Integrierte Produkt- und Erfahrungsdatenverwaltung

**Wolfgang Mahnke, Ulrich Marder, Stefan Queins,
Norbert Ritter, Bernd Schürmann**

SFB 501 Bericht 08/2000

Integrierte Produkt- und Erfahrungsdatenverwaltung

Wolfgang Mahnke[*], Ulrich Marder[*], Stefan Queins[+],
Norbert Ritter[*], Bernd Schürmann[++]

{mahnke,marder,queins,ritter,schuerma}@informatik.uni-kl.de

Sonderforschungsbereich 501
Technischer Bericht 08/2000

[*]AG Datenbanken und Informationssysteme

[+]AG VLSI-Entwurf und -Architektur

[++]AG Entwurfsmethodik Eingebetteter Systeme

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

Zusammenfassung

In diesem Bericht wird ausgehend von dem bestehenden Prototyp der SFB-501-Erfahrungsdatenbank der Frage nachgegangen, inwieweit eine Integration von Produktdatenstrukturen und zugehörigen Manipulationsfunktionen mit der bestehenden Erfahrungsdatenverwaltung angestrebt werden sollte. Da der Fall, dass Produktdaten, teilweise bereits vor Abschluss des zugehörigen Projekts, in den Status 'Erfahrung' angehoben werden, sehr häufig auftritt, halten wir eine entsprechende Integration nicht nur für sinnvoll sondern auch für notwendig. Neben der Klärung der Frage, wie diese Integration konzeptuell und realisierungstechnisch zu gestalten ist, muss weiter die Möglichkeit geschaffen werden, mehrere Projekt-spezifische Produktdatenmodelle zu unterstützen. Wir beschreiben einen in Zusammenarbeit der Teilprojekte A3 und D1 erarbeiteten Integrationsvorschlag.

Schlüsselwörter:

Erfahrungsdatenbank, Produktdatenverwaltung, Schemaintegration

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

In Zusammenarbeit der Teilprojekte A1 und A3 des SFB 501 wurde ein Ansatz (eines Reuse-Repositories [10]) zur Datenbank-gestützten Erfahrungsdatenverwaltung [2, 3, 14] entwickelt. Diese SFB-501-Erfahrungsdatenbank (EDB) unterstützt die Abwicklung von Softwareentwicklungsprojekten nach dem Quality-Improvement-Paradigma [1, 2], das einen ablaufbezogenen Rahmen zur umfassenden Wiederverwendung von Software-Artefakten anbietet. Erfahrungen sind also im Wesentlichen in SE-Prozessen entstandene, potenziell wiederverwendbare Artefakte. Das Spektrum reicht hier von Beschreibungen/Bewertungen von in früheren Projekten genutzten Methoden und Technologien über Prozessmodelle bis hin zu konkreten Produktdaten, die in dem neuen Prozess wiederverwendet bzw. zwecks neuartiger Verwendung angepasst werden sollen. Wie in [3, 13] beschrieben, wurde die SFB-501-EDB so konzipiert, dass sie eine geeignete Wiederverwendungs-Infrastruktur in allen Projektphasen bietet. Obwohl dieser Ansatz auch bereits erste Möglichkeiten der Verwaltung von Produktdaten beinhaltet, ist zunächst davon auszugehen, dass die eigentlichen in den Projekten zu manipulierenden Produktdaten in separaten, möglicherweise auf die Verwendung spezieller Werkzeuge zugeschnittenen Systemen verwaltet werden.

In diesem Artikel wollen wir der Frage nachgehen, inwieweit eine Integration der Verwaltung von Produkt- und Erfahrungsdaten im SFB 501 angestrebt werden soll bzw. wie eine solche Integration ggf. gestaltet werden sollte.

Die gegenwärtige (System-)Situation im SFB 501 kann folgendermaßen dargestellt werden. Es wird ein EDB-Prototyp entwickelt, der in den einzelnen Teilprojekten (während der Abwicklung experimenteller SE-Prozesse) bzw. auch außerhalb des SFB's gewonnene Erfahrungen zentral verwaltet und zur Verfügung stellt. Der aktuelle Prototyp legt damit eindeutig den Schwerpunkt auf die Verwaltung von Erfahrungsdaten, stellt aber auch die Möglichkeit der Verwaltung von Produktdaten bereit (siehe Kapitel 2). Letzteres geschieht jedoch in Form eines allgemeinen, generischen Mechanismus', der die Prinzipien der Erfahrungsdatenverwaltung auch auf Produktdaten anwendet, aber die in den einzelnen Teilprojekten genutzten, spezifischen Produktdatenmodelle nicht berücksichtigt.

Aus der Sicht eines einzelnen Teilprojekts halten wir aus folgenden Gründen eine integrierte Kontrolle von Erfahrungs- und Produktdaten (unter Berücksichtigung des spezifischen Produktdatenmodells) für sinnvoll:

- Sowohl die Übertragung von wiederzuverwendenden Erfahrungsdaten in einen dem Projekt zugänglichen Produktdatenbereich (z. B. während der Planungsphase) als auch die Gewinnung von Erfahrungsdaten aus Projekt-bezogenen Daten (z. B. Produktdaten) gestalten sich einfacher bei integrierter Verwaltung, da keine Systemgrenzen überwunden werden müssen.
- Zwischen Erfahrungs- und Produktdaten können während des Projektablaufs semantische Beziehungen geknüpft werden, die sowohl hilfreich für den Projektablauf selbst als auch für die Analyse von Projekt- und Erfahrungsdaten sein können. So können beispielsweise bei Wiederverwendung Abstammungsbeziehungen zwischen im Projekt manipulierten Daten und den zugehörigen, ursprünglichen Erfahrungsdaten gehalten werden. Auf diese Weise kann nach

Projektabschluss entschieden werden, welche der beiden Varianten für die weitere Wiederverwendung vorgehalten werden soll.

- Da Erfahrungsdaten meistens in Projekten gewonnen/erzeugt werden bzw. in Projekten Erfahrungsdaten wiederverwendet werden, nutzen Erfahrungsdaten-Manager und Projektentwickler oft dieselben Werkzeuge zur Manipulation der Daten. Entsprechend macht es Sinn, diese (Entwicklungs-)Werkzeuge an ein integriertes Repository anzubinden und damit die Möglichkeit zu schaffen, sie sowohl zur Erfahrungsdatenverwaltung als auch zur Entwicklungsarbeit in laufenden Projekten zu nutzen.

