtigt (vgl. Ind_L^{Ignore} und Ind_G^{Ignore} in Zeilen 10 und 11).
Für die Instanzen aus lokaler und globaler Produktontologie werden paarweise die definierten Schemakonzept-Attributregeln ausgewertet. Auf jeder Produktdatenintegrationsebene werden hierzu diejenigen Instanzen der lokalen sowie globalen Produktontologie ermittelt, die für einen paarweisen Vergleich in Frage kommen. Die Menge der Kandidaten der lokalen Produktontologie $IndCandidates_L$ und der globalen Produktontologie $IndCandidates_G$ sind zunächst leer (Zeilen 12, 13). Die Integration wird für die oberste Produktdatenintegrationsebene *pdc* gestartet. In diesem Fall ist die Menge $IndCandidates_L$ gleich der Menge der Instanzen für das aktuelle Schemakonzept $Ind_L^{sc_L^{cur}}$ (Zeilen 12 bis 14). Befindet sich die Integration auf einer anderen Produktdatenintegrationsebene, so werden nur diejenigen lokalen Instanzen für einen Vergleich herangezogen, für deren zugehörige Instanzen auf der nächst höheren Produktdatenintegrationsebene bereits Korrespondenzen ermittelt worden sind (Zeilen 12 bis 21).
Abbildung 6.27 illustriert ein Beispiel. Seien Korrespondenzen `cor1`, `cor2` und `cor3` zwischen den Instanzen `B`, `C`, `X`, `Y` und `Z` der Produktdatenintegrationsebene `Product Data Collection` durch eine Schemakonzept-Attributregel `SCAR 1` ermittelt. Algorithmus 1 befindet sich in der `Object`-Ebene. Die Menge der lokalen Instanzen, die für einen Vergleich in Betracht gezogen werden, sind in diesem Beispiel `B1` und `C1`, da für die zugehörige Instanzen der nächst höheren Produktdatenintegrationsebene (`B` und `C`) Korrespondenzen (`cor1`, `cor2` und `cor3`) vorhanden sind.

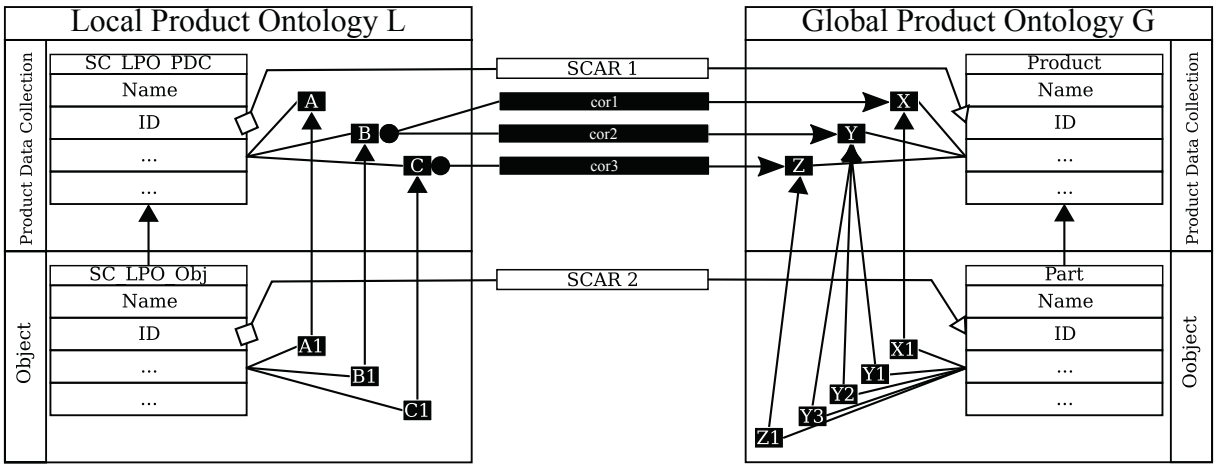


Abbildung 6.27.: Reduktion der Vergleiche von Instanzen

Anschließend wird die Menge der globalen Instanzen bestimmt, die für den Vergleich herangezogen werden. Befindet sich die Integration auf der obersten Produktdatenintegrationsebene, ist die Menge der Kandidaten, analog zur lokalen Produktontologie, gleich der Menge der globalen Instanzen für das aktuelle globale Schemakonzept $Ind_G^{sc_G^{cur},layer}$ (Zeilen 23 bis 25). Befindet sich der Algorithmus auf einer anderen Produktdatenintegrationsebene, wird die Menge der globalen Instanzkandidaten über Korrespondenzen zwischen lokalen und globalen Instanzen wie folgt ermittelt: Für jede lokale Instanz ind_L^{cur} der Menge der lokalen Instanzkandidaten $IndCandidates_L$ wird die lokale Instanz $parentInd_L$ ermittelt, die sich auf der nächst höheren Produktdatenintegrationsebene befindet. Danach wird für die Instanz $parentInd_L$ die Menge der korrespondierenden globalen Instanzen $ParentInd_G$ bestimmt (Zeilen 26 - 27). Die Menge der globalen Instanzkandidaten $IndCanidates_G$ ergibt sich aus den zugehörigen Instanzen aus $ParentInd_G$ der nächst tieferen Produktdatenintegrationsebene (Zeile 30).
Zur Veranschaulichung betrachten wir nochmals das Beispiel aus Abbildung 6.27. Für die lokale Instanz `B1` ergibt sich die Menge der globalen Instanzkandidaten über die Instanz `B` der nächst höheren Produktdatenintegrationsebene von `B`. Für diese Instanz existieren zwei korrespondierende globale Instanzen `X` und `Y`. Die Menge der zugehörigen Instanzen von `X` und `Y` auf der nächst tieferliegenden Produktdatenintegrationsebene (`X1`, `X2`, `Y1`, `Y2` und `Y3`) ist die gesuchte Menge der globalen Instanzkandidaten für einen Vergleich mit `B1`.
Nachdem die Mengen der lokalen Instanzkandidaten $IndCandidates_L$ und der globalen Instanzkandidaten $IndCandidates_G$ ermittelt sind, wird jedes Element der ersten Menge mit jedem Element der zweiten Menge verglichen (Zeile 22 und 33). Dazu wird für jede Schemakonzept-Attributregel, die zwischen lokalem Schemakonzept sc_L^{cur} und globalen Schemakonzept sc_G^{cur} definiert ist, die Menge der Attribut-Matching-Bedingungen (vgl. Abschnitt 6.2.3) für die lokale und globale Instanz (ind_L^{cur} und ind_G^{cur}) ausgewertet (Zeile 35). Die Auswertung erfolgt durch Algorithmus 2 $evaluateAMC$. Sind für die lokale und globale Instanz ind_L^{cur} und ind_G^{cur} alle Attribut-Matching-Bedingungen erfüllt, wird eine neue Korrespondenz ($newCor$) sowie Korrespondenz-Begründung ($Justification$) erstellt (Zeilen 36 bis 38). Schließlich werden für ind_L^{cur} und ind_G^{cur} korrespondierende Schemakonzept-Attributaktionen für lokale und globale Instanz ausgeführt (Zeile 39). Dazu wird Algorithmus 3 $executeSCAA$ ausgeführt. Neben Attribut-Begründungen (siehe Abschnitt 6.2.3) liefert Algorithmus 3 eine Menge an Aufgaben zurück, die durch Inte-

grationskonflikte enstanden sind (Zeilen 40 und 41). Der Algorithmus terminiert, nachdem die Auswertung von lokalen und globalen Instanzkandidaten für alle Schemakonzepte der untersten Produktdatenintegrationsebene beendet ist.
Die in Algorithmus 1 verwendete Funktion `evaluateAMC` ist in Algorithmus 2 definiert.

Algorithmus 2 (Attribut-Matching-Bedingung auswerten)

1: **procedure** EVALUATEAMC($ind_L, ind_G, scar$)
2: **for each** $curAmc \in scar.AMC$ **do**
3: $val_L = (attrVal \in AttrVal_L \mid \exists(attrVal, curAmc.attr_L) \in references_L$ mit $(ind_L, attrVal) \in hasValue_L)$
4: $val_G = (attrVal \in AttrVal_G \mid \exists(attrVal, curAmc.attr_G) \in references_G$ mit $(ind_G, attrVal) \in hasValue_G)$
5: **if** ! $match(val_L, val_G, curAmc.mf, curAmc.par)$ **then**
6: **return** $false$
7: **end if**
8: **end for**
9: **return** $true$
10: **end procedure**

Eingabe des Algorithmus 2 sind eine Schemakonzept-Attributregel *scar* sowie jeweils eine Instanz einer lokalen und globalen Produktontologie (ind_L und ind_G). Für jede Attribut-Matching-Bedingung (Zeile 1) der Schemakonzept-Attributregel werden die Attributwerte der lokalen Instanz val_L und der globalen val_G (Zeilen 2 und 3) bestimmt. Stimmen die Werte bei Auswertung der Matching-Funktion (vgl. Definition 27) der Schemakonzept-Attributregel nicht überein, ist mindestens eine Attribut-Matching-Bedingung dieser Regel nicht erfüllt. Somit wird als Ergebnis $false$ zurückgegeben (Zeile 5). Sind alle Bedingungen erfüllt, wird als Ergebnis *true* zurück geliefert (Zeile 7).
Die Auswertung der Match-Funktion einer Schemakonzept-Attributregel wird im Folgenden nicht weiter detailliert. Beispiele wurden bereits in Abschnitt 6.2.3 erläutert.
Schließlich definiert der Algorithmus 3 `executeSCAA` die Ausführung der Schemakonzept-Attributaktionen für die identifizierten Korrespondenzen:

Algorithmus 3 (SCAAs ausführen)

1: **procedure** EXECUTESCAA($cor, SCAA$)
2: $AttrValJustification = \varnothing$
3: $Tasklist = \varnothing$
4: **for each** $scaa^{cur} \in SCAA$ **do**
5: **for each** $aa^{cur} \in scaa^{cur}.aa$ **do**
6: $localAttrVal = (\exists attrVal \in AttrVal_L \mid$
$\exists(attrVal, scaa^{cur}.attr_L) \in references_L \wedge$
$\exists(cor.ind_L, attrVal) \in hasValue_L)$
7: $localChanged = (\exists metaAttrVal \in MetaAttrVal_L \mid$
$(localAttrVal, 'changed', metaAttrVal) \in hasMetaAttrVal_L)$
8: $globalAttrVal = (\exists attrVal \in AttrVal_G \mid$
$\exists(attrVal, scaa^{cur}.attr_G) \in references_G \wedge$
$\exists(cor.ind_G, attrVal) \in hasValue_G)$

```
9:          globalChanged = (∃metaAttrVal ∈ MetaAttrVal_L |
                (localAttributeValue,′ changed′, metaAttrVal)ιhasMetaAttrVal_L)
10:         switch aa^cur.behavior do
11:             case append :
12:                 globalAttrVal = globalAttrVal ⋃ localAttrVal
13:             case recent :
14:                 if localChaned > globalChanged then
15:                     globalAttrVal = localAttrVal
16:                 end if
17:             case local :
18:                 globalAttrVal = localAttrVal
19:             case global :
20:                 if globalAttrVal == ∅ then
21:                     globalAttrVal = localAttrVal
22:                 end if
23:             case interactive :
24:                 Tasklist = Tasklist ⋃ (cor, aa^cur)
25:         end for
26:         AttrValJustification = AttrValJustification ⋃ (globalAttrVal, scaa)
27:     end for
28:     return AttrValJustification, Tasklist
29: end procedure
```

Eingabe von Algorithmus 3 sind eine Menge an Schemakonzept-Attributaktionen $SCAA$ sowie eine Korrespondenz *cor* auf dessen Instanzen die Aktionen ausgeführt werden sollen. Die folgenden Schritte werden für alle Schemakonzept-Attributaktionen $SCAA$ und deren Attribut-Aktionen ausgeführt (Zeilen 4 und 5). Zunächst werden die Attributwerte (inklusive Änderungszeitpunkte) der lokalen sowie globalen Instanz ermittelt (Zeilen 6 bis 9). Anschließend wird, abhängig vom Integrationsverhalten (siehe Abschnitt 6.2.3), der globale Attributwert *globalAttrVal* angepasst. Bei *append* wird der lokale Attributwert in Form einer Liste an den globalen Attributwert angehängt (Zeilen 11 und 12). Ist der lokale Attributwert nach dem globalen Attributwert geändert worden, wird im Fall *recent* der globale durch den lokalen Attributwert ersetzt. Für *local* wird immer der lokale Attributwert in die globale Produktontologie kopiert, bei *global* genau dann, wenn der globale Attributwert leer ist. Im letzten Fall (*interactive*) wird ein Eintrag in der Aufgabenliste (*Tasklist*) hinzugefügt. Welcher Attributwert in die globale Produktontologie übernommen wird, wird nach der automatischen Auswertung manuell entschieden. Nachdem alle Attributaktionen ausgeführt worden sind, werden die Begründungen (*AttrValJustification*) sowie Aufgaben (*Tasklist*) zurückgegeben (Zeile 28).

6.4. Manuelle Instanzintegration

Obwohl die Integration von Instanzen möglichst automatisch erfolgen sollte, fallen nach einer automatischen Instanzintegration ggf. manuelle Aufgaben an. Folgende drei Arten von Aufgaben lassen sich unterscheiden: Erstens müssen für diejenigen lokalen Instanzen, für die keine Korrespondenzen in der globalen Produktontologie automatisch ermittelt werden konnten, entsprechende Korrespondenzen manuell durch Experten erstellt werden. Zweitens müssen existierende

Integrationskonflikte, die nicht automatisch aufgelöst werden konnten, manuell aufgelöst werden. Drittens können während der automatischen Instanzintegration auch Korrespondenzen ermittelt worden sein, die falsch sind. Experten können in diesem Fall die betroffenen Korrespondenzen manuell löschen. Ursachen sowie mögliche Lösungsmöglichkeiten werden im Folgenden erläutert.

6.4.1. Aufgabe 1: Fehlende Korrespondenzen erstellen

Nach einer automatischen Instanzintegration sind für einige Instanzen einer lokalen Produktontologie keine korrespondierenden globalen Instanzen mit Hilfe von Schemakonzept-Attributregeln gefunden worden. Folgende Ursachen können dafür verantwortlich sein.

Fehlender Attributwert einer Instanz, der durch eine SCAR ausgewertet wird

Ein Attribut, welches durch eine SCAR ausgewertet wird, ist nicht dokumentiert, d.h. sein Attributwert ist nicht gesetzt. Ein möglicher Grund dafür ist, dass die Entwicklung von Komponenten eines Produktes unterschiedlich weit fortgeschritten sein kann. Wie in Abschnitt 2.2.3 erläutert, werden Komponenten parallel durch eine Vielzahl von Entwicklern realisiert. Dadurch kann ihr Entwicklungsfortschritt zu einem bestimmten Zeitpunkt t unterschiedlich weit fortgeschritten sein. Während beispielsweise Komponente A zum Zeitpunkt t spezifiziert und realisiert sein kann, kann eine weitere Komponente B zum gleichen Zeitpunkt nur spezifiziert, jedoch nicht realisiert, sein. Ein weiterer Grund ist, dass eine Dokumentation oder Änderung eines Attributwertes in einem Informationssystem noch nicht an alle beteiligten Systeme entlang der Entwicklungskette weitergeben worden ist. Schließlich kann die Dokumentation eines Attributwertes vergessen worden sein.
Lösungsmöglichkeit: Eine Möglichkeit zur Reduzierung der Anzahl an Instanzen, für die keine Korrespondenzen auf Grund fehlender Attributwerte gefunden werden konnten, besteht darin, die automatische Instanzintegration erst dann durchzuführen, wenn möglichst viele Instanzen mit Attributwerten dokumentiert sind.

Unterschiedliche Datenqualität und inkonsistente Verwendung von Bezeichnern

Eine weitere Ursache für fehlende Korrespondenzen von lokalen Instanzen zu globalen Instanzen sind Unterschiede bzgl. der Datenqualität der Instanzattribute in den einzelnen Informationssystemen. Wichtigster Aspekt der Datenqualität ist die Konsistenz von Attributwerten zwischen Informationssystemen. So können für die selben Realweltobjekte unterschiedliche Bezeichner für Artefakte verwendet werden oder durch verschiedene Schreibweisen oder Dokumentationssprachen dokumentiert sein.

Beispiel 22 (Unterschiedliche Namenskonventionen). Baut zum Beispiel eine SCAR auf einem Attribut auf, das wiederum auf einem Muster basiert, welches durch eine globale Namenskonvention definiert ist, kann es vorkommen, dass diese Namenskonvention bei der Dokumentation von Artefakten nicht berücksichtigt worden ist. Tabelle 6.6 zeigt Beispiele von Attributwerten für die Bezeichnung eines Fahrersitzes, die semantisch das gleiche bedeuten, jedoch nicht durch eine SCAR zugeordnet werden können, die auf Mustern basiert.

Nicht identifizierte Korrespondenzen treten zudem auf, wenn eine SCAR auf einer Ähnlichkeitsmetrik basiert (vgl. Abschnitt 6.2.3). Ist der definierte Schwellwert, ab dem zwei Attributwerte als korrespondierend angesehen werden, zu hoch, werden nicht alle Korrespondenzen ermittelt.

Bezeichner
Sitz Vorne Links
Sitz VL
Seat Front Left
Sitz Fahrer
Fahrersitz Linkslenker
Driver Seat Left Steering

Tabelle 6.6.: Unterschiedliche Bezeichner aus verschiedenen Informationssystemen für dieselbe Komponente

Lösungsmöglichkeit: Inkonsistenzen und mangelnde Datenqualität können innerhalb von Informationssystemen vermieden werden, wenn Attribute auf eine zuvor definierte Konsistenzregel während der Dokumentation überprüft werden. Hierzu sollte eine Vereinheitlichung von Bezeichnen, unter Berücksichtigung von Dokumentationsmethodiken und Regeln für Abkürzungen, erfolgen.

Neue Komponenten werden lokal erstellt

Während für einige lokale Instanzen keine korrespondierenden Instanzen in der globalen Produktontologie ermittelt werden können, weil Attributwerte fehlen, so können auch lokale Instanzen existieren, für die keine Korrespondenzen bestimmt werden können, da diese nur im lokalen Informationssystem vorhanden sind.

Beispiel 23 (Neue Komponenten existieren nur lokal). Beispielsweise ist ein Sachnummernsystem / PDM-System als lokale Masterproduktontologie definiert. Wird eine neue Komponente zunächst nur versuchshalber in einem lokalen Informationssystem modelliert und getestet, existiert für diese Komponente noch keine Sachnummer in der lokalen Masterproduktontologie und folglich auch nicht in der globalen Produktontologie.

Lösungsmöglichkeit: Für diese lokale Instanz kann, solange keine korrespondierende Instanz in der globalen Produktontologie gefunden wurde, bis eine entsprechende Instanz in der lokalen Masterproduktontologie angelegt wird. Dies geschieht etwa, wenn entschieden wird die neue Komponente in ein Produkt zu integrieren. Dazu wird eine Sachnummer erstellt (in der lokalen Masterproduktontologie) und anschließend auch in die globale Produktontologie synchronisiert.

Entwicklung von neuen Komponenten noch nicht begonnen

Der letzte Fall ist als Sonderfall zu betrachten, da er nur dann auftreten kann, wenn die Instanzen einer globalen Produktontologie manuell gepflegt und nicht durch eine lokale Masterproduktontologie versorgt werden. Ein neue Komponente für ein Produkt wurde definiert und eine entsprechende Instanz in einer globalen Produktontologie angelegt. Zu diesem Zeitpunkt wurde noch nicht mit der Spezifikation (z.B. Anforderungen) und Realisierung begonnen, d.h. in lokalen Informationssystemen wurden noch keine entsprechenden Artefakte und damit Instanzen in den Produktontologien erstellt.

Lösungsmöglichkeiten: Eine Möglichkeit dieses Problem zu lösen besteht darin, eine lokale Masterproduktontologie zu definieren und auf die globale Produktontologie abzubilden. Neue Komponenten werden anschließend ausschließlich in dieser Masterproduktontologie erzeugt.

6.4.2. Aufgabe 2: Integrationskonflikte auflösen

Der zweite Grund für manuelle Aufgaben nach einer automatischen Instanzintegration sind Integrationskonflikte, die während ersterer aufgetreten sind und die manuell aufgelöst werden müssen.

Nicht synchronisierte Änderungen

Hauptursache ist das Integrationsverhalten von Schemakonzept-Attributaktionen. Wurden getätigte Änderungen in einem Informationssystem nicht an die anderen, im Entwicklungsprozess nachgelagerten, Informationssystemen synchronisiert, so existieren möglicherweise widersprüchliche Werte.
Wurde als Integrationsverhalten *manual* oder *append* für eine SCAA definiert, muss für alle lokalen Instanzen, für die Korrespondenzen entdeckt werden und deren Attributwerte teilweise widersprüchlich sind, der Integrationskonflikt manuell aufgelöst werden.
Während Instanzen mit inkonsistenten Attributwerten, die durch SCAAs mit einem *append* als Integrationsverhalten ermittelt wurden, Kandidaten für eine manuelle Konfliktauflösung sind, müssen inkonsistente Attributwerte von Instanzen im Fall von *manual* zwingend manuell aufgelöst werden, d.h. ein Data Quality Engineer (vgl. Abschnitt 6.2.5) muss einen Attributwert aus der Liste auswählen. Dadurch lässt sich unterscheiden, welche Instanzen am Ende des Integrationsprozesses vollständig konsistent sein müssen und welche Konflikte noch in der nachgelagerten Nutzungsphase durch Anwendungsfälle aufgelöst werden können.

6.4.3. Aufgabe 3: Fehlerhafte Korrespondenzen löschen

Bisher wurden nur die Fälle betrachtet, in denen Korrespondenzen von lokalen und globalen Produktontologien fehlten. Es kann aber auch vorkommen, dass während der automatischen Instanzintegration falsche Korrespondenzen ermittelt werden.

Zu niedrige Schwelle für Ähnlichkeitsfunktion

Analog zum Fall 1, bei dem bestimmte Korrespondenzen nicht gefunden werden, weil der Schwellwert einer Matching-Funktion zu hoch ist, kann eine zu niedrige Schwelle dazu führen, dass Korrespondenzen zwischen Instanzen ermittelt werden, die in der Realwelt gar nicht zueinander gehören (*False Positives*) [Faw06].
Zusammengefasst müssen nach Auswertung von Schemakonzept-Attributregeln und -Aktionen folgende manuellen Aufgaben erledigt werden:

1. Erstellen von Korrespondenzen zwischen Instanzen aus lokalen Produktontologien (falls dies möglich ist). Dazu müssen z.B. Data Quality Engineers über entsprechendes Hintergrundwissen verfügen.

2. Löschen von Korrespondenzen (erfordert ebenfalls Hintergrundwissen).

3. Auflösung von Integrationskonflikten (Hintergrundwissen notwendig).

4. Ausschließen oder Löschen von Instanzen (Diese Aufgabe sollte bereits vor einer automatischen Intanzintegration erledigt werden).

Für alle Aufgaben sind klar definierte Änderungsoperationen notwendig, die beschreiben, wie Änderungen behandelt werden. Die Ausführung der Aufgaben kann durch unterschiedliche Personengruppen erfolgen. Während das Erstellen und Löschen von Korrespondenzen fachliches Hintergrundwissen erfordert, sollten diese Aufgaben idealerweise von den betroffenen System- und Komponentenverantwortlichen durchgeführt werden. Für alle Instanzen, die keinen Verantwortlichen dokumentiert haben, sind Data Quality Engineers dafür verantwortlich, entsprechende Ansprechpartner mit dem benötigten Hintergrundwissen manuell zu identifizieren.
Die Dokumentation der Verantwortlichen kann beispielsweise durch Metainformationen erfolgen. Diese Verantwortlichen werden im Anschluss an die automatische Instanzintegration über das Integrationsergebnis informiert. Durch diese Aufteilung kann der Aufwand der manuellen Integrationsaufgaben verteilt und beschleunigt werden.
Man beachte, dass Produktontologien hierarchisch aufgebaut sind; d.h., das manuelle Erstellen von Korrespondenzen kann dazu führen, dass die Auswertung von Schemakonzept-Attributregeln und -Aktionen erneut für einen Teilausschnitt einer Produktontologie ausgeführt werden muss. Dies wiederum führt dazu, dass möglicherweise neue Korrespondenzen gefunden werden, die erneut zu Integrationskonflikten führen, welche dann ebenfalls manuell aufgelöst werden müssen. Anders formuliert kann eine Änderung eine Vielzahl an abhängigen Änderungen nach sich ziehen (sog. *Ripple-Effect*).
Die angesprochenen Änderungsoperationen werden in Kapitel 8 detailliert. Zunächst werden im Kapitel 7 Metriken eingeführt, die es ermöglichen, eine vorgenommene Integration quantitativ zu messen. Um bestimmen zu können, wie lange manuelle Integrationsaufgaben, die nach einer initialen automatischen Integration anfallen, müssen die Aufgaben (z.B. Erstellung fehlender Korrespondenzen, Auflösung von Integrationskonflikten) quantitativ beziffert werden können. Dazu werden im Kapitel 7 unterschiedliche Vollständigkeits- und Konsistenzmetriken eingeführt.

7. Bestimmung der Integrationsqualität

Um den Integrationsprozess zu überwachen und bewerten, werden Metriken benötigt, welche die Qualität der resultierenden Integration messen [TBHR15]. In diesem Kapitel werden für diesen Zweck verschiedene Metriken entwickelt und erläutert.
Ziel der Produktdatenintegration ist die Unterstützung verschiedener Anwendungsfälle (vgl. Definition 8). Diese können integrierte Daten erst dann nutzen, wenn die Integration abgeschlossen ist, d.h. wenn diese *vollständig* und *konsistent* ist. Der Nutzungszeitraum integrierter Produktdaten wird zuvor, in Abstimmung zwischen den Nutzern des integrierten Datenbestandes und den Integrationsexperten, festgelegt. Um garantieren zu können, dass die Integration vollständig und konsistent zum definierten Nutzungszeitpunkt ist, muss der Integrationsprozess gesteuert und überwacht werden.
Eine Voraussetzung für eine automatische Instanzintegration ist die vollständige Abbildung von Schemakonzepten der lokalen Produktontologien auf entsprechende Konzepte der globalen (vgl. Abschnitt 6.2.3). Eine lokale Produktontologie wiederum kann genau dann automatisch auf eine globale Produktontologie abgebildet werden, wenn für jedes Schemakonzept mindestens eine Schemakonzept-Attributregel definiert ist (vgl. Definition 29). Diese Eigenschaft wird im Folgenden als *lokale Schemakonzept-Vollständigkeit* bezeichnet. Darüber hinaus lassen sich weitere Metriken definieren, welche die Vollständigkeit von Artefakten (z.B. Schemakonzepten, Instanzen und Attributen) lokaler und globaler Produktontologien bestimmen und damit für die Überwachung des Integrationsprozesses genutzt werden können.
Im Abschnitt 7.1 werden die Begriffe Vollständigkeit und Konsistenz im Kontext der Integration von Produktdaten erläutert, bevor schließlich Metriken für die lokale Produktontologie (siehe Abschnitt 7.2) und die globale Produktontologien (siehe Abschnitt 7.3) definiert werden. Schließlich wird in Abschnitt 7.4 beschrieben, wie mit Hilfe dieser Metriken der Integrationsprozess gesteuert und überwacht werden kann.

7.1. Vollständigkeit und Konsistenz

Nachfolgend werden zunächst allgemeine Definitionen für *Vollständigkeit* und *Konsistenz* gegeben, bevor in den anschließenden Abschnitten verschiedene Vollständigkeits- und Konsistenzmetriken für Produktontologien eingeführt werden.

Definition 42 (Vollständigkeit). Seien M_1 eine Menge von Artefakten und M_2 eine Referenzmenge mit $M_1 \subset M_2$. Ferner sei $q = \frac{|M_1|}{|M_2|} \in [0,1]$. Dann: M_1 heißt vollständig bezogen auf M_2 genau dann und nur dann, wenn $q = 1$ gilt.

Definition 43 (Konsistenz). Sei ein bestimmter Sachverhalt (Objekt) der realen Welt in unterschiedlichen Informationssystemen abgebildet. Dann sind die entsprechenden Beschreibungen konsistent, wenn sie, bezogen auf den Sachverhalt, den selben Zustand referenzieren. D.h., in allen Informationssystemen ist das Objekt konsistent beschrieben. Umgekehrt nennen wir die Beschreibung inkonsistent, wenn in mindestens einem Informationssystem nicht der aktuell gültige Realweltzustand des Objekts abgebildet ist.

Die folgenden Abschnitte beschreiben unterschiedliche Metriken, die für verschiedene Artefakte lokaler und globaler Produktontologien bestimmt werden können. Für jede Metrik werden Nutzen, Anwendung und Nutzungszeitpunkt beschrieben, gefolgt von einer formalen Definition.

7.2. Metriken für lokale Produktontologien

Eine Vollständigkeitsmetrik wurde bereits im Kontext der automatischen Instanzintegration (siehe Abschnitt 6.3) angewendet, ohne diese formal zu definieren. Die *lokale Schemakonzept-Vollständigkeit* gibt an, wann eine automatische Instanzintegration durchgeführt werden kann. Während diese Metrik eine notwendige Bedingung darstellt, garantiert sie nicht, dass möglichst vielen Korrespondenzen automatisch ermittelt werden können. Wie erwähnt können für parallel entwickelte Komponenten zwischen lokalen und globalen Produktontologien die zugehörigen Produktdaten einen unterschiedlichen Reifegrad aufweisen.

7.2.1. Lokale Schemakonzept-Vollständigkeit

Domänen- und Integrationsexperten sind für die Erstellung und Pflege der Abbildungs- und Integrationsregeln (SCARs und SCAAs) zwischen den Schemakonzepten der lokalen Produktontologien und der globalen Produktontologie verantwortlich. Der initiale Integrationsprozess einer lokalen Produktontologie in die globale Produktontologie kann erst ausgeführt werden, wenn für alle Schemakonzepte der lokalen Produktontologie jeweils mindestens eine Abbildungs- und Integrationsregel definiert ist. Folglich beschreibt die *lokale Schemakonzept-Vollständigkeit* (engl. *Local Schema Concept Completeness*) das Verhältnis der abgebildeten Schemakonzepte einer lokalen Produktontologie zu allen vorhanden Schemakonzepten dieser Produktontologie. Sind beide Mengen gleich, ist die lokale Produktontologie *lokal Schemakonzept-vollständig*.
Lokale Schemakonzept-Vollständigkeit bildet die formale Voraussetzung für die Ausführung der automatischen Instanzintegration. Sie wird von Integrationsexperten vor der automatischen Instanzintegration überwacht. Wir definieren die lokale Schemakonzept-Vollständigkeit wie folgt:

Definition 44 (Lokale Schemakonzept-Vollständigkeit). Die *lokale Schemakonzept-Vollständigkeit* (*LocalSchemaConceptCompleteness*) beschreibt das Verhältnis der Schemakonzepte für die eine Schemakonzept-Attributregel definiert ist (SC_{Mapped}) zu allen Schemakonzepten. Schemakonzepte, die von einer Integration ausgeschlossen sind (SC_L^{Ignore}), werden nicht berücksichtigt (vgl. Abschnitt 6.2.3).

$$LocalSchemaConceptCompleteness = \frac{|SC_{Mapped}|}{|SC_L \backslash SC_L^{Ignore}|} \in [0,1]$$
mit $SC_{Mapped} = \{sc \in SC_L \backslash SC_L^{Ignore} \mid \exists\,(sc_L, sc_G, AA) \in SCAR\}$
mit $sc_G \in SC_G$sowie $AA \in AttributeAction$

7.2.2. Lokale SCAR-Vollständigkeit

Eine weitere Metrik für lokale Produktontologien ist die *lokale SCAR-Vollständigkeit*. Sie beschreibt die Anzahl an Instanzen für welche die für die Auswertung durch SCARs benötigten Attributen gesetzt sind in Relation zur Anzahl aller Instanzen. Mithilfe dieser Metrik lässt sich bestimmen, ob Korrespondenzen zwischen lokalen und globalen Instanzen automatisch ermittelt werden können. Wir definieren die *lokale SCAR-Vollständigkeit* wie folgt:

Definition 45 (Lokale SCAR-Vollständigkeit). Die *lokale SCAR-Vollständigkeit* (*LocalSCARCompleteness*) beschreibt das Verhältnis der Anzahl von Instanzen, deren SCAR-Attribute gesetzt sind (Ind_{SCAR}), in Relation zur Anzahl der Instanzen von Ind_L (abzgl. der Menge ausgeschlossener Instanzen Ind_L^{Ignore}):

$$LocalSCARCompleteness = \frac{|Ind_{SCAR}|}{|Ind_L \backslash Ind_L^{Ignore}|} \in [0,1]$$
mit $Ind_{SCAR} = \{ind \in Ind_L \mid \exists sc \in SC_L\ \exists attr_L \in Attr_L\ \exists attrVal \in AttrVal_L$
$\exists (attr_L, attr_G, AMC) \in SCAR : (sc, attr_L) \in hasAttribute_L \wedge$
$(attrVal, attr_L) \in references_L \wedge (ind, attrVal) \in hasValue_L\}$

7.2.3. Lokale SCAA-Vollständigkeit

Die *lokale SCAA-Vollständigkeit* beschreibt die Anzahl der Instanzen, für welche die für SCAAs benötigte Attribute gesetzt sind, in Relation zu allen Instanzen (exklusive der ausgeschlossenen Instanzen). Während die *lokale SCAR-Vollständigkeit* bestimmt, ob Korrespondenzen zwischen lokalen und globalen Instanzen automatisch ermittelt werden können, bestimmt die *lokale SCAA-Vollständigkeit*, ob lokale Attribute in die globale Produktontologie integrierbar sind.

Definition 46 (Lokale SCAA-Vollständigkeit). Die *lokale SCAA-Vollständigkeit* (*LocalSCAACompleteness*) ist das Verhältnis der Anzahl von Instanzen, dessen SCAA-Attribute gesetzt sind ($|Ind_{SCAA}|$) in Relation zur Anzahl aller Instanzen $|Ind_L|$ ohne die ausgeschlossenen Instanzen $|Ind_L^{Ignore}|$:

$$LocalSCAACompleteness = \frac{|Ind_{SCAA}|}{|Ind_L \backslash Ind_L^{Ignore}|} \in [0,1]$$

mit

$$Ind_{SCAA} = \{ind \in Ind_L \mid \exists(sc_L, sc_G, (attr_{L_1}, attr_{G_1}, behavior_1), \ldots,$$
$$(attr_{L_n}, attr_{G_n}, behavior_n)) \in SCAA : \forall attr \in (attr_{L_1}, \ldots, attr_{L_n})$$
$$\exists attrVal_L \in AttrVal_L \wedge (attrVal_L, attr) \in references_L$$
$$\wedge (ind, attrVal_L) \in hasValue_L\}$$

7.2.4. Kombination aus SCAR- und SCAA-Vollständigkeit

Für die Integration von Instanzen einer lokalen Produktontologie ist die *lokale SCAR-Vollständigkeit* nur eine hinreichende Bedingung. Zwar können Instanzen andere Attribute für SCARs beinhalten, ohne jedoch Werte für SCAAs zu besitzen. Erst wenn ausreichend Instanzen neben Attributwerten für SCARs auch Werte für SCAAs besitzen, damit eine automatische Integration gestartet werden. Die *lokale SCAR- und SCAA-Vollständigkeit* definieren wir wie folgt:

Definition 47 (Lokale SCAR- und SCAA-Vollständigkeit). Die *lokale SCAR- und SCAA-Vollständigkeit* (*LocalSCARSCAACompleteness*) ist die Anzahl der Instanzen, deren SCAR-Attribute (Ind_{SCAR}) und SCAA-Attribute (Ind_{SCAA}) gesetzt sind, im Verhältnis zur Anzahl aller Instanzen (exklusive ausgeschlossener):

$$LocalSCARSCAACompleteness = \frac{|Ind_{SCAR} \cap Ind_{SCAA}|}{|Ind_L \backslash Ind_L^{Ignore}|}$$

7.2.5. Lokale Instanz-Vollständigkeit

Da die Integration der Instanzen nicht vollständig automatisiert werden kann, müssen Korrespondenzen von Instanzen aus lokalen Produktontologien und globaler Produktontologie manuell gepflegt werden. Um diesen manuellen Integrationsprozess zu überwachen, wird die *lokale Instanz-Vollständigkeit* (engl. *Local Individual Completeness*) ermittelt. Diese ist das Verhältnis der Anzahl von Instanzen, die mindestens eine Korrespondenz zu einer Instanz in der globalen Produktontologie haben, zu allen vorhandenen Instanzen der lokalen Produktontologie. Sind beide Mengen gleich, wird die lokale Produktontologie als *lokal Instanz-Vollständig* bezeichnet. Die Metrik wird sowohl von Integrationsexperten als auch dem Integration Board während des gesamten Integrationsprozess genutzt.

Definition 48 (Lokale Instanz-Vollständigkeit). Die *lokale Instanz-Vollständigkeit* (*LocalIndividualCompleteness*) ist das Verhältnis der Anzahl von Instanzen Ind_L^{Mapped} einer lokalen Produktontologie, für die mindestens eine Korrespondenz auf eine Instanz in der globalen Produktontologie besteht, zur Anzahl aller Instanzen der lokalen Produktontologie Ind_L (abzgl. der Anzahl ausgeschlossener Instanzen Ind_L^{Ignore}):

$$LocalIndividualCompleteness = \frac{|Ind_L^{Mapped}|}{|Ind_L \backslash Ind_L^{Ignore}|} \in [0,1]$$
$$\text{mit } Ind_{Mapped} = \{ind_L \in Ind_L \mid \exists\, (ind_L, ind_G, scar) \in Correspondences\}$$
$$scar \in SCAR \text{ sowie } ind_G \in Ind_G$$

7.3. Metriken für globale Produktontologien

Während die bisher vorgestellten Metriken auf lokale Produktontologien fokussieren, können in eine globale Produktontologie eine Vielzahl lokaler Produktontologien integriert sein. Vollständigkeit- und Konsistenzmetriken beziehen sich im Kontext einer globalen Produktontologie demzufolge potenziell auf eine Vielzahl an lokalen Produktontologien.

7.3.1. Globale Schemakonzept-Vollständigkeit

Die *globale Schemakonzept-Vollständigkeit* (engl. *Global Schema Concept Completeness*) bestimmt den Anteil derjenigen Schemakonzepte einer globalen Produktontologie, für die mindestens eine Schemakonzept-Attributregel definiert ist. Mit Hilfe dieser Metrik kann überprüft werden, ob alle Schemakonzepte der globalen Produktontologie tatsächlich verwendet werden. Wird die globale Produktontologie beispielsweise nach dem Top-Down-Ansatz modelliert (vgl. Abschnitt 6.2.4), sollte nach Abschluss der manuellen Integration für jedes Schemakonzept der globalen Produktontologie mindestens eine Schemakonzept-Attributregel vorhanden sein. Anderenfalls wurde entweder ein globales Schemakonzept modelliert, welches nicht benötigt wird, oder die Integration eines lokalen Schemakonzepts wurde übersehen.

Definition 49 (Globale Schemakonzept-Vollständigkeit). Für eine Menge an Produktontologie-Mappings $\{POMapping_{L_1 \times G}, \ldots, POMapping_{L_n \times G}\}$ ist die *globale Schemakonzept-Vollständigkeit* (*GlobalSchemaConceptCompleteness*) das Verhältnis globaler Schemakonzepte, für die mindestens eine SCAR definiert ist (SC_G^{Mapped}), zur Menge aller Schemakonzepte (SC_G):

$$GlobalSchemaConceptCompleteness = \frac{|SC_G^{mapped}|}{|SC_G|}$$
$$\text{mit } SC_G^{mapped} = \{sc_G^1, \ldots, sc_G^k \in SC_G \mid (sc_{L_1}^1, sc_G^1), (sc_{L_1}^2, sc_G^2), \ldots, (sc_{L_n}^m, sc_G^k)$$
$$\in SCAR_{L_1 x G} \bigcup \ldots \bigcup SCAR_{L_n \times G}\}$$

7.3.2. Globale Instanz-Vollständigkeit

Hauptziel der Integration von Produktdaten ist es, fachliche Anwendungsfälle zu unterstützen. Beispiele für solche Anwendungsfälle sind die Erstellung physischer Prototypen (z.B. Versuchsfahrzeug) oder Plausibilitätsüberprüfungen. Da die Erstellung physischer Prototypen sehr aufwändig und teuer ist, müssen die integrierten Produktdaten vollständig integriert werden, d.h. es dürfen keine Informationen fehlen. Aus diesem Grund müssen alle Instanzen aus der globalen

Produktontologie mit mindestens einer Instanz aus einer lokalen Produktontologie assoziiert werden. Die Vollständigkeit der Korrespondenzen von Instanzen aus der globalen Produktontologie in die lokalen Produktontologien wird als *globale Instanz-Vollständigkeit* (engl. *Global Individual Completeness*) bezeichnet. Formal gibt diese Metrik das Verhältnis derjenigen Instanzen der globalen Produktontologie, die über mindestens eine Korrespondenz zu einer lokalen Produktontologie verfügen, zu allen Instanzen der globalen Produktontologie.
Durch die Definition einer lokalen Masterproduktontologie ist für jede Instanz der globalen Produktontologie eine Korrespondenz vorhanden. Folglich müssen die Instanzen der lokale Masterproduktontologie von der Bestimmung ausgenommen werden. Werden Instanzen in der globalen Produktontologie manuell angelegt, ist dieses Kriterium weiter notwendig. Die Metrik wird von Integrationsexperten und dem Integration Bord während des gesamten Integrationsprozess überwacht.

Definition 50 (Globale Instanz-Vollständigkeit). Für eine Menge an Produktontologie-Mappings $\{POMapping_{L_1xG}, \ldots, POMapping_{L_nxG}\}$ ist die globale Instanzvollständigkeit (*GlobalIndividualCompleteness*) die Anzahl der Instanzen der globalen Produktontologie, für die mindestens eine Korrespondenz auf eine Instanz in einer lokalen Produktontologie vorhanden ist, in Relation zu allen Instanzen.

$$GlobalIndividualCompleteness = \frac{|Ind_G^{Mapped}|}{|Ind_G|} \in [0,1]$$

Die Menge der Instanzen einer globalen Produktontologie Ind_L^{Mapped}, für die mindestens eine Korrespondenz auf eine lokale Instanz vorhanden ist wie folgt definiert:

$$Ind_G^{Mapped} = \{ind \in Ind_G \mid \exists\,(ind_L, ind, scar) \in Correspondences\}$$
$$\text{mit } scar \in SCAR \text{ sowie } ind_L \in Ind_L$$

7.3.3. Globale Instanzattribut-Vollständigkeit

Produktdaten entwickeln sich über die Zeit. Daher werden die einzelnen Attribute zu unterschiedlichen Zeitpunkten erfasst und dokumentiert. D.h., bei der initialen Integration fehlen ggf. Attributwerte für die Auswertung von SCARs oder SCAAs können nicht ausgeführt werden. Da diese Attribute ggf. für Anwendungsfälle benötigt werden, müssen sie auf Vollständigkeit überprüft werden. Die *globale Instanzattribut-Vollständigkeit* (engl. *Global Individual Attribute Completeness*) ist das Verhältnis von Instanzen einer globalen Produktontologie, die für jedes Attribute einen Wert besitzen, zu allen Instanzen dieser Produktontologie. Die Metrik wird sowohl von Integrationsexperten als auch vom Integration Board während des gesamten Integrationsprozesses kontrolliert.