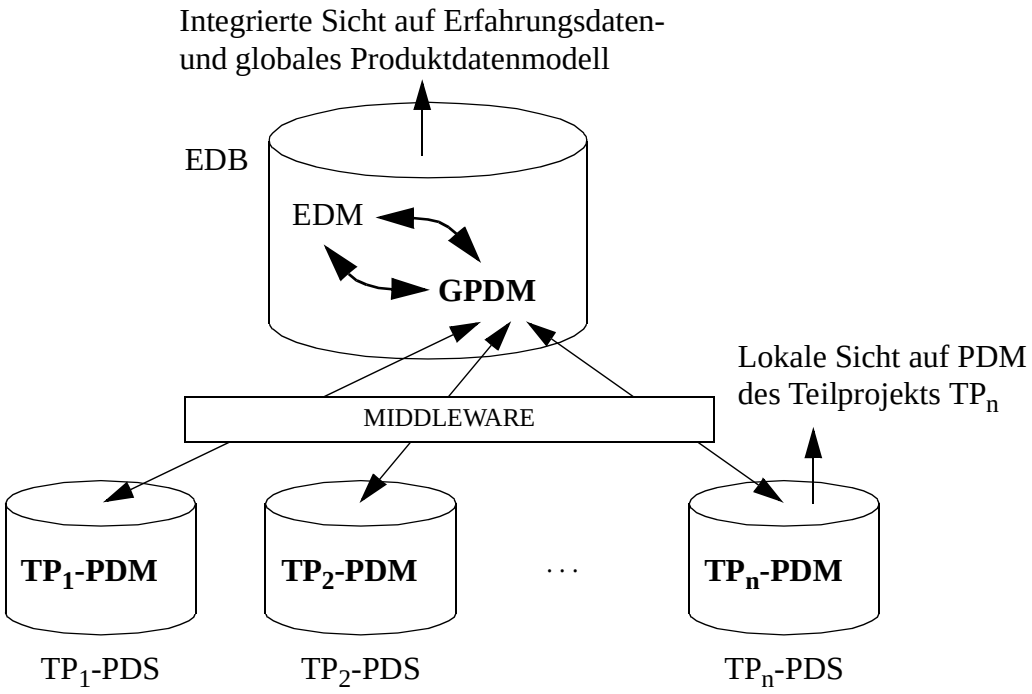
Aus diesen Gründen streben wir die integrierte Verwaltung von Erfahrungs- und Produktdaten an. Jedoch stellt sich neben der Frage nach der geeigneten Integration des Produktdatenmodells eines Teilprojekts mit der Erfahrungsdatenverwaltung das Problem, in welcher Weise die unterschiedlichen Produktdatenmodelle der verschiedenen Teilprojekte des SFB's berücksichtigt werden können. Hier ergeben sich aus Datenbanksicht zwei grundsätzliche Lösungsmöglichkeiten:

- Globales Produktdatenmodell

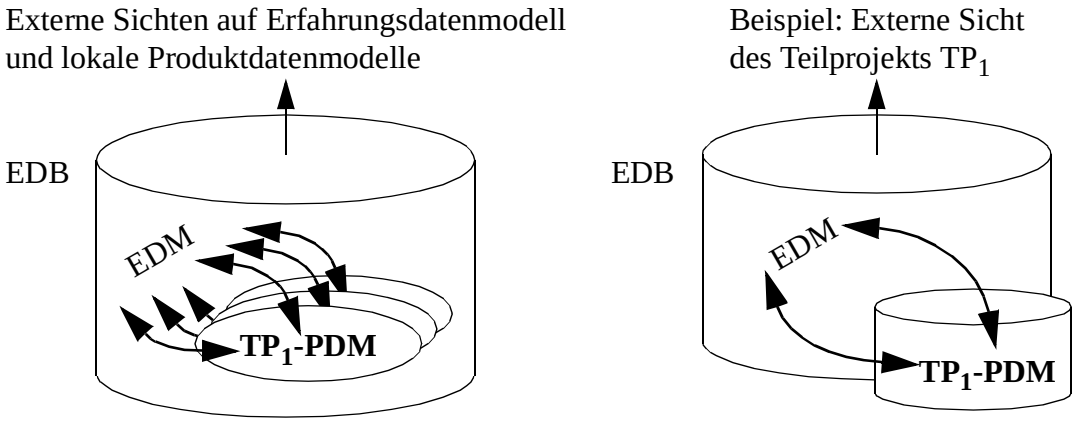
  Ein globales Produktdatenmodell erfordert die konzeptionelle Integration aller in den einzelnen zu betrachtenden Produktdatenmodellen enthaltenen Elemente. Abbildung 1 a) illustriert diesen Fall, bei dem in einem ersten Schritt mit Hilfe einer entsprechend geeigneten (DB-)Middleware, wie z. B. BRITTY [6], ein globales, d. h., alle semantischen Aspekte aller betroffenen lokalen Produktdatenschemata integrierendes Schema (GPDM) erzeugt wird und dieses dann in einem zweiten Schritt mit dem Erfahrungsdatenmodell (EDM) integriert wird. Nach diesen beiden Integrationsschritten bietet die EDB, wie in Abbildung 1 a) illustriert, eine integrierte Sicht auf Erfahrungs- und Produktdatenverwaltung und kann entsprechende Manipulationsfunktionen anbieten. Die an der EDB-Schnittstelle gewünschten Manipulationsfunktionen bestimmen die Komplexität der Middleware-Integration. Falls nicht nur über die lokalen (Teilprojekt-spezifischen) Produktdaten(verwaltungs)systeme (PDS) sondern auch über die EDB-Schnittstelle Produktdaten manipuliert werden sollen, muss die Middleware mächtig genug sein, gegen das GPDM spezifizierte Änderungsoperationen auf die einzelnen lokalen PDS aufzuspalten. Die bereits angesprochene Middleware BRITTY [6] leistet dies.

- Teilprojekt-spezifische, externe Produktdaten-Sichten

  Abbildung 1 b) illustriert diese zweite Möglichkeit, wobei auf der linken Seite der allgemeine Ansatz illustriert wird. Dieser besteht darin, jedem einzelnen Teilprojekt eine Integration seines spezifischen Produktdatenmodells mit dem EDM als externe EDB-Sicht zur Verfügung zu stellen. Somit erscheint jedem Teilprojekt die EDB so, als ob sie neben den allgemeinen Erfahrungsdatenstrukturen lediglich die eigenen Produktdatenstrukturen unterstützt. Entsprechend findet hier sowohl die Manipulation von Erfahrungsdaten als auch die Manipulation von Produktdaten (in jedem Fall) über die EDB-Schnittstelle statt. Damit legen wir uns jedoch realisierungstechnisch noch nicht darauf fest, ob dieses System mit Hilfe genau eines DBVS oder als Kopplung eines TP-PDS mit der EDB realisiert wird. Abbildung 1 b) deutet mit der Illustration auf der rechten Seite diesen beiden Möglichkeiten an.