Definition 51 (Globale Instanzattribut-Vollständigkeit).
Für eine Menge an Produktonto-logie-Mappings $\{POMapping_{L_1\times G}, \ldots, POMapping_{L_n\times G}\}$ ist die *globale Instanzattribut-Vollständigkeit* (*GlobalIndividualAttributeCompleteness*) das

Verhältnis der Anzahl globaler Instanzen für die alle Attribut einen Wert besitzen ($|Ind_G^{AttrVal}|$), zur Anzahl aller globalen Instanzen ($|Ind_G|$):

$$GlobalIndividualAttributeCompleteness = \frac{|Ind_G^{AttrVal}|}{|Ind_G|} \in [0,1]$$

mit $Ind_G^{AttrVal} = \{ind \in Ind_G \mid \exists sc_G \in SC_G \, \exists(sc, ind) \in hasIndividual$
$\exists(attr_1, \ldots, attr_n) \in Attr_G$ und es gilt $\forall attr \in (attr_1, \ldots, attr_n) \, \exists attrVal \in AttrVal_G$
$\wedge \, \exists(attrVal, attr) \in references_G \wedge \; \exists(ind, attrVal) \in hasValue_G\}$

7.3.4. Globale Instanzattribut-Konsistenz

Entwickler und Ingenieure arbeiten in der Regel mit lokalen Kopien von Produktdaten. Haben diese einen bestimmten Entwicklungsstand erreicht, werden sie über Änderungsmanagementprozesse bewertet. Freigegebene Produktdaten werden anschließend im Produktdatenmanagementsystem abgelegt. Nicht alle Applikationen (z.B. bereichsspezifische Spezialapplikationen) unterliegen einem zentralen Änderungsmanagement, bzw. Änderungen werden erst ab einem definierten Zeitpunkt im Entwicklungsprozess zentral verwaltet. So kann es vorkommen, dass in unterschiedlichen Informationssystemen unterschiedliche Entwicklungsstände von Produktdaten dokumentiert sind.
Nach der initialen Integration können Korrespondenzen von Instanzen mehrerer lokaler Produktontologien auf eine einzelne Instanz in der globalen Produktontologie existieren. Sind in den jeweiligen Schemakonzepten SCAAs definiert, die in das selbe Attribut in der globalen Produktontologie integrieren, kann es zu Integrationskonflikten kommen. Dies ist z.B. der Fall, wenn ein Produktaspekt in mehreren Informationssystemen unterschiedlich dokumentiert wird. Wird in einem dieser Informationssysteme der Attributwert geändert, kann es vorkommen, dass diese Änderung nicht an die anderen Informationssysteme weiter propagiert wird. Folglich wird eine Metrik benötigt, um derartige Integrationskonflikte zu erkennen.
Die *globale Instanzattribut-Konsistenz* (engl. *Global Individual Attribute Consistency*) ist das Verhältnis der Anzahl von Instanzen einer globalen Produktontologie, deren Attributwerte konsistent sind, d.h. die keine Integrationskonflikte aufweisen, zur Anzahl aller Instanzen der Ontologie. Eine Instanz ist *inkonsistent*, wenn es mindestens zwei SCAAs aus zwei unterschiedlichen lokalen Produktontologien gibt, die auf das gleiche Attribut eines Schemakonzeptes in der globalen Produktonologie verweisen und es mindestens zwei Instanzen aus diesen lokalen Produktontologien gibt, die auf die selbe Instanz in der globalen Produktontologie verweisen und unterschiedliche Attributwerte für die SCAAs besitzen. Sind die Attributwerte aller Instanzen der globalen Produktontologie konsistent, wird diese als *global Instanzattribut-Konsistent* bezeichnet. Die Metrik wird sowohl von Integrationsexperten als auch dem Integration Board während des gesamten Integrationsprozesses überwacht. Sie ist folgendermaßen definiert.

Definition 52 (Globale Instanzattribut-Konsistenz). Für eine Menge an Produktontologie-Mappings $\{POMapping_{L_1 \times G}, \ldots, POMapping_{L_n \times G}\}$ ist die *globale Instanzattribut-Konsistenz* (GlobalIndividualAttributeCompleteness) des Verhältnis der Anzahl inkonsistenter globalen Instanzen ($|Ind_G^{Inconsistent}|$) zur Anzahl aller globalen Instanzen ($|Ind_G|$):

$$GlobalIndividualAttributeCompleteness = 1 - \frac{|Ind_G^{Inconsistent}|}{|Ind_G|} \in [0,1]$$
mit $Ind_G^{Inconsistent} = \{ind \in Ind_G \mid \exists(ind_{L_1}, ind), \ldots, (ind_{L_n}, ind) \in Correspondences$
$\exists(attr_L, attr_G, AA) \in SCAA\ \exists attrVal_{L_1}, \ldots, attrVal_{L_m} \in AttrVal_L$
$\exists(attrVal_{L_1}, attr_L), \ldots, (attrVal_{L_m}, attr_L) \in references_L$
$\exists(ind_{L_1}, attrVal_{L_1}), \ldots, (ind_{L_m}, attrVal_{L_m}) \in hasValue_L$
und es gilt $\mid (attrVal_{L_1}, ldots, attrVal_{L_m})| > 1\}$für $m \leqslant n$

7.4. Integrationsplanung und -überwachung

Die manuelle Integration von Produktdaten erfordert, je nach Anzahl der Instanzen und Attributwerte, eine gewisse Zeit. Um den Integrationsprozess zu steuern und ihn zu überwachen, ist es notwendig, zuvor definierte Soll-Zustände der Integrationsmetriken mit vorhandenen Ist-Zuständen abzugleichen. Bei Abweichungen zwischen Soll- und Ist-Zuständen einzelner Metriken können dann geeignete Gegenmaßnahmen getroffen werden. Dies könnte z.B. die Erhöhung der Mitarbeiterzahl oder die Verschiebung des Nutzungszeitpunktes sein. Die Definition von Soll-Zuständen einzelner Metriken geschieht durch das Integration Bord in Zusammenarbeit mit den Fach- sowie Integrationsexperten. Ein Soll-Wert einer Integrationsmetrik ist wie folgt definiert:

Definition 53 (Soll-Wert). Für eine Menge an Produktontologie-Mappings $\{POMapping_{L_1 \times G}, \ldots, POMapping_{L_n \times G}\}$ ist ein Soll-Wert (*MetricReferenceValue*) ein Tupel
$(M, T, V) \in MetricReferenceValue$ mit

- $M \in \{LocalSCCompleteness, LocalSCARCompleteness, LocalSCAACompleteness, GlobalSchemaConceptCompleteness, \ldots\}$ ist eine Integrationsmetrik,
- T ist ein diskreter Zeitpunkt,
- $V \in [0,1]$ ist ein reeller Wert.

Die Metriken zur Bewertung der Integrationsqualität betrachten den Stand der Integration zum aktuellen Zeitpunkt. Sie werden jedoch in unterschiedlichen Phasen der Produktdatenintegration benötigt. Der Prozess lässt sich in zwei Phasen unterteilen, die durch jeweils zwei Zeitpunkte t_1 und t_2 abgeschlossen werden.
Zum Zeitpunkt t_0 beginnt die **Phase 1** mit der manuellen Schemaintegration. Hier werden folgende Metriken benötigt: *Lokale Schemakonzept-Vollständigkeit*, *Lokale SCAR-SCAA-Vollständigkeit* und *Globale Schemakonzept-Vollständigkeit*. Die erste Phase endet mit dem **Zeitpunkt** t_1, zu dem die automatische Instanzintegration gestartet wird. Der Zeitpunkt der automatischen Instanzintegration ist fest definiert. Um diesen Zeitpunkt einhalten zu können, müssen Soll-Zustände für die vorangehend genannten Metriken definiert werden. Bei Abweichungen zwischen Ist- zu Soll-Zuständen an definierten Zeitpunkten wiederum können geeignete Maßnahmen beschlossen werden, um den Termin der automatischen Instanzintegration einhalten zu können.

Voraussetzung für die automatische Instanzintegration sind folgende Werte für die einzelnen Metriken:

- $LocalSCCompleteness = 1$, d.h. für all Schemakonzepte aller lokalen Produktontologien sind SCARs definiert.
- $LocalSCARSCAACompleteness \geqslant threshold$, d.h. die Anzahl der Instanzen, die Attributwerte besitzen, die für das Matching sowie die Integration notwendig sind, überschreitet einen definierten Schwellenwert (*threshold*).

Damit zum Zeitpunkt t_1 diese Zustände erreicht werden, ist es notwendig, den Verlauf der Metriken zu überwachen. Dazu werden Soll-Zustände $s_0^1, \ldots, s_n^1$ zu Zeitpunkten zwischen t_0 und t_1 definiert. An diesen Zeitpunkten werden Soll- und Ist-Zustände der definierten Metriken abgeglichen. Soll- und Ist-Zustände lassen sich als Graph darstellen. Abbildung 7.1 illustriert beispielhaft den zeitlichen Verlauf von Ist-Zuständen für verschiedene Integrationsmetriken vom Zeitpunkt t_0 bis zur automatischen Instanzintegration zum Zeitpunkt t_1.

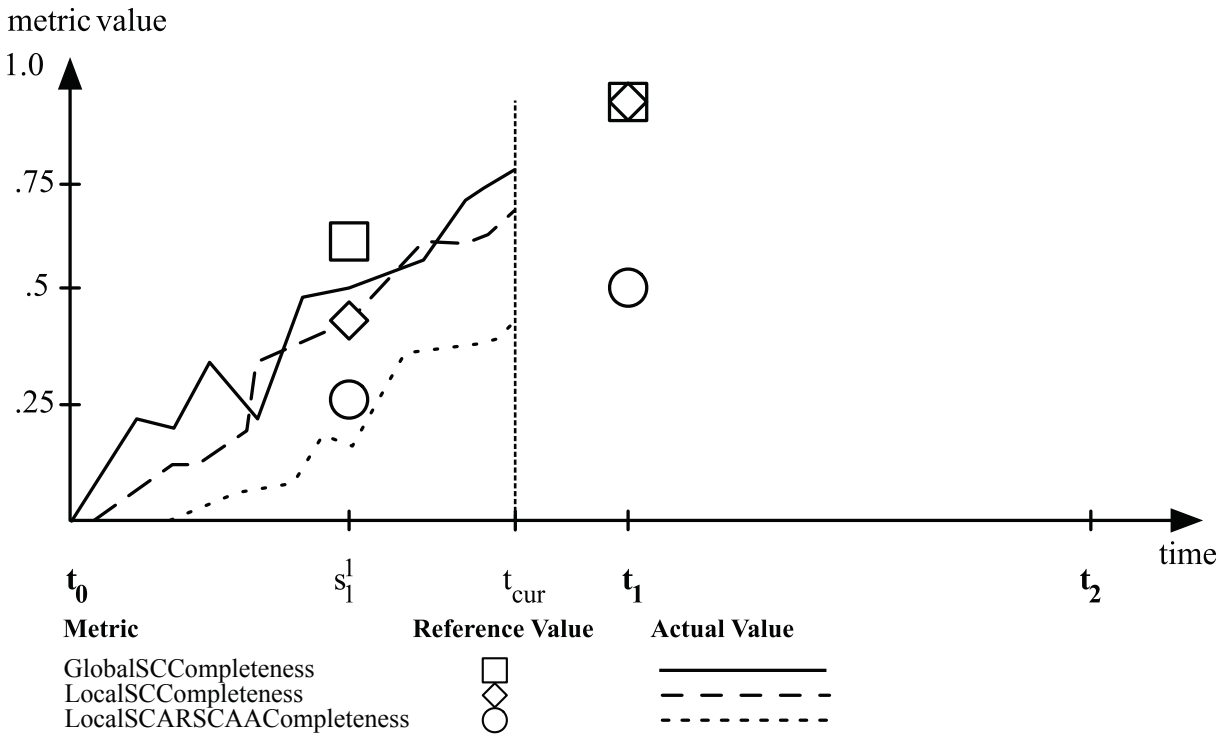


Abbildung 7.1.: Zeitlicher Verlauf der Metriken in Phase 1

Die Zustände für t_1 und t_2 sowie die Zeitpunkte der Soll-Zustände werden durch das *Integration Bord* in Zusammenarbeit mit Integrationsexperten definiert. Die Zustände basieren auf Erfahrungen dieser Experten, d.h. in den ersten Integrationsprojekten können die Abweichungen zwischen Soll- und Ist-Zuständen signifikant sein.
In Abbildung 7.1 sind, neben dem Verlauf der Ist-Stände über die Zeit zwischen t_0 und t_1, drei Soll-Zustände für folgenden Integrationsmetriken definiert: [1]

[1] Soll-Zustände können für jede Integrationsmetrik zu verschiedene Zeitpunkten definiert werden.

- $GlobalSCCompleteness = .53$
- $LocalSCCompleteness = .42$
- $LocalSCARSCAACompleteness = .28$

Während der Ist-Zustand der *lokalen Schemakonzept-Vollständigkeit* zum Zeitpunkt s_1^1 in etwa dem definierten Referenzwert entspricht, sind die Ist-Zustände der *globalen Schemakonzept-Vollständigkeit* sowie die lokale *SCAR-SCAA-Vollständigkeit* deutlich unter den definierten Soll-Zuständen. Zum Zeitpunkt s_1^1 analysiert das Integration Bord die Ist- und Soll-Zustände der verschiedenen Metriken und definiert ggf. notwendige Maßnahmen für den Integrationsprozess. Schließlich sind zum Zeitpunkt t_1 folgende Soll-Zustände definiert:

- $GlobalSCCompleteness = 1$
- $LocalSCCompleteness = 1$
- $LocalSCARSCAACompleteness = 0.5$

Nach der automatischen Instanzintegration beginnt die **Phase 2** mit der manuellen Instanzintegration, in der folgende Metriken für die *Globale Instanz-Vollständigkeit* und *Globale Instanz-Konsistenz* benötigt werden. Analog zur ersten Phase werden auch für die manuelle Instanzintegration Soll-Zustände der einzelnen Metriken zu verschiedenen Zeitpunkten definiert. Die zweite Phase endet mit dem Zeitpunkt t_2 ab dem die Nutzung beginnt. Um zum Zeitpunkt t_2 mit der Nutzung der integrierten Produktdaten beginnen zu können, ist es auch für die zweite Phase notwendig, Soll-Zustände $s_0^2, \ldots, s_m^2$ zu Zeitpunkten zwischen t_1 und t_2 zu definieren. Abbildung 7.2 illustriert den Verlauf des vorherigen Beispiels ab dem Zeitpunkt t_1 bis zur Nutzung ab Zeitpunkt t_2.
Zwischen t_1 und t_2 sind zwei Zeitpunkte s_1^2 und s_2^2 definiert, zu denen jeweils Soll-Zustände für die *globale Instanz-Vollständigkeit* sowie *globale Instanz-Konsistenz* definiert sind. Während der Ist-Zustand für die *globale Instanz-Konsistenz* zum Zeitpunkt s_1^2 etwa dem Soll-Zustand entspricht, ist der Ist-Zustand der *globalen Instanz-Vollständigkeit* geringer. Analog zur ersten Phase analysiert und bewertet das Integration Bord die Ist- und Soll-Zustände zu den einzelnen Zeitpunkten. Voraussetzung für die Anwendungsfallnutzung sind folgende Soll-Zustände zum Zeitpunkt t_2:

- $GlobalIndCompleteness = 1$
- $GlobalIndConsistency = 1$

Für das Beispiel aus Abbildung 7.2 sind in Tabelle 7.1 die Soll-Zustände für die Produktontologie-Mappings $POMapping_{L_1 \times G}, \ldots, POMapping_{L_n \times G}$ zusammengefasst:

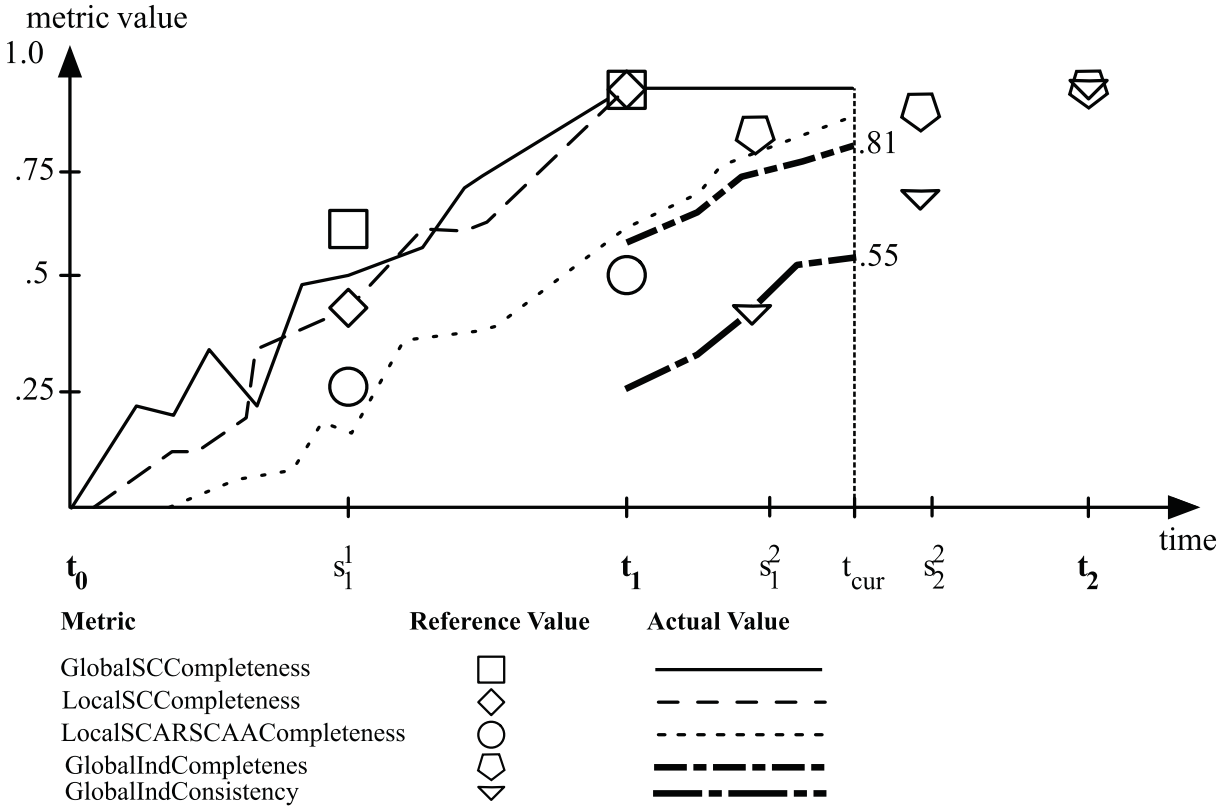


Abbildung 7.2.: Zeitlicher Verlauf der Metriken in Phase 2

Metrik	**Zeitpunkt**	**Soll-Zustand**
GlobalSCCompleteness	s_1^1	0.53
LocalSCCompleteness	s_1^1	0.42
LocalSCARSCAACompleteness	s_1^1	0.28
GlobalSCCompleteness	t_1	1
LocalSCCompleteness	t_1	1
LocalSCARSCAACompleteness	t_1	0.5
GlobalIndCompleteness	s_1^2	0.4
GlobalIndConsistency	s_1^2	0.84
GlobalIndCompleteness	s_1^2	0.7
GlobalIndConsistency	s_1^2	0.9
GlobalIndCompleteness	t_2	1
GlobalIndConsistency	t_2	1

Tabelle 7.1.: Soll-Zustände für das Beispiel aus Abbildung 7.2

8. Evolution integrierter Produktdaten

Produktdaten unterliegen kontinuierlichen Änderungen, sei es infolge von Optimierungen oder aufgrund neuer Standards, Regularien oder Gesetze [RW12, MRB08]. Folglich sollte ein Produktdatenintegrationsframework die Evolution von bereits integrierten Schemata und Instanzen unterstützen muss. Im Folgenden werden Änderungsszenarien beschrieben, die nach einer initialen Integration auftreten können. Für alle Szenarien wird angenommen, dass die Integrationsmenge vollständig und konsistent ist. Die in Kapitel 6 beschrieben Prozesse definieren die notwendigen Aufgaben der einzelnen Rollen, um Produktdaten vollständig und konsistent zu integrieren. Da Produktdaten kontinuierlichen Änderungen unterliegen ist diese statische Betrachtung nicht ausreichend. Vielmehr muss das PROCEED-Rahmenwerk die Möglichkeit bieten, sowohl auf Schema- als auch Instanzänderungen einzelner lokaler Produktontologien reagieren zu können.

Die in diesem Kapitel beschriebenen Änderungsprozesse sind reaktiv, d.h. wenn eine Änderung an Instanzen bzw. Schemakonzepten vorgenommen wird, werden diese vom PROCEED-Rahmenwerk erkannt. Anschließend werden Aktionen durchgeführt, um die Konsistenz und Vollständigkeit der globalen Produktontologie wieder herzustellen.

Die Attribute von Produktdaten werden meist zu unterschiedlichen Zeitpunkten festgelegt. Bei der initialen Integration etwa können Attributwerte fehlen, die für die Integration von Instanzen notwendig sind, (z.B. zwecks Auswertung von SCARs oder Ausführung von SCAAs). Ein weiteres Szenario für die Änderung von Instanzattributwerten ist die Beseitigung von Fehlern.

Neben Änderungen auf Instanzebene kann es vorkommen, dass die Datenmodelle lokaler Informationssysteme angepasst werden sollen, etwa nach der Erweiterung von Informationssystemen um neue Funktionen. Diese Änderungen an verschiedenen Artefakten lokaler Produktontologien (Schemakonzepte, Instanzen, Korrespondenzen, Abbildungs- und Integrationsregeln) haben Einfluss auf den Zustand der aktuellen Integration. Einige Änderungen lassen sich automatisieren, während andere manuelle Interaktionen erfordern.

Zusätzlich zu Änderungen an Produktdaten kann es notwendig werden, Änderungen an Konzepten oder gar Abbildungsregeln und -aktionen vorzunehmen. Im Folgenden werden die Änderungsoperationen, inklusive der notwendigen Schritte, beschrieben, um diese Änderungen zu realisieren.

Zunächst wird in Abschnitt der sog. *Ripple-Effect* erläutert. Demnach kann eine einzelne Änderung eine Welle weiterer Änderungen bedingen, ohne dass letztere zu Beginn der Änderung direkt ersichtlich waren. Im Abschnitt 8.2 werden Änderungsoperationen auf Datenebene, d.h. Änderungen für Instanzen, Korrespondenzen und Attributwerte behandelt und zwar sowohl für lokale als auch globale Produktontologien. Abschnitt 8.3 befaßt sich mit Änderungsoperationen auf Schemaebene, d.h. mit Änderungen an Schemakonzepten, Attributen, Schemakonzept-Attributregeln und -Attributaktionen.

8.1. Ripple-Effekt

Änderungen im Allgemeinen und das Löschen einer Korrespondenz im Speziellen können Auswirkungen auf weitere Korrespondenzen haben. Abbildung 8.1 zeigt drei unterschiedliche Szenarien für das Löschen von Korrespondenzen. Die Korrespondenz cor_2 zwischen lokaler Instanz `B` und globaler Instanz `Y` einer Produktdatenintegrationsebene `Layer` wird gelöscht. Auf der darunter liegenden Produktdatenintegrationsebene `Layer+1` befindet sich eine Instanz `B1`, die `B` zugeordnet ist. Für letztgenannte Instanz existieren korrespondierende globale Instanzen `Y1`, `Y2` und `Y3` über die Korrespondenzen `cor5`, `cor6` und `cor7`. Nachdem die Korrespondenz `cor2` gelöscht wurde, müssen folglich auch die Korrespondenzen `cor5`, `cor6` und `cor7` gelöscht werden. Existieren zu `B1` wiederum Instanzen auf der Produktdatenintegrationsebene unterhalb von `Layer+1` wird dieser Vorgang auch für diese Instanzen fortgesetzt.

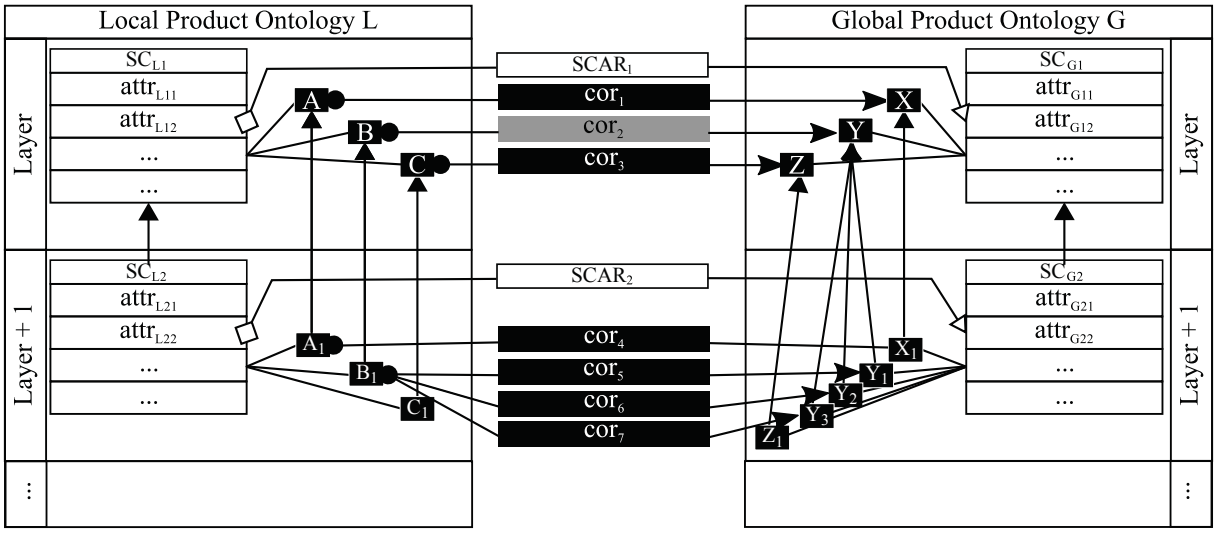


Abbildung 8.1.: Kaskadierender Effekt beim Löschen von Korrespondenzen

Änderungen wirken sich nicht nur kaskadierend innerhalb einer lokalen Produktontologie aus, sondern können sich auch über mehrere lokale Produktontologien erstrecken. Dies ist dann der Fall, wenn Attribute globaler Schemakonzepte durch SCAAs einer lokalen Produktontologie befüllt werden und gleichzeitig durch SCARs genutzt werden. In Abbildung 8.2 ist ein solches Beispiel dargestellt. Wird der Attributwert `REF` von `ind_M_1` geändert, muss diese Änderung in die globale Produktontologie propagiert werden. Das geänderte Attribut wird durch `SCAR2` und `SCAR3` der lokalen Produktontologien `L1` und `L2` verwendet. Folglich müssen die bestehende Korrespondenzen `cor2` und `cor3` gelöscht und die SCARs `SCAR2` und `SCAR3` erneut ausgewertet werden. Ggf. werden nun SCAAs (`SCAA2, SCAA3`) wiederholt ausgeführt. Infolge dieser neuen Ausführung werden in der globalen Produktontologie möglicherweise Attributwerte geändert (z.B. `Name` und `Requirement`), die wiederum durch SCARs (`SCAR3`) weiterer lokaler Produktontologien (`L2`) verwendet werden. Durch diese Änderung müssen existierende Korrespondenzen zwischen lokaler Produktontologie `L1` und globaler Produktontologie `G`, die durch die Schemakonzept-Attributregel `SCAR3` zustande gekommen sind, ggf. gelöscht bzw. neue Korrespondenzen erstellt werden.

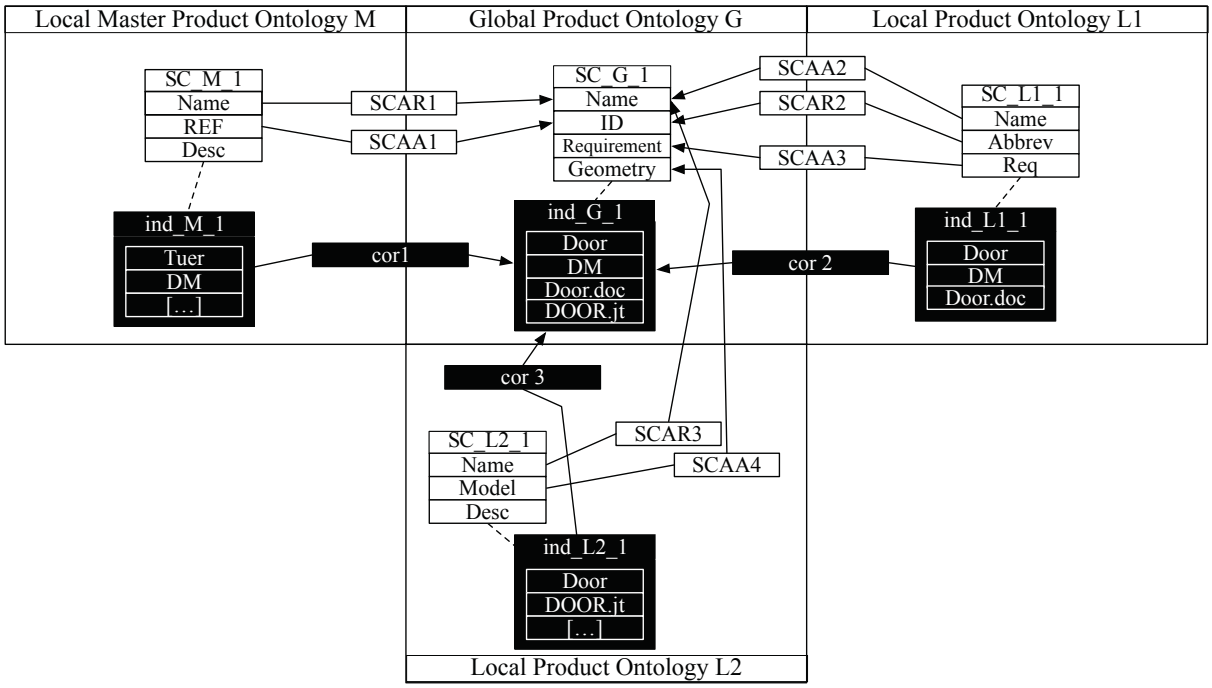


Abbildung 8.2.: Auswirkungen einer Instanzänderung

Die Auswirkungen von Änderungen lassen sich grafisch in Form von *ripple*-Graphen darstellen (siehe Abbildung 8.3). Im Zentrum befindet sich die Instanz der initialen Änderung `ind_M_1`. Auf dem nächst äußeren Ring befinden sich die Instanzen, die direkt von der Änderung betroffen sind (`ind_G_1`). Änderungen, die durch direkten Instanzen hervorgerufen werden, werden auf den nächst äußeren Ringen dargestellt (`ind_L1_1`, `ind_L2_1`).

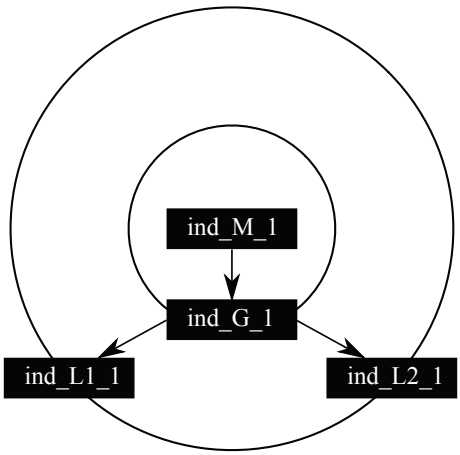


Abbildung 8.3.: Ripple-Graph für die Darstellung einer Instanzänderung

8.2. Instanzänderungen

Im Folgenden wird der Ablauf von Änderungen mittels BPMN[1]-Prozessmodellen illustriert. Abbildung 8.4 zeigt die verwendeten Symbole. Ein Prozess beginnt mit einem Startereignis und terminiert mit Erreichen eines Endereignisses. Es wird zwischen *User Tasks* und *Service Tasks* unterschieden. Während erstere die Interaktion mit Benutzern erfordern, laufen letztere automatisch ab. Zwischen den Prozessbeteiligten können Nachrichten ausgetauscht und so weitere Prozesse gestartet werden.
Bedingte Aufsplittung des Kontrollflusses werden in den Prozessmodellen über *XOR-Gateways* geregelt. Schließlich wird das Element *Zeitereignis* verwendet, das nach einer bestimmten Zeit ausgelöst wird.

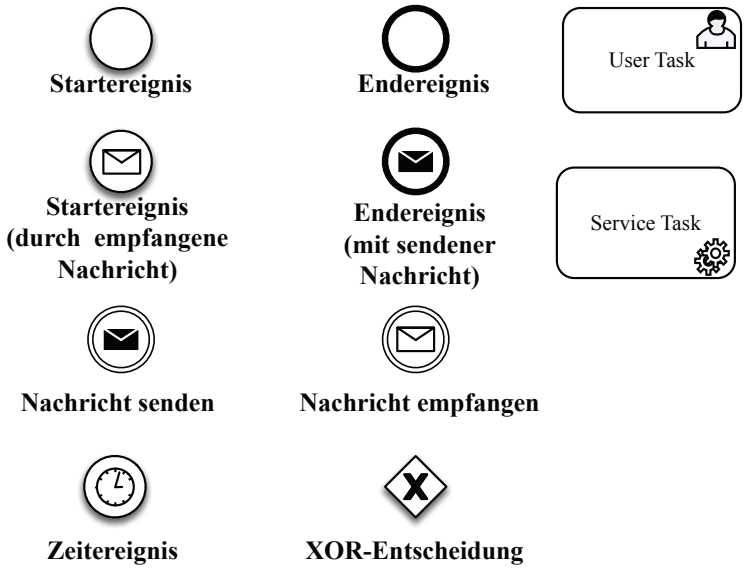


Abbildung 8.4.: Legende der BPMN Symbole

Im Folgenden werden Änderungen beschrieben, die sich auf Daten lokaler und globaler Produktdaten (etwa Korrespondenzen, Instanzen, Attributwerte) auswirken.

8.2.1. Hinzufügen einer Korrespondenz

Abbildung 8.5 zeigt die notwendigen Schritte nach Erstellung einer neuen Korrespondenz durch einen Data Quality Engineer (vgl. Abschnitt 6.2.5). Zunächst werden alle SCAAs des entsprechenden Schemakonzeptes der lokalen Produktontologie ausgeführt. Kommt es bei deren Ausführung zu Konflikten, die nicht automatisch aufgelöst werden können (siehe Abschnitt 6.2.3), muss ein Data Quality Engineer eingreifen. Danach werden diejenigen Instanzen der lokalen Produktontologie ermittelt, die sich auf der nächst tieferen Produktdatenintegrationsebene befinden. Wurde die unterste Ebene erreicht, ist die Änderung durchgeführt und der Data Quality Engineer wird entsprechend informiert. Anderenfalls werden für alle ermittelten Instanzen der nächst tiefe-

[1]Business Process Model and Notation (BPMN) ist eine grafische Sprache zur Modellierung von Geschäftsprozessen.

ren Ebene die SCARs des entsprechenden Schemakonzeptes ausgewertet und Korrespondenzen erstellt. Dadurch beginnt der Prozess wieder mit der Aktivität *SCAAs ausführen*.
Konnten für einzelne Instanzen keine Korrespondenzen durch die Regeln automatisch bestimmt werden, muss der Data Quality Engineer diese manuell erstellen oder einzelne Instanzen von der Integration ausschließen. Für jede gefundene Instanz werden die SCAAs ausgeführt, der Prozess beginnt dann wieder von vorne bis die letzte Produktdatenintegrationsebene erreicht worden ist.

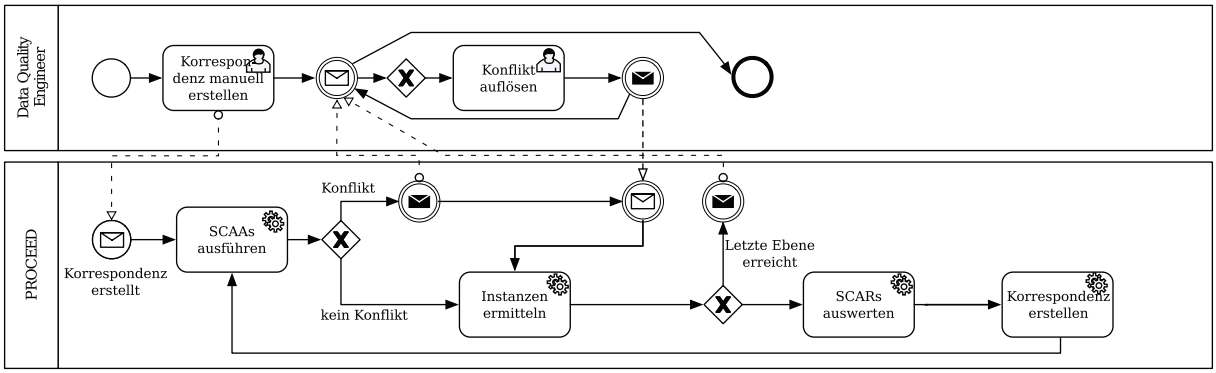


Abbildung 8.5.: Manuelles Anlegen einer Korrespondenz

Abbildung 8.6 illustriert die verschiedenen Artefakte, die von der Änderungsoperation betroffen sind. Die zu erstellende Korrespondenz *cor* wird für eine Instanz $ind_{L_{11}}$ eines Schemakonzepts sc_{L_1} und eine Instanz $ind_{G_{11}}$ eines Schemakonzeptes sc_{G_1} erstellt ①. Zunächst werden die entsprechenden SCAAs für die Instanz $ind_{L_{11}}$ ausgeführt ②. Anschließend werden die Instanzen Ind_{sub}, die zu $ind_{L_{11}}$ gehören und auf der nächst tieferliegenden Produktdatenintegrationsebene liegen, ermittelt ③. Für Ind_{sub} werden die SCARs ausgewertet und entsprechende Korrespondenzen erstellt ④. Für jede Korrespondenz werden die SCAAs ausgeführt ⑤. Die letzten drei Schritte werden solange wiederholt, bis die letzte Produktdatenintegrationsebene erreicht wird.

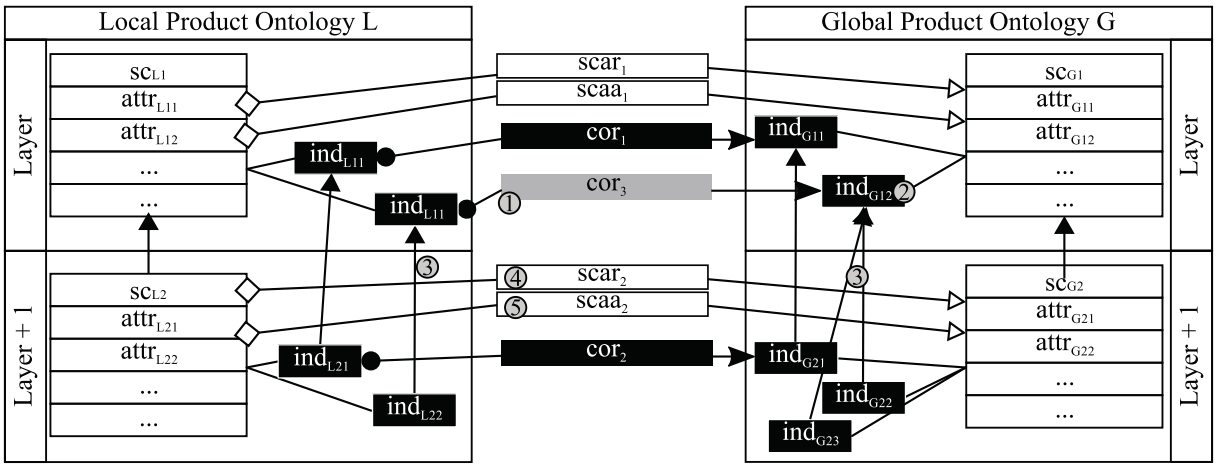


Abbildung 8.6.: Manuelles Erstellen neuer Korrespondenzen

Algorithmus 4 formalisiert den zuvor beschriebenen Ablauf. Dabei wird angenommen, dass neben der Korrespondenz *cor* eine Begründung erstellt wurde. Anschließend müssen für die Instanzen der Korrespondenz alle Schemakonzept-Attributaktionen ausgeführt werden (siehe Zei-

len 3 bis 6). Für die neue Korrespondenz wird nun überprüft, ob die betroffene Instanz sich nicht auf der Version-Integrationsebene befindet (siehe Zeile 9). Anschließend wird die Menge $IndCandidates_L$ aller lokalen Instanzen ermittelt, die sich in der nächst tiefer gelegenen Integrationsebene befinden und zur lokalen Instanz der Korrespondenz gehören. Analog enthält die Menge $IndCandidates_G$ die betroffenenden globalen Instanzen. Für das kartesische Produkt dieser beiden Mengen werden die entsprechenden Schemakonzept-Attributregeln ausgewertet (vgl. Zeile 17) und bei Treffern Korrespondenzen erstellt und Schemakonzept-Attributaktionen ausgeführt (Zeilen 20 und 21). Für die gefundenen Korrespondenzen wird der zuvor beschriebene Vorgang (Zeilen 9 bis 29) so lange wiederholt, bis die letzte Produktdatenintegrationsebene erreicht wird.

Algorithmus 4 (Korrespondenz hinzufügen)

```
1:  procedure AddCorrespondence(cor ∈ Correspondence)
2:      SCAA_cur = {scaa ∈ SCAA | ∃sc ∈ SC_L ∃(sc, cor.ind_L) ∈ hasIndividual_L
            ∃(sc, sc_G, AA) ∈ SCAA mit sc_G ∈ SC_G}
3:      for each scaa ∈ SCAA_cur do
4:          scaaResult = executeSCAA(cor, scaa)                          ▷ ②
5:          AttrValJustification = AttrValJustification ⋃
                scaaResult.AttrValJustification
6:      end for
7:      Correspondences^cur = cor
8:      for each cor^cur ∈ Correspondences^cur do
9:          if ! cor^cur.ind_L ∈ Ind_L^version then
10:             IndCandidates_L = (ind ∈ Ind_L | ∃(ind, cor^cur.ind_L) ∈ indBelow_L)   ▷ ③
11:             IndCandidates_G = (ind ∈ Ind_G | ∃(ind, cor^cur.ind_G) ∈ indBelow_G)
12:             for each ind_L^cur ∈ IndCandidates_L do
13:                 for each ind_G^cur ∈ IndCandidates_G do
14:                     sc_L^cur = {sc ∈ SC_L | ∃(sc, ind_L^cur) ∈ hasIndividual_L}
15:                     sc_G^cur = {sc ∈ SC_G | ∃(sc, ind_G^cur) ∈ hasIndividual_G}
16:                     for each scar^cur = (sc_L^cur, sc_G^cur, AMC) ∈ SCAR do
17:                         if evaluateAMC(ind_L^cur, ind_G^cur, scar^cur.AMC) then
18:                             newCor = (ind_L^cur, ind_G^cur, scar^cur)        ▷ ④
19:                             Justification = Justification ⋃(newCor, scar^cur)
20:                             Correspondences = Correspondences ⋃ newCor
21:                             scaaResult = executeSCAA(newCor)                 ▷ ⑤
22:                             AttrValJustification = AttrValJustification ⋃
                                    scaaResult.AttrValJustification
23:                             Tasklist = Tasklist ⋃ scaaResult.Tasklist
24:                         end if
25:                     end for
26:                 end for
27:             end for
28:         end if
29:     end for
30: end procedure
```

8.2.2. Löschen einer Korrespondenz

Abbildung 8.7 zeigt notwendige Schritte nach Löschen einer Korrespondenz durch einen Data Quality Engineer. Zunächst werden alle SCAAs des entsprechenden Schemakonzeptes in den korrespondierenden globalen Instanzen rückgängig gemacht. Danach werden die zugeordneten Instanzen der lokalen Produktontologie ermittelt, die sich auf der nächst tieferen Produkdatentintegrationsebene befinden. Für jede dieser Instanzen werden alle vorhandenen Instanzen gelöscht und der Prozess geht von vorne los, solange bis die letzte Ebene erreicht wird. Danach ist die Änderung abgeschlossen und der Data Quality Engineer wird entsprechend informiert.