a) Globales Produktdatenmodell



b) Teilprojekt-spezifische, externe Produktdaten-Sichten

Legende:

EDM:	Erfahrungsdatenmodell
GPDM:	globales Produktdatenmodell
TP_i-PDM:	spezifisches Produktdatenmodell des Teilprojekts TP_i
TP_i-PDS:	Produktdaten(verwaltungs)system des Teilprojekts TP_i

Abbildung 1: Integration von Produktdatemodellen und EDB

Dieses mit seinen zwei prinzipiellen Lösungsmöglichkeiten hier aufgezeigte Problem der Integration mehrerer PDM mit dem EDM sollte in diesem Bericht jedoch nur der Vollständigkeit halber ange-

sprochen werden. Wir werden in einem späteren Bericht genaue Aussagen über eine entsprechende Entscheidung machen. Zuvor müssen die Prinzipien der Integration von EDM und (G)PDM (also die Realisierung der gebogenen Pfeile in Abbildung 1) geklärt werden, was in diesem (vorliegenden) Bericht geschehen soll. Entsprechend ist dieser Bericht wie folgt gegliedert. Im nachfolgenden Kapitel werden wir zunächst den gegenwärtigen Ansatz der Erfahrungsdatenverwaltung, so wie er mit der SFB-501-EDB implementiert wurde, beschreiben. Daran anschließend wird Kapitel 3 die Anforderungen der Integration der Produktdatenverwaltung in die bereits existierende Erfahrungsdatenverwaltung herausstellen. Kapitel 4 beschreibt allgemeine Integrationskonzepte, bewertet diese und wählt den für unsere Zwecke geeigneten Ansatz aus. Kapitel 5 beschreibt den von uns gewählten Ansatz anhand der zugehörigen Systemarchitektur. Anschließend beschreibt Kapitel 6 das im Teilprojekt D1 des SFB 501 entwickelte Produktdatenmodell, das repräsentativ für andere Produktdatenmodelle zur Gestaltung des Integrationsansatzes betrachtet wurde. Die in Kapitel 7 diskutierten operationalen Aspekte vervollständigen die Beschreibung des integrierten Ansatzes. Vor der Zusammenfassung gibt Kapitel 8 noch Einblicke in die Nutzung des integrierten Systems in einem beispielhaften Prozess.

2 Erfahrungsdatenverwaltung (SFB-501-EDB)

Der aktuelle EDB-Prototyp verfolgt einen allgemeinen, generischen Ansatz hinsichtlich der Verwaltung von *Erfahrungen* und *Produktdaten*, berücksichtigt aber keine spezifischen Produktdatenmodelle, wie bereits oben angesprochen. Dennoch sollen in diesem Kapitel beide Arten von Daten und die zugehörigen Verwaltungsmechanismen des gegenwärtigen EDB-Prototypen angesprochen werden, um den aktuellen Stand zu beschreiben.

Erfahrungen seien durch in Projekten/Experimenten (wir benutzen diese Begriffe synonym) erzeugte Artefakte[1] repräsentiert, die eine potenzielle Wiederverwendbarkeit in nachfolgenden Projekten aufweisen. *Produktdaten* sind Daten, die sich in der Bearbeitung durch momentan laufende Projekte befinden und daher (aus der allgemeinen EDB-Nutzer-Sicht) natürlicherweise von beschränkter Sichtbarkeit/Gültigkeit sind.

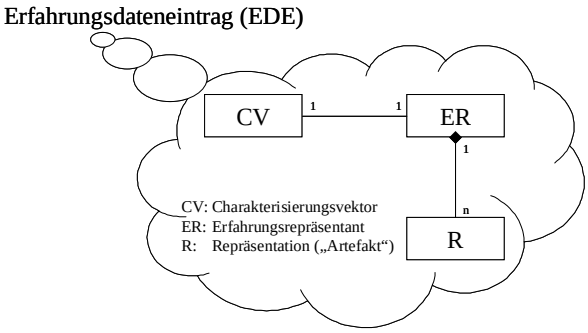


Abbildung 2: Konzeptionelles Schema eines Erfahrungsdateneintrags

2.1 Erfahrungsverwaltung

Zur Verwaltung von Erfahrungen werden in einem als sogenannte *OWS* (von engl.: *organization-wide section*) bezeichneten Bereich der EDB die im SFB-Bericht 08/99 [3] beschriebenen Klassifikationsstrukturen benutzt. Innerhalb dieser Strukturen wird jede einzelne erfasste Erfahrung als ein *Erfahrungsdateneintrag* (EDE) folgendermaßen gespeichert (vgl. Abbildung 2): Erfahrungen können in

1. Unter dem Begriff ‘Artefakt’ verstehen wir etwas Physisches, z. B. eine Datei oder eine Menge von Tupeln in einer DB, unter ‘Erfahrung’ hingegen etwas Abstraktes, nicht zuletzt wegen des eher subjektiven Charakters dieses Begriffs im allgemeinen Sprachgebrauch.

mehreren verschiedenen Repräsentationen abgelegt werden, zwischen denen jedoch von einer logischen Äquivalenzbeziehung auszugehen ist. Beispielsweise kann ein Dokument sowohl im Framemaker-Format, im Postscript-Format als auch im Tex-Format vorliegen, solange diese drei Repräsentationen dieselben Inhalte haben. Diese logische Äquivalenz wird durch einen sog. *Erfahrungsrepräsentanten* (ER) modelliert. Für die Speicherung der Repräsentationen kommt primär die Nutzung von LOB-Datentypen (*Large Objects*) im Datenbanksystem in Frage, da wiederverwendbare Artefakte von Entwicklungswerkzeugen in proprietären Formaten erzeugt bzw. zur Weiterverarbeitung erwartet werden. Um dennoch eine angemessene ähnlichkeitsbasierte Suche nach Erfahrungen (wie sie insbesondere zu Beginn neuer Projekte wichtig ist) anbieten zu können, wird mit jedem Erfahrungsrepräsentanten im Rahmen einer (1:1)-Beziehung ein sogenannter *Charakterisierungsvektor* (CV) assoziiert, dessen Inhalte den ER (bzw. den bedeutungstragenden Inhalt der dem ER zugeordneten Repräsentationen) beschreiben und dessen Struktur (Attribute, Beziehungen) so zur Formulierung von Suchkriterien herangezogen werden können. Die von der EDB angebotenen Benutzerfunktionen zur Manipulation von Erfahrungsdaten bzw. für die ähnlichkeitsbasierte Suche können an dieser Stelle aus Platzgründen nicht detailliert beschrieben werden; es sei auf [2, 3] verwiesen.