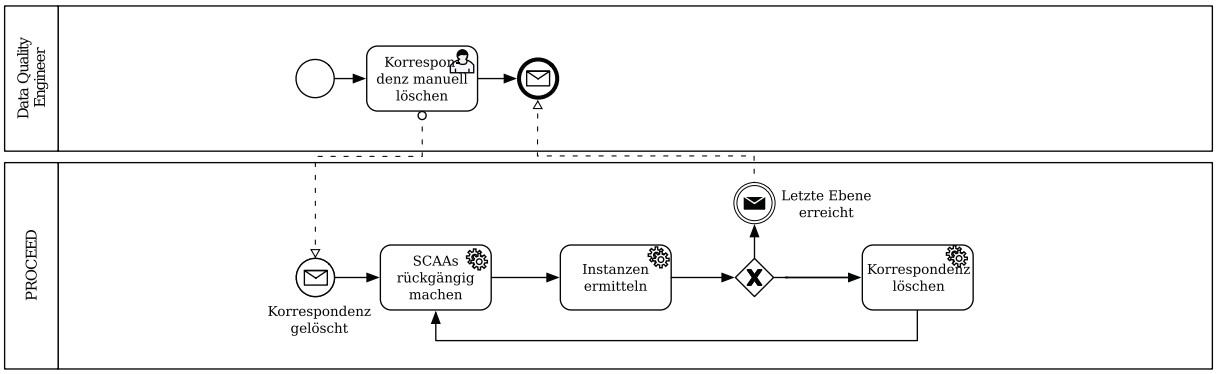


Abbildung 8.7.: Eine Korrespondenz wird manuell gelöscht

Abbildung 8.8 zeigt die Artefakte lokaler und globaler Produktontologien, die von einer Änderung betroffen sind. Die Korrespondenz cor_1 zwischen $ind_{L_{11}}$ und $ind_{G_{11}}$ wird gelöscht ①. Danach müssen zunächst die entsprechenden SCAAs in $ind_{G_{11}}$ der globalen Produktontologie rückgängig gemacht werden ②. In diesem Beispiel ist dies $scaa_1$. Anschließend werden diejenigen Instanzen ermittelt, die zu $ind_{L_{11}}$ gehören und die auf der nächst tieferen Produktdatenintegrationsebene liegen ③. Für jede dieser Instanzen ($ind_{L_{21}}$ und $ind_{L_{22}}$ im Beispiel) werden in den korrespondierenden Instanzen der globalen Produktontologie die SCAAs ebenfalls rückgängig gemacht ④. Schließlich werden die Korrespondenzen von $ind_{L_{21}}$ und $ind_{L_{22}}$ gelöscht ⑤. Die Schritte ③ bis ⑤ werden solange wiederholt, bis die letzte Produktdatenintegrationsebene erreicht ist.

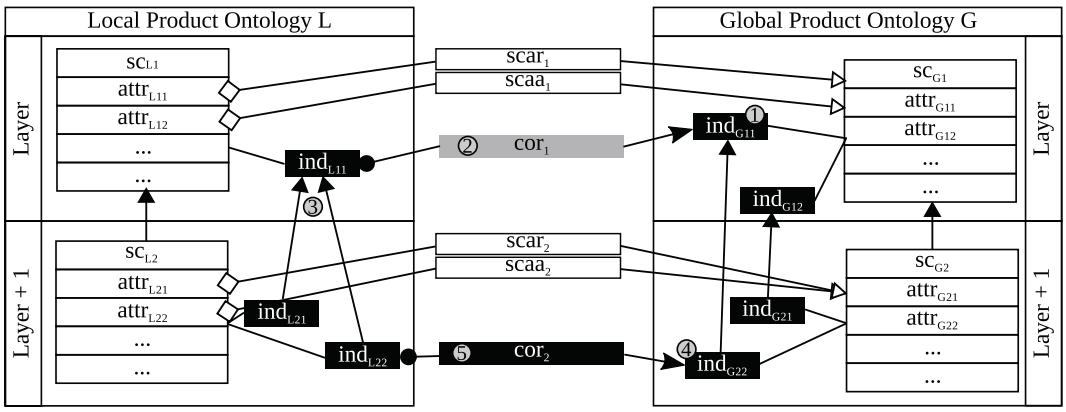


Abbildung 8.8.: Manuelles Löschen einer Korrespondenz

Algorithmus 5 beschreibt eine Hilfsfunktion für das Löschen globaler Attributwerte einer Instanz.

Eingabeparameter sind eine Instanz ind_G der globalen Produktontologie sowie eine Schemakonzept-Attributaktion *scaa*. Für jede Attributaktion werden mögliche Attributwerte der Instanz ermittelt und gelöscht (Zeilen 2 bis 4). Schließlich wird eine Begründung für einen Attributwert gelöscht (Zeilen 5 und 6).

Algorithmus 5 (Globale Attributwerte löschen)

1: **procedure** DeleteGlobalAttributValue($ind_G \in Ind_G, scaa \in SCAA$)
2: **for each** $aa^{cur} \in scaa^{cur}.AA$ **do**
3: $deletedAttrVal = \{attrVal \in AttrVal_G \mid \exists(ind_G, attrVal) \in hasVal_G \wedge \exists(attrVal, attr_G) \in references_G\}$
4: $AttrVal_G = AttrVal_G \setminus deletedAttrVal$
5: $deletedAttrValJust = \{attrValJust \mid \exists(attrValJust, scaa^{cur}) \in AttrValJustification\}$
6: $AttrValJustificationG = AttrValJustification \setminus deletedAttrValJust$
7: **end for**
8: **end procedure**

Algorithmus 6 löscht die Korrespondenz *cor*. Zunächst muss die Begründung der gelöschten Korrespondenz entfernt werden (Zeilen 4 bis 5). Für alle Schemakonzept-Attributaktionen müssen die entsprechenden Attribute (inklusive Begründungen) in den korrespondierenden globalen Instanzen entfernt werden (Zeilen 6 bis 9). Anschließend werden die Korrespondenz gelöscht und alle Instanzen unterhalb der lokalen Instanz ermittelt (Zeilen 10 und 11). Für jede Instanz, für die eine Korrespondenz existiert, wird der Algorithmus rekursiv aufgerufen (Zeilen 12 bis 16).

Algorithmus 6 (Korrespondenz löschen)

1: **procedure** DeleteCorrespondence($cor \in Correspondence$)
2: $Correspondences^{cur} = cor$
3: **for each** $cor^{cur} \in Correspondences^{cur}$ **do**
4: $justification^{cur} = \{j \in Justification \mid \exists(cor^{cur}, j) \in Justification\}$
5: $Justification = Justification \setminus justification^{cur}$
6: $SCAA^{cur} = \{scaa \in SCAA \mid \exists sc \in SC_L\ \exists sc_G \in SC_G\ \exists(sc, cor^{cur}.ind_L) \in hasIndividual_L\ \exists(sc, sc_G) \in SCAA\}$
7: **for each** $scaa^{cur} \in SCAA^{cur}$ **do**
8: $DeleteGlobalAttributValues(cor^{cur}.ind_G, scaa^{cur})$
9: **end for**
10: $Correspondences = Correspondences \setminus cor^{cur}$
11: $Ind_L^{sub} = \{ind \in Ind_L \mid \exists(ind, cor^{cur}.ind_L) \in indBelow_L\}$
12: **for each** $ind_L^{sub,cur} \in Ind_L^{sub}$ **do**
13: **if** $\exists(ind^{sub,cur}, ind_G) \in Correspondece$ **then**
14: $DeleteCorrespondence((ind^{sub,cur}, ind_G))$
15: **end if**
16: **end for**
17: **end for**
18: **end procedure**

8.2.3. Hinzufügen einer Instanz

Abbildung 8.9 illustriert den Prozess nach Erstellung einer neuen lokalen Instanz für ein Schemakonzept. Wird eine neue Instanz erstellt, werden für sie die vorhandenen SCARs ausgewertet. Für die gefundenen Korrespondenzen wiederum werden alle definierten SCAAs ausgewertet. Treten während dieser Auswertung Integrationskonflikte auf, die nicht automatisch aufgelöst werden können, wird ein Data Quality Engineer entsprechend informiert. Wurde die neue Instanz nicht auf der Version-Ebene hinzugefügt, muss für jede Ebene unterhalb der Änderung mindestens eine zugeordnete Instanz erstellt werden.

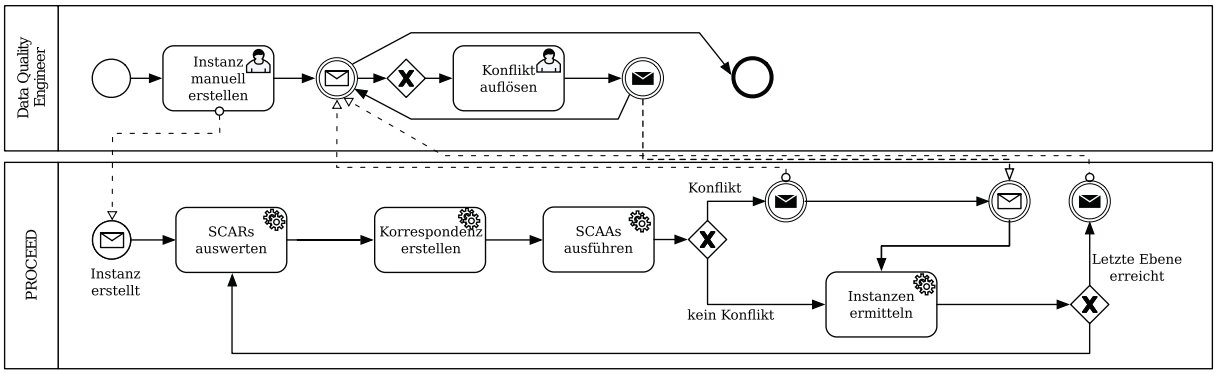


Abbildung 8.9.: Eine neue Instanz wurde hinzugefügt

Abbildung 8.10 zeigt exemplarisch das Erstellen einer neuen Instanz $ind_{L_{22}}$ für ein Schemakonzept $sc_{L2} \in SC_L$ ①. Zuerst werden die vorhandenen Schemakonzept-Attributregeln (SCARs) $scar_2$ für $ind_{L_{22}}$ und die potenziell korrespondierenden Instanzen aus der globalen Produktontologie $Ind_G^{candidates}$ ausgewertet. Weiter werden entsprechende Korrespondenzen erzeugt ② und für diese die jeweiligen SCARs als Begründung hinterlegt. Für die gefundenen Korrespondenzen werden die Aktionen $SCAA$ ausgeführt (③).

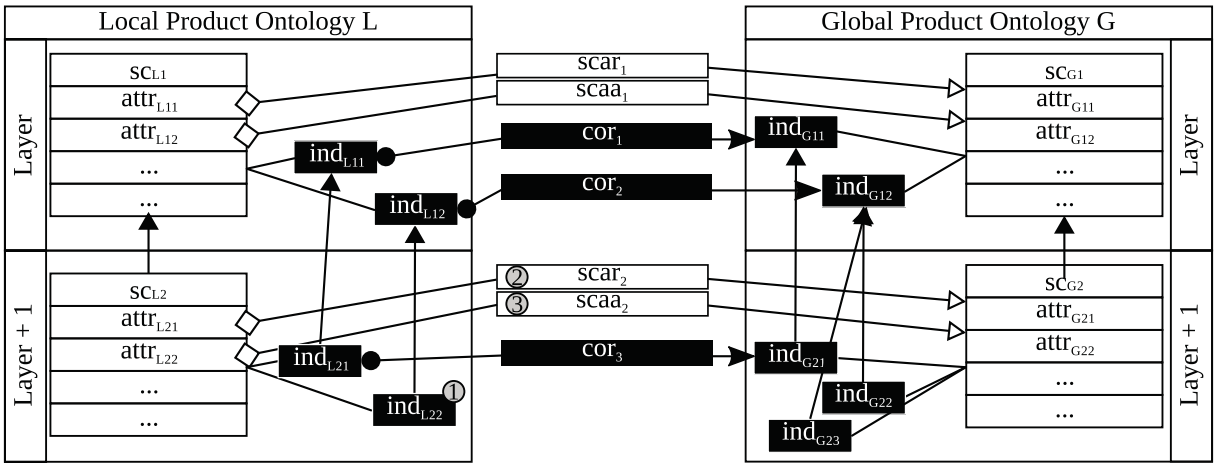


Abbildung 8.10.: Erstellen einer neuen Instanz

Algorithmus 7 beschreibt diesen Einfügeprozess formal. Die Instanz ind_L einer lokalen Produktontologie wird dem Schemakonzept sc_L hinzugefügt. Sie kann unterhalb einer existierenden

Instanz ind_L^{parent} erstellt werden. Zunächst werden die SCARs der Instanz ermittelt (Zeile 2). Für jede SCAR werden die potenziell korrespondierenden Instanzen $Ind_G^{candidates}$ der globalen Produktontologie bestimmt (Zeile 4). Für das Kreuzprodukt aus ind_L und $Ind_G^{candidates}$ werden die Attribut-Matching-Bedingungen der SCAR ausgewertet (Zeile 6). Wurden Korrespondenzen identifiziert, werden diese, zusammen mit einer Begründung, hinzugefügt und der Algorithmus 7 aufgerufen.

Algorithmus 7 (Instanz hinzufügen)

```
1: procedure AddIndividual(ind_L, ind_L^parent, sc_L)
2:     SCAR^{sc_L} = {scar ∈ SCAR | ∃(sc_L, sc_G, AMC) ∈ SCAR}
3:     for each scar^cur ∈ SCAR^{sc_L} do
4:         Ind_G^candidates = {ind ∈ Ind_G | ∃(scar^cur.sc_G, ind) ∈ hasIndividual_G}
5:         for each ind_G^cur ∈ Ind_G^candidates do
6:             if evaluateAMC(ind_L, ind_G^cur, scar^cur.AMC) then
7:                 newCor = (ind_L, ind_G^cur, scar^cur)
8:                 Correspondences = Correspondences ⋃ newCor
9:                 Justification = Justification ⋃ (newCor, scar^cur)
10:                AddCorrespondence(newCor)
11:            end if
12:        end for
13:    end for
14: end procedure
```

8.2.4. Löschen einer Instanz

Abbildung 8.11 zeigt den Prozess, der auf das Löschen einer Korrespondenz zwischen Instanzen aus lokalen und globalen Produktontologien reagiert. Auslöser dieser Änderung ist das manuelle Löschen durch einen Data Quality Engineer oder Domänenexperten. Wie erwähnt, muss beim Löschen von Artefakten zunächst die Startebene ermittelt werden, von der aus das Löschen kaskadierend dann von oben nach unten durchgeführt wird. Für die korrespondierende Instanz der globalen Produktontologie werden alle SCAAs der lokalen Produktontologie rückgängig gemacht, die Korrespondenz wird anschließend gelöscht. Danach werden die zugeordneten Instanzen auf der nächst tieferen Produktdatenintegrationsebene ermittelt, für diese werden die SCAAs ebenfalls rückgängig gemacht. Dieser Vorgang wird solange wiederholt, bis die letzte Ebene erreicht wird.
Die Korrespondenz wurde somit kaskadierend entfernt, die Integrationsmenge ist nun jedoch nicht mehr vollständig. Damit die Fachanwender die Produktdaten nutzen können, müssen für die gelöschten Korrespondenzen neue angelegt werden, oder aber die entsprechenden Instanzen müssen von der Integration ausgeschlossen werden. Die Korrespondenz wurde initial durch den Data Quality Engineer gelöscht, da die automatische Auswertung der SCARs und SCAAs fehlerhaft war. Entweder kann dieser eine „richtige“ korrespondierende Instanz in der globalen Produktontologie ermitteln oder diese von der Integration ausschließen. Legt der Data Quality Engineer eine neue Korrespondenz an, wird ein weiter Änderungsprozess aufgerufen.

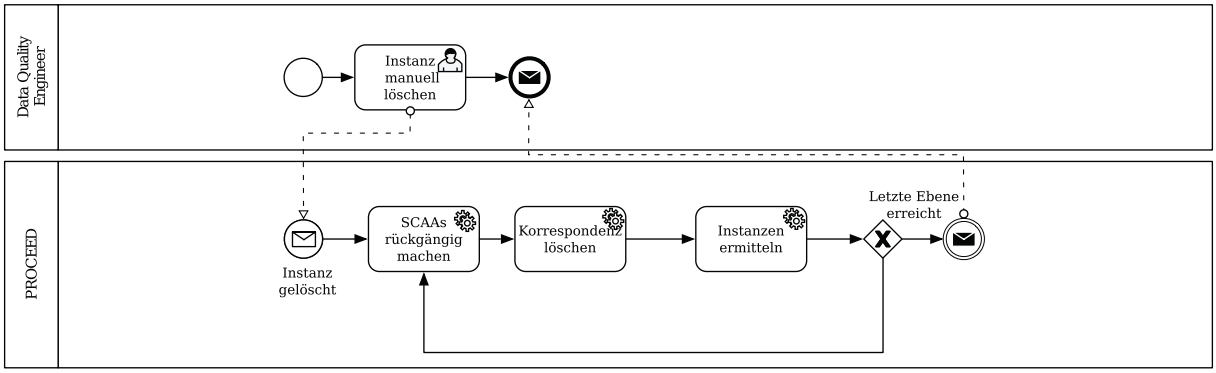


Abbildung 8.11.: Eine Instanz wurde gelöscht

Abbildung 8.12 veranschaulicht das Löschen einer lokalen Instanz $ind_{L_{12}}$. Zunächst müssen die SCAAs für korrespondierende Instanzen ($ind_{G_{12}}$) in der globalen Produktontologie rückgängig gemacht werden ①. Anschließend werden die Korrespondenzen ② gelöscht. Danach werden die Individuals, die zu $ind_{L_{12}}$ gehören und auf der nächst tieferen Produktdatenintegraionsebene liegen, $(ind_{L_{21}}, ind_{L_{22}})$ ermittelt ③. Schließlich wird die Instanz $ind_{L_{12}}$ aus der lokalen Produktontologie entfernt ④. Die Schritte ① bis ④ werden für die Instanzen $ind_{L_{21}}$ und $ind_{L_{22}}$ wiederholt, bis die letzte Produktdatenintegrationsebene erreicht wird.

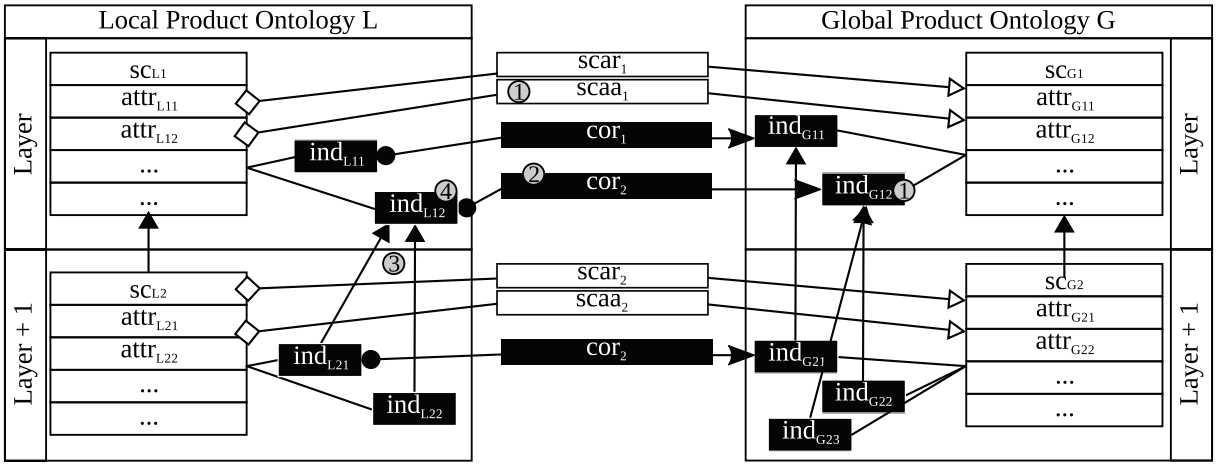


Abbildung 8.12.: Löschen einer Instanz

Formal wird dieser Prozess durch Algorithmus 8 beschrieben. Für die zu löschende Instanz ind_L werden die bestehenden Korrespondenzen Cor^{ind_L} bestimmt (Zeile 2). Für alle Korrespondenzen wird Algorithmus 8 aufgerufen, anschließend wird die Instanz entfernt (Zeilen 4 und 5).

Algorithmus 8 (Instanz löschen)

1: **procedure** DELETEINDIVIDUAL(ind_L)
2: $Cor^{ind_L} = \{cor \in Correspondence \mid \exists(ind_L, ind_G) \in Correspondence\}$
3: **for each** $cor^{cur} \in Cor^{ind_L}$ **do**
4: $DeletedCorrespondence(cor^{cur})$
5: $Ind_L = Ind_L \backslash cor^{cur}.ind_L$
6: **end for**
7: **end procedure**

8.2.5. Löschen eines Attributwertes

Wird ein Attributwert einer Instanz ind_L einer lokalen Produktontologie gelöscht, nachdem der Attributwert bereits über eine Korrespondenz in die globale Produktontologie integriert worden ist, sind drei Fälle zu unterscheiden. Entweder wird das Attribut von einer Schemakonzept-Attributaktion oder -Attributregel verwendet (Fälle 1 und 2). Schließlich kann es sich um ein Attribut handeln, welches von keinem der beiden verwendet wird (Fall 3).
Abbildung 8.13 zeigt den Änderungsprozess eines Attributwertes einer lokalen Instanz. Sobald der Attributwert durch einen Nutzer geändert wurde, muss ermittelt werden, ob es sich um ein Attribut handelt, dass entweder von einer Schemakonzept-Attributaktion oder -Attributregel verwendet wird. Im ersten Fall müssen über bestehende Korrespondenzen diejenigen globalen Instanzen ermittelt werden, für die existierende Schemakonzept-Attributregeln rückgängig gemacht werden. Im zweiten Fall werden die existierenden Korrespondenzen ebenfalls ermittelt. Diese werden anschließend mittels Algorithmus 5 gelöscht.

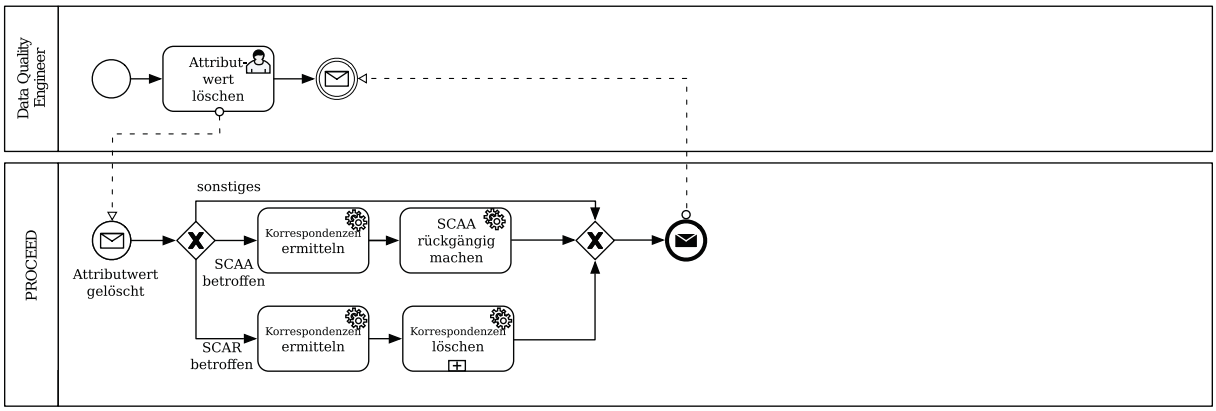


Abbildung 8.13.: Attributwert löschen

Abbildung 8.14 illustriert einen Ausschnitt einer lokalen Produktontologie, in der ein Attributwert $attrVal_{L12}$ einer Instanz ind_{L11} geändert wurde (①). Das geänderte Attribut wird durch eine Schemakonzept-Attributaktion $scaa_1$ verwendet. Zunächst müssen die korrespondierenden Instanzen in einer globalen Produktontologie ermittelt werden (②). Anschließend wird für jede gefundene Instanz die entsprechende Schemakonzapt-Attributaktion rückgängig gemacht (③).

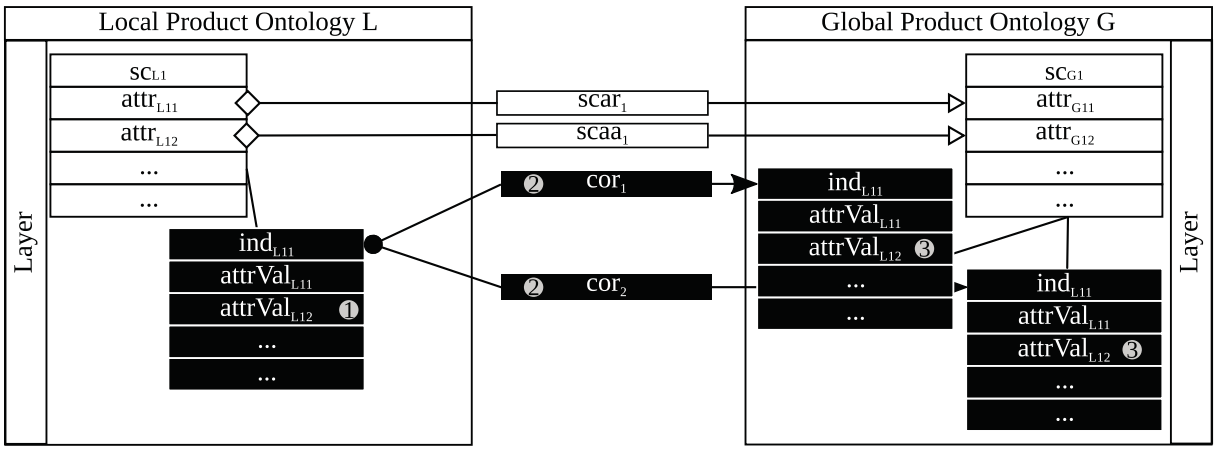


Abbildung 8.14.: Löschen des Attributwerts eines SCAA-Attributs

Analog dazu illustriert Abbildung 8.15 eine lokale Produktontologie, in welcher der geänderte Attributwert durch eine Schemakonzept-Attributregel verwendet wird. Der Wert des Attributs $attrVal_{L12}$ wurde geändert ①. Folglich müssen alle Korrespondenzen der Instanz ind_{L11} ermittelt und mittels Algorithmus 6 gelöscht werden ②.

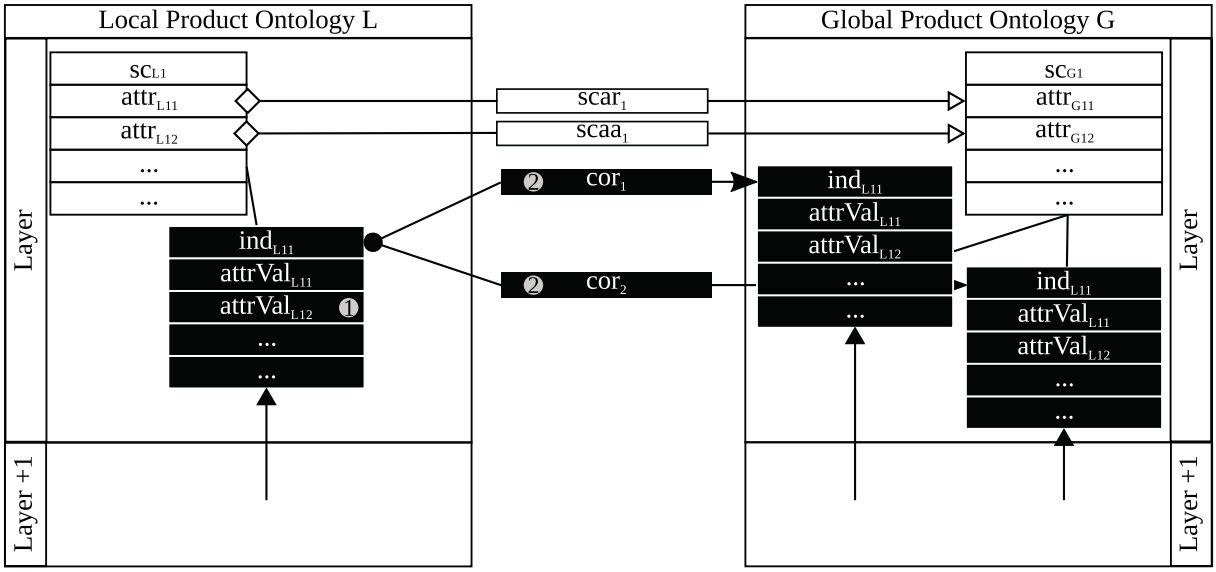


Abbildung 8.15.: Löschen des Attributwerts eines SCAR-Attributs

Formal beschreibt Algorithmus 9 die Änderungsoperation. Zunächst wird das Schemakonzept des geänderten Attributs bestimmt (Zeile 2). Davon ausgehend werden die Schemakonzept-Attributregeln und -Attributaktionen ermittelt (Zeilen 3 und 4). Es wird über alle SCARs, inklusive deren Attribut-Matching-Bedingungen, iteriert. Ist das lokale Attribut der Attribut-Matching-Bedingung gleich dem geänderten Attribut, wird Algorithmus 11 aufgerufen (Zeilen 5 bis 11). Analog wird über die SCAAs iteriert und Algorithmus 10 ausgeführt (Zeilen 12 bis 18).

Algorithmus 9 (Lösche Attributwert)

1: **procedure** DeleteAttributeValue($ind_L, attr_L$)
2: $sc_L^{ind_L} = \{sc \in SC_L \mid \exists(sc, ind_L) \in hasIndividual_L\}$
3: $SCAR^{ind_L} = \{scar \in SCAR \mid \exists(sc^{ind_L}, sc_G, AMC) \in SCAR\}$
4: $SCAA^{ind_L} = \{scaa \in SCAA \mid \exists(sc^{ind_L}, sc_G, AA) \in SCAA\}$
5: **for each** $scar^{cur} \in SCAR^{ind_L}$ **do**
6: **for each** $amc^{cur} \in scar^{cur}.AMC$ **do**
7: **if** $amc^{cur}.attr_L == attr_L$ **then**
8: $DeleteSCAARttributeValue(ind_L, attr_L)$
9: **end if**
10: **end for**
11: **end for**
12: **for each** $scaa^{cur} \in SCAA^{ind_L}$ **do**
13: **for each** $aa^{cur} \in scaa^{cur}.AA$ **do**
14: **if** $aa^{cur}.attr_L == attr_L$ **then**
15: $DeleteSCAAAttributeValue(ind_L, attr_L)$
16: **end if**
17: **end for**
18: **end for**
19: **end procedure**

Algorithmus 10 formalisiert die in Abbildung 8.14 erläuterten Schritte. Zunächst werden die Korrespondenzen der Instanz ind_L bestimmt (Zeile 2). Mit Hilfe der Schemakonzept-Attributaktionen wird das betroffene Attribut $attr_G$ des Schemakonzepts in der globalen Produktontologie ermittelt (Zeilen 3 bis 11). Schließlich wird für alle korrespondierenden Instanzen von ind_L der korrespondierende Attributwert entfernt (Zeilen 12 bis 15).

Algorithmus 10 (Lösche SCAA-Attributwert)

1: **procedure** DeleteSCAAAttributeValue($ind_L, attr_L$)
2: $Cor^{ind_L} = \{cor \in Correspondence \mid \exists(ind_L, ind_G) \in Correspondence\}$
3: $sc_L^{ind_L} = \{sc_L \in SC_L \mid \exists(sc_L, ind_L) \in hasIndividual_L\}$
4: $SCAA^{ind_L} = \{scaa \in SCAA \mid \exists sc_G \in SC_G \wedge \exists(sc_L^{ind_L}, sc_G) \in SCAA\}$
5: **for each** $scaa^{cur} \in SCAA^{ind_L}$ **do**
6: **for each** $aa^{cur} \in scaa^{cur}.AA$ **do**
7: **if** $aa^{cur}.attr_L == attr_L$ **then**
8: $attr_G = aa^{cur}.attr_G$
9: **end if**
10: **end for**
11: **end for**
12: **for each** $ind_G^{cur} \in Cor^{ind_L}.ind_G$ **do**
13: $attrVal^{cur} = \{attrVal_G \in AttrVal_G \mid \exists(attrVal_G, attr_G) \in references_G$
$\wedge \exists(ind_G^{cur}, attrVal_G) \in hasValue_G\}$
14: $AttrVal_G = AttrVal_G \backslash attrVal_G^{cur}$
15: **end for**
16: **end procedure**

Algorithmus 11 definiert die Schritte, die in Abbildung 8.15 erläutert wurden. Zunächst wer-

den die bestehenden Korrespondenzen der geänderten Instanz ermittelt (Zeile 2). Jede dieser Instanzen wird mit Hilfe von Algorithmus 5 entfernt (Zeilen 3 bis 5).

Algorithmus 11 (Lösche SCAR-Attributwert)

1: **procedure** DELETESCARATTRIBUTEVALUE($ind_L, attr_L$)
2: $Cor^{ind_L} = \{cor \in Correspondence \mid \exists(ind_L, ind_G) \in Correspondence\}$
3: **for each** $cor^{cur} \in Cor^{ind_L}$ **do**
4: $DeletedCorrespondence(cor^{cur})$
5: **end for**
6: **end procedure**

8.2.6. Ändern eines Attributwerts

Das Ändern eines Attributwerts ist ähnlich dem vorher beschriebenen Löschen. Abbildung 8.16 zeigt die notwendigen Aktivitäten für das Ändern eines Attributwerts. Analog muss zwischen dem Ändern eines SCAR- und SCAA-Attributs unterschieden werden. Wird ein von einer Schemakonzept-Attributaktion verwendetes Attribut geändert, müssen zunächst die korrespondierenden Instanzen ermittelt werden. Anschließend sind für diese die Attributwerte korrespondierender globaler Instanzen anzupassen. Handelt es sich um ein Attribut, welches durch eine Schemakonzept-Attributregel verwendet wurde, und existieren Korrespondenzen in eine globale Produktontologie, müssen diese Korrespondenzen zunächst gelöscht werden. Anschließend werden die Schemakonzept-Attributregeln neu ausgewertet und neue Korrespondenzen erstellt sowie Schemakonzept-Attributaktionen ausgeführt.

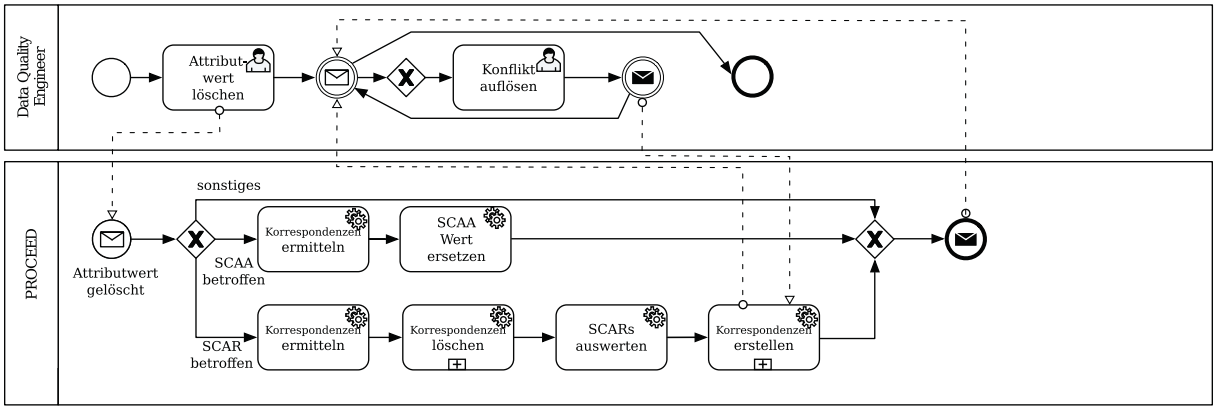


Abbildung 8.16.: Änderung eines Attributwerts

Abbildung 8.17 zeigt eine lokale Produktontologie, bei der für Instanz ind_{L11} der Attributwert von $attr_{L12}$ geändert wurde. Der Attributwert wird durch die Schemakonzept-Attributaktion $scaa_1$ verwendet. Zunächst werden alle korrespondierenden globalen Instanzen ermittelt ① und anschließend deren Attributwerte an den neuen lokalen Wert angepasst ②.

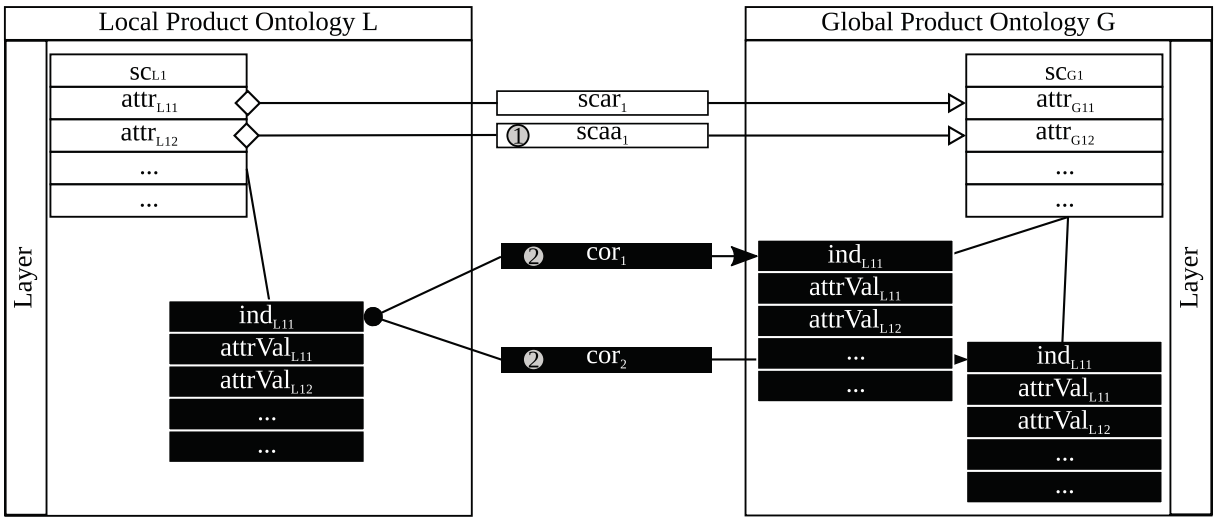


Abbildung 8.17.: Änderung des Werts eines SCAA-Attributs

Abbildung 8.18 zeigt die notwendigen Schritte nach der Änderung eines von einer SCAR verwendeten Attributwerts. Zunächst müssen alle korrespondierenden Instanzen der globalen Produktontologie ermittelt werden. Zu beachten ist, dass eine Korrespondenz zwischen Instanzen lokaler Produktontologien und der globalen Produktontologie durch mehrere SCAR zustande kommen können. Um die Integration nachvollziehbar zu machen, werden für jede Korrespondenz die SCARs hinterlegt, die zu deren Erstellung geführt haben. Folglich kann eine Korrespondenz nur dann gelöscht werden, wenn genau eine SCAR zur Erstellung geführt hat. Anderenfalls wird lediglich die Referenz auf die SCAR entfernt. Für alle Instanzen der lokalen Produktontologie, deren Korrespondenzen zu Instanzen der globalen Produktontologie gelöscht werden sollen, werden entsprechende SCAAs des zugrundeliegenden Schemakonzeptes rückgängig gemacht. Anschließend werden die Korrespondenzen gelöscht und für die Instanzen die zugehörigen Instanzen auf der nächst tieferen Produktdatenintegrationsebene ermittelt. Für all diese Instanzen werden jeweils korrespondierende Instanzen der globalen Produktontologie ermittelt und die vorherigen Schritte werden solange wiederholt, bis die letzte Produktdatenintegrationsebene erreicht worden ist. Am Ende dieses Vorgangs sind die Korrespondenzen zwischen lokalen Produktontologien und globaler Produktontologie kaskadierend gelöscht worden. Nun müssen für die Instanz, deren Attributwert initial geändert wurde, die vorhandenen SCARs neu ausgewertet und entsprechende Korrespondenzen erstellt werden. Wurden Korrespondenzen gefunden bzw. manuell erstellt, müssen die SCAAs ausgeführt werden.

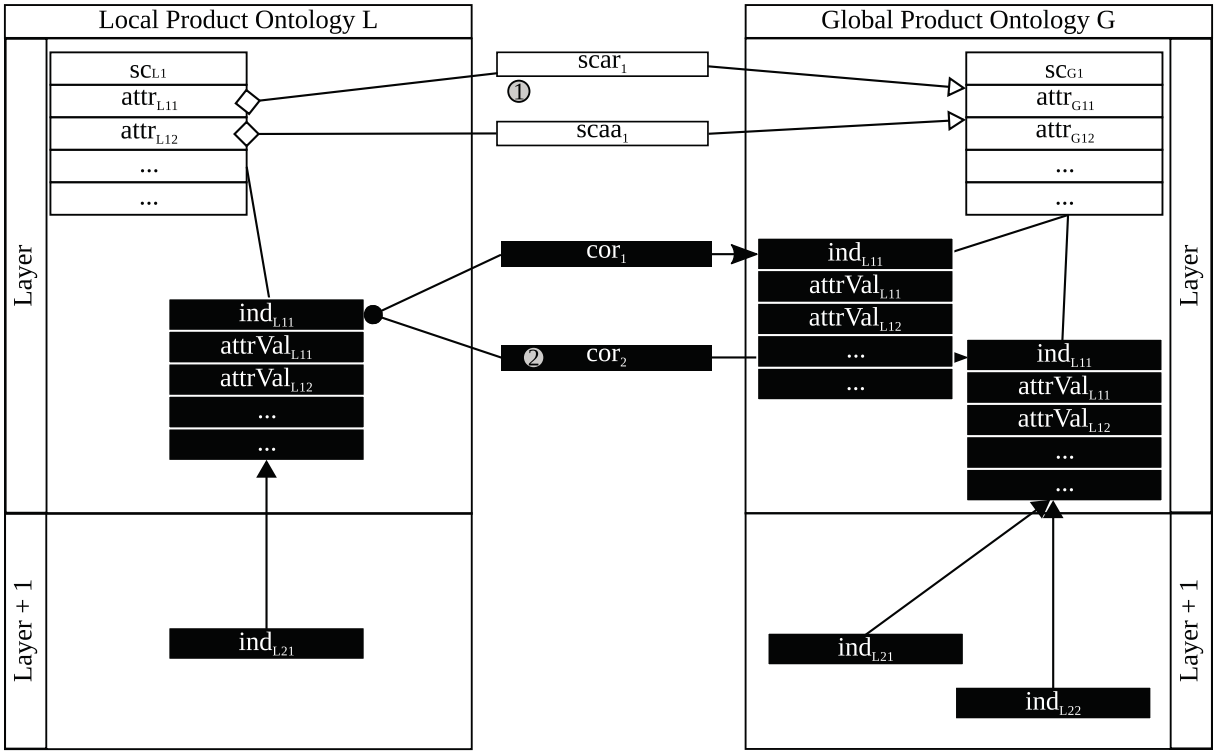


Abbildung 8.18.: Änderung des Attributwerts eines SCAR-Attributs

Algorithmus 12 formalisiert das Ändern eines Attributes. Mit Hilfe des lokalen Schemakonzeptes der geänderten Instanz werden die Schemakonzept-Attributregeln und -Attributaktionen bestimmt (Zeilen 2 bis 4). Alle Schemakonzept-Attributregeln, inklusive Attribut-Matching-Bedinungen, werden iteriert. Stimmt das lokale Attribut einer Attribut-Matching-Bedingung mit dem betroffenen Attribut überein, wird Algorithmus 13 ausgeführt (Zeilen 5 bis 11). Analog dazu werden in den Zeilen 12 bis 18 die Schemakonzept-Attributaktionen iteriert und bei Übereinstimmung der Attribute der Algorithmus 14 aufgerufen.

Algorithmus 12 (Ändere Attributwert)

1: **procedure** ChangeAttributeValue($ind_L, attr_L$)
2: $sc_L^{ind_L} = \{sc \in SC_L \mid \exists(sc, ind_L) \in hasIndividual_L\}$
3: $SCAR^{ind_L} = \{scar \in SCAR \mid \exists(sc^{ind_L}, sc_G, AMC) \in SCAR\}$
4: $SCAA^{ind_L} = \{scaa \in SCAA \mid \exists(sc^{ind_L}, sc_G, AA) \in SCAA\}$
5: **for each** $scar^{cur} \in SCAR^{ind_L}$ **do**
6: **for each** $amc^{cur} \in scar^{cur}.AMC$ **do**
7: **if** $amc^{cur}.attr_L == attr_L$ **then**
8: $ChangeSCAARttributeValue(ind_L, attr_L)$
9: **end if**
10: **end for**
11: **end for**
12: **for each** $scaa^{cur} \in SCAA^{ind_L}$ **do**

```
13:         for each aa^cur ∈ scaa^cur.AA do
14:             if aa^cur.attr_L == attr_L then
15:                 ChangeSCAAAttributeValue(ind_L, attr_L)
16:             end if
17:         end for
18:     end for
19: end procedure
```

Algorithmus 13 formalisiert die in Abbildung 12 beschriebenen Schritte. Zunächst müssen die bestehenden Korrespondenzen rückgängig gemacht werden (Zeilen 2 bis 5). Jede Attributregel des Schemakonzeptes sc_L wird für die geänderte Instanz ind_L erneut für alle Kandidaten der globalen Produktontologie ausgewertet (Zeilen 6 bis 10). Wird eine Korrespondenz ermittelt, wird sie, zusammen mit einer Begründung, hinzugefügt und Algorithmus 4 wird ausgeführt.

Algorithmus 13 (SCAR-Attributwert geändert)

```
1: procedure ChangeSCARAttributeValue(ind_L, attr_L)
2:     Cor^ind_L = {cor ∈ Correspondence | ∃(ind_L, ind_G) ∈ Correspondence}
3:     for each cor^cur ∈ Cor^ind_L do
4:         DeletedCorrespondence(cor^cur)
5:     end for
6:     SCAR^sc_L = {scar ∈ SCAR | ∃(sc_L, sc_G, AMC) ∈ SCAR}
7:     for each scar^cur ∈ SCAR^sc_L do
8:         Ind_G^candidates = {ind ∈ Ind_G | ∃(scar^cur.sc_G, ind) ∈ hasIndividual_G}
9:         for each ind_G^cur ∈ Ind_G^candidates do
10:            if evaluateAMC(ind_L, ind_G^cur, scar^cur.AMC) then
11:                newCor = (ind_L, ind_G^cur, scar^cur)
12:                Correspondences = Correspondences ⋃ newCor
13:                Justification = Justification ⋃ (newCor, scar^cur)
14:                AddCorrespondence(newCor)
15:            end if
16:        end for
17:    end for
18: end procedure
```

Algorithmus 14 formalisiert die Schritte aus Abbildung 8.17. Zuerst werden die Korrespondenzen sowie das Schemakonzept der geänderten Instanz bestimmt (Zeilen 2 und 3). Mit Hilfe des Schemakonzeptes werden die Schemakonzept-Attributaktionen ermittelt. Diese werden iteriert, um das globale Attribut des korrespondierenden Schemakonzeptes zu identifizieren (Zeilen 4 bis 11). Schließlich wird für alle korrespondierenden globalen Instanzen der betroffene Attributwert aktualisiert (Zeilen 12 bis 1).

Algorithmus 14 (Ändere SCAA-Attributwert)

```
1: procedure ChangeSCAAAttributeValue(ind_L, attr_L)
2:     Cor^ind_L = {cor ∈ Correspondence | ∃(ind_L, ind_G) ∈ Correspondence}
3:     sc_L^ind_L = {sc_L ∈ SC_L | ∃(sc_L, ind_L) ∈ hasIndividual_L}
4:     SCAA^ind_L = {scaa ∈ SCAA | ∃sc_G ∈ SC_G ∧ ∃(sc_L^ind_L, sc_G) ∈ SCAA}
```

```
5:    for each scaa^cur ∈ SCAA^ind_L do
6:        for each aa^cur ∈ scaa^cur.AA do
7:            if aa^cur.attr_L == attr_L then
8:                attr_G = aa^cur.attr_G
9:            end if
10:       end for
11:   end for
12:   for each cor^cur ∈ Cor^ind_L do
13:       executeSCAA(cor^cur, SCAA^ind_L)
14:   end for
15: end procedure
```

8.3. Schemaänderungen

Neben Änderungen von Instanzen und deren Attributwerte können Änderungen an Schemakonzepten und Attributen auftreten. Die Gründe für solche Änderungen können vielschichtig sein. Dazu zählen etwa Anpassungen von Informationssystemen an neue Technologien, neue gesetzliche Vorgaben oder neue Unternehmensstrategien. Die folgenden Abschnitte beschreiben Änderungen unterschiedlicher Schemaartefakte.
Während Datenänderungen durch Data Quality Engineers bzw. Anwender durchgeführt werden, müssen Änderungen auf Schemaebene durch das Integration Board entschieden werden. Die folgenden Abschnitte befassen sich, analog zu den Datenänderungen, mit Änderungsoperationen für Artefakte der Schemaebene. Für jede Operation wird wieder der Ablauf als BPMN-Prozessmodell illustriert. Anschließend wird dieser mittels eines Beispiels dargestellt, bevor er formalisiert wird.

8.3.1. SCAA hinzufügen

Abbildung 8.19 zeigt den Prozess für das Anlegen einer neuen Schemakonzept-Attributaktion. Zunächst wird diese durch einen Integrationsexperten modelliert. Ist dieser Vorgang abgeschlossen, werden die Instanzen des betroffenen Schemakonzeptes sowie deren Korrespondenzen bestimmt. Für alle Korrespondenzen der Instanzen wird die neue Schemakonzept-Attributaktion ausgeführt.

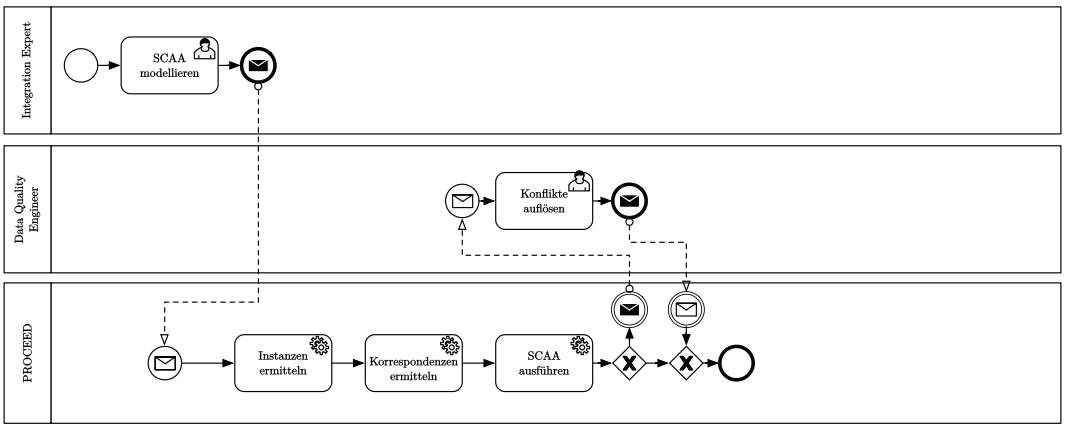


Abbildung 8.19.: Hinzufügen neuer SCAAs

Abbildung 8.20 zeigt beispielhaft eine lokale und die globale Produktontologie, für die eine neue Schemakonzept-Attributaktion $scaa_1$ erstellt wurde ①. Zunächst werden das betroffene Schemakonzept sc_{L1} sowie dessen Instanzen (ind_{L11} bis ind_{L15}) bestimmt ②. Anschließend werden für jede Instanz die korrespondierenden globalen Instanzen ermittelt. Für jede dieser Instanzen wird die neue SCAA ausgeführt ③.

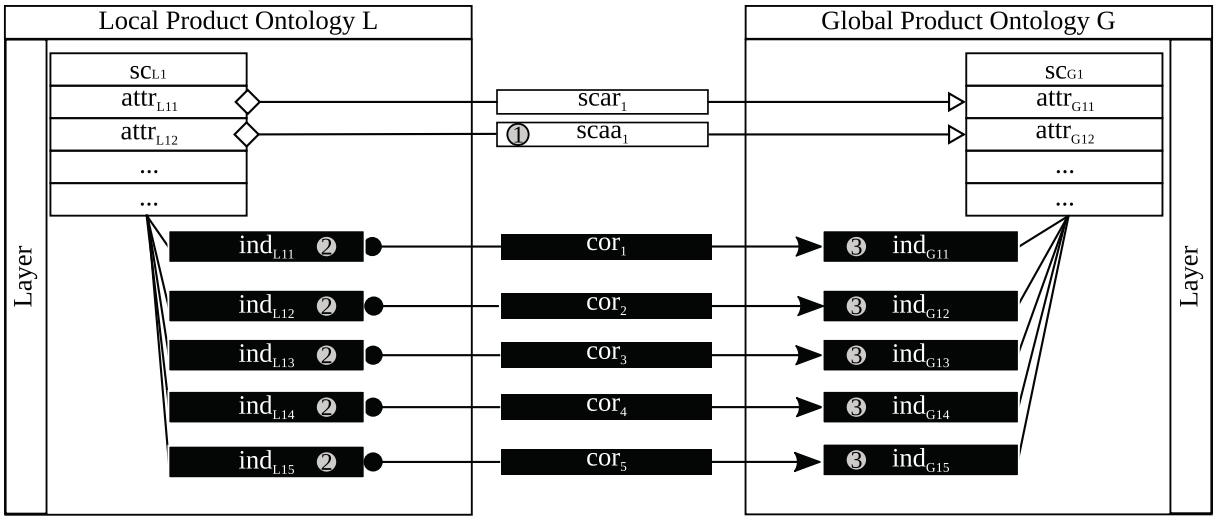


Abbildung 8.20.: Neue SCAA

Algorithmus 15 formalisiert die Schritte aus Abbildung 8.20. Für die neue Schemakonzept-Attributaktion wird die Menge der betroffenen Instanzen bestimmt (Zeile 2). Für jede dieser Instanzen wiederum wird die Menge der Korrespondenzen auf globale Instanzen ermittelt (Zeilen 3 und 4). Schließlich wird für jede dieser Korrespondenzen die neue Schemakonzept-Attributaktion ausgeführt (Zeile 6).

Algorithmus 15 (SCAA hinzufügen)

1: **procedure** ADDSCAA($scaa$)
2: $Ind_L^{sc_L} = \{ind_L \in Ind_L \mid \exists(scaa.sc_L, ind_L) \in hasIndividual_L\}$
3: **for each** $ind_L^{cur} \in Ind_L^{sc_L}$ **do**
4: $Cor^{ind^{cur}} = \{\exists(ind_L^{cur}, ind_G) \in Correspondences\}$
5: **for each** $cor^{cur} \in Cor^{ind^{cur}}$ **do**
6: $executeSCAA(cor^{cur}, scaa)$
7: **end for**
8: **end for**
9: **end procedure**

8.3.2. Löschen einer SCAA

Wird ein Attribut aus der globalen Produktontologie nicht mehr für Anwendungsfälle benötigt, kann die entsprechende Schemakonzept-Attributregel durch Integrationsexperten gelöscht werden. Abbildung 8.21 zeigt den Ablauf der einzelnen Schritte dieser Änderungsoperation. Nach Entfernen der Schemakonzept-Attributaktion durch einen Integrationsexperten, werden die hiervon betroffenen Instanzen bestimmt. Für jede von ihnen werden die globalen Instanzen über die

existierenden Korrespondenzen ermittelt. Schließlich wird für jede dieser korrespondierenden globalen Instanzen der Wert des Attributs der SCAA gelöscht.

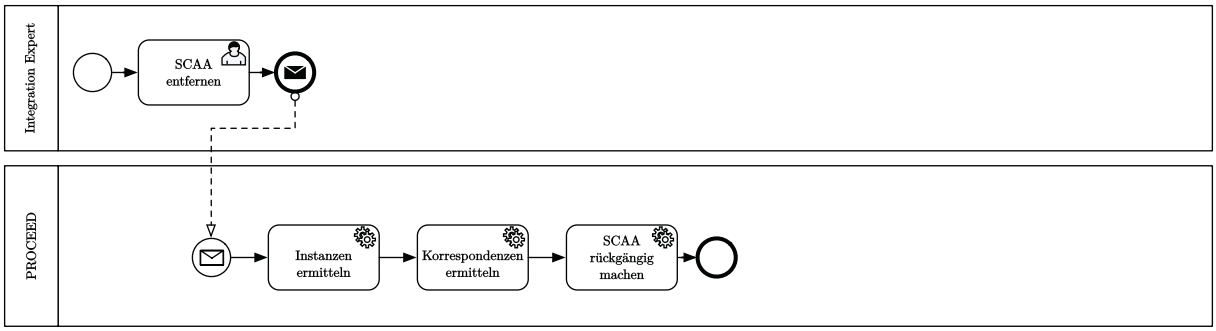


Abbildung 8.21.: SCAA löschen

Abbildung 8.22 zeigt ein Beispiel, in dem eine Attributaktion eines Schemakonzepts sc_{L1} entfernt wird ①. Zunächst werden alle betroffenen Instanzen über die Korrespondenzen bestimmt ②. Anschließend werden die Aktionen in den globalen Instanzen rückgängig gemacht ③ und die entsprechende SCAA gelöscht.

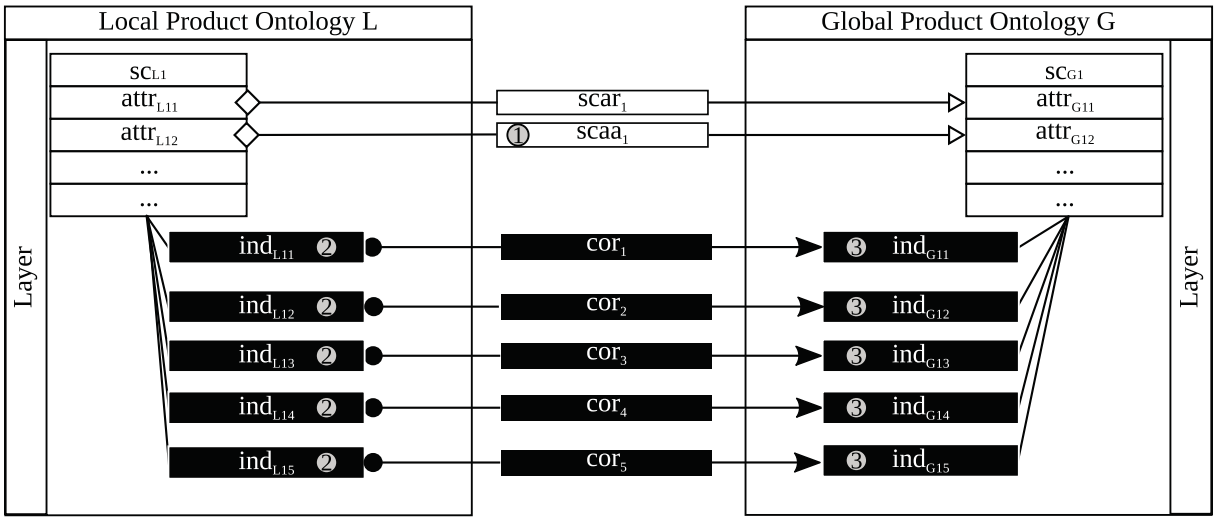


Abbildung 8.22.: Löschen einer SCAA

Algorithmus 16 formalisiert die Schritte aus Abbildung 8.22. Zunächst wird die Menge der betroffenen Instanzen $Ind_L^{sc_L}$ ermittelt (Zeile 2). Für jede Instanz wird dann über die existierenden Korrespondenzen die Menge der betroffenen globalen Instanzen ermittelt. Schließlich werden für jede der so ermittelten globalen Instanzen mit Hilfe von Algorithmus 5 die Attributwerte der Schemakonzept-Attributaktion entfernt (Zeilen 3 bis 8).

Algorithmus 16 (SCAA löschen)

1: **procedure** DELETESCAA($scaa$)
2: $Ind_L^{sc_L} = \{ind_L \in Ind_L \mid \exists(scaa.sc_L, ind_L) \in hasIndividual_L\}$
3: **for each** $ind_L^{cur} \in Ind_L^{sc_L}$ **do**
4: $Cor^{ind^{cur}} = \{cor \in Correspondences \mid \exists(ind_L^{cur}, ind_G) \in Correspondences\}$
5: **for each** $cor^{cur} \in Cor^{ind^{cur}}$ **do**
6: $DeleteGlobalAttributValue(cor^{cur}, scaa)$
7: **end for**
8: **end for**
9: **end procedure**

8.3.3. Änderung einer SCAA

Neben dem Löschen bzw. Hinzufügen einer Schemakonzept-Attributaktion kann eine solche auch geändert werden, indem mindestens eine Attribut-Aktion angepasst wird. Die Schritte der Änderungsoperation setzen sich aus dem Löschen und Hinzufügen einer SCAA zusammen (siehe Abbildung 8.23).

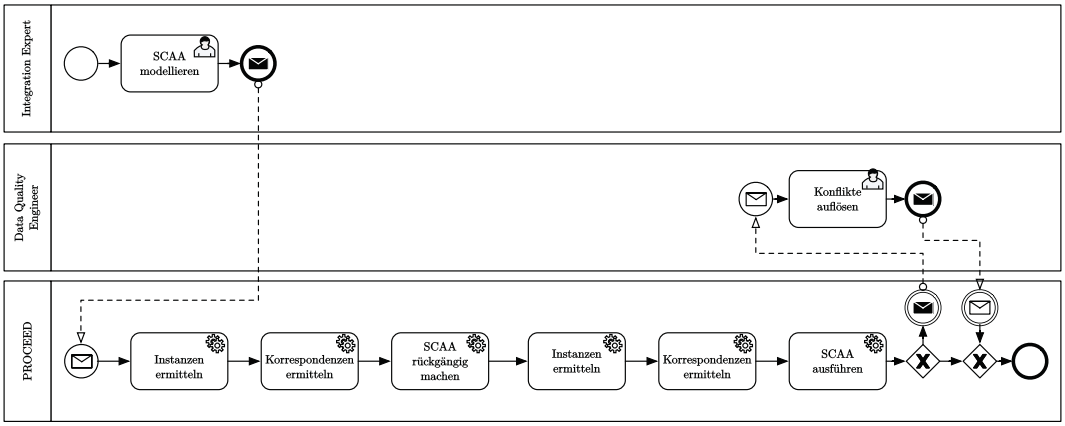


Abbildung 8.23.: Ändern eines SCAA-Attributs

Algorithmus 20 beschreibt den Prozess aus Abbildung 8.23 formal und setzt sich aus den beiden Algorithmen 15 und 16 zusammen (Zeilen 2 und 3).

Algorithmus 17 (SCAA ändern)

1: **procedure** CHANGESCAA($scaa$)
2: $DeleteSCAA(scaa)$
3: $AddSCAA(scaa)$
4: **end procedure**

8.3.4. Hinzufügen einer SCAR

Wird für ein Schemakonzept ein in einem neuen SCAR zu verwendendes Attribut erstellt, stellen sich die benötigten Änderungsaktivitäten deutlich komplexer dar als im vorherigen Fall. Die erforderlichen Aktivitäten sind in Abbildung 8.24 dargestellt. Die neue SCAR muss zunächst

durch einen Integrationsexperten modelliert werden. Für alle Instanzen des geänderten Schemakonzeptes der lokalen Produktontologie wird die neu erstellte SCAR ausgewertet, bei Treffern werden dann jeweils entsprechende Korrespondenzen erstellt. Für alle gefundenen Korrespondenzen werden alle SCAAs des geänderten Schemakonzeptes ausgeführt. Treten bei der Ausführung von SCAAs um Integrationskonflikte auf, müssen diese durch einen Data Quality Engineer aufgelöst werden. Anschließend werden für alle Instanzen der lokalen Produktontologie, für die neue Korrespondenzen gefunden wurden, die zugehörigen Instanzen der nächst tieferen Produktdatenintegrationsebene ermittelt. Die SCARs des entsprechenden Schemakonzeptes werden für diese Instanzen ausgewertet, und bei Treffern werden Korrespondenzen erstellt und SCAAs ausgeführt. Dieser Prozess wird solange wiederholt, bis die letzte Produktdatenintegrationsebene erreicht worden ist.

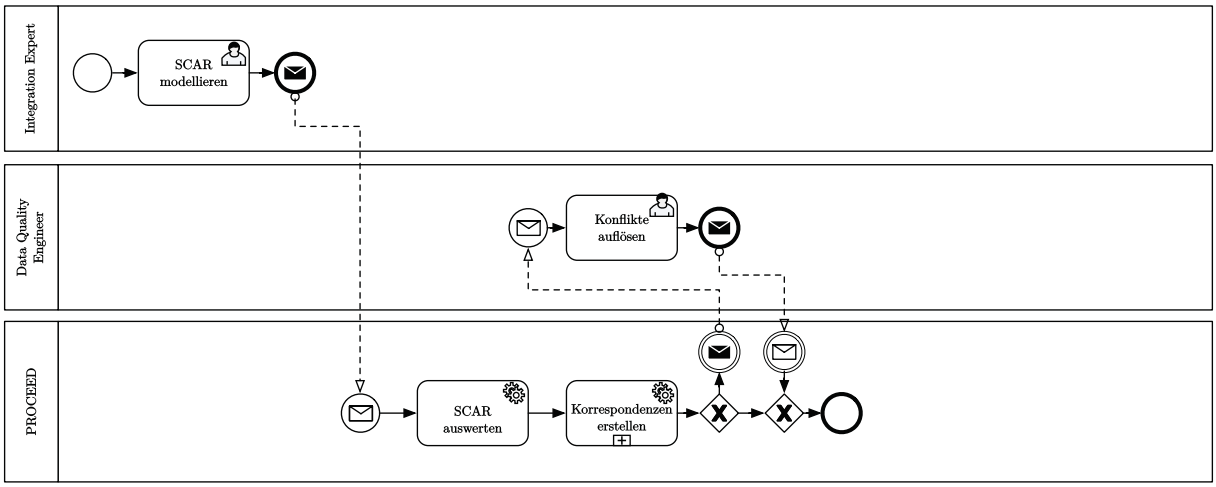


Abbildung 8.24.: Hinzufügen einer neuen SCAR

Abbildung 8.25 zeigt ein Beispiel für eine lokale und globale Produktontologie, für die eine neue Schemakonzept-Attributregel $scar_2$ zwischen den Schemakonzepten sc_{L1} und sc_{G1} modelliert wurde ①. Es werden die lokalen Instanzen des Schemakonzeptes sc_{L1} ② und die globalen Instanzen von sc_{G1} ③ bestimmt. Für das Kreuzprodukt aus beiden Instanzmengen wird die neue Schemakonzept-Attributregel ausgewertet ④. Für jede Übereinstimmung werden eine neue Korrespondenz angelegt ⑤ und die bestehenden Schemakonzept-Attributaktionen ausgeführt ⑥. Für die neuen Korrespondenzen wird dieser Vorgang für die weiteren Integrationsebenen wiederholt ⑦, bis schließlich die letzte Ebene erreicht worden ist.

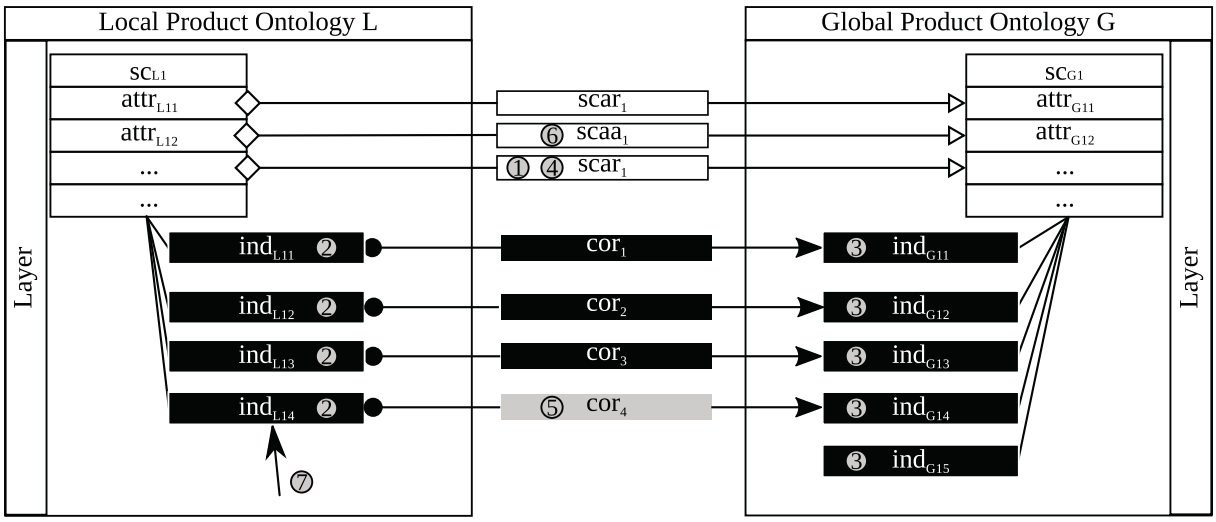


Abbildung 8.25.: Einfügen einer neuen SCAR

Algorithmus 18 formalisiert die Schritte aus Abbildung 8.25. Zuerst werden diejenigen Instanzen aus einer lokalen und der globalen Produktontologie bestimmt, für welche die neue Schemakonzept-Attributregel ausgewertet werden soll (Zeilen 2 und 3). Für das Kreuzprodukt aus beiden Mengen werden die Attribut-Matching-Bedingungen der neuen SCAR ausgewertet und bei Übereinstimmung eine neue Korrespondenz erstellt (Zeilen 4 bis 7). Zusätzlich wird eine Begründung erzeugt (Zeile 8). Abschließend wird der Algorithmus 4 für die neue Korrespondenz ausgeführt (Zeile 9).

Algorithmus 18 (SCAR hinzufügen)

```
1:  procedure AddSCAR(scar)
2:      Ind_L^candidates = {ind_L ∈ Ind_L | ∃(scar.sc_L, ind_L) ∈ hasIndividual_L}
3:      Ind_G^candidates = {ind_G ∈ Ind_G | ∃(scar.sc_G, ind_G) ∈ hasIndividual_G}
4:      for each ind_L^cur ∈ Ind_L^candidates do
5:          for each ind_G^cur ∈ Ind_G^candidates do
6:              if evaluateAMC(ind_L, ind_G^cur, scar^cur.AMC) then
7:                  newCor = (ind_L, ind_G^cur, scar^cur)
8:                  Correspondences = Correspondences ⋃ newCor
9:                  Justification = Justification ⋃ (newCor, scar^cur)
10:                 AddCorrespondence(newCor)
11:             end if
12:         end for
13:     end for
14: end procedure
```

8.3.5. Löschen einer SCAR

Attribute von Schemakonzepten können auch in SCARs referenziert werden. Abbildung 8.26 zeigt die Aktivitäten, die notwendig sind, um die Integrationsmenge wieder vollständig und konsistent zu machen.

Zuerst müssen für alle Instanzen des geänderten Schemakonzeptes der lokalen Produktontologie die korrespondierenden Instanzen der globalen Produktontologie ermittelt werden. Für letztere werden alle SCAAs des geänderten Schemakonzeptes rückgängig gemacht. Danach werden die Korrespondenzen zwischen den Instanzen aus lokalen Produktontologien und globaler Produktontologie gelöscht. Für alle Instanzen des geänderten Schemakonzeptes werden die zugehörigen Instanzen der nächst tieferen Produktdatenintegrationsebene ermittelt. Für all diese Instanzen wiederum werden erneut die korrespondierenden Instanzen der globalen Produktontologie bestimmt und die SCAAs rückgängig gemacht. Dieser Vorgang wird solange wiederholt, bis die letzte Produktdatenintegrationsebene erreicht worden ist. Nach diesen Schritten existieren Instanzen in der lokalen Produktontologie, die nicht auf Instanzen in der globalen Produktontologie abgebildet sind. Nun müssen entweder Korrespondenzen manuell durch Data Quality Engineers erstellt werden, oder diese Instanzen müssen von der Integration ausgeschlossen werden, damit die Integrationsmenge vollständig und konsistent ist.

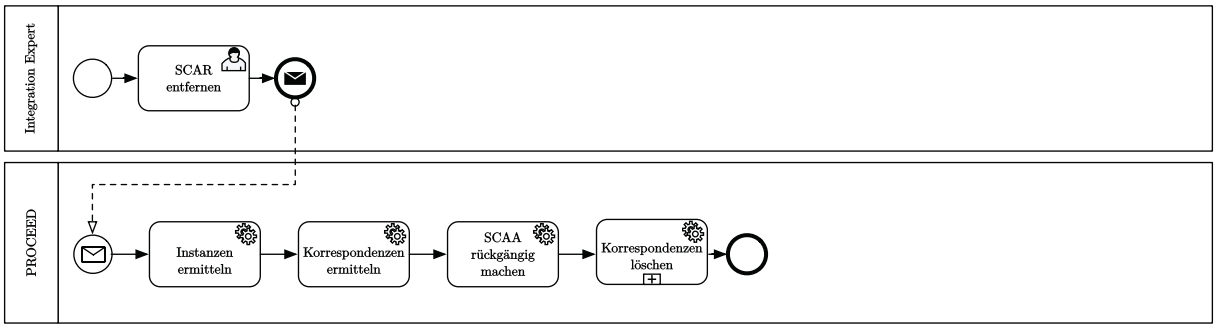


Abbildung 8.26.: Löschen einer SCAR

Abbildung 8.27 zeigt exemplarisch ein Szenario mit einer lokalen und globalen Produktontologie, bei dem eine Schemakonzept-Attributregel $scar_1$ gelöscht wurde ①. Zuerst müssen die potenziell betroffenen Instanzen des lokalen Schemakonzeptes bestimmt werden ②. Für jede dieser Instanzen werden vorhandene Korrespondenzen dahingehend untersucht, ob sie durch die gelöschte Schemakonzept-Attributregel erzeugt worden sind ③. Für diese Korrespondenzen werden bestehende Schemakonzept-Attributaktionen rückgängig gemacht ④. Anschließend werden die betroffenen Korrespondenzen gelöscht ⑤ und der Vorgang für die darunterliegenden Ebenen so lange wiederholt, bis die letzte Ebene erreicht wird ⑥.

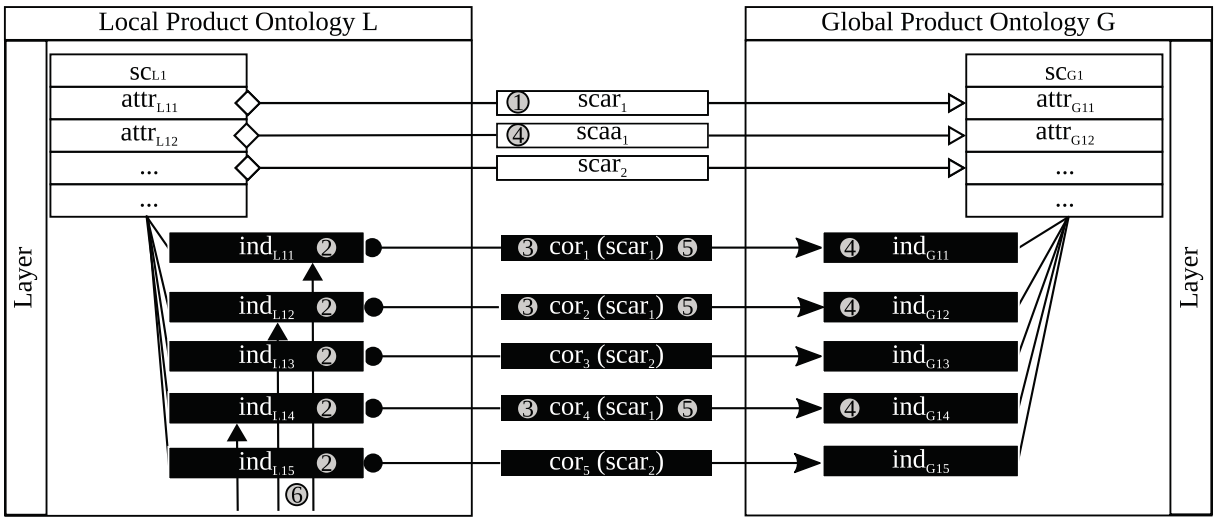


Abbildung 8.27.: Löschen einer SCAR

Algorithmus 19 formalisiert die Schritte aus Abbildung 8.27. Zunächst wird die Menge der potenziell betroffenen lokalen Instanzen ($Ind_L^{candidates}$) bestimmt (Zeile 2). Für jede Instanz wird die Menge der existierenden Korrespondenzen iteriert (Zeilen 3 bis 5). Falls eine Korrespondenz durch die gelöschte Schemakonzept-Attributregel erzeugt wurde (Zeile 6), werden Algorithmus 6 ausgeführt und die Korrespondenz anschließend entfernt (Zeilen 7 und 8).

Algorithmus 19 (SCAR löschen)

```
1:  procedure DeleteSCAR(scar)
2:      Ind_L^candidates = {ind_L ∈ Ind_L | ∃(scar.sc_L, ind_L) ∈ hasIndividual_L}
3:      for each ind_L^cur ∈ Ind_L^candidates do
4:          Cor^ind^cur = {cor ∈ Correspondences | ∃(ind_L^cur, ind_G) ∈ Correspondences}
5:          for each cor^cur ∈ Cor^ind^cur do
6:              if ∃(cor^cur, scar) ∈ Justification then
7:                  DeleteCorrespondence(cor^cur)
8:                  Correspondences = Correspondences\cor^cur
9:              end if
10:         end for
11:     end for
12: end procedure
```

8.3.6. Änderung einer SCAR

Abbildung 8.28 zeigt die benötigen Aktivitäten, wenn ein innerhalb einer SCAR verwendetes Attribut eines Schemakonzeptes geändert wird. Analog zum Ändern einer Schemakonzept-Attributaktion setzt sich das Ändern einer Schemakonzept-Attributregel aus dem Löschen und Einfügen von Korrespondenzen zusammen. Zunächst muss für alle Instanzen der lokalen Produktontologie des geänderten Schemakonzeptes die Menge der korrespondierenden Instanzen der globalen Produktontologie ermittelt werden. Für jedes Element dieser Menge werden all diejenigen SCAAs der lokalen Produktontologie rückgängig gemacht, die das gelöschte Attribut

referenzieren. Anschließend werden die existierenden Korrespondenzen gelöscht. Danach werden die geänderte Schemakonzept-Attributregel für die lokalen und globalen Instanzen neu ausgewertet und bei Treffern entsprechende Korrespondenzen erzeugt.

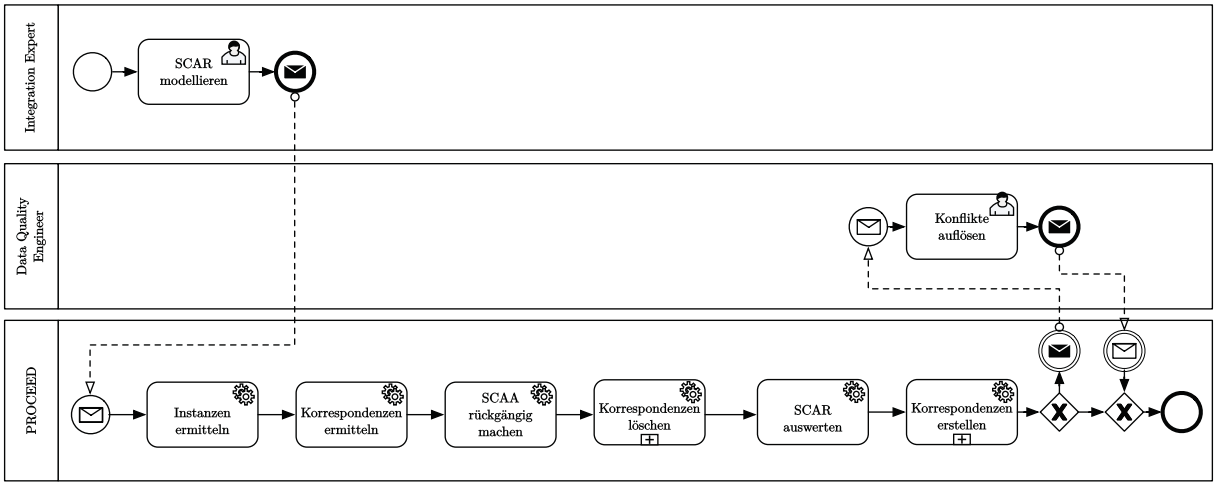


Abbildung 8.28.: Änderung einer SCAR

Abbildung 8.29 illustriert eine lokale und globale Produktontologie, in der die Schemakonzept-Attributregel $scar_1$ geändert wurde ①. Die Schritte ② bis ⑥ laufen identisch zum Abschnitt 8.3.5 ab. Nach Löschen derjenigen Korrespondenzen, die durch die alte Schemakonzept-Attributregel erstellt wurden, wird die neue Schemakonzept-Attributregel für die lokalen und globalen Instanzen ausgewertet ⑦. Bei Übereinstimmungen werden Korrespondenzen erstellt ⑧ und der Vorgang für die nächst tiefere Integrationsebene fortgesetzt ⑨.

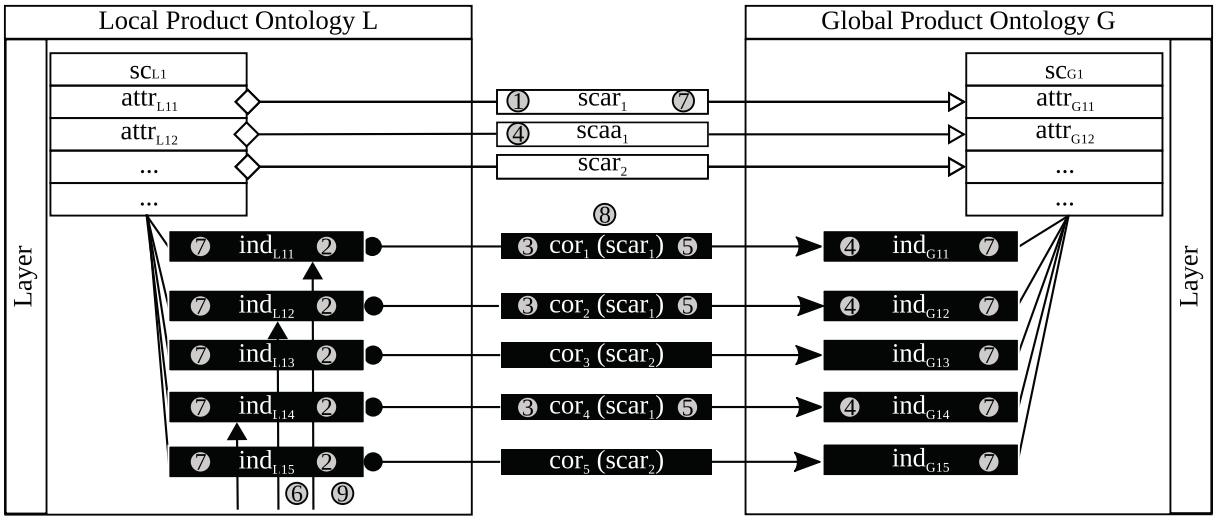


Abbildung 8.29.: Bearbeitung einer SCAR

Algorithmus 20 formalisiert die zuvor beschriebenen Schritte aus Abbildung 8.29. Zunächst wird Algorithmus 19, gefolgt von Algorithmus 18, aufgerufen.

Algorithmus 20 (SCAR ändern)

```
1: procedure ChangeSCAR(scar)
2:     DeleteSCAR(scar)
3:     AddSCAR(scar)
4: end procedure
```

Teil III

Evaluation

9. Proof-of-Concept Implementierung

In diesem Kapitel wird die Architektur des PROCEED-Systems erläutert, das die Konzepte aus den Kapiteln 6 bis 8 realisiert. Des Weiteren wird die technische Realisierung der Architektur vorgestellt, mit dem Ziel, deren technische Machbarkeit zu demonstrieren. Schließlich werden verwandte Arbeiten diskutiert.

9.1. PROCEED Systemarchitektur

Abbildung 9.1 illustriert die zu Grunde liegende Architektur des PROCEED-Systems mittels UML-Komponentendiagramm. Die einzelnen Komponenten der Architektur, die nachfolgend vorgestellt werden, ergeben sich aus den in den Kapiteln 6 bis 8 vorgestellten Konzepten.