2.2 Produktdatenverwaltung

Zur Verwaltung von Produktdaten wird (im gegenwärtigen EDB-Prototyp) jedem laufenden Projekt ein spezieller Bereich zugeordnet (engl.: *experiment-specific section*, kurz ESS), der zunächst, d. h. bis auf explizite, selektive Freigabe einzelner Inhalte, der Sichtbarkeit von Nicht-Projektbeteiligten entzogen ist. In diesen Bereich können natürlich (von Projektbeteiligten) Daten aus der OWS als Ausgangspunkt für Projekt-spezifische Aktivitäten ‘hineinkopiert’ werden (Wiederverwendung). Entsprechend werden auch Produktdaten in der bereits oben angesprochenen Form als LOBs mit zugehörigen Charakterisierungsvektoren verwaltet. Die Verarbeitung von Produktdaten geschieht in aller Regel durch Anwendung von Entwicklungswerkzeugen, die in einer Art ‘losen Kopplung’ mit der EDB zusammenarbeiten (Eingabedaten werden vor Aufruf des Werkzeugs an eine bestimmte ‘Stelle’ im Dateisystem gespeichert und die durch den Werkzeuglauf modifizierte(n) Datei(en) wird/ werden anschließend wieder in die EDB ‘hineingesaugt’). Mit dem in diesem Bericht angestrebten höheren Integrationsgrad der Produktdatenverwaltung werden sich weitere Möglichkeiten der Kopplung von Entwurfswerkzeugen und EDB ergeben, wie wir später sehen werden.

3 Grundlegendes Problem der Integration von Produktdatenmodellen mit der EDB

In Erweiterung des im vorangegangenen Kapitel beschriebenen und im gegenwärtigen EDB-Prototyp realisierten Integrationsgrad betrachten wir im Folgenden den Fall, dass das Produktdatenmodell[2] vollständig mit der EDB integriert wird (anstelle einer einfachen Speicherung von Produktdaten als LOBs).

2. Wir abstrahieren zunächst davon, ob ein repräsentatives Teilprojekt-spezifisches Produktdatenmodell oder ein globales Produktdatenmodell in die EDB integriert wird, da dies, wie bereits in Kapitel 1 beschrieben, keine unterschiedlichen Integrationsmechanismen erfordert.

Generell erfordert eine solche Integration folgende Maßnahmen:

- Bereitstellung von EDB-Schemastrukturen zur Aufnahme der Teilprojekt-spezifischen Produktdatenmodelle; hierbei sind Versionierungsgesichtspunkte zu beachten.
- Identifikation der Eigenschaften der in den Teilprojekten genutzten Werkzeuge hinsichtlich ihres Datenzugriffsverhaltens und ihrer Aufrufmöglichkeiten. Diese Eigenschaften bestimmen die Art der Integration der Werkzeuge mit der EDB.

Während die integrierte Produktdatenverwaltung für laufende Projekte zweifellos den Idealfall darstellt, gilt dies vom Standpunkt der Erfahrungsdatenverwaltung nicht mehr uneingeschränkt. Dies liegt im Wesentlichen daran, dass Produktdaten, betrachtet man sie einmal als Erfahrungen, nicht mehr nur für Anwendungen bzw. Werkzeuge interessant sind, die die Semantik des entsprechenden Produktdatenmodells kennen. Vielmehr steht nun die inhaltsbezogene Suche nach (im Prinzip beliebigen) Erfahrungen und deren (z. B. für Projektmanager) aussagekräftige Präsentation im Vordergrund.

Um solcherart verwaltete Produktdaten als Erfahrungen nutzen zu können, liegt es nahe, das entsprechende Produktdatenmodell (genauer: definierte Teile der darin verwalteten Produktdaten) einfach als eine weitere mögliche Repräsentation, die einem ER zugordnet wird, anzusehen. Diese Lösung birgt jedoch einige Probleme bezüglich der Konsistenzerhaltung, die im Wesentlichen darauf zurückzuführen sind, dass schwer feststellbar ist, inwieweit sich Aktualisierungen bei den Produktdaten auf die Gültigkeit eventuell existierender anderer (semantisch äquivalenter) Repräsentationen auswirken. Das nachfolgende Kapitel diskutiert diese Probleme und erläutert prinzipielle Lösungsmöglichkeiten.

4 Erfahrungsrepräsentation durch Sichtenbildung auf Produktdatenmodellen

Die Integration der Produktdatenmodelle in die EDB macht es prinzipiell möglich, Erfahrungen durch die Bildung von Sichten auf die Produktdaten zu repräsentieren. Das bisherige EDB-Modell darf hierdurch jedoch nicht grundsätzlich verändert werden, da sonst der Vorteil verloren geht, dass mit relativ einfachen, universellen Werkzeugen (z. B. Web-Browser) nach Erfahrungen gesucht und diese dann auch dargestellt werden können (sofern eine Repräsentation in einem leicht darstellbaren Format wie Plain Text, HTML oder Postscript verfügbar ist). Kennzeichnend für diese Art der Modellierung ist u. a., dass Erfahrung Objektcharakter (hier insbesondere Objektidentität) erhält. Anwendungen, die mit diesem Modell arbeiten, können (Erfahrungs-)repräsentationen nur als Ganzes lesen oder schreiben. Wird eine Repräsentation geändert oder hinzugefügt, kann davon die Gültigkeit der übrigen Repräsentationen unter *demselben* ER sowie des dazugehörigen Charakterisierungsvektors und dessen Beziehungen zu anderen Erfahrungen betroffen sein (ob dies tatsächlich so ist, kann nur ein Anwender von Fall zu Fall entscheiden). Die Integrität anderer EDE bleibt jedoch generell unberührt.

Möchte man bei der PDM-Integration die oben genannten Eigenschaften des EDB-Modells möglichst erhalten, so ist das im folgenden Abschnitt beschriebene Konsistenzproblem zu lösen. Wir stellen im Anschluss daran zwei mögliche Lösungsansätze vor. Der für die Praxis interessantere sog. extensionale Ansatz wird dann in den nachfolgenden Kapiteln unter Betrachtung eines konkreten PDMs aus dem SFB weiter vertieft.