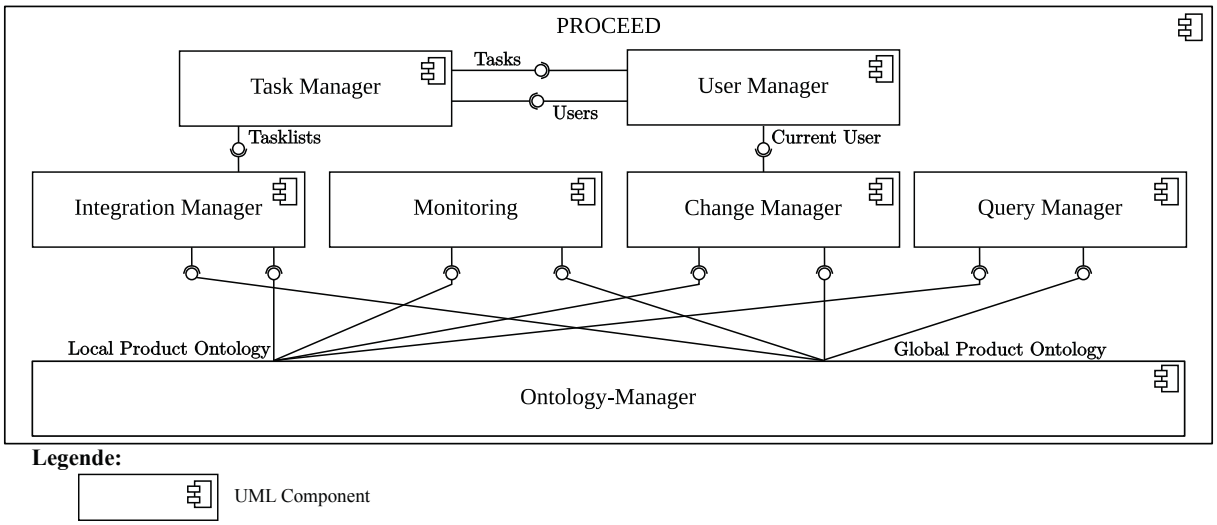


Abbildung 9.1.: Architektur des PROCEED-Systems

Der `Ontology Manager` verwaltet die lokalen und globalen Produktontologien und stellt die strukturelle Konsistenz einzelner Produktontologien sicher (vgl. Abschnitt 6.2.1).
Startpunkt des PROCEED-Systems ist ein *Integrationsprojekt*, welches Anwendungsfälle realisiert. Ein Integrationsprojekt besteht aus genau einer globalen und mindestens einer lokalen Produktontologie. Der `Integration Manager` realisiert den Integrationsalgorithmus aus Abschnitt 6.3 und interagiert mit dem `Change Manager`, welcher die Änderungsoperationen aus Kapitel 8 umsetzt. Das `Monitoring` wiederum berechnet und überwacht definierte Qualitätsmetriken (siehe Kapitel 7). Der `Query Manager` verwaltet Abfragen, die an die integrierten Produktdaten gestellt werden. Der `User Manager` verwaltet die verschiedenen Rollen und Nutzer (siehe Abschnitt 6.2.5). Er übergibt Informationen zum aktuell registrierten Benutzer des PROCEED-Systems und hinterlegt Metadaten für Artefakte von Produktontologien, falls Änderungen durchgeführt wer-

den. Der `Task Manager` weist Integrationsaufgaben, die während der Integration anfallen, an verschiedene Rollen und Nutzer zu (siehe Abschnitt 6.4).
Das PROCEED-System gliedert sich in eine existierende IT-Systemlandschaft eines Unternehmens ein. Folglich müssen bestehende Systeme und die ihnen zugrundliegende Prozesse (z.B. Änderungsmanagement) bei einer Integration berücksichtigt werden. Für die Population lokaler Produktontologien mit Instanzen sind unterschiedliche Lösungen möglich. Diese Lösungsalternativen werden im folgenden Abschnitt diskutiert.
Im Bezug auf Änderungen in lokalen Informationssystemen muss entschieden werden, in welchen Abständen diese an das PROCEED-System gemeldet werden. Dabei sind mehrere Szenarien möglich:

1. Nach Freigabe einer Änderung in einem Informationssystem wird diese sofort an das PROCEED-System übermittelt. Im weiteren Verlauf bezeichnen wir dies als *Ad-hoc-Integration*.

2. Eine weitere Möglichkeit ist es, Änderungen an das PROCEED-System zu übermitteln, wenn eine zuvor definierte Schwelle bzgl. der Anzahl an Änderungen überschritten worden ist. Diese Möglichkeit wird als *Schwellen-Integration* bezeichnet. Damit ein lokales Informationssystem diese Variante unterstützen kann, muss es in der Lage sein, die Anzahl an Änderungen zu überwachen.

3. Unter *Zeitbasierter Integration* schließlich bezeichnen wir die Möglichkeit, dass lokale Informationssysteme Änderungen in einem festdefinierten Zeitintervall aufzeichnen und diese am Ende des Intervalls dem PROCEED-System bereitstellen.

Neben dem zeitlichen Aspekt von Änderungen müssen die Rollen lokaler Informationssysteme und des PROCEED-Systems definiert werden. Ein Informationssystem kann sowohl sich passiv als auch aktiv in einem Systemverbund, das mehrere Systeme integriert, verhalten. Ein Informationssystem interagiert aktiv mit einem anderen, wenn es Informationen an dieses andere Informationssystem übermittelt. Für die Integration lokaler Informationssysteme in das PROCEED-System können Szenarien definiert werden, in denen lokale Informationssysteme aktiv oder passiv sind.

- Im ersten Szenario sind lokale Informationssysteme aktiv, d.h. sie melden Änderungen aktiv an das PROCEED-System (*Push-Prinzip*). Dies kann etwa über API-Aufrufe oder nachrichtenbasiert erfolgen. Im zweiten Szenario ist das PROCEED-System aktiv und ruft die lokalen Informationssysteme auf (*Pull-Prinzip*). Technisch kann dies auf Grundlage von API-Aufrufen der lokalen Informationssysteme oder nachrichtenbasiert erfolgen.

- Eine weitere Möglichkeit besteht darin, dass lokale Informationssysteme ihre Änderungen in einer strukturierten Form exportieren und als Datei zur Verfügung stellen. Das PROCEED-System greift dann auf diese Dateien zu und reagiert bei Erkennung von Änderungen.

Die beiden Szenarien und ihre verschiedenen technischen Realisierungen bieten Vor- und Nachteile. Im ersten Szenario muss das PROCEED-System eine klar definierte API für Änderungsoperationen zur Verfügung stellen. Weiter muss jedes lokale Informationssystem, welches in das PROCEED-System integriert werden soll, um Änderungsoperationen erweitert werden, so dass

bei Änderungen die API des PROCEED-Systems aufgerufen wird. Im zweiten Szenario muss jedes lokale Informationssystem, das in das PROCEED-System integriert werden soll, ein API realisieren. Nur dann kann das PROCEED-System mögliche Änderungen abfragen. Darüber hinaus muss jedes Informationssystem erweitert werden, um Änderungen an Produktdaten aufzuzeichnen, so dass diese über die API bereitgestellt werden können. Eine weitere Möglichkeit besteht darin, dass die lokalen Informationssysteme keine Änderungen übermitteln, sondern nur den aktuell gültigen Stand von Produktdaten bereitstellen. In diesem Fall muss das PROCEED-System einen Differenzvergleich zwischen den aktuellen Daten der lokalen Produktontologien und den gemeldeten Ständen der lokalen Informationssysteme durchführen, um mögliche Änderungen zu ermitteln.

9.2. Grundlagen

Technisch ist die Architektur des PROCEED-Systems durch Teile des *Semantic Web Stacks* [Bra07] realisiert. Dieser ist in Abbildung 9.2 illustriert. Die für die Realisierung des PROCEED-Systems verwendeten Standards (vgl. graue Kästen) werden im Folgenden kurz erläutert.

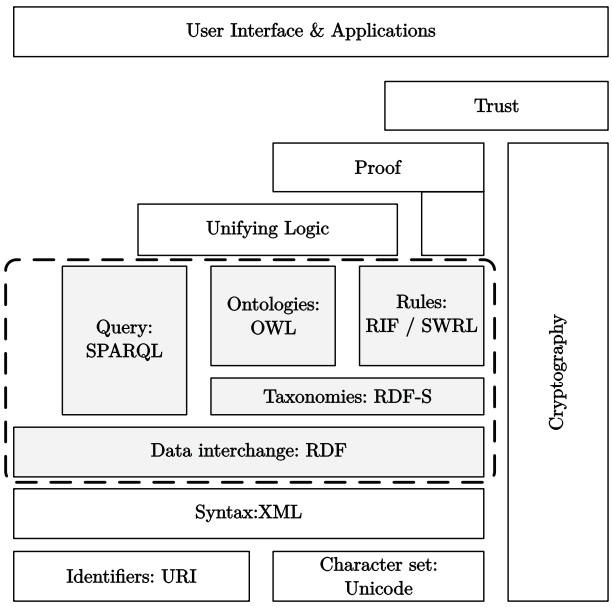


Abbildung 9.2.: Semantic Web Stack (In Anlehnung an [Bra07])

Das *Resource Description Framework* (RDF) [LSo98] ist eine Spezifikation des World Wide Web Konsortiums für die konzeptuelle Beschreibung von Informationen. Letztere bestehen aus eindeutig identifizierbaren Ressourcen und deren Beziehungen untereinander. Aussagen über Ressourcen werden als Tripel (Subjekt-Prädikat-Objekt) mit vordefinierter Semantik modelliert. Da RDF aus weit verbreiteten Web-Protokollen, wie z.B. URI, HTTP und XML basiert, ermöglicht es Interoperabilität. Abbildung 9.3 zeigt ein Beispiel eines RDF-Graphen, der zwei Aussagen enthält.

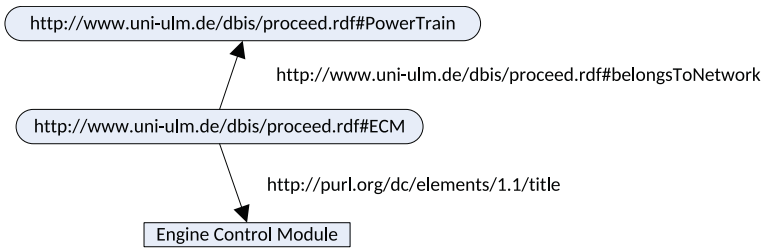


Abbildung 9.3.: Beispiel eines RDF-Graphs

Zusätzlich zu ihrer grafischen Repräsentation, lassen sich RDF-Graphen unterschiedlich serialisieren. Mögliche Syntaxformate sind zum Beispiel XML und Turtle [BBLP08]. Im Folgenden werden alle Ontologien mit Hilfe der Turtle-Syntax dargestellt. In Listing 9.1 wird das Beispiel aus Abbildung 9.3 in Turtle-Syntax serialisiert.

```
1  @prefix gpo:<http://www.uni-ulm.de/proceed/gpo.owl#> .
2  @prefix dc:<http://purl.org/dc/elements/1.1/> .
3    gpo:ECM dc:title "Engine Control Module " .
4    gpo:ECM belongsToNetwork gpo:PowerTrain .
```

Listing 9.1: Beispiel eines RDF-Graphen in Turtle-Syntax

Während RDF die Interoperabilität von Ontologien ermöglicht, lassen sich damit Aussagen über Klassen und Klassenbeziehungen (Relationen) nicht modellieren. Aus diesem Grund wurde die *RDF Vocabulary Description Language* (RDFS) [BG00] entworfen. Sie definiert grundlegende Elemente für die Beschreibung von Ontologien. In Einzelnen lassen sich mit Hilfe von RDFS Klassen, Hierarchien und Relationen zwischen Klassen definieren.
Listing 9.2 zeigt ein Beispiel für eine RDFS Ontologie. Es wird die Klasse `ElectronicContolUnit` definiert, für die zwei Instanzen existieren (`EngineControlModule` und `BodyControlModule`). Weiter wird die Eigenschaft `communicatesWith` definiert, die zwischen `ElectronicControlUnit` spezifiziert werden kann.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  @base <http://www.example.org#> .
4
5    :ElectronicControlUnit rdf:type rdfs:class .
6    :EngineControlModule rdf:type :ElectronicControlUnit .
7    :BodyControlModule rdf:type :ElectronicControlUnit .
8    :communicatesWith rdfs:domain :ElectronicControlUnit .
9    :communicatesWith rdfs:range :ElectronicControlUnit .
10   :EngineControlModule :communicatesWith :BodyControlUnit .
```

Listing 9.2: Beispiel einer RDFS-Ontologie

Mit RDFS ist es nicht möglich, Disjunktheit auszudrücken. Zudem existieren weitere Einschränkungen [Tat09], die dazu geführt haben, dass zusätzliche Ontologie-Beschreibungssprachen mit höherer Ausdrucksmächtigkeit entstanden sind. Basierend auf RDF und RDFS ermöglichen es die *Web Ontology Language* (OWL) [MVH+04] und deren Nachfolger OWL2 [MPSP+09], Ontologien unterschiedlicher Ausdrucksmächtigkeit zu modellieren. Die OWL-Spezifikation definiert dazu drei sog. OWL-Profile: OWL Lite, OWL DL und OWL Full. Zu den wichtigsten Entitäten einer OWL-Ontologie zählen *owl:Class*, *owl:ObjectProperty*, *owl:DatatypeProperty* und

owl:NamedIndividual. In Listing 9.3 ist eine OWL-Ontologie in Turtle-Syntax dargestellt; die Ontologie umfasst die beiden Klassen `ECU` und `ECUVariant` somit die Beziehung `hasVariant` zwischen ihnen. Für beide Klassen ist die Eigenschaft `hasName` als `xsd:string` definiert. Für beide Klassen ist jeweils ein Individual vorhanden (`Engine` und `Diesel Engine`). Abbildung 9.4 illustiert die OWL-Ontologie aus Listing 9.3 als RDF-Graphen.

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
3  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
6  @prefix : <http://www.example.org/ontologyexample.owl#> .
7  @base <http://www.example.org/ontologyexample.owl>
8
9  <http://www.example.org/ontologyexample.owl> rdf:type owl:Ontology ;
10                                                rdfs:label "Example" .
11 :ECU rdf:type owl:Class .
12 :ECUVariant rdf:type owl:Class .
13 :hasVariant rdf:type owl:ObjectProperty ;
14                rdfs:domain :ECU ;
15                rdfs:range :ECUVariant .
16 :hasName rdf:type owl:DatatypeProperty ;
17            rdfs:domain :ECU, ECUVariant ;
18            rdfs:range xsd:string .
19
20 :ind2 a :ECUVariant ;
21       :hasName "Diesel Engine" .
22
23 :ind1 a :ECU ;
24          :hasName "Engine" ;
25          :hasVariant :ind2 .
```

Listing 9.3: Beispiel einer OWL-Ontologie in Turtle-Syntax

Mit sog. *Reasonern* lässt sich aus bestehendem Wissen einer Ontologie neues Wissen automatisch ableiten [Bec09]. So ist es mit ihrer Hilfe möglich, den Modellierungsaufwand von Produktontologien zu reduzieren. Bspw. muss die Beziehung zwischen zusammengehörenden Instanzen, die sich auf unterschiedlichen Produkdatenintegrationsebenen befinden, nur in eine Richtung modelliert werden, während die Gegenrichtung durch den Einsatz eines Reasoners automatisch ermittelt wird. So muss zwischen zwei Instanzen ind_{L1} und ind_{L2} nur die Beziehung $(ind_{L1}, ind_{L2}) \in isBelow$ (s. Definition 21) modelliert werden, die Beziehung $(ind_{L2}, ind_{L1}) \in isAbove$ wird durch eine entsprechende Modellierung in der lokalen Produktontologie automatisch abgeleitet. Auch die Sicherstellung der strukturellen Konsistenz (siehe Abschnitt 6.2.1) kann durch die Verwendung von Reasonern unterstützt werden (s. Abschnitt 9.3.4).
Ein weiter Teil des Semantic Web Stacks ist SPARQL [HSP13] eine Sprache, mit der sich Abfragen an RDF-Graphen formulieren lassen. Da Schemata in RDF intrinsisch sind, d.h. selber Bestandteil des RDF-Graphen sind, lassen sich neben der Abfrage von Instanzen auch Schema-Informationen einfach mit Hilfe von SPARQL bestimmen. Listing 9.4 zeigt ein Beispiel für eine SPARQL-Abfrage für den RDF-Graphen aus Listing 9.2. Zunächst werden die benötigten Präfixe definiert, bevor die Abfrageoperation formuliert wird. Die `SELECT`-Operation liefert, analog zu einer SQL-Abfrage, als Ergebnis eine Tabelle zurück (siehe Tabelle 9.1). Neben `SELECT` existieren noch weitere Operationen wie `CONSTRUCT`, `ASK` und `DESCRIBE`, die im folgenden aber nicht weiter erläutert werden.

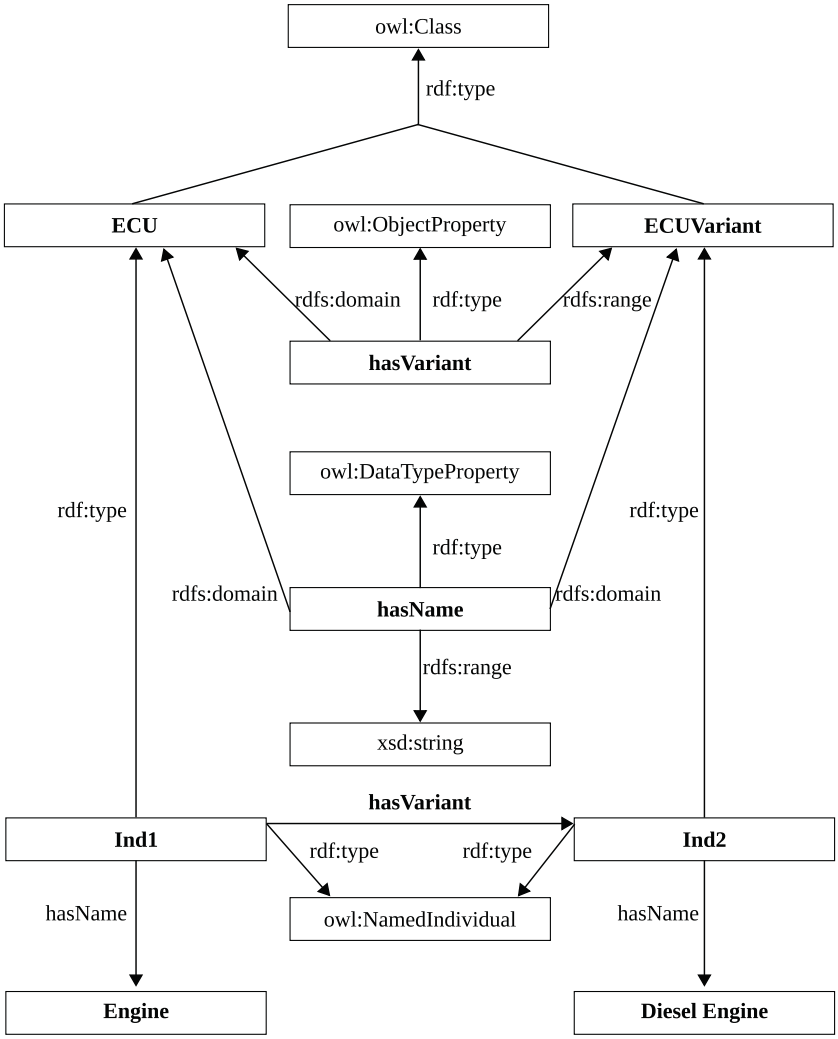


Abbildung 9.4.: RDF-Graph der OWL-Ontologie aus Listing 9.3

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX owl: <http://www.w3.org/2002/07/owl#>
3  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5  SELECT DISTINCT ?Class ?Ind
6  WHERE
7  {
8     ?Class rdf:type owl:Class .
9     ?Ind rdf:type owl:NamedIndividual
10    ?Ind rdf:type ?Class .
11 }
```

Listing 9.4: Beispiel einer SPARQL-Abfrage

Class	Ind
ECU	Ind1
ECUVariant	Ind2

Tabelle 9.1.: Ergebnis der SPARQL-Abfrage aus Listing 9.4

Ein weiterer Baustein des Semantic Web Stacks sind Regeln. Neben der Definition von Ontologien mit Hilfe der verfügbaren Beschreibungselemente (z.B. Klassen, Hierarchien und Relationen zwischen Klassen) existiert die Möglichkeit, komplexere Regeln für Ontologien mit Hilfe der *Semantic Web Rule Language* (SWRL) zu definieren. In Listing 9.5 ist ein Beispiel für eine SWRL-Regel dargestellt. Eine SWRL-Regel besteht immer aus einem *Antecedent* und einem *Consequent*, anders ausgedrückt aus einer Bedingung und einer Folgerung. Im Beispiel besteht die Bedingung aus der Konjunktion zweier OWL-Objecttypes `hasParent(?x1, ?x2)` ∧ `hasBrother(?x2,?x3)` sowie der Folgerung `hasUncle(?x1,?x3)`. Konkret heißt dies, wenn es in einer Ontologie drei Individuals `a, b` und `c` gibt, für die `hasParent(a, b)` ∧ `hasBrother(b,c)` gilt, muss auch `hasUncle(a,c)` gelten.

```
1 hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) → hasUncle(?x1,?x3)
```

Listing 9.5: Beispiel einer SWRL-Regel

Tabelle 9.2 fasst die Abbildungen der PROCEED-Konzepte auf OWL-Konzepte zusammen.

PROCEED	OWL
Schema Koncept	owl:Class
SCAR, SCAA	owl:ObjectProperty
Attribut	owl:DatatypeProperty
Instanz	owl:NamedIndividual

Tabelle 9.2.: Beziehungen zwischen den Konzepten von PROCEED und OWL

9.3. Realisierung der Lösungskonzepte

In den folgenden Abschnitten werden zunächst die Metaproduktontologie und anschließend die Umsetzung von SCARs und SCAAs in OWL-2 erläutert, bevor die Sicherstellung der strukturellen Konsistenz von Produktontologien detailliert wird. Anschließend wird die Umsetzung der weiteren Komponenten der Architektur beschrieben.

9.3.1. Metaproduktontologie

Abbildung 9.5 veranschaulicht die Serialisierung der einzelnen Produktontologien. Alle Produktontologien basieren auf einer sog. *Metaproduktontologie*, welche die strukturellen Vorgaben aus Abschnitt 6.2.1 als OWL-2-Ontologie umsetzt. Jede lokale und globale Produktontologie sowie alle Produktontologie-Mapings importieren diese Metaproduktontologie. Schemakonzept-Attributregeln und -Aktionen sowie die Korrespondenzen zwischen einer lokalen und globalen Produktontologie werden in einer separaten OWL2-Ontologie gespeichert (siehe `Product Ontology Mapping`). Diese Ontologie wiederum importiert die zugrundeliegende lokale Produktontologie und die globale.

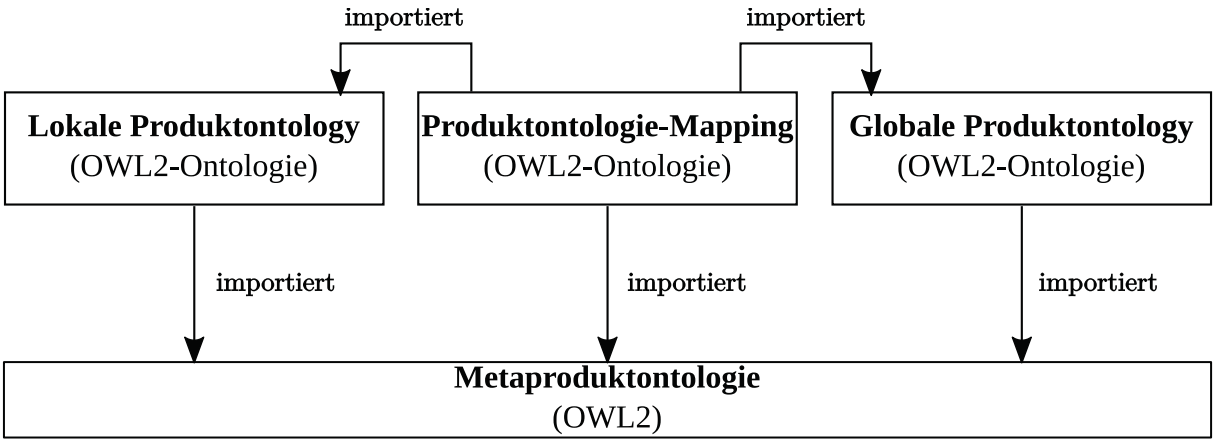


Abbildung 9.5.: Technische Realisierung lokaler und globaler Produktontologien mit OWL-2

Zusammengefasst werden in der Metaproduktontologie folgende Konzepte des PROCEED-Rahmenwerkes definiert:

- Produktdatenintegrationsebene,
- Lokale und globale Produktontologie,
- Strukturelle Konsistenz einer Produktontologie,
- Produktontologie-Mapping,
- Referenzwerte und
- Integrationsausschluss.

Die nachfolgende Beschreibung der OWL-2-Ontologie erfolgt mittels Turtle-Syntax. Der Übersicht halber gelten die Präfixe in Listing 9.7 für alle folgenden Listings und werden nicht jedes mal neu aufgelistet.

```
1 @prefix owl: <http://www.w3.org/2002/07/owl#> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

Listing 9.6: Metaproduktontologie als OWL-Ontologie

Die Metaproduktontologie ist vom Type `owl:Ontology`. Sie hat einen definierten Ontologie-Typ `hasOntologyType`, der als `owl:AnnotationProperty` definiert ist. Es wird zwischen Metaproduktontologie `MetaProductOntologyType`, globaler Produktontologie `GlobalProductOntologyType`, lokaler Produktontologie `LocalProductOntologyType` und Produktontologie-Mapping `ProductOntologyMappingZype` unterschieden.

```
1 @prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
2 @base <http://www.uni-ulm.de/proceed/metaontology> .
3 <http://www.uni-ulm.de/proceed/metaontology> rdf:type owl:Ontology ;
4                                              rdfs:label "Meta Product Ontology" ;
```

```
                                                    :hasOntologyType :MetaProductOntologyType .
:OntologyType rdf:type owl:Class .
:MetaOntologyType rdf:type owl:Class ;
                  rdfs:subClassOf :OntologyType .
:LocalProductOntologyType rdf:type owl:Class ;
                          rdfs:subClassOf :OntologyType .
:LocalMasterProductOntologyType rdf:type owl:Class ;
                          rdfs:subClassOf :OntologyType .
:GlobalProductOntologyType rdf:type owl:Class ;
                           rdfs:subClassOf :OntologyType .
:ProductOntologyMappingType rdf:type owl:Class ;
                            rdfs:subClassOf :OntologyType .
:hasOntologyType rdf:type owl:AnnotationProperty .
```

Listing 9.7: Metaproduktontologie als OWL-Ontologie

Schemakonzepte werden als `owl:Class` modelliert und es wird zwischen lokalem und globalem Schemakonzept (`LocalSchemaConcept`, `GlobalSchemaConcept`) unterschieden (siehe Listing 9.8). Attribute werden als `owl:DatatypeProperty` realisiert (`Attribute`). DieProduktdatenintegrationsebenen (siehe Abschnitt 6.2.1) werden als OWL-Klasse realisiert (`ProductDataIntegrationLayer`). Attribute sind `rdfs:subPropertyOf :Attribute`, welches ein `owl:DatatypeProperty` ist. Für jede Produktdatenintegrationsebene (`ProductDataIntegrationLayer`) sind einander entsprechende lokale und globale OWL-Klassen als Subklassen definiert.

```
@prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
@base <http://www.uni-ulm.de/proceed/metaontology> .

:SchemaConcept rdf:type owl:Class .

:LocalSchemaConcept rdf:type owl:Class ;
                    rdfs:subClassOf :SchemaConcept .

:GlobalSchemaConcept rdf:type owl:Class ;
                     rdfs:subClassOf :SchemaConcept .

:Attribute rdf:Type owl:DatatypeProperty.

:ProductDataIntegrationLayer rdf:type owl:Class .
:PDCLayer rdf:type owl:Class ;
          rdfs:subClassOf :ProductDataIntegrationLayer .

:ObjectLayer rdf:type owl:Class ;
             rdfs:subClassOf :ProductDataIntegrationLayer .

:VariantLayer rdf:type owl:Class ;
              rdfs:subClassOf :ProductDataIntegrationLayer .

:VersionLayer rdf:type owl:Class ;
              rdfs:subClassOf :ProductDataIntegrationLayer .

:LocalPDCLayer rdf:type owl:Class ;
               rdfs:subClassOf :PDCLayer .

:LocalObjectLayer rdf:type owl:Class ;
                  rdfs:subClassOf :ObjectLayer.

```

```
33 :LocalVariantLayer rdf:type owl:Class ;
34                       rdfs:subClassOf :VariantLayer .
35
36 :LocalVersionLayer rdf:type owl:Class ;
37                       rdfs:subClassOf :VersionLayer .
38
39 :GlobalPDCLayer rdf:type owl:Class ;
40                     rdfs:subClassOf :PDCLayer .
41
42 :GlobalObjectLayer rdf:type owl:Class ;
43                       rdfs:subClassOf :ObjectLayer .
44
45 :GlobalVariantLayer rdf:type owl:Class ;
46                        rdfs:subClassOf :VariantLayer .
47
48 :GlobalVersionLayer rdf:type owl:Class ;
49                        rdfs:subClassOf :VersionLayer .
```

Listing 9.8: Metaproduktontologie als OWL-Ontologie

Listing 9.9 definiert die notwendigen OWL-Klassen und Object-Properties, um die Hierarchie von Schemakonzepten zu modellieren. Beziehungen zwischen Schemakonzepten unterschiedlicher Produktdatenintegrationsebenen werden als `owl:ObjectProperty` modelliert. Letztere werden definiert als `rdfs:subPropertyOf` von `isBelow` und `isAbove`. Dabei ist `isAbove` als `owl:inverseOf` von `isBelow` definiert. Dadurch muss nur eine der beiden Hierarchie-Richtungen modelliert werden, während die andere durch Inferenz (mittels eines Reasoners) automatisch bestimmt wird.

```
1 @prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
2 @base <http://www.uni-ulm.de/proceed/metaontology> .
3
4 :isBelow rdf:Type owl:ObjectProperty .
5 :isAbove rdf:Type owl:ObjectProperty ;
6          owl:inverseOf :isBelow .
```

Listing 9.9: Modellierung der Hierarchie von Produktontologien

Listing 9.10 fasst die notwendigen Klassen und Beziehungen zusammen, die notwendig sind, um Schemakonzept-Attributregeln in OWL zu modellieren. SCARs zwischen Schemakonzepten aus lokaler und globaler Produktontologie werden als `owl:ObjectProperties` abgebildet. Die notwendigen Informationen der SCARs werden als `owl:AnnotationProperty` definiert. Dazu zählen die Matching-Funktion (`hasMatchingFunction`), mögliche Parameter (`hasMatchingParameter`) sowie das lokale und globale Attribut (`hasLocalAttribute`, `hasGlobalAttribute`). Jede SCAR wird entsprechend der Produktdatenintegrationsebene einem OWL-Objectproperty (`hasPDCSCARTo`, `hasObjectSCARTo`, `hasVariantSCARTo`, `hasVersionSCARTo`) untergeordnet.

```
1  @prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
2  @base <http://www.uni-ulm.de/proceed/metaontology> .
3
4  :MatchingFunction rdf:type owl:Class .
5
6  :MappingTable rdf:type owl:Class ;
7                  rdfs:subClassOf :MatchingFunction .
8
9  :RegularExpression rdf:type owl:Class ;
10                       rdfs:subClassOf :MatchingFunction .
```

```
11
12 :StringEquivalence rdf:type owl:Class ;
13                       rdfs:subClassOf :MatchingFunction .
14
15 :Levenshtein rdf:type owl:Class ;
16                rdfs:subClassOf :MatchingFunction .
17
18 :hasMatchingFunction rdf:type owl:AnnotationProperty .
19
20 :hasMatchingParameter rdf:type owl:AnnotationProperty .
21
22 :hasLocalAttribute rdf:type owl:AnnotationProperty .
23
24 :hasGlobalAttribute rdf:type owl:AnnotationProperty .
25
26 :hasSCARTo rdf:type owl:ObjectProperty .
27
28 :hasPDCSCARTo rdf:type owl:ObjectProperty ;
29                 rdfs:subPropertyOf :hasSCARTo .
30
31 :hasObjectSCARTo rdf:type owl:ObjectProperty ;
32                    rdfs:subPropertyOf :hasSCARTo .
33
34 :hasVariantSCARTo rdf:type owl:ObjectProperty ;
35                     rdfs:subPropertyOf :hasSCARTo .
36
37 :hasVersionSCARTo rdf:type owl:ObjectProperty ;
38                     rdfs:subPropertyOf :hasSCARTo .
39
40 :hasCorrespondenceProbability rdf:type owl:AnnotationProperty .
41 :hasCorrespondenceCreationType rdf:type owl:AnnotationProperty .
42
43 :CorrespondenceCreationType rdf:type owl:Class .
44
45 :AutomaticCreation rdf:type owl:Class ;
46                      rdfs:subClassOf :CorrespondenceCreationType .
47
48 :ManualCreation rdf:type owl:Class ;
49                      rdfs:subClassOf :CorrespondenceCreationType .
```

Listing 9.10: Abbildung von SCARs

Analog definiert Listing 9.11 die notwendigen Klassen und Beziehungen zur Definition von SCAAs. Neben dem lokalen und globalen Attribut (bereits in Listing 9.9 definiert) wird das Integrationsverhalten (`hasIntegrationBehavior`) benötigt. Entsprechend der Produktdatenintegrationsebene werden SCAAs OWL-Objectproperties (`hasPDCSCAATo`, `hasObjectSCAATo`, `hasVariantSCAATo`, `hasVersionSCAATo`) untergeordnet.

```
1 @prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
2 @base <http://www.uni-ulm.de/proceed/metaontology> .
3
4 :IntegrationBehavior rdf:type owl:Class .
5
6 :LatestIntegrationBehavior rdf:type owl:Class ;
7                              rdfs:subClassOf :IntegrationBehavior .
8
9 :LocalIntegrationBehavior rdf:type owl:Class ;
```

```
10                                  rdfs:subClassOf :IntegrationBehavior .
11
12  :GlobalIntegrationBehavior rdf:type owl:Class ;
13                                   rdfs:subClassOf :IntegrationBehavior .
14
15  :hasSCAATo rdf:type owl:ObjectProperty .
16  :hasPDCSCAATo rdf:type owl:ObjectProperty ;
17                      rdfs:subPropertyOf :hasSCAATo .
18
19  :hasObjectSCAATo rdf:type owl:ObjectProperty ;
20                          rdfs:subPropertyOf :hasSCAATo .
21
22  :hasVariantSCAATo rdf:type owl:ObjectProperty ;
23                          rdfs:subPropertyOf :hasSCAATo .
24
25  :hasVersionSCAATo rdf:type owl:ObjectProperty ;
26                          rdfs:subPropertyOf :hasSCAATo .
27
28  :hasIntegrationBehavior rdf:type owl:AnnotationProperty .
29
30  :hasAttributeJustification rdf:type owl:AnnotationProperty .
```

Listing 9.11: Abbildung von SCAAs

Listing 9.12 fasst grundlegende Metaattribute zusammen, die für alle Schemakonzepte, Instanzen, SCARs, SCAAs und Korrespondenzen definiert werden. Zu ihnen zählt der Ersteller eines Artefakts (`Creator_Metattr`, der Erstellungszeitpunkt (`CreationDate_Metattr`) sowie der letzte Änderungszeitpunkt (`LastModified_Metattr`). Des weiteren werden Metaattribute definiert, um Varianten- und Versionstypen für Schemakonzepte definieren zu können (`VariantType_Metattr` und `VersionType_Metattr`).

```
1   @prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
2   @base <http://www.uni-ulm.de/proceed/metaontology> .
3
4   @prefix : <http://www.uni-ulm.de/proceed/metaontology.owl#> .
5
6   :Creator_Metattr rdf:type owl:DatatypeProperty ;
7                    rdfs:subClassOf :Metaattribute ;
8                    rdfs:domain :SchemaConcept .
9
10  :CreationDate_Metattr rdf:type owl:DatatypeProperty ;
11                        rdfs:subClassOf :Metaattribute ;
12                        rdfs:domain :SchemaConcept .
13
14  :LastModified_Metattr rdf:type owl:DatatypeProperty ;
15                        rdfs:subClassOf :Metaattribute ;
16                        rdfs:domain :SchemaConcept .
17
18  :hasVariantType_Metaattr rdf:type owl:AnnotationProperty ;
19                           rdfs:subClassOf :Metaattribute ;
20                           rdfs:domain :LocalVariantLayer .
21
22  :hasVersionType_Metaattr rdf:type owl:AnnotationProperty ;
23                           rdfs:subClassOf :Metaattribute ;
24                           rdfs:domain :LocalVersionLayer .
```

Listing 9.12: Definition von Metaattributen

In Listing 9.13 sind die notwendigen Klassen dargestellt, um Instanzen von einer automatischen Integration auszuschließen. Alle Schemakonzepte, die als Subklasse von `ExcludeSCFromIntegration` definiert sind, werden bei der Integration sowie der Bestimmung der Integrationsqualität nicht berücksichtigt. Gleiches gilt für alle Instanzen des Typs `ExcludeIndFromIntegration`.

```
@prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
@base <http://www.uni-ulm.de/proceed/metaontology> .

  :ExcludeFromIntegration rdf:type owl:Class .
  :ExcludeSCFromIntegration rdf:type owl:Class ;
                            rdfs:subClassOf :ExcludeFromIntegration .
  :ExcludeIndFromIntegration rdf:type owl:Class ;
                             rdfs:subClassOf :ExcludeFromIntegration .
```

Listing 9.13: Integrationsauschluss von Instanzen

Listing 9.14 illustriert die notwendigen Klassen, um Referenzwerte für lokale und globale Produktontologien definieren zu können. Jede Integrationsqualitätsmetrik aus Kapitel 7 ist der OWL-Klasse `IntegrationQualityMetric` untergeordnet. Die Referenzwerte der einzelnen Metriken sind Instanzen der jeweiligen Klassen. Sie besitzen einen Zeitpunkt (`hasDueDate`) und einen Wert (`hasValue`).

```
@prefix : <http://www.uni-ulm.de/proceed/metaontology#> .
@base <http://www.uni-ulm.de/proceed/metaontology> .

  :IntegrationQualityMetric rdf:type owl:Class .
  :hasDueDate rdf:type owl:DatatypeProperty ;
              rdfs:domain :IntegrationQualityMetric ;
              rdfs:range xsd:Date .

  :hasValue rdf:type owl:DatatypeProperty ;
            rdfs:domain :IntegrationQualityMetric ;
            rdfs:range xsd:Float .

  :LocalSCCompleteness rdf:type owl:Class ;
                       rdfs:subClassOf :IntegrationQualityMetric .

  :LocalSCARCompleteness rdf:type owl:Class ;
                         rdfs:subClassOf :IntegrationQualityMetric .

  :LocalSCAACompleteness rdf:type owl:Class ;
                         rdfs:subClassOf :IntegrationQualityMetric .

  :LocalSCARSCAACompleteness rdf:type owl:Class ;
                             rdfs:subClassOf :IntegrationQualityMetric .

  :LocalIndCompleteness rdf:type owl:Class ;
                        rdfs:subClassOf :IntegrationQualityMetric .

  :GlobalSCCompleteness rdf:type owl:Class ;
                        rdfs:subClassOf :IntegrationQualityMetric .

  :GlobalIndCompleteness rdf:type owl:Class ;
                         rdfs:subClassOf :IntegrationQualityMetric .

  :GlobalIndAttrCompleteness rdf:type owl:Class ;
                             rdfs:subClassOf :IntegrationQualityMetric .
```

```
36
37  :GlobalIndAttrConsistency rdf:type owl:Class ;
38                            rdfs:subClassOf :IntegrationQualityMetric .
```

Listing 9.14: Definition von Referenzwerten

9.3.2. Abbildung einer Produktontologie

Basierend auf der Metaproduktontologie werden in den folgenden Listings beispielhaft eine lokale und die globale Produktontologie mit Hilfe von OWL-2 abgebildet. In Listing 9.15 sind die grundlegenden Eigenschaften der lokalen Produktontologie `lpo1` spezifiziert. Um die Lesbarkeit zu erhöhen, enden die Bezeichnungen der Ontologie-Artefakte auf das entsprechende Konzept des PROCEED-Rahmenwerkes. Beispielsweise enden Schemakonzepte auf `SC`, Instanzen auf `Ind` und Attribute auf `Attr`.

```
1  @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
2  @prefix : <http://www.uni-ulm.de/proceed/lpo1#> .
3  @base <http://www.uni-ulm.de/proceed/lpo1> .
4
5    lpo1 rdf:type owl:Ontology ;
6         rdfs:label "Local Product Ontology 1" ;
7         owl:versionInfo "1.0" ;
8         rdfs:comment "Local Product Ontology 1" ;
9         owl:imports <http://www.uni-ulm.de/proceed/metaontology.owl#> ;
10        mo:ontologyType mo:LocalProductOntology .
```

Listing 9.15: Produktontologie als OWL-Ontologie

Die lokale Produktontologie `lpo1` besitzt je ein Schemakonzept für jede Produktdatenintegrationsebene (siehe `Product_SC`, `ECU_SC`, `ECUVariant_SC` und `ECUVersion_SC` in Listing 9.16). Die Zuordnung eines Schemakonzeptes zu einer Ebene erfolgt durch Unterordnung in die entsprechenden OWL-Klassen, die zuvor in der Metaproduktontologie definiert wurden.

```
1  @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
2  @prefix : <http://www.uni-ulm.de/proceed/lpo1#> .
3  @base <http://www.uni-ulm.de/proceed/lpo1> .
4
5    :Product_SC rdf:type owl:Class ;
6                rdfs:label "Product" ;
7                rdfs:subClassOf mo:LocalPDCLayer .
8    :ECU_SC rdf:type owl:Class ;
9                rdfs:label "ECU" ;
10               rdfs:subClassOf mo:LocalObjectLayer .
11   :ECUVariant_SC rdf:type owl:Class ;
12                   rdfs:label "ECU Variant" ;
13                   rdfs:subClassOf mo:LocalVariantLayer .
14   :ECUVersion_SC rdf:type owl:Class ;
15                   rdfs:label "ECU Version" ;
16                   rdfs:subClassOf mo:LocalVersionLayer .
```

Listing 9.16: Produktontologie als OWL-Ontologie

Die Hierarchie der Schemakonzepte zeigt Listing 9.16. Zwischen jeweils zwei Schemakonzepten, die sich auf über- bzw. untereinanderliegenden Produktdatenintegrationsebenen befinden, sind entsprechende OWL-Objectproperties definiert.

```
1  @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
2  @prefix : <http://www.uni-ulm.de/proceed/lpo1#> .
3  @base <http://www.uni-ulm.de/proceed/lpo1> .
4
5    :ecuBelowProduct rdf:type owl:ObjectProperty ;
6                     rdfs:label "ECUIsBelowProduct" ;
7                     rdfs:subPropertyOf mo:isBelow ;
8                     rdfs:domain :ECU_SC ;
9                     rdfs:range :Product_SC .
10
11   :productAboveEcu rdf:type owl:ObjectProperty ;
12                    rdfs:label "ProductIsAboveECU" ;
13                    rdfs:subPropertyOf mo:isAbove ;
14                    rdfs:domain :Product_SC ;
15                    rdfs:range :ECU_SC .
16
17   :ecuBelowProduct owl:inversOf lpo1:productAboveEcu .
18
19   :ecuVariantBelowEcu rdf:type owl:ObjectProperty ;
20                       rdfs:label "ECUVariantIsBelowECU" ;
21                       rdfs:subPropertyOf mo:isBelow ;
22                       rdfs:domain :ECUVariant_SC ;
23                       rdfs:range :ECU_SC .
24
25   :ecuAboveEcuVariant rdf:type owl:ObjectProperty ;
26                       rdfs:label "EcuIsAboveECUVariant" ;
27                       rdfs:subPropertyOf mo:isAbove ;
28                       rdfs:domain :ECU_SC ;
29                       rdfs:range :ECUVariant_SC .
30
31   :ecuVariantBelowEcu owl:inversOf lpo1:ecuAboveEcuVariant .
32
33   :ecuVersionBelowEcuVariant rdf:type owl:ObjectProperty ;
34                              rdfs:label "ECUVersionIsBelowECUVariant" ;
35                              rdfs:subPropertyOf mo:isBelow ;
36                              rdfs:domain :ECUVersion_SC ;
37                              rdfs:range :ECUVariant_SC .
38
39   :ecuVariantAboveEcuVersion rdf:type owl:ObjectProperty ;
40                              rdfs:label "EcuVariantIsAboveECUVersion" ;
41                              rdfs:subPropertyOf mo:isAbove ;
42                              rdfs:domain :ECUVariant_SC ;
43                              rdfs:range :ECUVersion_SC .
```

Listing 9.17: Produktontologie als OWL-Ontologie

Für die Schemakonzepte der lokalen Produktontologie `lpo1` sind folgende Attribute definiert (siehe Listing 9.18). Jeder Instanz der Schemakonzepte besitzt einen Namen (`name_attr`). Für das Schemakonzept `ECUVariant` ist ein Attribut definiert, welches die Beschreibung der Variante beinhaltet (`variant_Attr`). Für `ECUVersion` sind sowohl eine Versionsummer (`versionId_Attr` als auch eine Referenz auf ein Dokument (`versionReference`) spezifiziert. Schließlich definieren `mo:hasVariantType_Metaattr` und `mo:hasVersionType_Metaattr` die Art des zugrundeliegenden Variantenmechanismus und der Versionierung.

```
@prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
@prefix : <http://www.uni-ulm.de/proceed/lpo1#> .
@base <http://www.uni-ulm.de/proceed/lpo1> .

:name_Attr rdf:type owl:DatatypeProperty ;
                    rdfs:subPropertyOf mo:Attribute ;
                    rdfs:domain :Product_SC, :ECU_SC, :ECUVariant_SC, :ECUVersion_SC ;
                    rdfs:range xsd:string .

:variant_Attr rdf:type owl:DatatypeProperty ;
                rdfs:subPropertyPf mo:Attribute ;
                rdfs:domain :ECUVariant ;
                rdfs:range xsd:string .

:versionId_Attr rdf:type owl:DatatypeProperty ;
                rdfs:subPropertyPf mo:Attribute ;
                rdfs:domain :ECUVersion ;
                rdfs:range xsd:integer .

:versionReference_Attr rdf:type owl:DatatypeProperty ;
                rdfs:subPropertyPf mo:Attribute ;
                rdfs:domain :ECUVersion ;
                rdfs:range xsd:string .

:ECUVariant mo:hasVariantType_Metaattr "Typ1"^^xsd:string .

:ECUVersion mo:hasVersionType_Metaattr "Baseline"^^xsd:string .
```

Listing 9.18: Attribut als OWL-DatatypeProperty

Für die einzelnen Schemakonzepte sind die Instanzen aus Listing 9.21 in `lpo1` modelliert. Für die Hierarchiebeziehungen zwischen Instanzen ist nur eine Richtung modelliert, die andere kann mittels Inferenz durch einen Reasoner bestimmt werden. Eine Instanz wird als `owl:NamedIndividual` modelliert. Die Zuordnung zu einem Schemakonzept erfolgt durch die Beziehung `rdf:type` sowie Attributwerte mittels Referenz entsprechender OWL-Datatype-Properties.

```
@prefix : <http://www.uni-ulm.de/proceed/lpo1#> .
@base <http://www.uni-ulm.de/proceed/lpo1> .

  :PDC1_Ind rdf:type owl:NamedIndividual, :Product_SC ;
            rdfs:label "Limousine2017" ;
            :name_Attr "Limousine"^^xsd:string .

  :Object1_Ind rdf:type owl:NamedIndividual, :ECU_SC ;
               rdfs:label "ECM" ;
               :name_Attr "Engine Control Module"^^xsd:string .

  :Object1_Ind :ecuBelowProduct :PDC1_Individual .

  :Variant1_Ind rdf:type owl:NamedIndividual, :ECUVariant_SC ;
                rdfs:label "ECM Diesel" ;
                :name_Attr "Engine Control Module for Diesel"^^xsd:string .

  :Variant2_Ind rdf:type owl:NamedIndividual, :ECUVariant_SC ;
                rdfs:label "ECM Gasoline" ;
                :name_Attr "Engine Control Module for Gasoline"^^xsd:string .