4.1 Problem der Konsistenzerhaltung

Die Verwendung von gemäß eines Produktdatenmodells verwalteten Produktdaten als Erfahrungsrepräsentation (kurz: *PDM-Repräsentation* bzw. R_{PDM}) bedeutet, dass es zwei unterschiedliche miteinander konkurrierende Arten von Sichten auf die Produktdaten gibt: die (primäre) Werkzeug-Sicht und die (sekundäre) Erfahrungs-Sicht. Da sowohl die Werkzeug-Sicht als auch die Erfahrungs-Sicht grundsätzlich Änderungsfunktionalität beinhalten, ergibt sich folglich das Problem der Konsistenzerhaltung zwischen den beiden Sichten (vgl. Abbildung 3). Dies lässt sich wie folgt charakterisieren:

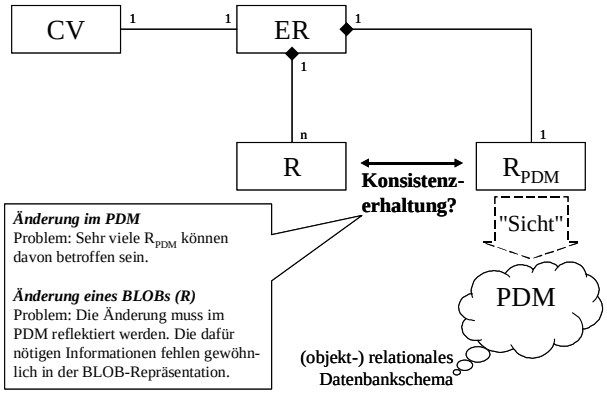


Abbildung 3: Sichten auf Produktdaten als Erfahrungsrepräsentation

- *Verlust des Objektcharakters*. Die konkrete bzw. aktuelle PDM-Repräsentation eines ER kann (ohne besondere Anforderungen an das PDM zu stellen) nur dynamisch, z. B. durch Auswertung eines SQL-Ausdrucks, ermittelt werden. Es ist also im allgemeinen Fall nicht möglich, eine derartige Repräsentation analog der LOB-Repräsentationen als Ganzes aus- oder einzuchecken.
- *Drohender Verlust der semantischen Datenintegrität*. Dies folgt unmittelbar aus dem vorhergehenden Punkt. Anwendungen, die die Werkzeug-Sicht verwenden, können Produktdaten in der EDB beliebig feingranular ändern und damit beliebig viele EDE zugleich (semantisch) ungültig machen. Dies wäre nur überprüfbar, wenn für jedes einzelne Produktdatum feststellbar wäre, welchen ER es zugeordnet ist. Das ist jedoch nicht praktikabel, wenn entsprechende PDM-Repräsentationen (wie bei Sichtenbildung üblich) nur deskriptiv definiert sind, denn bei jeder noch so geringen Änderung der Produktdaten müssten sämtliche PDM-Repräsentationen evaluiert werden, um mögliche Änderungen der EDE nachzuvollziehen.
- *Problematische Änderungssemantik der EDB-Sicht.* In der Erfahrungs-Sicht können die LOB-Repräsentationen vom Anwender editiert und die Änderungen anschließend wieder propagiert werden. Derartige Änderungen müssten nunmehr in der primären PDM-Repräsentation nachgeführt werden. Dies kann prinzipiell auf zwei Arten geschehen: automatisch oder manuell.

  Im ersten Fall wäre eine Prozedur erforderlich, die den (geänderten) LOB-Inhalt interpretiert und entsprechende Aktualisierungen der betroffenen Daten im Produktdatenschema veranlasst. Dabei kann es aber wiederum passieren, dass von den Aktualisierungen auf PDM-Seite noch andere EDE betroffen sind, mit den gleichen Konsequenzen, wie schon oben beschrieben. Der kritischste Punkt dieses Ansatzes ist allerdings, dass die LOB-Repräsentation alle Informationen enthalten muss, die für die eindeutige Zuordnung der enthaltenen Produktdaten zu Objekten im PDM-Teil der EDB erforderlich sind (z. B. DB-interne Objektidentifikatoren).

  Im anderen Fall müsste der Bearbeiter der LOB-Repräsentation (oder ein anderer Verantwortlicher) vom System darauf hingewiesen werden, dass er die vorgenommenen Änderungen manuell bei der PDM-Repräsentation nachziehen muss. Es ist natürlich denkbar, dass sich eine Änderung auf rein

formale Aspekte der sekundären Repräsentation beschränkt, die die primäre Repräsentation überhaupt nicht betreffen. In einem solchen Fall ist normalerweise keine Aktualisierung der PDM-Repräsentation notwendig. Das kritische Problem des manuellen Ansatzes bleibt dennoch bestehen: Ein EDB-Anwender, der eine LOB-Repräsentation editieren möchte, muss auch das dahinter stehende PDM kennen sowie gegebenenfalls auch die dafür verwendbaren Spezialwerkzeuge zur Verfügung haben und bedienen können.

4.2 Intensional orientierter Lösungsansatz für den allgemeinen Fall

Wir gehen bei unseren Lösungsansätzen prinzipiell davon aus, dass PDM-Repräsentationen beliebig deklarativ spezifiziert sein können, und zwar durch einen sog. *Repräsentationsdeskriptor* (kurz RD, i. Allg. eine Menge von SQL-Ausdrücken).

Der Repräsentationsdeskriptor (RD_{PDM}) wird nun als die *primäre* Repräsentation unterhalb des ER betrachtet (vgl. Abbildung 4). Dementsprechend muss ein CV für einen derartigen ER die *Intension* des Repräsentationsdeskriptors beschreiben. Dies hat den Vorteil, dass direkte Änderungen der Produktdaten über die Werkzeug-Sicht keine Relevanz für die Gültigkeit des CV bzw. des gesamten EDE haben, denn der RD_{PDM} bleibt – unabhängig von den aktuellen Produktdaten – stets gültig und evaluierbar. Nur bei einer Änderung des Repräsentationsdeskriptors selbst (oder des PDM) ist auch die Gültigkeit des CV (und seiner Beziehungen) zu überprüfen (vom Anwender). Von Nachteil ist natürlich, dass Erfahrungen nur sehr vage charakterisierbar sind und die Generierung der Repräsentationsdeskriptoren i. d. R. Expertenwissen erfordert.

Sekundäre (LOB-)Repräsentationen werden überhaupt nicht intern materialisiert. Es kann somit nicht zu Konsistenzproblemen zwischen primären und sekundären Repräsentationen kommen. Um den Anwendungen trotzdem sekundäre Repräsentationen als *Sichten auf EDE* anbieten zu können, müssen geeignete *Formatgeneratoren* eingesetzt werden. Ein Formatgenerator evaluiert den Repräsentationsdeskriptor und generiert aus dem gewonnenen Anfrageergebnis eine LOB-Repräsentation in dem von der Anwendung gewünschten Format.

Die Änderungssemantik der Erfahrungs-Sicht sollte sich auf die Möglichkeit, die Sichtendefinitionen (d. h., die Repräsentationsdeskriptoren) ändern zu können, beschränken. Falls erforderlich, können aber auch *Check-in-Filter* für sekundäre Repräsentationen eingesetzt werden. In diesem Fall können Anwendungen sekundäre LOB-Repräsentationen editieren und als Aktualisierung zurück an die EDB senden. Der Check-in-Filter interpretiert das LOB, um daraus entsprechende Aktualisierungsanweisungen für die Produktdaten zu

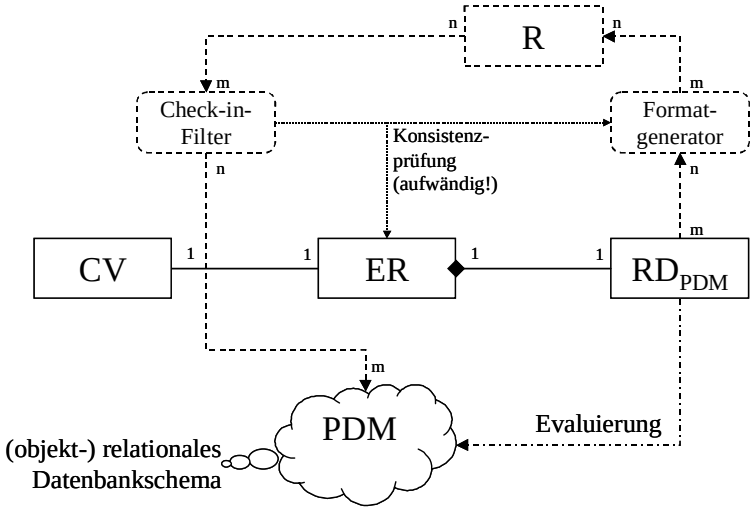


Abbildung 4: Konzeptionelles Schema für den intensional orientierten Lösungsansatz

generieren. Vor der Ausführung dieser Anweisungen müsste jedoch überprüft werden, ob das geänderte LOB mit dem Repräsentationsdeskriptor konsistent ist, d. h., nach erfolgter Produktdatenaktualisierung müsste der entsprechende Formatgenerator ein LOB erzeugen, das inhaltlich mit dem von der Anwendung gesendeten übereinstimmt. Für diese Überprüfung ist eine normalisierte interne Repräsentation erforderlich, die sowohl aus dem externen Format als auch aus dem Repräsentationsdeskriptor generierbar ist. Die Überprüfung ist aber auch dann erst *nach* der Änderung der Produktdaten möglich, d. h., der ganze Vorgang muss innerhalb einer langen Transaktion erfolgen, die am Ende zurückgesetzt werden müsste, falls der Konsistenzcheck scheitert. Ob derartige Check-In-Filter in einem gegebenen Anwendungsfall wirklich erforderlich sind, sollte daher gründlich geprüft werden. Wenn es sich nicht umgehen lässt, könnte natürlich auch der Konsistenzbegriff in geeigneter Weise abgeschwächt werden.

4.3 Extensional orientierter Lösungsansatz

Bei einem konkret gegebenen Produktdatenmodell könnte man prüfen, ob die *Extension* des Repräsentationsdeskriptors (kurz RD-Extension bzw. RDX) als primäre (PDM-)Repräsentation eventuell besser geeignet ist. In diesem Fall würden CVs (d. h., deren Inhalte) sich direkt auf die Produktdaten beziehen können. Um die dadurch entstehenden Konsistenzprobleme (vgl. Abschnitt 4.1) zu überwinden, muss es möglich sein, die aktuell gültige Zuordnung von Produktdaten zu RD-Extensionen, ausgehend von jedem einzelnen Produktdatum, „mit vertretbarem Aufwand“ zu bestimmen[3]. Dies wäre z. B. der Fall, wenn das PDM über Zugriffspfadstrukturen verfügt, die eine effiziente Berechnung dieser Zuordnung ermöglichen, oder solche Strukturen ohne wesentliche Beeinträchtigung der PDM-basierten Anwendungen hinzugefügt werden können (als „wesentliche Beeinträchtigung“ könnte man hierbei z. B. einen hohen Verwaltungsaufwand für die zusätzlichen Strukturen ansehen). Diese Vorraussetzungen sind am wahrscheinlichsten bei objektorientierten PDM gegeben, wenn es möglich ist, Erfahrung durch einzelne Objektausprägungen zu repräsentieren.

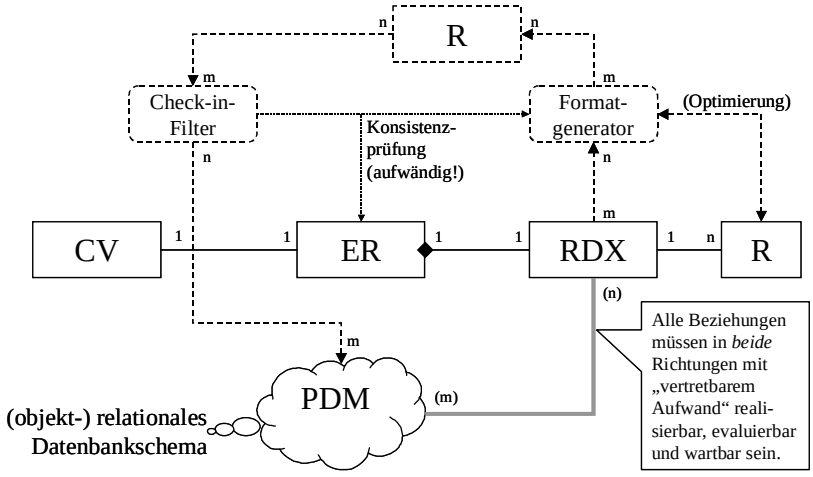


Abbildung 5: Konzeptionelles Schema für den extensional orientierten Lösungsansatz

In Bezug auf die sekundären Repräsentationen ändert sich grundsätzlich nichts gegenüber dem allgemeinen Lösungsansatz (vgl. Abschnitt 4.2). Es wäre jedoch nun möglich, aus Gründen der Optimie-

3. Die Evaluierung aller Repräsentationsdeskriptoren für eine solche Überprüfung betrachten wir als nicht vertretbaren Aufwand. Wird hierzu eine brauchbare Alternative geschaffen, dann könnte man jedoch nicht nur RD-Extensionen als Erfahrungsrepräsentation nutzen, sondern auch auf die RD selbst verzichten, da sie implizit (vor-)gegeben wären.

rung sekundäre Repräsentationen intern zu speichern, da festgestellt werden kann, wann eine solche interne Materialisierung durch die nachfolgende Aktualisierung eines Produktdatums ungültig wird. Nur dann wäre sie (spätestens bei der nächsten Anforderung durch eine Anwendung) per Formatgenerator neu zu erzeugen (natürlich aber auch im Falle der Aktualisierung des EDE über einen Check-in-Filter, falls diese Option realisiert wird).