```

```
22  :Variant1_Ind :ecuVariantBelowEcu :Object1_Ind .
23
24  :Variant2_Ind :ecuVariantBelowEcu :Object1_Ind .
25
26  :Version1_Ind rdf:type owl:NamedIndividual, :ECUVersion_SC ;
27                rdfs:label "ECM Diesel V1" ;
28                :name_Attr "Engine Control Module for Diesel V1"^^xsd:string ;
29                :VersionId_Attr 1^^xsd:integer ;
30                :reference_Attr "\prj\...\ECM"^xsd:string
31
32  :Version2_Ind rdf:type owl:NamedIndividual, :ECUVersion_SC ;
33                rdfs:label "ECM Gasoline V1" ;
34                :name_Attr "Engine Control Module for Gasoline V1"^^xsd:string .
35
36  :Version2_Ind :VersionId_Attr 1^^xsd:string
37
38  :Version1_Ind :ecuVariantBelowEcu :Variant1_Ind .
39
40  :Version2_Ind :ecuVariantBelowEcu :Variant2_Ind .
```

Listing 9.19: Instanz als OWL-NamedIndividual

Zusammengefasst illustriert Abbildung 9.6 die zuvor beschriebe lokale Produktontologie `lpo1` mit Bezug auf die einzelnen Artefakte der OWL-Ontologie. Analog zeigt Abbildung 9.7 eine mögliche globale Produktontologie, deren OWL-2-Definition der Übersicht halber im Anhang A (siehe Listing 11.4) zu finden ist.

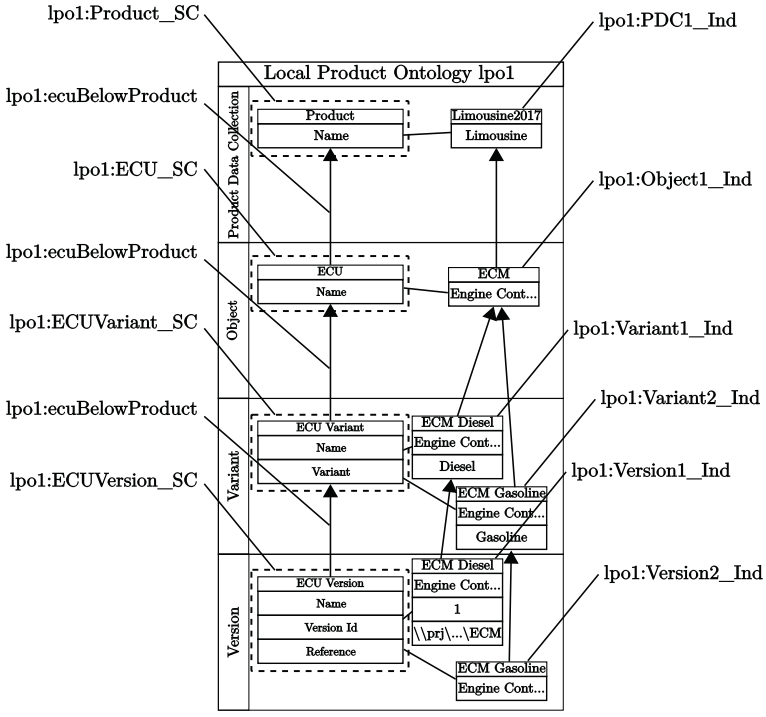


Abbildung 9.6.: Grafische Darstellung der lokalen Produktontologie `lpo1`

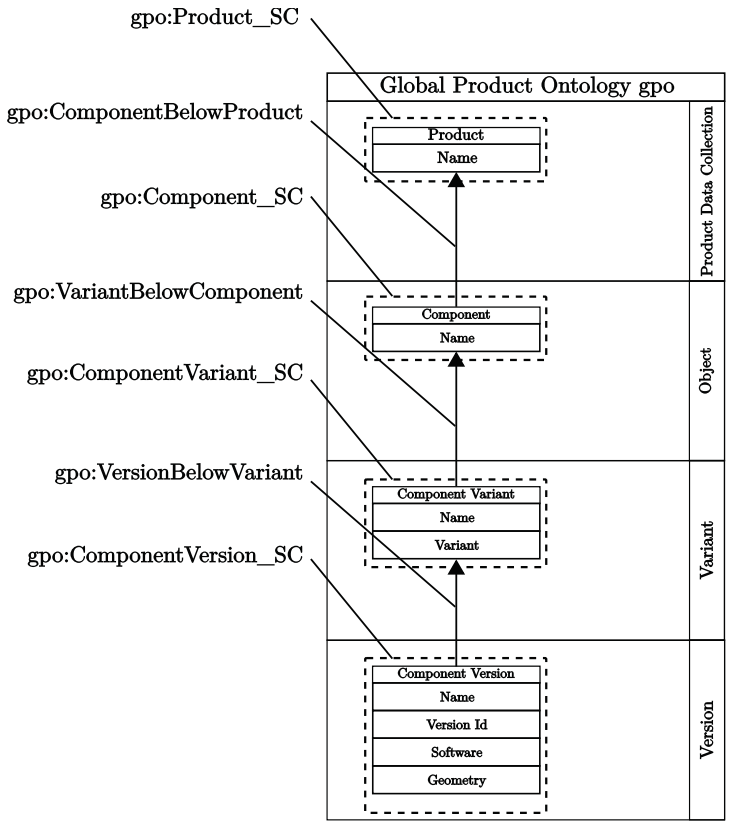


Abbildung 9.7.: Grafische Darstellung der globalen Produktontologie `gpo`

9.3.3. Abbildung des Produktontologie-Mappings

Nach Definition von lokaler und globaler Produktontologie, können Schemakonzept-Attributregeln und -Aktionen im Produktontologie-Mapping spezifiziert werden. Listing 9.22 zeigt einen Ausschnitt des Produktontologie-Mappings `pomapping1` zwischen der lokalen Produktontologie `lpo1` und der globalen Produktontologie `gpo`. Die Schemakonzept-Attributregel `productCorrespondensProject_SCAR` definiert für die Attribute `lpo1:name_Attr` und `gpo:label_Attr` eine Matching-Funktion `mo:StringEquivalence`.

```
1  @prefix lpo1: <http://www.uni-ulm.de/proceed/lpo1.owl#> .
2  @prefix gpo: <http://www.uni-ulm.de/proceed/gpo.owl#> .
3  @prefix : <http://www.uni-ulm.de/proceed/pomapping1#> .
4  @base <http://www.uni-ulm.de/proceed/pomapping1> .
5
6  :productCorrespondesProject_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty ;
7                                    rdfs:subPropertyOf mo:hasPDCSCARTo .
8                                    :hasMatchingFunction mo:StringEquivalence ;
9                                    :hasMatchingParameter "" ;
10                                   rdfs:domain lpo1:Product_SC ;
11                                   rdfs:range gpo:Product_SC ;
12                                   mo:hasLocalAttribute lpo1:name_Attr .
13                                   mo:hasGlobalAttribute gpo:label_Attr .
```

Listing 9.20: Beispiel einer SCAR in OWL2

Analog dazu zeit Listing 9.21 ein Beispiel für eine Schemakonzept-Attributaktion. Die SCAA `ecuVersion2ComponentVersion_SCAA` definiert eine Attributaktion zwischen den Attributen `lpo1:reference_Attr` und `gpo:ECUVersion_Ref_Attr` mit dem Integrationsverhalten `mo:LatestIntegrationBehavior`.

```
1  @prefix lpo1: <http://www.uni-ulm.de/proceed/lpo1.owl#> .
2  @prefix gpo: <http://www.uni-ulm.de/proceed/gpo.owl#> .
3  @prefix : <http://www.uni-ulm.de/proceed/pomapping1#> .
4  @base <http://www.uni-ulm.de/proceed/pomapping1> .
5
6  :ecuVersion2ComponentVersion_SCAA rdf:type owl:ObjectProperty ;
7                                    rdfs:subPropertyOf mo:hasVersionSCAATo
8                                    :hasIntegrationBehavior mo:LatestIntegrationBehavior ;
9
10                                   rdfs:domain lpo1:ECU_VERSION_SC ;
11                                   rdfs:range gpo:ComponentVersion_SC ;
12                                   mo:hasLocalAttribute lpo1:reference_Attr ;
13                                   mo:hasGlobalAttribute gpo:ECUVersion_Ref_Attr .
```

Listing 9.21: Beispiel einer SCAA in OWL2

Werden mit Hilfe des Integrationsalgorithmus (siehe Abschnitt 6.3.1) oder mit den Änderungsoperation (s. Kapitel 8) Korrespondenzen ermittelt, müssen sie im Produktontologie-Mapping gespeichert werden. Listing 9.22 zeigt ein Beispiel für eine Korrespondenz, die auf der SCAR `:productCor-respondesProject_SCAR` basiert. Die Korrespondenz ist zwischen der lokalen Instanz `lpo1:PDC1_Ind` und der globalen Instanz `gpo:PDC2_Ind` definiert, der Erstellungstyp (`mo:hasCorrespondenceCreationType`) ist `mo:AutomaticCreation`.

```
1  @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
2  @prefix lpo1: <http://www.uni-ulm.de/proceed/lpo1.owl#> .
3  @prefix gpo: <http://www.uni-ulm.de/proceed/gpo.owl#> .
4  @prefix : <http://www.uni-ulm.de/proceed/pomapping1#> .
5  @base <http://www.uni-ulm.de/proceed/pomapping1> .
6
7  [] a owl:Axiom ;
8     owl:annotatedProperty :productCorrespondesProject_SCAR ;
9     owl:annotatedSource lpo1:PDC1_Ind ;
10    owl:annotatedTarget gpo:PDC2_Ind ;
11    mo:hasCorrespondenceProbability "1.0"^^xsd:float ;
12    mo:hasCorrespondenceCreationType mo:AutomaticCreation .
```

Listing 9.22: Beispiel einer Korrespondenz in OWL2

Schließlich illustriert Listing 9.23 die Modellierung einer Attribut-Begründung (siehe Definition 36) in OWL-2. Der Attributwert `Link` des Attributs `ECU_Ref_Attr` für die Instanz `ComponentVersion1_Ind` ist durch die Schemakonzept-Attributaktion `ecuVersion2ComponentVersion_SCAA` zustande gekommen.

```
1  @prefix lpo1: <http://www.uni-ulm.de/proceed/lpo1.owl#> .
2  @prefix gpo: <http://www.uni-ulm.de/proceed/gpo.owl#> .
3  @prefix : <http://www.uni-ulm.de/proceed/pomapping1#> .
4  @base <http://www.uni-ulm.de/proceed/pomapping1> .
5
6  [ rdf:type owl:Axiom ;
7    owl:annotatedTarget "Link"^^xsd:string ;
8    owl:annotatedSource :ComponentVersion1_Ind ;
9    owl:annotatedProperty :ECU_Ref_Attr ;
```

```
10    :hasAttributeJustification :ecuVersion2ComponentVersion_SCAA
11  ] .
```

Listing 9.23: Beispiel einer Attribut-Begründung in OWL2

9.3.4. Sicherstellung der strukturellen Konsistenz

Die in Definition 24 (siehe Abschnitt 6.2.1) spezifizierten Eigenschaften sind Voraussetzung für die Umsetzung der Konzepte des PROCEED-Rahmenwerkes (z.B. Integrationsalgorithmus, Änderungsoperationen). Im Folgenden wird erläutert, wie die strukturelle Konsistenz einer Produktontologie im PROCEED-System technisch umgesetzt wird.
Mit Hilfe von SWRL lassen sich Regeln für OWL-Ontologien definieren, die dem OWL-DL-Profil (basierend auf Beschreibungslogiken) entsprechen. Zwar lassen sich mit SWRL Regeln definieren, die über die Ausdrucksmächtigkeit von OWL-DL-Ontologien hinaus gehen, jedoch werden zum Beispiel Ausdrücke wie `rdf:type` oder `rdfs:Class` nicht unterstützt. Aus diesem Grund wird die Sicherstellung der strukturellen Konsistenz von Produktontologien mit Hilfe von SPARQL-Abfragen realisiert. Zunächst werden die Schemakonsistenzregeln (vgl. Kapitel 6.2.1) wiederholte, bevor ihre Definition mittels SPAQRL erläutert wird.

Schemakonsistenzregel 1: Jede Produktdatenintegrationsebene besitzt mindestens ein Schemakonzept. Ein Schemakonzept kann nur mit einem Schemakonzept verbunden werden, das sich auf einer darunter liegenden Produktdatenintegrationsebene befindet.

Listing 9.24 realisiert einen Teil der *Schemakonsistenzregel 1* indem die Produktdatenintegrationsebenen bestimmt werden, für die keine Schemakonzepte definiert sind. `?l1` gibt die Produktdatenintegrationsebene und `?perLayer` die Anzahl der Schemakonzepte zurück. Da letztere gefiltert werden (`HAVING (?perLayer` *leq* `1)`), ist diese Anzahl immer `0`.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT ?l1 (COUNT(?sc) as ?perLayer)
4  WHERE {
5    ?sc rdfs:subClassOf ?l1 .
6    FILTER (?l1 = mo:LocalPDCLayer || ?l1 = mo:LocalObjectLayer || ?l1 = mo:LocalVariantLayer || ?l1 =
         mo:LocalVersionLayer)
7  }
8  GROUP BY ?l1
9  HAVING (?perLayer < 1)
```

Listing 9.24: SPARQL-Query zur Ermittlung fehlender Schemakonzepte

Listing 9.25 realisiert den restlichen Teil von *Schemakonsistenzregel 1*. Die Abfrage liefert alle Schemakonzept-Paare zurück, die sich auf übereinanderliegenden Produktdatenintegrationsebenen befinden, für die jedoch keine Hierarchie-Beziehung `isBelow` definiert ist. Da die Hierarchie-Beziehungen als `rdfs:subPropertyOf` definiert werden und SPARQL keine Inferenzen berücksichtigt, muss mit Hilfe eines Reasoners die Typisierung der Klassen vorgenommen werden.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT DISTINCT ?sc ?sc2
4  WHERE {
5    ?sc rdfs:subClassOf ?l1 .
```

```
6    ?sc2 rdfs:subClassOf ?l2 .
7    FILTER ( (?l1 = mo:LocalPDCLayer && ?l2=mo:LocalObjectLayer) || (?l1 = mo:LocalObjectLayer &&
         ?l2=mo:LocalVariantLayer) || (?l1 = mo:LocalVariantLayer && ?l2=mo:LocalVersionLayer))
8    FILTER NOT EXISTS {
9      ?op1 rdf:type owl:ObjectProperty .
10     ?op1 rdfs:subPropertyOf mo:isBelow .
11     ?op1 rdfs:range ?sc .
12     ?op1 rdfs:domain ?sc2 .
13   }
14 }
```

Listing 9.25: SPARQL-Query zur Bestimmung fehlender Hierarchie-Beziehungen zwischen Schemakonzepten

Schemakonsistenzregel 2: Für jedes Schemakonzept der Variant-Ebene ist der Variabilitätstyp definiert. **Schemakonsistenzregel 3:** Für jedes Schemakonzept der Versions-Ebene ist der Versionstyp definiert.

Die SPARQL-Abfrage in Listing 9.26 liefert die Liste `?sc`, welche diejenigen Schemakonzepte der Variant-Ebene umfasst, für die kein Metaattribut zur Beschreibung eines Variantentypes definiert ist. Analog liefert Listing 9.27 die Liste entsprechend **Schemakonsistenzregel 3**.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT ?sc
4  WHERE {
5    ?sc rdfs:subClassOf mo:LocalVariantLayer .
6  FILTER (not exists {?sc mo:hasVariantType_Metaattr ?x})
7  }
```

Listing 9.26: SPARQL-Query zur Bestimmung von Varianten-Schemakonzepten ohne Variantentyp

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT ?sc
4  WHERE {
5    ?sc rdfs:subClassOf mo:LocalVersionLayer .
6  FILTER (not exists {?sc mo:hasVersionType_Metaattr ?x})
7  }
```

Listing 9.27: SPARQL-Query zur Bestimmung von Versions-Schemakonzepten ohne Versionstyp

Instanzkonsistenzregel 1: Die Instanzen einer globalen Produktontologie sind zwischen den Produktdatenintegrationsebenen, analog zu ihren Schemakonzepten, über Beziehungen miteinander verbunden.

In Listing 9.28 werden diejenigen Instanzen `?ind1` und `?ind2` bestimmt, die sich in übereinander liegenden Produktdatenintegrationsebenen befinden, für die jedoch keine Beziehung definiert ist.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT DISTINCT ?ind1 ?ind2
4  WHERE {
5    ?ind1 rdf:type ?l1 .
6    ?ind2 rdf:type ?l2 .
```

```
7   ?ind1 rdf:type owl:NamedIndividual .
8   ?ind2 rdf:type owl:NamedIndividual .
9   FILTER ((not exists {?ind2 mo:isBelow ?ind1} || not exists {?ind1 mo:isAbove ?ind2}) && ( (?l1 = mo:
        LocalPDCLayer && ?l2 = mo:LocalObjectLayer) || (?l1 = mo:LocalObjectLayer && ?l2 = mo:
        LocalVariantLayer) || (?l1 = mo:LocalVariantLayer && ?l2 = mo:LocalVersionLayer)) )
10  }
```

Listing 9.28: SPARQL-Query zur Bestimmung fehlender Hierarchie-Beziehungen zwischen Instanzen

Instanzkonsistenzregel 2: Die Instanzen eines Schemakonzeptes auf der *Version*-Ebene sind durch *Vorgänger*- und *Nachfolger*-Beziehungen geordnet.

Die Vorgänger- und Nachfolgerbeziehungen zwischen Instanzen der Versions-Ebene sind genau dann nicht konsistent, wenn Instanzen existieren, für die weder Vor- noch Nachfolgerbeziehungen definiert sind. Die entsprechende SPARQL-Abfrage ist in Listing 9.29 dargestellt.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT DISTINCT ?ind1
4  WHERE {
5    ?sc rdfs:subClassOf mo:LocalVersionLayer .
6    ?ind1 rdf:type ?sc .
7    ?ind1 rdf:type owl:NamedIndividual .
8    FILTER (not exists{?ind1 mo:isSuccessorOf ?x} && not exists{?ind1 mo:isPredecessorOf ?x})
9  }
```

Listing 9.29: SPARQL-Query zur Ermittlung von Versionsinstanzen ohne Vorgänger und ohne Nachfolger

9.3.5. Realisierung von Integrationsqualitätsmetriken

Analog zur Sicherstellung der strukturellen Konsistenz von Produktontologien lassen sich die Integrationsqualitätsmetriken mit Hilfe von SPARQL-Abfragen umsetzen. In Listing 9.30 wird exemplarisch die lokale Schemakonzept-Vollständigkeit (vgl. Definition 44 in Kapitel 7) als SPARQL-Abfrage definiert. Im Detail werden mit Hilfe von `UNION` sowohl die Anzahl der Schemakonzepte `?scmapped` ermittelt, für die eine Schemakonzept-Attributregel vorhanden ist, als auch die Anzahl `?sc` aller Schemakonzepte, die nicht von der Integration ausgeschlossen sind. Das Ergebnis `?result` entspricht der Differenz aus beiden vorherigen Zahlen. Analog lassen sich alle Integrationsqualitätsmetriken aus Kapitel 7 mit Hilfe von SPARQL-Abfragen definieren.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
3  SELECT (count (distinct ?sc) as ?scmapped) (count (distinct ?sc2) as ?sc) ( (?scmapped/?sc) as ?result)
4  WHERE {
5
6    {
7    ?sc rdfs:subClassOf ?scl .
8    ?scar rdfs:subPropertyOf ?scarl .
9    ?scar rdfs:domain ?sc
10   FILTER ( (?scl = mo:LocalPDCLayer && ?scarl = mo:hasPDCSCARTo) || (?scl = mo:
         LocalObjectLayer && ?scarl = mo:hasObjectSCARTo) || (?scl = mo:LocalVariantLayer && ?scarl
         = mo:hasVariantSCARTo) || (?scl = mo:LocalVersionLayer && ?scarl = mo:hasVersionSCARTo))
11   }
```

```
12
13  UNION
14    {
15    ?sc2 rdfs:subClassOf ?scl2 .
16    FILTER ( ?scl2 = mo:LocalPDCLayer || ?scl2 = mo:LocalObjectLayer || ?scl2 = mo:LocalVariantLayer
          || ?scl2 = mo:hasVariantSCARTo || ?scl2 = mo:LocalVersionLayer)
17    FILTER NOT EXISTS {
18      ?sc2 rdfs:subClass mo:ExcludeSCFromIntegration
19    }
20    }
21
22  }
```

Listing 9.30: SPARQL-Query zur Bestimmung der lokalen Schemakonzept-Vollständigkeit

9.3.6. Realisierung des Integrationsalgorithmus sowie der Änderungsoperationen

Während die strukturelle Konsistenz und Qualitätsmetriken durch SPARQL-Abfragen realisiert werden, reicht dies für die Umsetzung des Integrationsalgorithmus (siehe Abschnitt 6.3.1) aus. Folglich wurde dieser Algorithmus softwareseitig mithilfe der OWL API [HB11] umgesetzt. Im Detail wurde dazu ein Plugin für den Ontologie-Editor Protégé [GMF+03] implementiert. Während Protégé ein weit verbreiteter Editor zur Modellierung von Ontologien ist, reichen die existierenden Plugins nicht aus, um alle Konzepte des PROCEED-Rahmenwerkes direkt zu unterstützen. Folglich wurde für Protégé ein Plugin realisiert, welches für die Umsetzung des PROCEED-Systems zuständig ist.
Abbildung 9.8 illustriert die technische Architektur von Protégé in der Version 4. Basierend auf Java und dem OSGi-Framework [All03] besteht Protégé aus OSGi-Plugins. Während `The Protégé 4 OWL Editor Plugin` für die Modellierung von OWL-Ontologien und die Umsetzung einer grafischen Oberfläche zuständig ist, wird die Serialisierung und Deserialisierung einer OWL-Ontologie mittels der `OWL API` umgesetzt. Darüber hinaus gibt es eine Vielzahl weiterer Plugins. Für die Realisierung des PROCEED-Systems wurden das `Hermit Reasoner Plugin` [SMH08] und das `Protégé SPARQL Plugin` verwendet. Basierend auf diesem Plugin baut das PROCEED-System als weiteres OSGI-Plugin auf den vorangehend genannten Plugins auf. Im Detail realisiert das PROCEED Plugin die Architektur aus Abbildung 9.1.

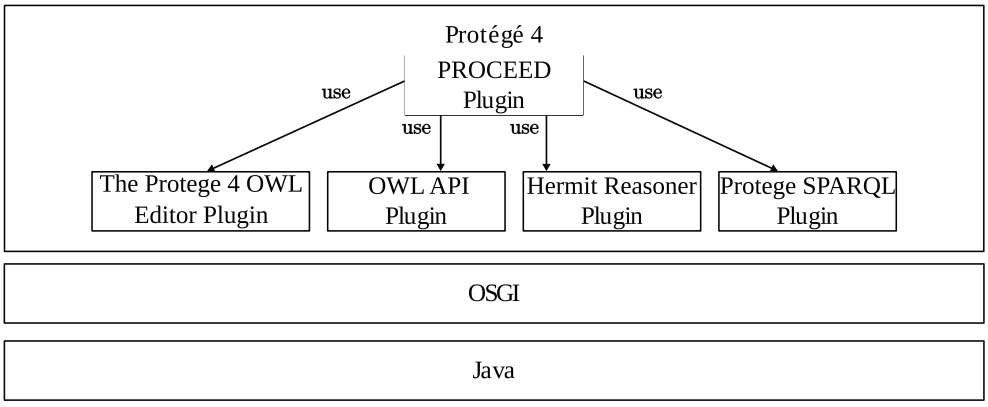


Abbildung 9.8.: Technische Architektur des PROCEED-Systems

Während der Editor von Protégé die Möglichkeit bietet, OWL-Ontologien zu modellieren, ist dieser Editor für Domänenexperten zu komplex, um Produktontologie zu modellieren. Zur einfachen Umsetzung der strukturellen Konsistenz wurden Konzepte für Benutzeroberflächen entworfen, mittels der sich Produktontologien sowohl in Form von Graphen als auch mit Hilfe von Listen zu erstellen lassen [Mer14]. Das Konzept definiert für verschiedene Nutzerrollen des PROCEED-Rahmenwerkes (siehe Abschnitt 6.2.5) unterschiedliche Sichten (z.B. Listen, Baumstrukturen und Graphen). Zur Reduktion der Komplexität wurde zusätzlich zu diesen Sichten ein Farb- und Formkonzept zur Darstellung von und Navigation in komplexen Produktontologien entworfen[1] [Hau15] (Grafiken zum Usability-Konzept befinden sich im Anhang G).
Teile dieser Konzepte wurden im PROCEED-Plugin für Protégé umgesetzt. Abbildung 9.9 zegt die Oberfläche, in der eine lokale und globale Produktontologie sowie Schemakonzept-Attributregeln und -aktionen modelliert werden können. Auf der linken Seite ist eine lokale Produktontologie mit entsprechenden Schemakonzepten und Instanzen illustiert. Für jede Produktdatenintegrationsebene existieren zwei Listen zur Darstellung von Schemakonzepten und Instanzen. Je nachdem, welches Element (Schemakonzept, Attribut, Instanz, Attributwert) in den Listen ausgewählt wird, sind bestimmte Änderungsoperationen (als Buttons dargestellt) ausführbar. Analog zu Schemakonzepten und Instanzen werden auch die Schemakonzept-Attributregeln und -aktionen den Integrationsebenen entsprechend angeordnet (siehe Abbildung 9.9 Mitte).

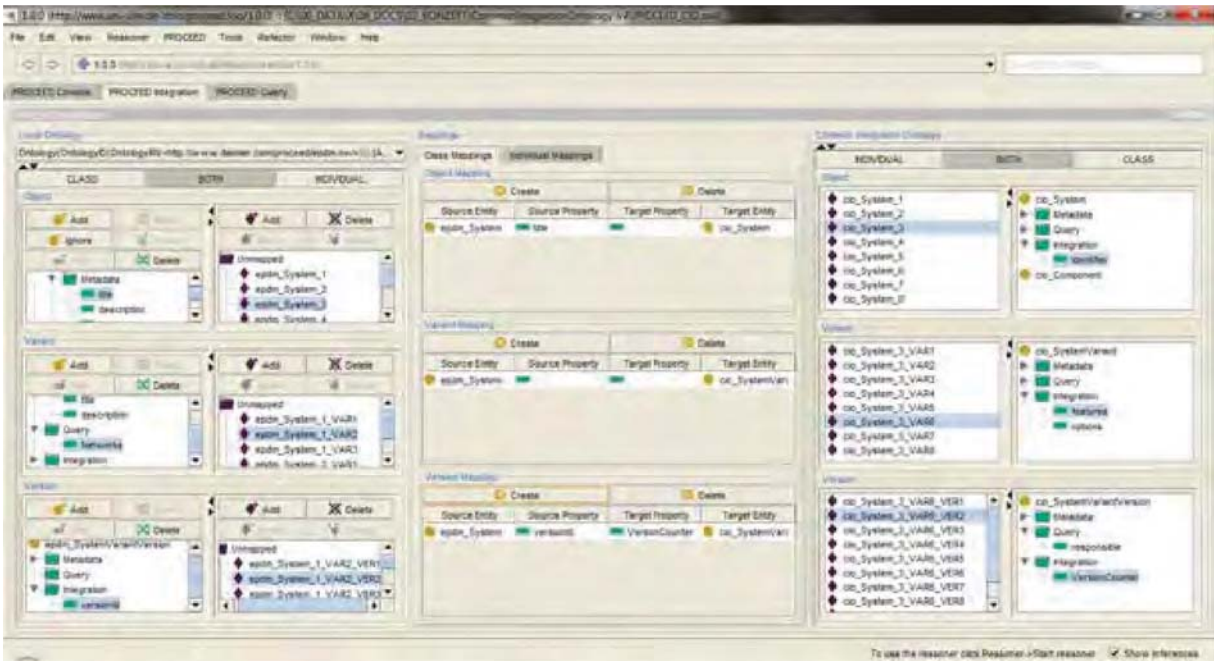


Abbildung 9.9.: Oberfläche des PROCEED-Plugins für Protégé

Abbildung 9.10 zeigt die Oberfläche für die Ausführung der Abfragen zur Realisierung der Anwendungsfälle, die mit Hilfe der integrierten Produktdaten realisiert werden sollen (vgl. Definition 8). Auf der linken Seite ist eine Liste mit definierten SPARQL-Abfragen dargestellt. Durch Auswahl aus der Liste wird die hinterlegte SPARQL-Abfrage in den Editor des SPARQL-Plugins

[1] Ein ähnlicher Ansatz wurde im Projekt niPRO für die Visualisierung von und Navigation in Prozess-Repositorien- und Ontologien realisiert [HMR12, MMR12, HMMR14].

geladen. Da die Integrationsqualitätsmetriken auf SPARQL basieren, kann der Abfrageeditor auch dazu verwendet werden, die einzelnen Metriken zu bestimmen.

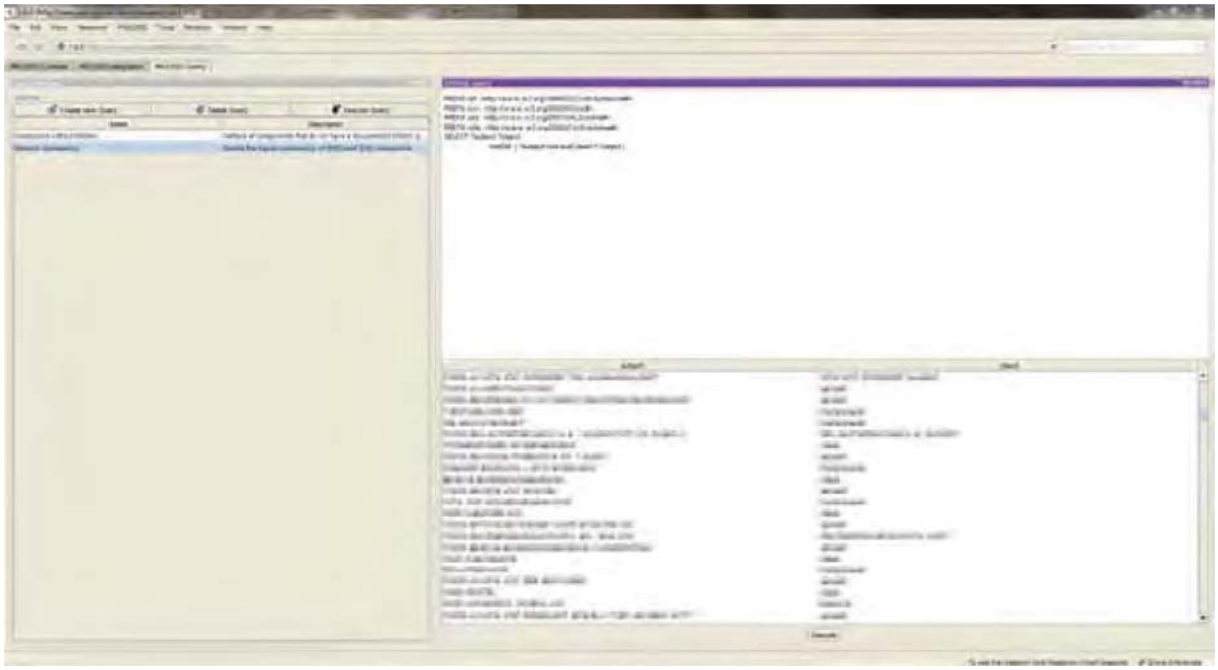


Abbildung 9.10.: Oberfläche zur Ausführung von Abfragen

10. Fallstudie

Zusätzlich zur skizzierten prototypischen Implementierung (vgl. Kapitel 9) wurde des PROCEED-Rahmenwerkes in Fallstudien in der Automobilindustrie evaluiert. Dieses Kapitel stellt eine solche Fallstudie vor und erläutert, welche Erkenntnisse sich aus der Anwendung des PROCEED-Rahmenwerkes ableiten lassen. Für jedes Integrationsszenario werden zunächst die Notwendigkeit zur Integration von Produktdaten motiviert und die zu realisierenden Anwendungsfälle beschrieben. Anschließend werden die zu integrierenden Informationssysteme und deren konzeptionelle Datenmodelle erläutert. Schließlich wird eine Lösung zur Integration der Informationssysteme mit Hilfe des PROCEED-Rahmenwerkes detailliert.

10.1. Abgleich von Stücklisten mit PROCEED

10.1.1. Motivation

Wie erläutert, besteht ein Produkt aus einer Vielzahl von Komponenten, deren Beziehungen zueinander eine hierarchische Produktstruktur bilden. Die Verwaltung dieser Komponenten und ihrer Beziehungen (d.h. der Produktstruktur) erfolgt mit Hilfe von PDMS. Die konstruktive Sicht auf ein Produkt wird als *Stückliste* bezeichnet und durch das Traversieren der Produktstruktur von oben nach unten erzeugt.
Entwickelt nun ein Hersteller ein Produkt in Kooperation mit einem anderen Hersteller, müssen die Produktdaten beider Hersteller in regelmäßigen Abständen abgeglichen werden. Typischerweise verwaltet jeder Hersteller ein eigenes PDMS für die Dokumentation. Gleiches gilt, wenn einzelne Aspekte der Entwicklung (z.B. Konstruktion, Software-Entwicklung) in unterschiedlichen Informationssystemen dokumentiert werden. Beispielsweise werden während der Entwicklung von E/E-Komponenten (z.B. Steuergeräte, Sensoren, Aktuatoren) geometrische Modelle und deren Informationen zu Hardware und Software meist in zwei separaten PDMS verwaltet. Diese Informationen werden von unterschiedlichen Personengruppen dokumentiert. Folglich kann es zu Inkonsistenzen in der Dokumentation von E/E-Komponenten in den beiden Systemen kommen. So können in einen PDMS geometrische Modelle für E/E-Komponenten vorhanden seien, entsprechende Informationen zu Hard- und Software im anderen PDMS fehlen und umgekehrt.

10.1.2. Anwendungsfälle

Folgende Anwendungsfälle sollen mit Hilfe der integrierten Produktdaten der beiden Informationssysteme realisiert werden:

- *Anwendungsfall 1*: Ausgehend von zwei PDMS sollen die Mengen der dokumentierten E/E-Komponenten für ein Fahrzeug abgeglichen werden. Dabei sollen diejenigen Komponenten ermittelt werden, die nur in einem der beiden Systeme dokumentiert sind.
- *Anwendungsfall 2*: Es sollen diejenigen E/E-Komponenten ermittelt werden, für die keine Geometriemodelle oder Software-Artefakte dokumentiert sind.

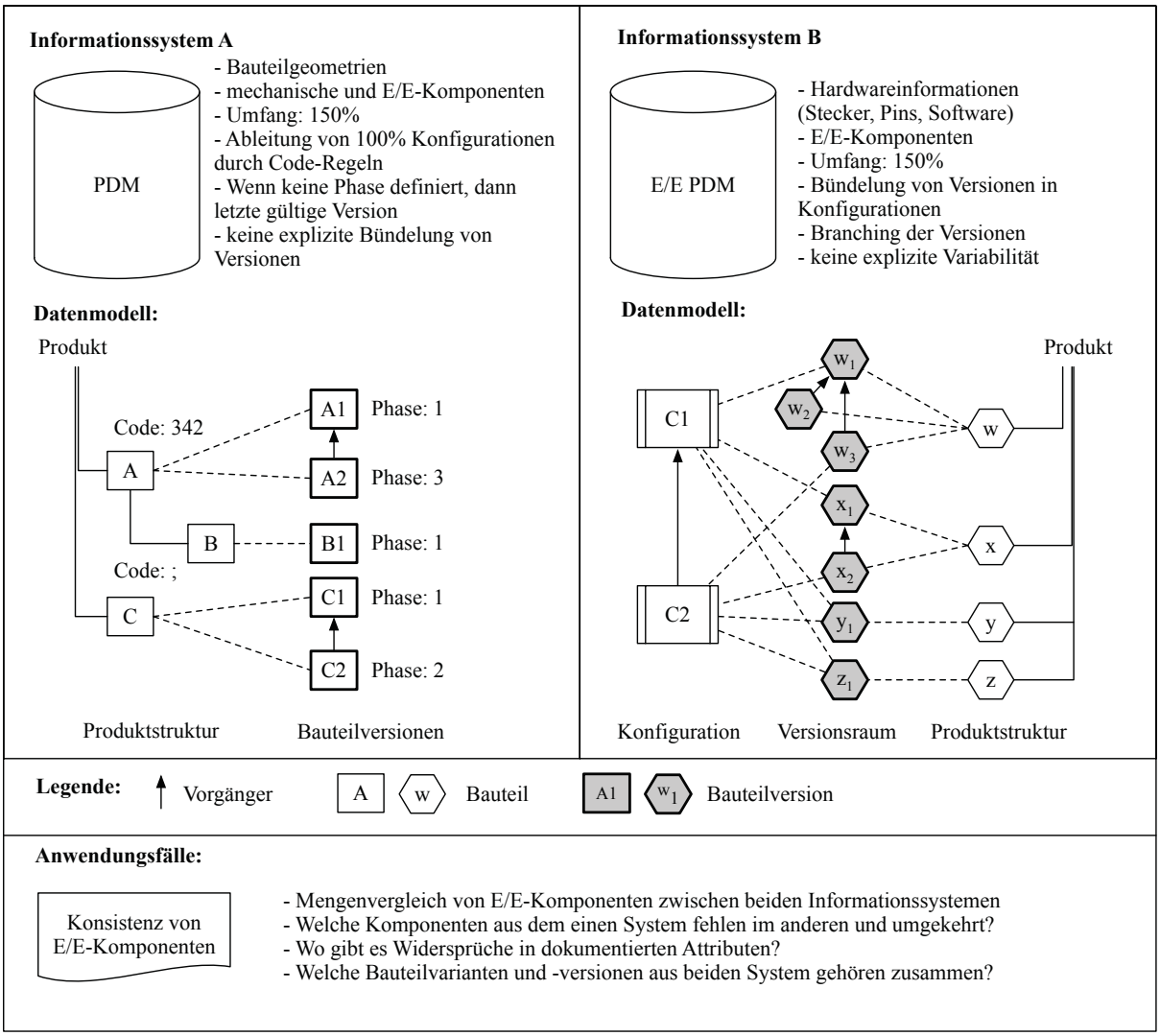


Abbildung 10.1.: Integrationsszenario - Konsistenzcheck zweier PDMS über Stücklisten

- *Anwendungsfall 3*: Es sollen diejenigen E/E-Komponenten ermittelt werden, die sich bzgl. der Anzahl ihrer Varianten unterscheiden. Obwohl beide Systeme unterschiedliche Mechanismen zur Dokumentation von Bauteilvarianten und -versionen verwenden können, kann die Auflistung von zusammengehörigen Varianten und Versionen aus beiden Systemen Hinweise auf mögliche Inkonsistenzen geben.

10.1.3. Systeme

Abbildung 10.1 illustriert die konzeptionellen Datenmodelle der beiden zu integrierenden Informationssysteme. Während *Informationssystem A* die Geometriedaten von sowohl mechanische als auch E/E-Bauteilen speichert, werden im *Informationssystem B* detaillierte Informationen zur Hard- und Software von E/E-Komponenten dokumentiert. In beiden Systemen werden die E/E-Komponenten mittels unterschiedlicher Strukturen und Namenskonventionen dokumen-

tiert.

- Im *Informationssystem A* werden Bauteile hierarchisch gespeichert. Dabei kommt es vor, dass Bauteile wiederum hierarchisch durch weitere Bauteile spezifiziert sind.
- Das *Informationssystem B* hingegen speichert die E/E-Komponenten in einer Liste.

Beide Systeme verwenden unterschiedliche Konventionen zur Dokumentation von Variabilität und Versionierung:

- Im *Informationssystem A* sind Bauteile mit Code-Regeln versehen, mit deren Hilfe sich Produkte konfigurieren lassen (Variabilität). Das heißt, aus einer Produktstruktur mit allen Bauteilen lässt sich ein konkretes Produkt ableiten. Alle Bauteile lassen sich versionieren, wodurch Versionsbäume entstehen können. Neben den Versionen lassen sich Bauteile zusätzlich zu projektspezifischen Meilensteinen zuordnen.
- *Informationssystem B* hingegen bietet keine explizite Unterstützung zur Dokumentation von Variabilität. Versionen von E/E-Komponenten werden zu Konfigurationen gebündelt, ohne dabei auf ein konkretes Produkt Bezug zu nehmen. Auch im *Informationssystem B* können Versionsbäume einzelner E/E-Komponenten entstehen.

Einzige Gemeinsamkeit beider Systeme ist eine Zuordnung der Artefakte zu globalen Bauteilnummern. In der Praxis unterscheidet sich die Qualität der Dokumentation dieser Bauteilnummern in beiden Systemen. Während in *Informationssystem A* die Zuordnung von Bauteilen zu Bauteilnummern sehr gut dokumentiert ist, fehlen diese Informationen im *Informationssystem B* für frühe Phasen der Produktentwicklung. Häufig werden bei der Entwicklung neuer E/E-Komponenten bestehende Informationen und Attribute vorheriger Komponentenversionen kopiert und erst im weiteren Verlauf der Entwicklung nachdokumentiert bzw. angepasst. So kann es vorkommen, dass für E/E-Komponenten im *Informationssystem B* Bauteilnummern dokumentiert sind, die nicht gültig sind bzw. noch nicht angepasst wurden.
Beide Systeme haben unterschiedliche Mengengerüste: Während *Informationssystem A* sowohl mechanische Bauteile als auch E/E-Bauteile verwaltet, fallen pro Produkt (in diesem Fall Fahrzeuge) bis zu Zehntausend Bauteile an. Durch die Variabilität und Versionierung ergeben sich somit mehrere Zehntausende Artefakte, die integriert werden müssen. Auf der anderen Seite verwaltet *Informationssystem B* nur E/E-Komponenten. Pro Fahrzeug ergeben sich hier bis zu 150 Bauteile. Bezieht man alle Versionen mit ein, sind aus diesem System mehrere Tausend Artefakte pro Fahrzeug zu integrieren.

10.1.4. Integration mit Hilfe des PROCEED-Rahmenwerkes

Um die beiden Informationssysteme mit Hilfe des PROCEED-Rahmenwerkes zu integrieren und die geschilderten Anwendungsfälle zu realisieren, sind folgende Schritte notwendig:

1. Auswählen einer Integrationsmethodik
2. Definition einer globalen Produktontologie
3. Definition einer lokaler Masterproduktontologie

4. Definition einer lokalen Produktontologie inklusive Abbildung in die globale Produktontologie

5. Spezifikation von Integrationsqualitäts-Metriken

6. Formulierung von Abfragen für Anwendungsfälle

1. Integrationsmethodik: Da nur zwei Informationssysteme zu integrieren sind, für die bisher noch keine lokalen Produktontologien existieren, wird als Integrationsmethodik eine globale Produktontologie Top-Down definiert.

2. Globale Produktontologie: Die globale Produktontologie besteht aus jeweils einem Schemakonzept pro Produktdatenintegrationsebene und orientiert sich am lokalen Informationssystem `A`: Ein Produkt (`Product`) besteht aus mehreren Bauteilen (`Component`), welche unterschiedliche Varianten (`Component Variant`) besitzen können. Für jede Bauteilvariante werden verschiedene Entwicklungsstände dokumentiert (`Component Version`). Eine Bauteilvariante lässt sich durch eine eindeutige Identifikation (`Code`) beschreiben. Jeder Entwicklungsstand einer Bauteilvariante ist ebenfalls durch eine eindeutige Identifikation (`PartId`) sowie einer fortlaufenden Nummer (`Version Id`) gekennzeichnet. Ein Entwicklungsstand der globalen Produktontologie umfasst sowohl eine geometrische Zeichnung (`Geometry`) als auch einen Softwarestand (`Software`). Für `Component Variant` wird der Variabilitätstyp 2 festgelegt (vgl. Abschnitt 6.2.1), d.h. für jede Bauteilvariante wird eine eigene, explizite Instanz in der globalen Produktontologie dokumentiert. Die Entwicklungsstände der Bauteilvarianten werden kontinuierlich dokumentiert.

3. Lokale Masterproduktontologie: Da die Datenqualität der dokumentierten Komponenten, inklusive ihrer eindeutigen Identifikationsnummern, im lokalen Informationssystem `A` sehr hoch ist, wird Informationssystem `A` auch zur Definition einer Masterproduktontologie verwendet. In Abbildung 10.2 sind die entsprechende lokale Masterproduktontologie `M` sowie Abbildungen in die globale Produktontologie `G` illustriert.

4. Lokale Produktontologie: Entsprechend wird für das lokale Informationssystem `B` eine lokale Produktontologie definiert. Abbildung 10.3 illustriert die lokale Produktontologie `L` für das lokale `Informationssystem B`.
Ein Produkt (`Product`) setzt sich auf mehreren E/E-Komponenten (`EE Component`) zusammen. Für eine E/E-Komponente können mehrere Varianten (`EE Component Variant`) vorhanden sein, die wiederum in unterschiedlichen Versionen (`EE Component Version`) existieren können. Jedes dieser Schemakonzepte besitzt als Attribut einen Namen (`Name`), für Varianten existiert ein Attribut, das einen Variantenbezeichner (`Variant`) enthält; analog beschreibt das Attribut `Version Id` eine Versionsnummer. Schließlich ist die Referenz der Software der E/E-Komponente in der lokalen Produktontologie modelliert (`Reference`).

5. Integrationsqualitätsmetriken: Da nur eine lokale Produktontologie vorhanden ist, ist zur Realisierung von Anwendungsfall 1 nur die lokale und globale Instanzvollständigkeit notwendig (siehe Kapitel 7).

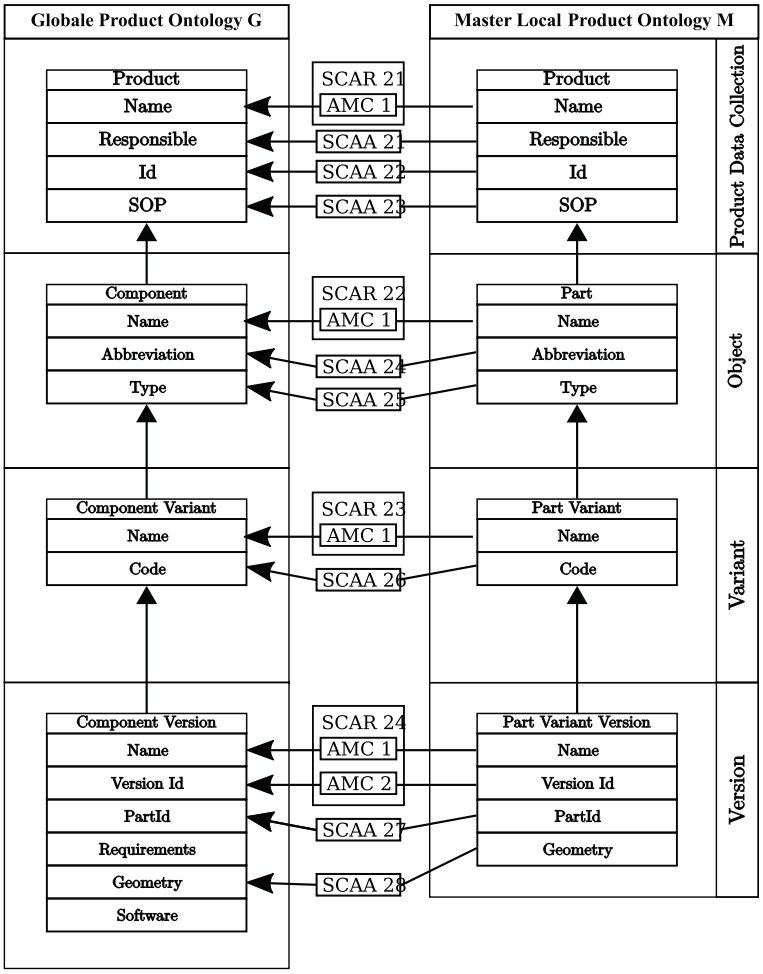


Abbildung 10.2.: Produktontologien des Integrationsszenarios

6. Abfragen Im folgenden Abschnitt werden die notwendigen Abfragen an die globale Produktontologie zur Realisierung der drei Anwendungsfälle beschreiben.
Listing 10.1 illustriert die SPARQL-Abfragen zur Realisierung des `Anwendungsfalls 1`.

```
1  PREFIX mo: <http://www.uni-ulm.de/proceed/metaontology#>
2  PREFIX gpo: <http://www.uni-ulm.de/proceed/gpo#>
3  PREFIX lpo1: <http://www.uni-ulm.de/proceed/lpo1#>
4  PREFIX pom1: <http://www.uni-ulm.de/proceed/pom1#>
5  SELECT DISTINCT ?component
6  WHERE {
7    ?component rdf:type gpo:Component_SC .
8    ?component gpo:ComponentType "ECU"^^xsd:string .
9    FILTER NOT EXISTS {
10     ?component pom:CorrespondesToSCAR ?lpo_Ind .
11     ?lpo_Ind rdf:type lpo1:LocalSchemaConcept .
12   }
13 }
```

Listing 10.1: SPARQL-Query für den `Anwendungsfall 1`

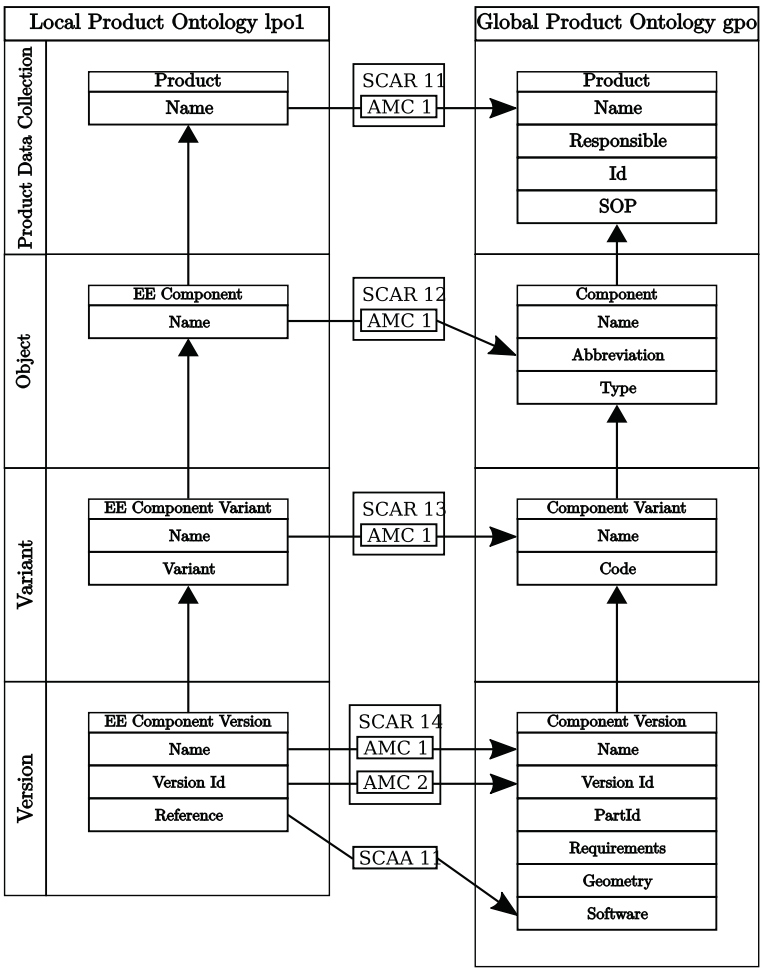


Abbildung 10.3.: Lokale Produktontologie LPO

In Listing 10.2 ist die SPARQL-Abfrage für den `Anwendungsfall 2` spezifiziert.

```
1  PREFIX gpo: <http://www.uni-ulm.de/proceed/gpo#>
2  SELECT DISTINCT ?componentVersion
3  WHERE {
4    ?componentVersion rdf:type gpo:ComponentVersion_SC .
5    FILTER (not exists {?componentVersion gpo:Geometry_Attr ?value } ||
6            not exists {?componentVersion gpo:Software_Attr ?value })
7    }
8  }
```

Listing 10.2: SPARQL-Query für den `Anwendungsfall 2`

Listing 10.3 beschreibt die SPARQL-Abfrage für den `Anwendungsfall 3`. Es werden zunächst diejenigen lokalen und globalen Instanzen ermittelt, die zueinander korrespondieren (Zeile 6). Für die lokale Instanz werden die Anzahl der zugehörigen Instanzen auf der Varianten-Ebene ermittelt (Zeilen 7 bis 13). Analog werden die globalen Instanzen bestimmt (Zeilen 14 bis 20). Es werden alle Paare von lokalen und globalen Instanzen herausgefiltert, für die die Anzahl der Varianten gleich ist (siehe Zeile 21).

```
1  PREFIX gpo: <http://www.uni-ulm.de/proceed/gpo#>
2  SELECT DISTINCT ?localInd ?globalInd
3  WHERE {
4    ?localInd rdf:type mo:LocalObjectLayer .
5    ?globalInd rdf:type mo:GlobalObjectLayer .
6    ?localInd pom1:correspondesTo ?globalInd .
7    {
8      SELECT (COUNT (?var) as ?localVarCnt)
9      WHERE {
10       ?localInd mo:isAbove ?var
11     }
12     GROUP BY ?var
13   }
14   {
15     SELECT (COUNT (?var) as ?globalVarCnt)
16     WHERE {
17       ?globalInd mo:isAbove ?var
18     }
19     GROUP BY ?var
20   }
21   FILTER (?localVarCnt != ?globalVarCnt)
22 }
```

Listing 10.3: SPARQL-Query für den `Anwendungsfall 3`

10.1.5. Erkenntnisse

Die Fallstudie zeigt die Integration eines häufigen Szenarios in der Produktentwicklung. Da beide Informationssysteme sehr heterogene Datenmodelle besitzen, ist zunächst ein initialer Aufwand für die Modellierung der Produktontologien notwendig. Die Auswahl von Variabilitäts- und Versionierungsmechanismus muss sorgsam gewählt werden, da eine spätere Änderung die Anpassung von bereits integrierten Instanzen in der globalen Produktontologie nach sich ziehen. Vor allem die Nutzung der Anwendungsfälle über einen längeren Zeitraum der Produktentwicklung, bei dem sich Produktdaten kontinuierlich ändern, bietet das PROCEED-Rahmenwerk den Vorteil, dass diese Änderungen sehr einfach über die diskutierten Änderungsoperationen (vgl. Kapitel 8) in der globalen Produktontologie realisiert werden können.
Außerdem zeigt sich an diesem Beispiel, dass bereits mit der Integration von zwei Informationssystemen eine Vielzahl komplexer Anwendungsfälle realisiert werden kann. In der Praxis ist es sinnvoll mit der Integration weniger Informationssysteme zu beginnen. Der initiale Modellierungsaufwand amortisiert sich mit zunehmender Anzahl zu integrierender Informationssysteme.

Teil IV

Zusammenfassung

11. Zusammenfassung und Ausblick

Die Integration von Produktdaten ist essentiell, um die Prozesse rund um die Produktentwicklung bestmöglich zu unterstützen. Die Beseitigung von Fehlern in späteren Entwicklungsphasen ist mit erheblichen Kosten verbunden. Deshalb ist besonders in frühen Phasen der Produktentwicklung der regelmäßige Abgleich der Entwicklungsstände, d.h. die Überprüfung der Vollständigkeit und Konsistenz von Produktdaten, notwendig. Hierzu bedarf es einer Integration autonomer, heterogener Informationssysteme.
Concurrent Engineering und die verschiedenen Anforderungen und Aufgaben der Produktentwicklung führen zur Verwendung unterschiedlicher Methoden um Produktvariabilität und -versionierung zu dokumentieren. Existierende Konzepte und Rahmenwerke fokussieren entweder auf Schema- oder Instanzintegration, und bieten keine ausreichende Unterstützung, um verschiedenen Konzepte für Produktvariabilität und -versionierung angemessen bei der Produktdatenintegration zu berücksichtigen.

11.1. Zusammenfassung

Die vorliegende Arbeit liefert mit PROCEED ein Rahmenwerk zur Integration komplexer Produktdaten, das den gesamten Integrationslebenszyklus unterstützt. Von der initialen Integration, über die Steuerung und Überwachung des Integrationsprozesses hin zur Evolution integrierter Produktdaten:

- **Umgang mit Heterogenität durch Abstraktion:** Durch die Einführung von Produktontologien kann der Umgang mit technischer und semantischer Heterogenität der zu integrierenden Produktdaten erheblich vereinfacht werden. Gleichzeitig wird durch die einheitliche Strukturierung innerhalb der Produktontologien in vier Integrationsebenen der Aufwand einer automatischen Auswertung von Integrationsregeln verringert. Die Dokumentation von implizitem Wissen in den Produktontologien kann zudem den manuellen Integrationsaufwand während der Schema- und Instanzintegration reduzieren.
- **Definition von Abbildungsregeln und -aktionen:** Neben den Abbildungsregeln zwischen Produktontologien, die eine automatische Bestimmung von Korrespondenzen zwischen Instanzen erlaubt, können mit Hilfe von Integrationsaktionen eine integrierte Sicht von Produktdaten geschaffen werden.
- **Automatische Bestimmung von Korrespondenzen:** Durch die Definition von Abbildungsregeln zwischen lokaler und globaler Produktontologie lässt sich die Ermittlung korrespondierender Instanzen automatisieren.
- **Steuerung und Überwachung des Integrationsprozesses:** Die Definition von Qualitätsmetriken für die Integration lokaler Produktontologien in eine globale Produktontologie erlauben es, den Integrationsprozess zu steuern und überwachen.

- **Definition von Änderungsoperationen:** Durch die Bereitstellung von Änderungsoperationen, sowohl für Schemakonzepte als auch Instanzen, ist es möglich, eine bereits erstellte globale Produktontologie stets mit dem Entwicklungstand der lokalen Produktontologien zu synchronisieren. Dadurch kann die Evolution von integrierten Produktdaten bewerkstelligt werden.
- **Domänenunabhängiger Ansatz:** Produktdatenintegration und der Umgang mit der Variabilität und Versionierung von Produktdaten sind keine Konzepte, die nur in der Automobilindustrie anzutreffen sind. Das entwickelte Konzept der Produktontologien orientiert sich an den Problemstellungen, die auch von gängigen Austauschstandards (z.B. STEP) adressiert werden. Genau wie STEP ist PROCEED ein Ansatz, der auf Produktdaten im Allgemeinen fokussiert und folglich in anderen Domänen, etwa der Luftfahrtindustrie oder dem Schiffbau Anwendung finden kann.
- **Konzepte zur Benutzerunterstützung:** Neben der technischen Lösung wurden auch Konzepte durch Unterstützung von Endanwendern betrachtet.

11.2. Ausblick

Über die diskutierten Aspekte der vorliegenden Arbeit hinaus gibt es weitere Themenbereiche, die Gegenstand weiterer Untersuchungen seien können:

- Die Integration im PROCEED-Rahmenwerk fokussiert auf einzelne Schemakonzepte, eine Integration mehrerer Schemakonzepte sowie Beziehungen zwischen diesen werden nicht berücksichtigt. Neben einzelnen Aspekten von Produktdaten ist die Integration von Beziehungen zwischen Produktdaten sinnvoller Gegenstand weiterer Untersuchungen. Die Berücksichtigung dieser Beziehungen bietet zudem vielversprechende Perspektiven hinsichtlich der Koordination der zugehörigen Teilentwicklungensprozesse (für entsprechende Arbeiten siehe [RR06, MRH07, KR09, SKAR17, SAR18]).
- Ziel der vorliegenden Arbeit ist die Unterstützung von lesenden Anwendungsfällen autonomer, heterogener Produktdaten. Werden Inkonsistenzen, unvollständige Attribute oder fehlende Instanzen in der globalen Produktontologie identifiziert, kann eine Behebung nur in den entsprechenden lokalen Informationssystemen erfolgen. Die Möglichkeit, Änderungen in der globalen Produktontologie vorzunehmen, die anschließend in die einzelnen lokalen Informationssysteme propagiert werden, kann den Prozess von Änderungen beschleunigen.
- Die Arbeit hat verschiedene Teilprozesse der Produktdatenintegration und -evolution explizit modelliert. Hier bietet eine Automation dieser Prozesse mittels Prozess-Management-Technologien [MRB08] vielversprechende Perspektiven, etwa in Bezug auf Koordination [MWR08], Flexibilität [Rei00, RRMD09, BBTR14, BBTR15], Variabilität [LRW09, ATW+15], Effektivität [LR16, LR12], Mobilität [PTKR10, PTR10] und Compliance [KRK17, KR17].
- Schließlich sollten weitere Fallstudien vorgenommen werden, um Erfahrungswerte für die Auswahl sowohl der zu verwendenden Integrationsmethodik als auch die Festlegung von Soll-Werten für die Steuerung des Integrationsprozesses zu ermitteln.

Literaturverzeichnis

[Ack89] R.L. Ackoff. From Data to Wisdom. *Journal of Applied Systems Analysis*, 16(1):3–9, 1989.

[AKRS06] C. Amelunxen, A. Königs, T. Rötschke, and A. Schürr. MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In *European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA 2006)*, pages 361–375. Springer, 2006.

[All03] OSGi Alliance. *OSGi Service Platform, Release 3*. IOS Press, Inc., 2003.

[AT00] R. Anderl and D. Trippner. *STEP – STandard for the Exchange of Product Model Data: Eine Einführung in die Entwicklung, Implementierung und industrielle Nutzung der Normenreihe ISO 10303 (STEP)*. B.G. Teubner Stuttgart, 2000.

[ATW+15] C. Ayora, V. Torres, B. Weber, M. Reichert, and V. Pelechano. VIVACE: A Framework for the Systematic Evaluation of Variability Support in Process-Aware Information Systems. *Information and Software Technology*, 57:248–276, January 2015.

[BBK+13] C. Ballard, M. Bhide, H. Kache, B. Kitzberger, B. Porst, Y.H. Sheng, H.C. Smith, and IBM Redbooks. *IBM Information Server: Integration and Governance for Emerging Data Warehouse Demands*. IBM Redbooks. IBM Redbooks, 2013.

[BBLP08] D. Beckett, T. Berners-Lee, and E. Prud'hommeaux. Turtle-terse RDF triple language. *W3C Team Submission*, 14(7), 2008.

[BBR11] S. Buchwald, T. Bauer, and M. Reichert. Bridging the Gap Between Business Process Models and Service Composition Specifications. In *Service Life Cycle Tools and Technologies: Methods, Trends and Advances*, pages 124–153. Idea Group Referenc, November 2011.

[BBTR14] T. Bauer, S. Buchwald, J. Tiedeken, and M. Reichert. Konzeption eines SOA-Repository mit Analysefähigkeiten. In *Proceedings Informatik 2014*, number P-232 in Lecture Notes in Informatics (LNI), pages 1565–1576. Koellen-Verlag, September 2014.

[BBTR15] T. Bauer, S. Buchwald, J. Tiedeken, and M. Reichert. A SOA Repository with Advanced Analysis Capabilities - Improving the Maintenance and Flexibility of Service-Oriented Applications. In *17th International Conference on Enterprise Information Systems (ICEIS 2015)*, pages 238–248, April 2015.

[Bec09] S. Bechhofer. OWL: Web Ontology Language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

[Bel02] Z. Bellahsene. Schema Evolution in Data Warehouses. *Knowledge and Information Systems*, 4(3):283–304, 2002.

[Ben09] P.R. Benson. ISO 8000 Data Quality: The fundamentals, part 1. *Real-World Decision Support (RWDS) Journal*, 3(4), 2009.

[Beu02] T. Beuter. *Workflow-Management für Produktentwicklungsprozesse*. PhD thesis, Universität Ulm, 2002.

[BG00] D. Brickley and R. V. Guha. Resource Description Framework (RDF) Schema Specification 1.0: W3C Candidate Recommendation. Technical report, W3C, Cambridge, MA, 2000.

[BGN+04] S. Burmester, H. Giese, J. Niere, M. Tichy, J.P. Wadsack, R. Wagner, L. Wendehals, and A. Zündorf. Tool integration at the meta-model level: the Fujaba approach. *International Journal on Software Tools for Technology Transfer*, 6(3):203–218, 2004.

[BHR05] U. Bestfleisch, J. Herbst, and M. Reichert. Requirements for the Workflow-based Support of Release Management Processes in the Automotive Sector. In *12th European Concurrent Engineering Conference (ECEC'05)*, pages 130–134, April 2005.

[BLL10] J. Blechinger, F. Lauterwald, and R. Lenz. Supporting the Production of High-Quality Data in Concurrent Plant Engineering Using a MetaDataRepository. In *AMCIS*, page 95, 2010.

[BN09] J. Bleiholder and F. Naumann. Data Fusion. *ACM Computing Surveys*, 41(1):1, 2009.

[Bob08] R. Bobrik. *Konfigurierbare Visualisierung komplexer Prozessmodelle*. PhD thesis, Universität Ulm, 2008.

[BR05] M. Broy and A. Rausch. Das neue V-Modell XT - Ein anpassbares Vorgehensmodell für Software und System Engineering. *Informatik-Spektrum*, 28(3):220–229, 2005.

[Bra07] S. Bratt. Semantic Web and Other Technologies to Watch. *Talks at W3C, January*, 2007.

[BRB05] R. Bobrik, M. Reichert, and T. Bauer. Requirements for the Visualization of System-Spanning Business Processes. In *1st Int'l Workshop on Business Process Monitoring and Performance Management (BPMPM'05) in conjunction with (DEXA'05)*, pages 948–954. IEEE Computer Society Press, August 2005.

[BRB06] R. Bobrik, M. Reichert, and T. Bauer. Proviado - Personalized and Configurable Visualizations of Business Processes. In *7th Int'l Conf on Electronic Commerce and Web Technologies (EC-WEB'06)*, number 4082 in LNCS, pages 61–71. Springer, September 2006.

[BS06] C. Batini and M. Scannapieco. Data Quality - Concepts, Methodologies and Techniques, 2006.

[CAD03] I. Crnkovic, U. Asklund, and A.P. Dahlqvist. *Implementing and Integrating Product Data Management and Software Configuration Management*. Artech House, 2003.

[CBK13] R. Capilla, J. Bosch, and K.-C. Kang. *Systems and Software Variability Management - Concepts, Tools and Experiences*. Springer, 2013.

[CD97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[CGR+12] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12)*, pages 173–182. ACM, 2012.

[CKR14] C.M. Chiao, V. Künzle, and M. Reichert. Towards Schema Evolution in Object-aware Process Management Systems. In *International Workshop on the Evolution of Information Systems and their Design Methods (EMISA 2014)*, number P-234 in Lecture Notes in Informatics (LNI), pages 101–115. Koellen-Verlag, September 2014.

[Col97] R.M. Colomb. Impact of Semantic Heterogeneity on Federating Databases. *The Computer Journal*, 40(5):235–244, 1997.

[Con97] S. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag, 1997.

[Con12] Roland Berger Strategy Consultants. `http://www.rolandberger.de/media/pdf/Roland_Berger_Master_product_complexity_20121108.pdf`, November 2012.

[Coo90] R.G. Cooper. Stage-Gate Systems: A New Tool for Managing New Products. *Business Horizons*, 33(3):44–54, 1990.

[CRS+13] S.K. Chandrasegaran, K. Ramani, R.D. Sriram, I. Horváth, A. Bernard, R.F. Harik, and W. Gao. The evolution, challenges, and future of knowledge representation in product design systems. *Computer-Aided Design*, 45(2):204–228, 2013.

[CW98] R. Conradi and B. Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, 30(2):232–282, 1998.

[DHI12] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Elsevier, 2012.

[DSDH08] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping Pay-As-You-Go Data Integration Systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08)*, pages 861–874. ACM, 2008.

[EIA98] EIA-649 - National Consensus Standard for Configuration Management. Norm, ANSI, 1430 Broadway, New York, NY 10018, USA, 1998.

[EN09] M. Eigner and M.F. Nem. Common Versioning of Product Data and Engineering Processes. In *Proceedings of the 2nd International Conference on Research into Design (ICORD 09)*, 2009.

[Faw06] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[GMF+03] J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson, N.F. Noy, and S.W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[GOP02] M.P. Gallaher, A.C. O'Connor, and T. Phelps. Economic Impact Assessment of the International Standard for the Exchange of Product Model Data (step) in Transportation Equipment Industries. *NIST planning report*, pages 02–5, 2002.

[GOR12] G. Grambow, R. Oberhauser, and M. Reichert. Towards Automated Process Assessment in Software Engineering. In *7th Int'l Conf on Software Engineering Advances (ICSEA'12)*, pages 289–295, November 2012.

[Gra16] G. Grambow. *Context-aware Process Management for the Software Engineering Domain*. PhD thesis, Ulm University, July 2016.

[Gru93] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Hal01] A.Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[Hal10] A. Hallerbach. *Management von Prozessvarianten*. PhD thesis, Universität Ulm, 2010.

[Hau15] S. Hausladen. Visualisierung und Navigation komplexer Produktdaten. Bachelor thesis, Universität Ulm, 2015.

[HB11] M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[HBR08] A. Hallerbach, T Bauer, and M. Reichert. Context-based Configuration of Process Variants. In *3rd International Workshop on Technologies for Context-Aware Business Process Management (TCoB 2008)*, pages 31–40, June 2008.

[HBR09] A. Hallerbach, T. Bauer, and M. Reichert. Guaranteeing Soundness of Configurable Process Variants in Provop. In *11th IEEE Conference on Commerce and Enterprise Computing (CEC'09)*, pages 98–105. IEEE Computer Society Press, July 2009.

[HBR10] A. Hallerbach, T. Bauer, and M. Reichert. Capturing Variability in Business Process Models: The Provop Approach. *Journal of Software: Evolution and Process*, 22(6-7):519–546, 2010.

[Hei95] S. Heiler. Semantic Interoperability. *ACM Computing Surveys*, 27(2):271–273, 1995.

[HMMR14] M. Hipp, B. Michelberger, B. Mutschler, and M. Reichert. Navigating in Process Model Repositories and Enterprise Process Information. In *IEEE 8th International Conference on Research Challenges in Information Science (RCIS 2014)*, pages 1–12. IEEE Computer Society Press, May 2014.

[HMN15] H.R. Hansen, J. Mendling, and G. Neumann. *Wirtschaftsinformatik*. Walter de Gruyter GmbH & Co KG, 2015.

[HMPR04] A.R. Hevner, S.T. March, J. Park, and S. Ram. Design Science Research in Information Systems. *Management Information Systems Quarterly*, 28:75–105, 2004.

[HMR12] M. Hipp, B. Mutschler, and M. Reichert. Navigating in Complex Business Processes. In *23rd Int'l Conf on Database and Expert Systems Applications (DEXA'12), Part II*, number 7447 in LNCS, pages 466–480. Springer, 2012.

[Hoy09] D. Hoyle. *ISO 9000: Quality Systems Handbook*. Elsevier, 6. edition, 2009.

[HPS+18] B. Hoppenstedt, R. Pryss, B. Stelzer, F. Meyer-Brötz, K. Kammerer, A. Treß, and M. Reichert. Techniques and Emerging Trends for State of the Art Equipment Maintenance Systems - A Bibliometric Analysis. *Applied Sciences*, 8:1–29, June 2018.

[HRZ+08] K. Hose, A. Roth, A. Zeitz, K.-U. Sattler, and F. Naumann. A Research Agenda for Query Processing in Large-Scale Peer Data Management Systems. *Information Systems*, 33(7):597–610, 2008.

[HS98] M.A. Hernández and S.J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[HSP13] S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 query language. *W3C Recommendation*, 21(10), 2013.

[Inm05] W.H. Inmon. *Building the Data Warehouse*. John Wiley & sons, 3rd edition, 2005.

[ISO03a] ISO 10007: Quality management systems – Guidelines for configuration management. Standard, International Organization for Standardization, Geneva, CH, 2003.

[ISO03b] ISO 18876-1: Integration of industrial data for exchange, access and sharing – Part 1: Architecture overview and description. Standard, International Organization for Standardization, Geneva, CH, 2003.

[ISO03c] ISO 18876-2: Integration of industrial data for exchange, access and sharing – Part 2: Integration and mapping methodology. Standard, International Organization for Standardization, Geneva, CH, 2003.

[ISO08] ISO 9001: Quality Management Systems – Requirements. Standard, International Organization for Standardization, Geneva, CH, 2008.

[JDB+98] C.S. Jensen, C.E. Dyreson, M. Böhlen, J. Clifford, R. Elmasri, S.K. Gadia, F. Grandi, P. Hayes, S. Jajodia, W. Käfer, et al. The Consensus Glossary of Temporal Database Concepts — February 1998 Version. In *Temporal Databases: Research and Practice*, pages 367–405. Springer, 1998.

[JG99] J. M. Juran and A. B. Godfrey. *Juran's Quality Handbook*. McGraw-Hill, 1999.

[Kat15] A Katzenbach. Automotive. In *Concurrent Engineering in the 21st Century*, pages 607–638. Springer, 2015.

[KCH+90] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellong University - Software Engineering Institute, 1990.

[KCH+03] W. Kim, B. Choi, E. Hong, S. Kim, and D. Lee. A Taxonomy of Dirty Data. *Data Mining and Knowledge Discovery*, 7(1):81–99, 2003.

[KPSR18] K. Kammerer, R. Pryss, K. Sommer, and M. Reichert. Towards Context-aware Process Guidance in Cyber-Physical Systems with Augmented Reality. In *4th International Workshop on Requirements Engineering for Self-Adaptive, Collaborative, and Cyber Physical Systems (RESACS'18)*. IEEE Computer Society Press, 2018.

[KR09] V. Künzle and M. Reichert. Towards Object-aware Process Management Systems: Issues, Challenges, Benefits. In *10th Int'l Workshop on Business Process Modeling, Development, and Support (BPMDS'09)*, number 29 in LNBIP, pages 197–210. Springer, June 2009.

[KR11] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, 2011.

[KR17] D. Knuplesch and M. Reichert. A Visual Language for Modeling Business Process Compliance Rules. *Software & Systems Modeling*, 16:715–736, 2017.

[KRK17] D. Knuplesch, M. Reichert, and A. Kumar. A framework for visually monitoring business process compliance. *Information Systems*, 64:381–409, March 2017.

[KRS08] F. Klar, S. Rose, and A. Schürr. A Meta-model-Driven Tool Integration Development Process. In *International United Information Systems Conference*, pages 201–212. Springer, 2008.

[KS91] W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, 1991.

[KS06] A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars - A Survey. *Electronic Notes in Theoretical Computer Science*, 148(1):113–150, 2006.

[LBK07] R. Lenz, M. Beyer, and K.A. Kuhn. Semantic Integration in Healthcare Networks. volume 76, pages 201–207. Elsevier, 2007.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[Len03] R. Lenz. Integration und Evolution ablaufunterstützender Informationssysteme im Krankenhaus. *GI-Arbeitskreis EA - Frühjahrskonferenz*, pages 34–41, 2003.

[LN06] U. Leser and F. Naumann. *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt, 2006.

[Los09] D. Loshin. *Master Data Management*. Morgan Kaufmann, 2009.

[LR07] R. Lenz and M. Reichert. IT Support for Healthcare Processes - Premises, Challenges, Perspectives. *Data and Knowledge Engineering*, 61(1):39–58, May 2007.

[LR12] M. Lohrmann and M. Reichert. Efficacy-aware Business Process Modeling. In *20th International Conference on Cooperative Information Systems (CoopIS 2012)*, number 7565 in LNCS, pages 38–55. Springer, September 2012.

[LR16] M. Lohrmann and M. Reichert. Effective application of process improvement patterns to business processes. *Software & Systems Modeling*, 15(2):353–375, May 2016.

[LRW09] C. Li, M. Reichert, and A. Wombacher. Discovering Reference Models by Mining Process Variants Using a Heuristic Approach. In *7th Int'l Conference on Business Process Management (BPM'09)*, number 5701 in LNCS, pages 344–362. Springer, September 2009.

[LSo98] O. Lassila, R.R. Swick, and other. Resource Description Framework (RDF) Model and Syntax Specification, 1998.

[Mer14] J. Merkel. Entwurf eines Usability-Konzeptes für das Produktdatenintegrationssystem PROCEED. Master thesis, Universität Ulm, 2014.

[MHHR06] D. Müller, J. Herbst, M. Hammori, and M. Reichert. IT Support for Release Management Processes in the Automotive Industry. In *International Conference on Business Process Management (BPM'06)*, pages 368–377. Springer, 2006.

[MHR06] D. Müller, J. Herbst, and M. Reichert. Flexibility of Data-Driven Process Structures. In *BPM'06 Int'l Workshops, Workshop on Dynamic Process Management (DPM'06)*, number 4103 in LNCS, pages 181–192. Springer, September 2006.

[MHR07] D. Müller, J. Herbst, and M. Reichert. Data-driven Modeling and Coordination of Large Process Structures. In *15th Int'l Conf on Cooperative Information Systems (CoopIS 2007)*, number 4803 in LNCS, pages 131–149. Springer, November 2007.

[MHR08] D. Müller, J. Herbst, and M. Reichert. A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In *20th Int'l Conf on Advanced Information Systems Engineering (CAiSE'08)*, number 5074 in LNCS, pages 48–63. Springer, June 2008.

[MMR12] B. Michelberger, B. Mutschler, and M. Reichert. Process-oriented Information Logistics: Aligning Enterprise Information with Business Processes. In *16th IEEE International EDOC Conference (EDOC 2012)*, pages 21–30. IEEE Computer Society Press, September 2012.

[MPSP+09] B. Motik, P.F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, et al. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. *W3C Recommendation*, 27(65):159, 2009.

[MRB08] B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the Effectiveness of Process-oriented Information Systems: Problem Analysis, Critical Success Factors and Implications. *IEEE Transactions on Systems, Man, and Cybernetics (Part C)*, 38(3):280–291, May 2008.

[MRH07] D. Müller, M. Reichert, and J. Herbst. Data-driven Modeling and Coordination of Large Process Structures. In *15th Int'l Conf on Cooperative Information Systems (CoopIS 2007)*, number 4803 in LNCS, pages 131–149. Springer, November 2007.

[MSR13] C. Manz, M. Stupperich, and M. Reichert. Towards Integrated Variant Management in Global Software Engineering: An Experience Report. In *8th IEEE International Conference on Global Software Engineering (ICGSE'13)*, pages 168–172. IEEE Computer Society Press, August 2013.

[Mül09] D. Müller. *Management datengetriebener Prozessstrukturen*. PhD thesis, Universität Ulm, 2009.

[MVH+04] D.L. McGuinness, F. Van Harmelen, et al. OWL Web Ontology Language Overview. *W3C Recommendation*, 10(10), 2004.

[MWR08] B. Mutschler, B. Weber, and M. Reichert. Workflow Management versus Case Handling: Results from a Controlled Software Experiment. In *23rd Annual ACM Symposium on Applied Computing (SAC'08), Special Track on Coordination Models, Languages and Architectures*, pages 82–89. ACM Press, March 2008.

[Nob07] S.H. Nobari. *Methode zur wirkungsorientierten Beschreibung variantenreicher Erzeugnisse (WIBER): Handhabung der Komplexität in der technischen Erzeugnisdokumentation*. Shaker, 2007.

[ORH05] P. Oliveira, F. Rodrigues, and P.R. Henriques. A Formal Definition of Data Quality Problems. In *Proceedings of the 2005 International Conference on Information Quality (MIT IQ Conference)*, 2005.

[ORMC23] C.K. Ogden, I. A. Richards, B. Malinowski, and F. G. Crookshank. *The Meaning of Meaning*. Kegan Paul London, 1923.

[PMR93] B. Prasad, R. S Morenc, and R.M. Rangan. Information Management for Concurrent Engineering: Research Issues. *Concurrent Engineering*, 1(1):3–20, 1993.

[PNSD07] T. Prefi, I. Neumärker, R. Schmitt, and K. Daniel. Qualitätsmanagement in der Produktentwicklung. *Masing Handbuch Qualitätsmanagement*, 5:405–433, 2007.

[PS14] T. Pfeifer and R. Schmitt. *Masing Handbuch Qualitätsmanagement*. Carl Hanser Verlag GmbH Co KG, 2014.

[PTKR10] R. Pryss, J. Tiedeken, U. Kreher, and M. Reichert. Towards Flexible Process Support on Mobile Devices. In *Proc CAiSE'10 Forum - Information Systems Evolution*, number 72 in LNBIP, pages 150–165. Springer, 2010.

[PTR10] R. Pryss, J. Tiedeken, and M. Reichert. Managing Processes on Mobile Devices: The MARPLE Approach. In *CAiSE'10 Demos*, June 2010.

[PVSV07] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. What-If Analysis for Data Warehouse Evolution. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2007)*, pages 23–33. Springer, 2007.

[RBRB06] S. Rinderle, R. Bobrik, M. Reichert, and T. Bauer. Businesss Process Visualization - Use Cases, Challenges, Solutions. In *8th Int'l Conf on Enterprise Information Systems (ICEIS'06) Track on Information Systems Analysis and Specification*, pages 204–211, May 2006.

[Red97] T.C. Redman. *Data Quality for the Information Age*. Artech House, Inc., Norwood, MA, USA, 1997.

[Rei00] M. Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. PhD thesis, Universität Ulm, 2000.

[Roc75] M.J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, (4):364–370, 1975.

[Rod95] J.F. Roddick. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, 37(7):383–393, 1995.

[Row07] J.E. Rowley. The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 2007.

[Roz97] G. Rozenberg. *Handbook of Graph Grammars and Computing*, volume 1. World Scientific, 1997.

[RR06] S. Rinderle and M. Reichert. Data-Driven Process Control and Exception Handling in Process Management Systems. In *18th Int'l Conf on Advanced Information Systems Engineering (CAiSE'06)*, number 4001 in LNCS, pages 273–287. Springer, June 2006.

[RRD03] M. Reichert, S Rinderle, and P. Dadam. On the Common Support of Workflow Type and Instance Changes under Correctness Constraints. In *11th Int'l Conf Cooperative Information Systems (CooplS 2003)*, number 2888 in LNCS, pages 407–425. Springer, November 2003.

[RRD04a] S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria for Dynamic Changes in Workflow Systems: A Survey. *Data & Knowledge Engineering*, 50(1):9–34, 2004.

[RRD04b] S. Rinderle, M. Reichert, and P. Dadam. Flexible Support of Team Processes by Adaptive Workflow Systems. *Distributed and Parallel Databases*, 16(1):91–116, July 2004.

[RRMD09] M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in Process-aware Information Systems. *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-aware Information Systems*, 2:115–135, March 2009.

[RS13] A. Roth and S. Skritek. Peer Data Management. *Dagstuhl Follow-Ups*, 5, 2013.

[RSB+08] S. Rachuri, E. Subrahmanian, A. Bouras, S.J. Fenves, S. Foufou, and R.D. Sriram. Information sharing and exchange in the context of product lifecycle management: Role of standards. *Computer-Aided Design*, 40(7):789–800, 2008.

[RTW15] Georg Rock, Karsten Theis, and Patrick Wischnewski. Variability Management. In *Concurrent Engineering in the 21st Century*, pages 491–519. Springer, 2015.

[RW12] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, Berlin-Heidelberg, 2012.

[SAR18] S. Steinau, K. Andrews, and M. Reichert. The Relational Process Structure. In *30th Int'l Conference on Advanced Information Systems Engineering (CAiSE 2018)*, number 10816 in LNCS, pages 53–67. Springer, 2018.

[SAS05] SASIG. Product Data Quality Guidelines for the Global Automotive Industry, Version 2 revision 1. Technical report, SASIG, 2005.

[SI08] A. Saaksvuori and A. Immonen. *Product Lifecycle Management*. Springer Science & Business Media, 2008.

[Sim96] H.A. Simon. *The Sciences of the Artificial*. MIT press, 3rd edition, 1996.

[SKAR17] S. Steinau, V. Künzle, K. Andrews, and M. Reichert. Coordinating Business Processes Using Semantic Relationships. In *19th IEEE Conference on Business Informatics (CBI 2017)*, pages 33–42. IEEE Computer Society Press, July 2017.

[SL90] A.P. Sheth and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[SMH08] R. Shearer, B. Motik, and I. Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *OWLED*, volume 432, page 91, 2008.

[SRF+05] E. Subrahmanian, S. Rachuri, S.J. Fenves, S. Foufou, and R.D. Sriram. Challenges in Supporting Product Design and Manufacturing in a Networked Economy: A PLM perspective. In *Proceedings of the International Conference on Product Lifecycle Management (PLM 05)*, pages 495–506, 2005.

[Sta11] J. Stark. *Product Lifecycle Management*. Springer, 2011.

[Ste12] V. Steller. OpenConfigurator - A Practical Approach to Configurator Implementation. Master's thesis, Ulm University, 2012.

[SWB12] M. Schulze, J. Weiland, and D. Beuche. Automotive Model-Driven Development and the Challenge of Variability. In *Proceedings of the 16th International Software Product Line Conference (SPLC'12) – Volume 1*, pages 207–214. ACM, 2012.

[Tat09] A. Tatnall. *Web Technologies: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2009.

[TBHR15] J. Tiedeken, T. Bauer, J. Herbst, and M. Reichert. Determining the Quality of Product Data Integration. In *23rd International Conference on Cooperative Information Systems (CoopIS 2015)*, number 9415 in Lecture Notes in Computer Science, pages 267–284. Springer, 2015.

[Tho09] J. W. Thomas. *Data-Exchange Standards and International Organizations: Adoption and Diffusion*. IGI Global, 2009.

[THR13a] J. Tiedeken, J. Herbst, and M. Reichert. Managing Complex Data for Electrical/Electronic Components: Challenges and Requirements. In *Proc BTW'13 Workshops, 15th Conf on Database Systems, Technology, and Web (BTW'13)*, number 216 in Lecture Notes in Informatics (LNI), pages 141–150. Koellen-Verlag, March 2013.

[THR13b] J. Tiedeken, J. Herbst, and M. Reichert. On the Integration of Electrical/Electronic Product Data in the Automotive Domain. *Datenbank Spektrum*, 13(3):189–199, September 2013.

[TRS15] D. Trippner, S. Rude, and A. Schreiber. Challenges to Digital Product and Process Development Systems at BMW. In *Concurrent Engineering in the 21st Century*, pages 555–569. Springer, 2015.

[UB02] M. Ungerer and K. Buchanan. Usage Guide for the STEP PDM Schema v4.3, 202.

[Ver97] W.W.M. Vermeer. Semantic Interoperability for Legacy Databases. 1997.

[Vi09] VDA and ProSTEP iViP. ECM Recommendation Part 0 (ECM), Version 2.0. Technical report, SASIG, 2009.

[VSW15] W.J.C. Verhagen, J. Stjepandić, and N. Wognum. Challenges of CE. In *Concurrent Engineering in the 21st Century*, pages 807–833. Springer, 2015.

[Was90] A.I. Wasserman. Tool Integration in Software Engineering Environments. In *Software Engineering Environments*, pages 137–149. Springer, 1990.

[Web14] J. Weber. *Automotive Development Processes*. Springer, 2014.

[Wes96] M. West. Integration of Industrial Data for Exchange, Access, and Sharing (IDEAS). Technical report, 1996. Accessed: 2016-07-15.

[Whi08] S.A. White. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Inc., 2008.

[Win99] W.E Winkler. The State of Record Linkage and Current Research Problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

[WKLW98] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin Core Metadata for Resource Discovery. Technical report, 1998.

[WS96] Richard Y Wang and Diane M Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996.

[WZL06] R.Y. Wang, M. Ziad, and Y.W. Lee. *Data Quality*. Springer Science & Business Media, 2006.

[XN09] X. Xu and A.Y.C. Nee. *Advanced Design and Manufacturing Based on STEP*. Springer Science & Business Media, 2009.

[YB03] A. Yassine and D. Braha. Complex Concurrent Engineering and the Design Structure Matrix Method. *Concurrent Engineering*, 11(3):165–176, 2003.

Anhang

A Globale Produktontologie gpo

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
6  @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
7  @prefix : <http://www.uni-ulm.de/proceed/gpo#> .
8  @base <http://www.uni-ulm.de/proceed/gpo> .
9
10   gpo rdf:type owl:Ontology ;
11        rdfs:label "Global Product Ontology" ;
12        owl:versionInfo "1.0" ;
13        rdfs:comment "Global Product Ontology" ;
14        owl:imports <http://www.uni-ulm.de/proceed/metaontology.owl#> ;
15        mo:ontologyType mo:GlobalProductOntology .
16
17   :Product_SC rdf:type owl:Class ;
18                rdfs:label "Product" ;
19                rdfs:subClassOf mo:GlobalPDCLayer .
20
21   :Component_SC rdf:type owl:Class ;
22                rdfs:label "Component" ;
23                rdfs:subClassOf mo:GlobalObjectLayer .
24
25   :ComponentVariant_SC rdf:type owl:Class ;
26                    rdfs:label "Component Variant" ;
27                    rdfs:subClassOf mo:GlobalVariantLayer .
28
29   :ComponentVersion_SC rdf:type owl:Class ;
30                    rdfs:label "Component Version" ;
31                    rdfs:subClassOf mo:GlobalVersionLayer .
32
33   :ComponentBelowProduct rdf:type owl:ObjectProperty ;
34                       rdfs:label "ComponentIsBelowProduct" ;
35                       rdfs:subPropertyOf mo:isBelow ;
36                       rdfs:domain :Component_SC ;
37                       rdfs:range :Product_SC .
38
39   :ProductAboveComponent rdf:type owl:ObjectProperty ;
40                       rdfs:label "ProductIsAboveComponent" ;
41                       rdfs:subPropertyOf mo:isAbove ;
42                       rdfs:domain :Product_SC ;
43                       rdfs:range :Component_SC .
44
45   :ComponentBelowProduct owl:inversOf :ProductAboveComponent .
```

```
46
47    :VariantBelowComponent rdf:type owl:ObjectProperty ;
48                           rdfs:label "ComponentVariantIsBelowComponent" ;
49                           rdfs:subPropertyOf mo:isBelow ;
50                           rdfs:domain :ComponentVariant_SC ;
51                           rdfs:range :Component_SC .
52
53    :ComponentAboveVariant rdf:type owl:ObjectProperty ;
54                           rdfs:label "ComponentIsAboveComponentVariant" ;
55                           rdfs:subPropertyOf mo:isAbove ;
56                           rdfs:domain :Component_SC ;
57                           rdfs:range :ComponentVariant_SC .
58
59    :VariantBelowComponent owl:inversOf :ComponentAboveVariant .
60
61    :VersionBelowVariant rdf:type owl:ObjectProperty ;
62                                 rdfs:label "ComponentVersionIsBelowComponentVariant" ;
63                                 rdfs:subPropertyOf mo:isBelow ;
64                                 rdfs:domain :ComponentVersion_SC ;
65                                 rdfs:range :ComponentVariant_SC .
66
67    :VariantAboveVersion rdf:type owl:ObjectProperty ;
68                                 rdfs:label "ComponentVariantIsAboveComponentVersion" ;
69                                 rdfs:subPropertyOf mo:isAbove ;
70                                 rdfs:domain :ComponentVariant_SC ;
71                                 rdfs:range :ComponentVersion_SC .
72    :VersionBelowVariant owl:inversOf :VariantAboveVersion .
73
74  :ComponentVersionNr_Attr rdf:type owl:DatatypeProperty ;
75                               rdfs:domain :ComponentVersion_SC ;
76                               rdfs:range xsd:string ;
77                               rdfs:subPropertyOf mo:Attribute .
78
79  :ECU_Ref_Attr rdf:type owl:DatatypeProperty ;
80                rdfs:domain :ComponentVersion_SC ;
81                rdfs:range xsd:string ;
82                rdfs:subPropertyOf mo:Attribute .
83
84  :GeometryModelRef_Attr rdf:type owl:DatatypeProperty ;
85                             rdfs:domain :ComponentVersion_SC ;
86                             rdfs:subPropertyOf mo:Attribute .
87
88  :Label_Attr rdf:type owl:DatatypeProperty ;
89              rdfs:range xsd:string ;
90              rdfs:subPropertyOf mo:Attribute .
91
92  :VariantCoding_Attr rdf:type owl:DatatypeProperty ;
93                          rdfs:domain :ComponentVariant_SC ;
94                          rdfs:range xsd:string ;
95                          rdfs:subPropertyOf mo:Attribute .
```