5 Systemarchitektur für die extensional orientierte EDB-PDM-Integration

Für die Integration des Produktdatenmodells aus dem SFB-Teilprojekt D1 (s. Kapitel 6 und7) eignet sich der extensional orientierte Lösungsansatz, da zum einen der Objektcharakter im PDM gegeben ist und zum anderen relativ einfach eine Zuordnung von Produktdaten zu den Repräsentationen der EDB möglich ist. Vor einer detaillierten Betrachtung dieses Integrationsbeispiels stellen wir daher eine allgemeine logische Systemarchitektur für die extensional orientierte EDB-PDM-Integration vor (s. Abbildung 6).

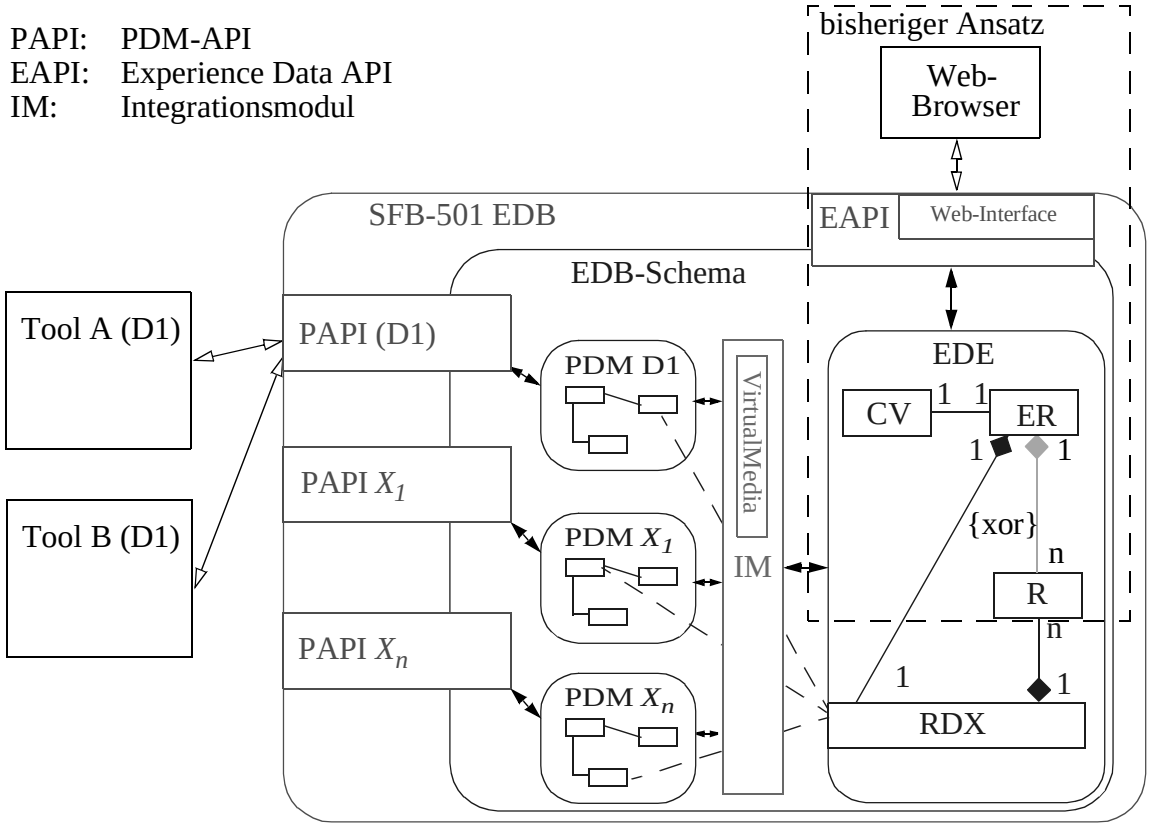


Abbildung 6: Logische Systemarchitektur der integrierten EDB

Der bisherige Ansatz der EDB sieht einen Zugriff (eines Web-Browsers) über das Web-Interface der EDB vor. Das Web-Interface ist Teil der *Experience Data API* (EAPI) der EDB, über die auf die Erfahrungsdateneinträge (EDE) zugegriffen werden kann. In diesem, im vorangehenden Kapitel 4 als Erfahrungs-Sicht bezeichneten Teil der EDB werden im Wesentlichen EDE verwaltet, die aus einem CV, einem ER und einer Menge von Repräsentationen bestehen. Die EDB-PDM-Integration bedeutet nun, dass daneben noch (beliebig viele) Produktdatenmodelle (PDM) verwaltet werden können. Auf

diese PDMs kann über je eine spezifische *PDM-API* (PAPI) direkt von externen Werkzeugen aus zugegriffen werden (Werkzeug-Sicht).

Anders als in der nicht-integrierten EDB werden dem ER die Repräsentationen nicht mehr direkt zugeordnet. Stattdessen besitzt der ER genau eine RDX, die maximal eine primäre Repräsentation und beliebig viele sekundäre Repräsentationen enthält. Eine primäre Repräsentation ist eine nicht-leere Menge von Produkten, die gemäß dem extensional orientierten Ansatz durch Beziehungen zwischen RDX und geeigneten PDM-Entities modelliert wird (gestrichelte Linien in Abbildung 6). Ein sog. Integrationsmodul (IM) überwacht alle über diese Beziehungen direkt oder indirekt miteinander verknüpften Objekte. Soll eines dieser Objekte geändert oder gelöscht werden, ist es Aufgabe des IM, Maßnahmen zur Erhaltung der semantischen Konsistenz einzuleiten (vgl. Kapitel 7). Das IM stellt außerdem die erforderlichen Formatgeneratoren (sowie gegebenenfalls auch Check-in-Filter) für die sekundären Repräsentationen zur Verfügung. Dies ist beispielsweise durch den Einsatz des VirtualMedia Frameworks [8, 9] für medienspezifische Datentypen realisierbar

6 Das Produktdatenmodell des SFB-Teilprojekts D1

Im Folgenden wird ein konkretes, repräsentatives Produktdatenmodell skizziert, um so in den Kapiteln 7 und 8 die Integration anhand eines Beispiels diskutieren zu können. Dafür dient uns das PDM des SFB-501-Teilprojekts D1.

In Abbildung 7 ist ein Ausschnitt des PDMs von D1 gegeben. Alle Objekte des PDMs sind Unterklassen von `Artifact` (die Instanzen der Klasse `Container` stellen lediglich den Inhalt der `Feature`-Objekte dar und sind keine eigenständigen Objekte). Im PDM ist die Klasse `Feature` noch in weitere Unterklassen untergliedert, wobei diese Unterklassen auch Beziehungen zueinander haben.