Listing 11.1: Globale Produktontologie als OWL-Ontologie

B Produktontologien des Integrationsszenarios 1

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
3  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
6  @prefix mo: <http://www.uni−ulm.de/proceed/metaontology.owl#> .
7  @prefix : <http://www.uni−ulm.de/proceed/gpo#> .
8  @base <http://www.uni−ulm.de/proceed/gpo> .
9
10   gpo rdf:type owl:Ontology ;
11        rdfs:label "Global Product Ontology" ;
12        owl:versionInfo "1.0" ;
13        rdfs:comment "Global Product Ontology" ;
14        owl:imports <http://www.uni−ulm.de/proceed/metaontology.owl#> ;
15        mo:ontologyType mo:GlobalProductOntology .
16
17   :Product_SC rdf:type owl:Class ;
18                rdfs:label "Product" ;
19                rdfs:subClassOf mo:GlobalPDCLayer .
20
21   :Component_SC rdf:type owl:Class ;
22                rdfs:label "Component" ;
23                rdfs:subClassOf mo:GlobalObjectLayer .
24
25   :ComponentVariant_SC rdf:type owl:Class ;
26                     rdfs:label "Component Variant" ;
27                     rdfs:subClassOf mo:GlobalVariantLayer .
28
29   :ComponentVersion_SC rdf:type owl:Class ;
30                     rdfs:label "Component Version" ;
31                     rdfs:subClassOf mo:GlobalVersionLayer .
32
33   :ComponentBelowProduct rdf:type owl:ObjectProperty ;
34                       rdfs:label "ComponentIsBelowProduct" ;
35                       rdfs:subPropertyOf mo:isBelow ;
36                       rdfs:domain :Component_SC ;
37                       rdfs:range :Product_SC .
38
39   :ProductAboveComponent rdf:type owl:ObjectProperty ;
40                       rdfs:label "ProductIsAboveComponent" ;
41                       rdfs:subPropertyOf mo:isAbove ;
42                       rdfs:domain :Product_SC ;
43                       rdfs:range :Component_SC .
44
45   :ComponentBelowProduct owl:inversOf :ProductAboveComponent .
46
47   :VariantBelowComponent rdf:type owl:ObjectProperty ;
48                          rdfs:label "ComponentVariantIsBelowComponent" ;
49                          rdfs:subPropertyOf mo:isBelow ;
50                          rdfs:domain :ComponentVariant_SC ;
51                          rdfs:range :Component_SC .
52
53   :ComponentAboveVariant rdf:type owl:ObjectProperty ;
54                          rdfs:label "ComponentIsAboveComponentVariant" ;
```

```
                        rdfs:subPropertyOf mo:isAbove ;
                        rdfs:domain :Component_SC ;
                        rdfs:range :ComponentVariant_SC .

  :VariantBelowComponent owl:inversOf :ComponentAboveVariant .

  :VersionBelowVariant rdf:type owl:ObjectProperty ;
                              rdfs:label "ComponentVersionIsBelowComponentVariant" ;
                              rdfs:subPropertyOf mo:isBelow ;
                              rdfs:domain :ComponentVersion_SC ;
                              rdfs:range :ComponentVariant_SC .

  :VariantAboveVersion rdf:type owl:ObjectProperty ;
                              rdfs:label "ComponentVariantIsAboveComponentVersion" ;
                              rdfs:subPropertyOf mo:isAbove ;
                              rdfs:domain :ComponentVariant_SC ;
                              rdfs:range :ComponentVersion_SC .
  :VersionBelowVariant owl:inversOf :VariantAboveVersion .


:Name_Attr rdf:type owl:DatatypeProperty ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:Responsible_Attr rdf:type owl:DatatypeProperty ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:Id_Attr rdf:type owl:DatatypeProperty ;
           rdfs:domain :Product_SC ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:SOP_Attr rdf:type owl:DatatypeProperty ;
           rdfs:domain :Product_SC ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:Abbreviation_Attr rdf:type owl:DatatypeProperty ;
           rdfs:domain :Component_SC ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:Type_Attr rdf:type owl:DatatypeProperty ;
           rdfs:domain :Component_SC ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:Code_Attr rdf:type owl:DatatypeProperty ;
           rdfs:domain :ComponentVariant_SC ;
           rdfs:range xsd:string ;
           rdfs:subPropertyOf mo:Attribute .

:VersionId_Attr rdf:type owl:DatatypeProperty ;
           rdfs:domain :ComponentVersion_SC ;
           rdfs:range xsd:integer ;
```

```
111                 rdfs:subPropertyOf mo:Attribute .
112
113 :PartId_Attr rdf:type owl:DatatypeProperty ;
114                 rdfs:domain :ComponentVersion_SC ;
115                 rdfs:range xsd:integer ;
116                 rdfs:subPropertyOf mo:Attribute .
117
118 :Requirements_Attr rdf:type owl:DatatypeProperty ;
119                 rdfs:domain :ComponentVersion_SC ;
120                 rdfs:range xsd:string ;
121                 rdfs:subPropertyOf mo:Attribute .
122
123 :Geometry_Attr rdf:type owl:DatatypeProperty ;
124                 rdfs:domain :ComponentVersion_SC ;
125                 rdfs:range xsd:string ;
126                 rdfs:subPropertyOf mo:Attribute .
127
128 :Software_Attr rdf:type owl:DatatypeProperty ;
129                 rdfs:domain :ComponentVersion_SC ;
130                 rdfs:range xsd:string ;
131                 rdfs:subPropertyOf mo:Attribute .
```

Listing 11.2: Globale Produktontologie als OWL2-Ontologie

C Lokale Produktontologie `lpo1`

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
6  @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
7  @prefix : <http://www.uni-ulm.de/proceed/lpo1#> .
8  @base <http://www.uni-ulm.de/proceed/lpo1> .
9
10   lpo1 rdf:type owl:Ontology ;
11        rdfs:label "Local Product Ontology 1" ;
12        owl:versionInfo "1.0" ;
13        rdfs:comment "Local Product Ontology 1" ;
14        owl:imports <http://www.uni-ulm.de/proceed/metaontology.owl#> ;
15        mo:ontologyType mo:LocalProductOntologyType .
16
17   :Product_SC rdf:type owl:Class ;
18                rdfs:label "Product" ;
19                rdfs:subClassOf mo:LocalPDCLayer .
20
21   :ECU_SC rdf:type owl:Class ;
22                rdfs:label "ECU" ;
23                rdfs:subClassOf mo:LocalObjectLayer .
24
25   :ECUVariant_SC rdf:type owl:Class ;
26                    rdfs:label "ECU Variant" ;
27                    rdfs:subClassOf mo:LocalVariantLayer .
28
29   :ECUVersion_SC rdf:type owl:Class ;
30                    rdfs:label "ECU Version" ;
```

```
31                    rdfs:subClassOf mo:LocalVersionLayer .
```

Listing 11.3: Lokale Produktontologie `lpo1` als OWL-Ontologie

D Lokale Masterproduktontologie `mpo`

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
3  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
6  @prefix mo: <http://www.uni−ulm.de/proceed/metaontology.owl#> .
7  @prefix : <http://www.uni−ulm.de/proceed/mpo#> .
8  @base <http://www.uni−ulm.de/proceed/mpo> .
9
10   mpo rdf:type owl:Ontology ;
11        rdfs:label "Local Master Product Ontology" ;
12        owl:versionInfo "1.0" ;
13        rdfs:comment "Local Master Product Ontology" ;
14        owl:imports <http://www.uni−ulm.de/proceed/metaontology.owl#> ;
15        mo:ontologyType mo:LocalMasterProductOntologyType .
16
17   :Product_SC rdf:type owl:Class ;
18                rdfs:label "Product" ;
19                rdfs:subClassOf mo:LocalPDCLayer .
20
21   :Part_SC rdf:type owl:Class ;
22                rdfs:label "Part" ;
23                rdfs:subClassOf mo:LocalObjectLayer .
24
25   :PartVariant_SC rdf:type owl:Class ;
26                    rdfs:label "Part Variant" ;
27                    rdfs:subClassOf mo:LocalVariantLayer .
28
29   :PartVersion_SC rdf:type owl:Class ;
30                    rdfs:label "Part Variant Version" ;
31                    rdfs:subClassOf mo:LocalVersionLayer .
```

Listing 11.4: Globale Produktontologie als OWL-Ontologie

E Mapping zwischen `lpo1` und `gpo`

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
3  @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
6  @prefix mo: <http://www.uni−ulm.de/proceed/metaontology.owl#> .
7  @prefix : <http://www.uni−ulm.de/proceed/pom1#> .
8  @base <http://www.uni−ulm.de/proceed/pom1> .
9
10   pom1 rdf:type owl:Ontology ;
11        rdfs:label "Product Ontology Mapping 1" ;
12        owl:versionInfo "1.0" ;
13        rdfs:comment Product Ontology Mapping" ;
14        owl:imports <http://www.uni−ulm.de/proceed/metaontology.owl#> ;
```

```
15        owl:imports <http://www.uni-ulm.de/proceed/lpo1#> ;
16        owl:imports <http://www.uni-ulm.de/proceed/gpo#> ;
17        mo:ontologyType mo:ProductOntologyMappingType .
18
19
20  :productCorrespondesProduct_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty ;
21                                    rdfs:label "SCAR11"
22                                    rdfs:subPropertyOf mo:hasPDCSCARTo .
23                                    mo:hasMatchingFunction mo:StringEquivalence ;
24                                    mo:hasMatchingParameter "" ;
25                                    rdfs:domain lpo1:Product_SC ;
26                                    rdfs:range gpo:Product_SC ;
27                                    mo:hasLocalAttribute lpo1:Name_Attr ;
28                                    mo:hasGlobalAttribute gpo:Name_Attr .
29
30  :ECUCorrespondesComponent_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty ;
31                                    rdfs:label "SCAR12"
32                                    rdfs:subPropertyOf mo:hasObjectSCARTo .
33                                    mo:hasMatchingFunction mo:StringEquivalence ;
34                                    mo:hasMatchingParameter "" ;
35                                    rdfs:domain lpo1:ECU_SC ;
36                                    rdfs:range gpo:Component_SC ;
37                                    mo:hasLocalAttribute lpo1:Name_Attr ;
38                                    mo:hasGlobalAttribute gpo:Abbreviation_Attr .
39
40  :ECUVariantCorrespondesComponentVariant_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty
        ;
41                                    rdfs:label "SCAR13"
42                                    rdfs:subPropertyOf mo:hasVariantSCARTo .
43                                    mo:hasMatchingFunction mo:StringEquivalence ;
44                                    mo:hasMatchingParameter "" ;
45                                    rdfs:domain lpo1:ECUVariant_SC ;
46                                    rdfs:range gpo:ComponentVariant_SC ;
47                                    mo:hasLocalAttribute lpo1:Name_Attr ;
48                                    mo:hasGlobalAttribute gpo:Name_Attr .
49
50
51  :ECUVersionCorrespondesComponentVersion_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty
        ;
52                                    rdfs:label "SCAR14"
53                                    rdfs:subPropertyOf mo:hasVersionSCARTo ;
54                                    rdfs:domain lpo1:ECUVersion_SC ;
55                                    rdfs:range gpo:ComponentVersion_SC .
56
57
58  :Name_AMC rdf:type owl:ObjectProperty ;
59            rdf:type mo:AttributeMatchingCondition ;
60
61            rdfs:subPropertyOf :ECUVersionCorrespondesComponentVersion_SCAR ;
62            mo:hasMatchingFunction mo:StringEquivalence ;
63            mo:hasMatchingParameter "" ;
64            mo:hasLocalAttribute lpo1:Name_Attr ;
65            mo:hasGlobalAttribute gpo:Name_Attr .
66
67  :VersionId_AMC rdf:type owl:ObjectProperty ;
68            rdf:type mo:AttributeMatchingCondition ;
```

```
69              rdfs:subPropertyOf :ECUVersionCorrespondesComponentVersion_SCAR ;
70              mo:hasMatchingFunction mo:StringEquivalence ;
71              mo:hasMatchingParameter "" ;
72              mo:hasLocalAttribute lpo1:VersionId_Attr ;
73              mo:hasGlobalAttribute gpo:VersionId_Attr .
74
75  :ECUVersion2ComponentVersion_SCAA rdf:type owl:ObjectProperty ;
76                                      rdfs:label "SCAA1"
77                                      rdfs:subPropertyOf mo:hasVersionSCAATo
78                                      mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
79
80                                      rdfs:domain lpo1:ECUVersion_SC ;
81                                      rdfs:range gpo:ComponentVersion_SC ;
82                                      mo:hasLocalAttribute lpo1:reference_Attr ;
83                                      mo:hasGlobalAttribute gpo:Software_Attr .
```

Listing 11.5: Produktontologie-Mapping zwischen `lpo1` und `gpo`

F Mapping zwischen `mpo` und `gpo`

```
1   @prefix owl: <http://www.w3.org/2002/07/owl#> .
2   @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3   @prefix xml: <http://www.w3.org/XML/1998/namespace> .
4   @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
6   @prefix mo: <http://www.uni-ulm.de/proceed/metaontology.owl#> .
7   @prefix : <http://www.uni-ulm.de/proceed/pom2#> .
8   @base <http://www.uni-ulm.de/proceed/pom2> .
9
10    pom2 rdf:type owl:Ontology ;
11        rdfs:label "Product Ontology Mapping 2" ;
12        owl:versionInfo "1.0" ;
13        rdfs:comment Product Ontology Mapping" ;
14        owl:imports <http://www.uni-ulm.de/proceed/metaontology.owl#> ;
15        owl:imports <http://www.uni-ulm.de/proceed/mpo#> ;
16        owl:imports <http://www.uni-ulm.de/proceed/gpo#> ;
17        mo:ontologyType mo:ProductOntologyMappingType .
18
19
20  :productCorrespondesProduct_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty ;
21                                      rdfs:label "SCAR21"
22                                      rdfs:subPropertyOf mo:hasPDCSCARTo .
23                                      mo:hasMatchingFunction mo:StringEquivalence ;
24                                      mo:hasMatchingParameter "" ;
25                                      rdfs:domain mpo:Product_SC ;
26                                      rdfs:range gpo:Product_SC ;
27                                      mo:hasLocalAttribute mpo:Name_Attr ;
28                                      mo:hasGlobalAttribute gpo:Name_Attr .
29
30  :partCorrespondesComponent_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty ;
31                                      rdfs:label "SCAR22"
32                                      rdfs:subPropertyOf mo:hasObjectSCARTo .
33                                      mo:hasMatchingFunction mo:StringEquivalence ;
34                                      mo:hasMatchingParameter "" ;
35                                      rdfs:domain mpo:Part_SC ;
36                                      rdfs:range gpo:Component_SC ;
```

```
37                                      mo:hasLocalAttribute mpo:Name_Attr ;
38                                      mo:hasGlobalAttribute gpo:Name_Attr .
39
40  :partVariantCorrespondesComponentVariant_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty
        ;
41                                      rdfs:label "SCAR23"
42                                      rdfs:subPropertyOf mo:hasVariantSCARTo .
43                                      mo:hasMatchingFunction mo:StringEquivalence ;
44                                      mo:hasMatchingParameter "" ;
45                                      rdfs:domain mpo:PartVariant_SC ;
46                                      rdfs:range gpo:ComponentVariant_SC ;
47                                      mo:hasLocalAttribute mpo:Name_Attr ;
48                                      mo:hasGlobalAttribute gpo:Name_Attr .
49
50  :PartVersionCorrespondesComponentVersion_SCAR rdf:type owl:ObjectProperty , owl:SymmetricProperty
        ;
51                                      rdfs:label "SCAR24"
52                                      rdfs:subPropertyOf mo:hasVersionSCARTo ;
53                                      rdfs:domain mpo:ECUVersion_SC ;
54                                      rdfs:range gpo:ComponentVersion_SC .
55
56
57  :Name_AMC rdf:type owl:ObjectProperty ;
58              rdf:type mo:AttributeMatchingCondition ;
59              rdfs:label "SCAA1"
60              rdfs:subPropertyOf :PartVersionCorrespondesComponentVersion_SCAR ;
61              mo:hasMatchingFunction mo:StringEquivalence ;
62              mo:hasMatchingParameter "" ;
63              mo:hasLocalAttribute mpo:Name_Attr ;
64              mo:hasGlobalAttribute gpo:Name_Attr .
65
66  :VersionId_AMC rdf:type owl:ObjectProperty ;
67              rdf:type mo:AttributeMatchingCondition ;
68              rdfs:subPropertyOf :PartVersionCorrespondesComponentVersion_SCAR ;
69              mo:hasMatchingFunction mo:StringEquivalence ;
70              mo:hasMatchingParameter "" ;
71              mo:hasLocalAttribute mpo:VersionId_Attr ;
72              mo:hasGlobalAttribute gpo:VersionId_Attr .
73
74  :Product2Product1_SCAA rdf:type owl:ObjectProperty ;
75                                      rdfs:label "SCAA21"
76                                      rdfs:subPropertyOf mo:hasPDCSCAATo
77                                      mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
78
79                                      rdfs:domain mpo:Product_SC ;
80                                      rdfs:range gpo:Product_SC ;
81                                      mo:hasLocalAttribute mpo:responsible_Attr ;
82                                      mo:hasGlobalAttribute gpo:responsible_Attr .
83
84  :Product2Product2_SCAA rdf:type owl:ObjectProperty ;
85                                      rdfs:label "SCAA22"
86                                      rdfs:subPropertyOf mo:hasPDCSCAATo
87                                      mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
88
89                                      rdfs:domain mpo:Product_SC ;
90                                      rdfs:range gpo:Product_SC ;
```

```
91                                      mo:hasLocalAttribute mpo:Id_Attr ;
92                                      mo:hasGlobalAttribute gpo:Id_Attr .
93
94  :Product2Product3_SCAA rdf:type owl:ObjectProperty ;
95                                      rdfs:label "SCAA23"
96                                      rdfs:subPropertyOf mo:hasPDCSCAATo
97                                      mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
98
99                                      rdfs:domain mpo:Product_SC ;
100                                     rdfs:range gpo:Product_SC ;
101                                     mo:hasLocalAttribute mpo:SOP_Attr ;
102                                     mo:hasGlobalAttribute gpo:SOP_Attr .
103
104
105 :Part2Component1_SCAA rdf:type owl:ObjectProperty ;
106                                     rdfs:label "SCAA24"
107                                     rdfs:subPropertyOf mo:hasObjectSCAATo
108                                     mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
109
110                                     rdfs:domain mpo:Part_SC ;
111                                     rdfs:range gpo:Component_SC ;
112                                     mo:hasLocalAttribute mpo:Abbreviation_Attr ;
113                                     mo:hasGlobalAttribute gpo:Abbreviation_Attr .
114
115 :Part2Component2_SCAA rdf:type owl:ObjectProperty ;
116                                     rdfs:label "SCAA25"
117                                     rdfs:subPropertyOf mo:hasObjectSCAATo
118                                     mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
119
120                                     rdfs:domain mpo:Part_SC ;
121                                     rdfs:range gpo:Component_SC ;
122                                     mo:hasLocalAttribute mpo:Type_Attr ;
123                                     mo:hasGlobalAttribute gpo:Type_Attr .
124
125 :PartVariant2ComponentVariant1_SCAA rdf:type owl:ObjectProperty ;
126                                     rdfs:label "SCAA26"
127                                     rdfs:subPropertyOf mo:hasVariantSCAATo
128                                     mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
129
130                                     rdfs:domain mpo:PartVariant_SC ;
131                                     rdfs:range gpo:ComponentVariant_SC ;
132                                     mo:hasLocalAttribute mpo:Code_Attr ;
133                                     mo:hasGlobalAttribute gpo:Code_Attr .
134
135 :PartVersion2ComponentVersion1_SCAA rdf:type owl:ObjectProperty ;
136                                     rdfs:label "SCAA27"
137                                     rdfs:subPropertyOf mo:hasVersionSCAATo
138                                     mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
139
140                                     rdfs:domain mpo:PartVersion_SC ;
141                                     rdfs:range gpo:ComponentVersion_SC ;
142                                     mo:hasLocalAttribute mpo:PartId_Attr ;
143                                     mo:hasGlobalAttribute gpo:PartId_Attr .
144
145 :PartVersion2ComponentVersion2_SCAA rdf:type owl:ObjectProperty ;
146                                     rdfs:label "SCAA28"
```

```
147                                rdfs:subPropertyOf mo:hasVersionSCAATo
148                                mo:hasIntegrationBehavior mo:LatestIntegrationBehavior ;
149
150                                rdfs:domain mpo:PartVersion_SC ;
151                                rdfs:range gpo:ComponentVersion_SC ;
152                                mo:hasLocalAttribute mpo:Geometry_Attr ;
153                                mo:hasGlobalAttribute gpo:Geometry_Attr .
```

Listing 11.6: Produktontologie-Mapping zwischen `mpo` und `gpo`

G Usability Konzept

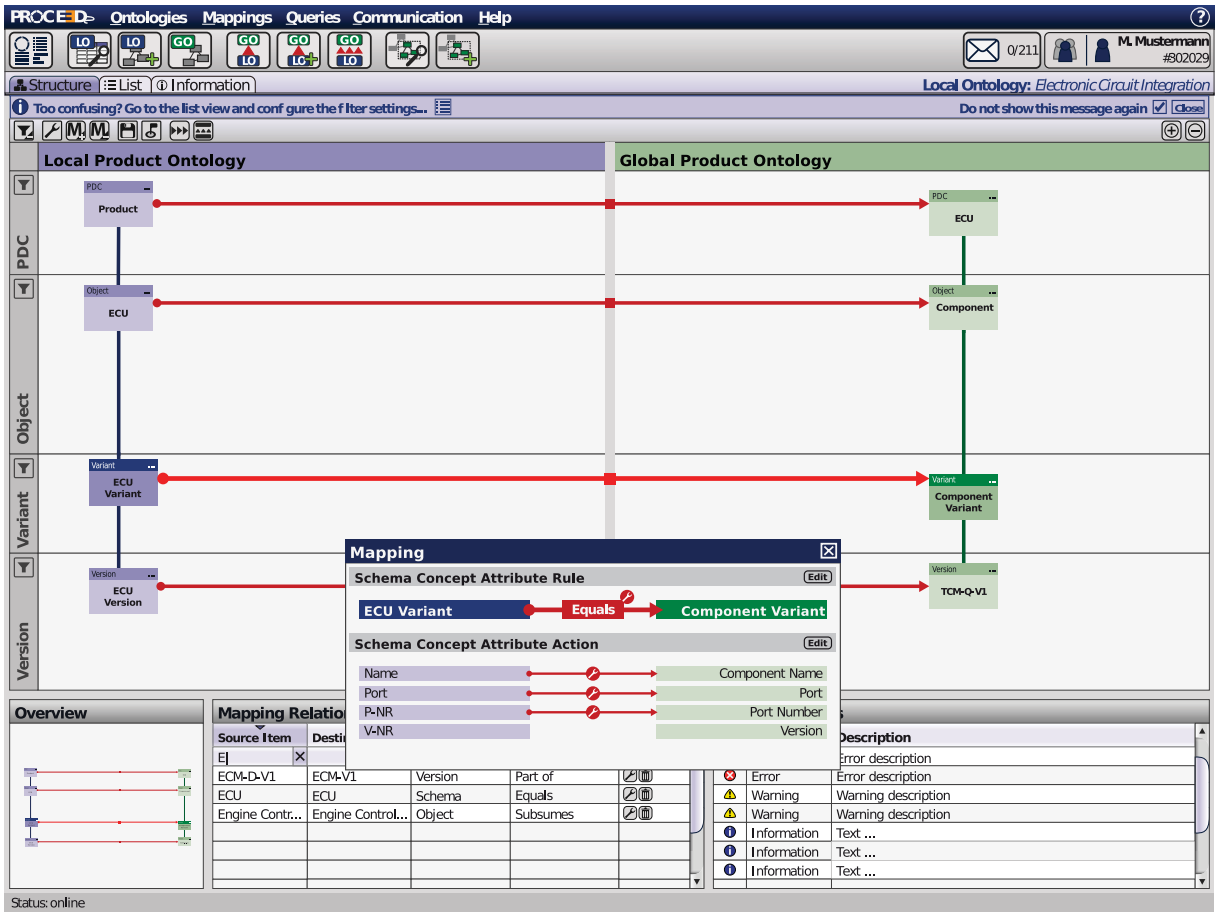


Abbildung 11.1.: Grafische Darstellung von lokaler und globaler Produktontologie inklusive SCARs

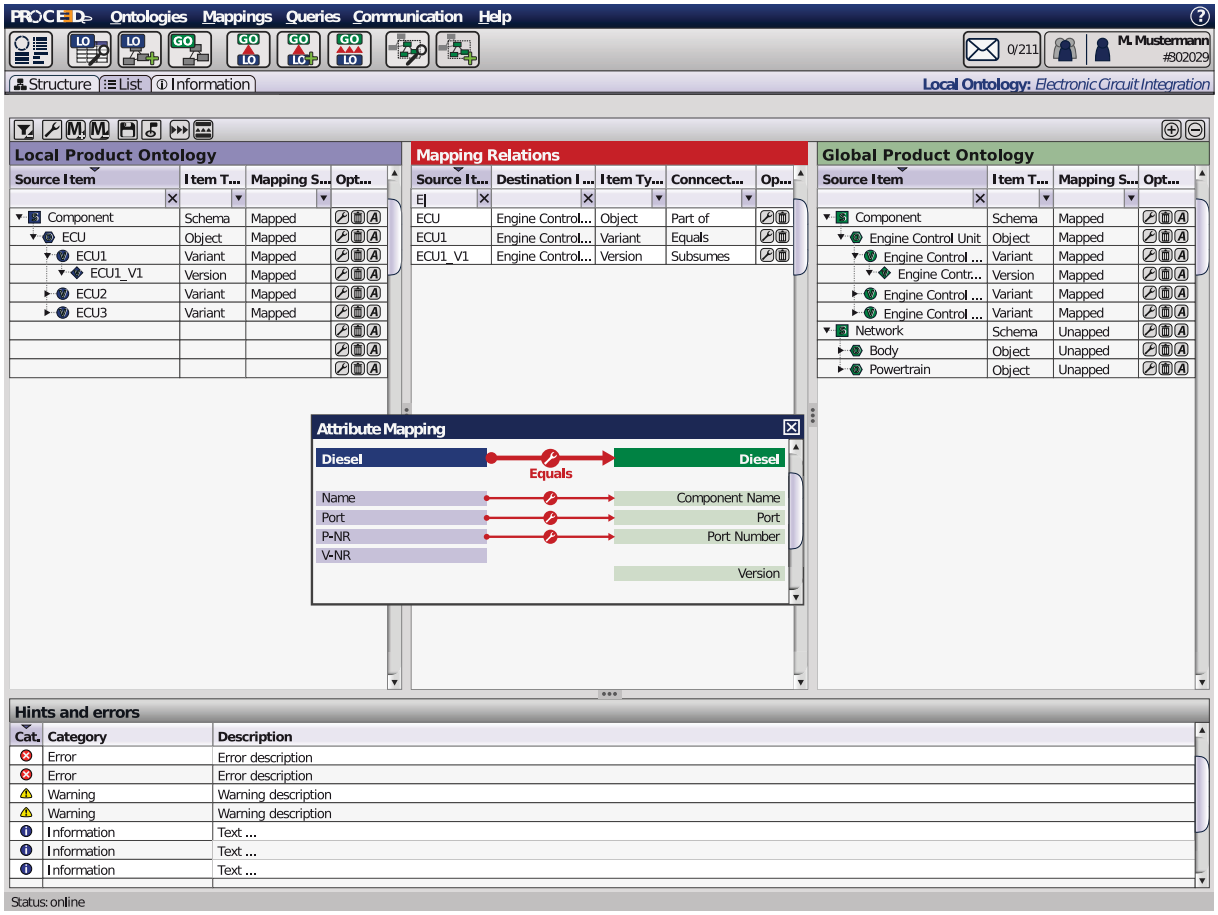


Abbildung 11.2.: Baumdarstellung von lokaler und globaler Produktontologie

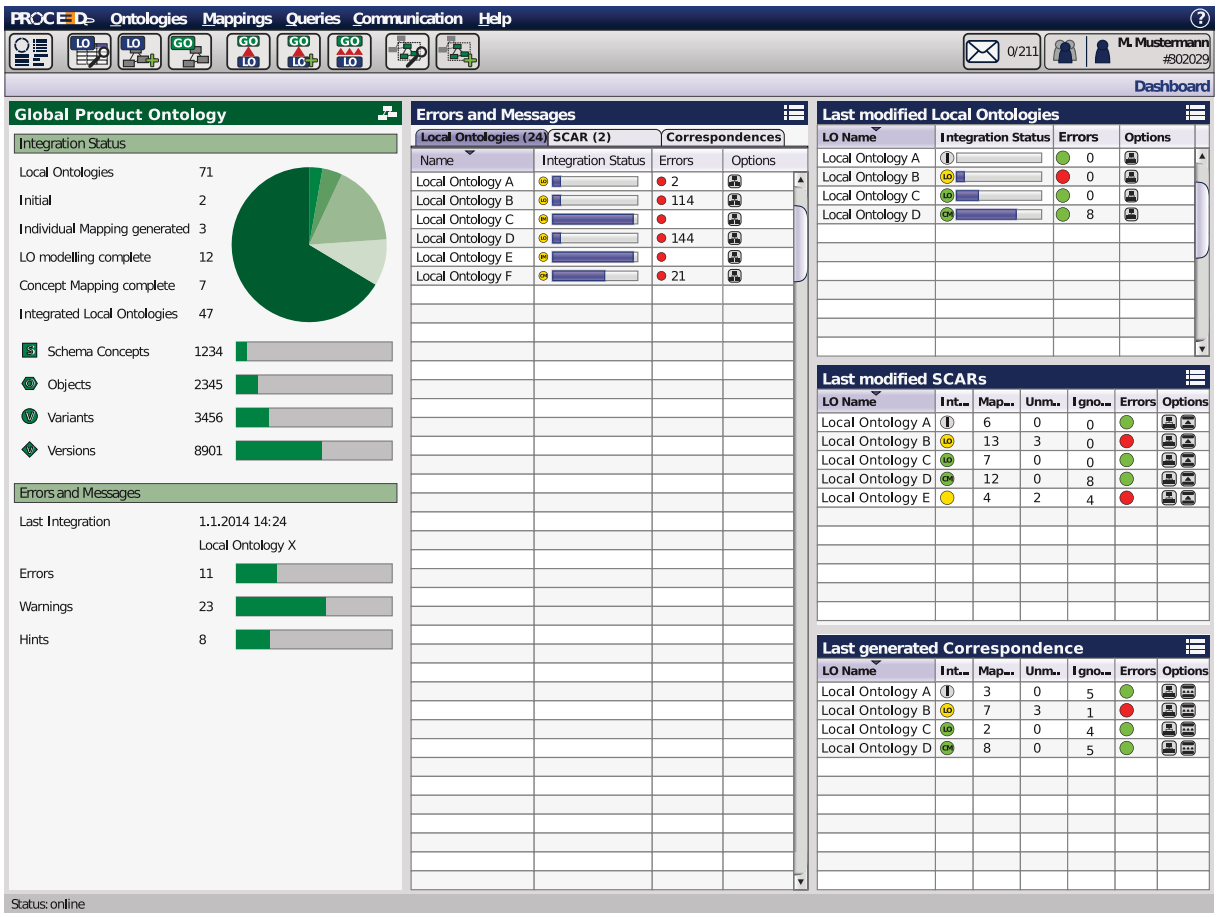


Abbildung 11.3.: Dashboard-Ansicht für die Anzeige von Fehlern, Hinweisen und Metriken

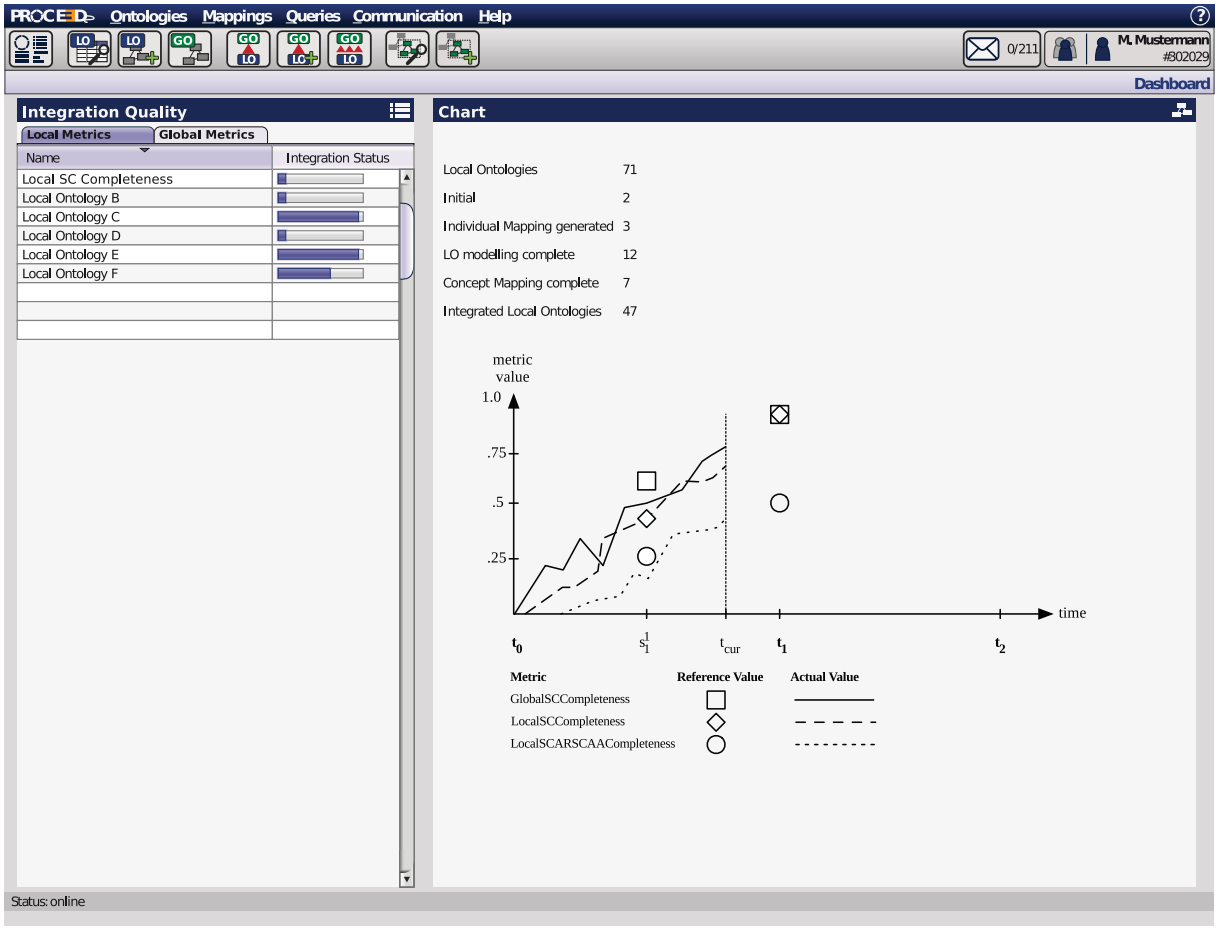


Abbildung 11.4.: Darstellung des grafischen Verlaufs von Qualitätsmetriken