Im PDM von D1 ist eine einfache Form der Versionierung von Objekten der Klasse `Artifact` bzw. ihren Subklassen vorgesehen. Statt einer Änderung eines `Artifact`-Objekts wird eine neue Version des `Artifact`-Objekts geschrieben. Der Abstammungsgraph eines `Artifact`-Objekts kann die Gestalt eines gerichteten Graphen annehmen, d. h. von einer Version können mehrere neue Versionen abgeleitet werden, und mehrere Versionen können zu einer neuen Version zusammengeführt werden.

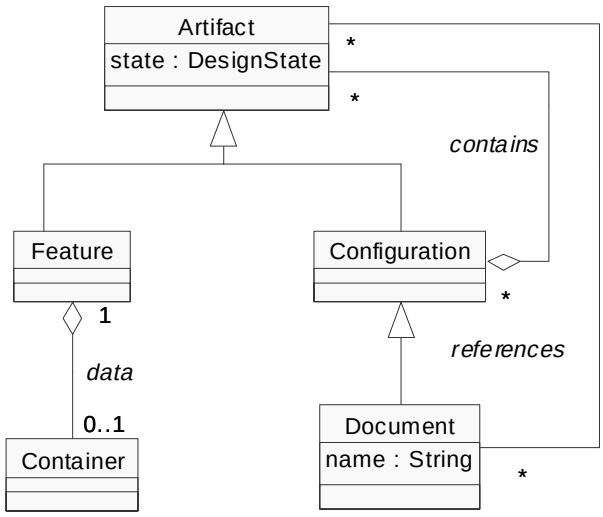


Abbildung 7: Ausschnitt des PDMs von D1

Alle Beziehungen einer `Artifact`-Version (inkl. Aggregations-/Kompositionsbeziehungen) übertragen sich nicht automatisch auf die neue Version, sondern müssen - sofern gewünscht - explizit für die neue Version neu angelegt werden. Bei der Nutzung dieses PDM ist also zwischen versionsfortschreibenden Attributwertände-

rungen und nicht-versionsfortschreibenden Beziehungsänderungen zu unterscheiden. Zur einfacheren Handhabbarkeit sind *VPropagate*-Operationen zu folgenden Zwecken vorgesehen: Beziehungen einer gegebenen `Artifact`-Version zu anderen Versionen können durch Beziehungen zu den jeweils aktuellsten Versionen der in Beziehung stehenden `Artifact`-Objekte ersetzt werden; bei der Ableitung einer neuen `Artifact`-Version können Beziehungen zu den jeweils aktuellsten Versionen aller mit ihrer Vorgängerversion in Beziehung stehenden `Artifact`-Objekte hergestellt werden. In den Fällen, in denen nicht automatisch die aktuellste Version eines Objektes ermittelt werden kann, muss der Benutzer diesen Status explizit zuweisen.

Neben der Versionierung ist noch ein weiterer Beziehungstyp zwischen den Versionen eines `Artifact`-Objekts vorgesehen: die Verfeinerung. Verfeinerungsbeziehungen zeigen qualitative Verbesserungen von Versionen in Bezug auf die Ziele des zugehörigen Entwurfsprozesses auf. Diese Beziehungen stehen orthogonal zur Versionierung und sind inbesondere bei der Integration einer entsprechenden Prozessunterstützung zu betrachten.

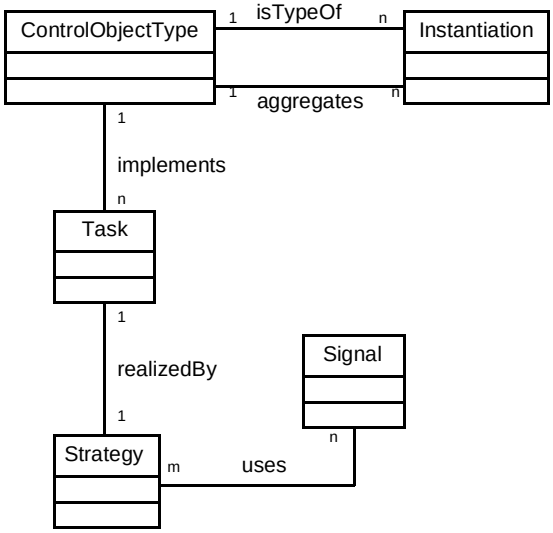


Abbildung 8: D1-Entwicklungprodukte

Abbildung 8 stellt einige der im D1-Prozess vorkommenden Entwicklungsprodukte miteinander in Beziehung. Ein `ControlObjectType` ist ein Objekttyp innerhalb des Kontrollsystems. Es besteht aus mehreren aggregierten Komponenten (`Instantiation`) und verwirklicht mehrere Aufgaben (`Task`). Wie eine Aufgabe gelöst wird, wird durch die zugehörige Strategie (`Strategy`) beschrieben. Zur Lösung der Aufgabe kann es auch gehören, dass Signale (`Signal`) für eine Kommunikation mit anderen `ControlObjectType`-Objekten konsumiert bzw. produziert werden. Dies wird dann durch Beziehungen vom Typ `uses` (zwischen `Strategy` und `Signal`) ausgedrückt.

Die in Abbildung 8 dargestellten Objekttypen stehen in einer Spezialisierungsrelation zu den in Abbildung 7 eingeführten. `ControlObjectType` ist abgeleitet von `Configuration`; die anderen vier spezialisieren den Objekttyp `Feature`. Weiterhin kann ein `ControlObjectType`-Objekt in zwei verschiedenen Darstellungen repräsentiert werden. Zunächst wird eine informelle HTML-Beschreibung erzeugt, die dann in einem SDL-Modell formalisiert wird. Diese beiden Darstellungsarten sind Spezialisierungen des Objekttyps `Document` aus Abbildung 7.

7 Möglichkeiten der integrierten Erfahrungs- und Produktdatenverwaltung am Beispiel des D1-PDMs

Wie bereits in Kapitel 5 erwähnt wurde, eignet sich für die Integration des Produktdatenmodells aus dem SFB-Teilprojekt D1 der extensional orientierte Lösungsansatz.

Die Werkzeug-Sicht und damit die Anbindung über die PAPI wurden bereits in Kapitel 5 skizziert. Die Anbindung des PDMs an die EDB (und damit die Erfahrungssicht) erfolgt, indem einzelne `Artifact`-Versionen zu Erfahrungen ‘angehoben’ werden können. Dafür ist diese `Artifact`-Version mit einem EDE - genauer einer RDX - zu verbinden. Die Auswahl der `Artifact`-Version sowie die Belegung des CVs erfolgt über die EAPI, wobei das IM entsprechende PDM-spezifische Funktionen zur Auswahl der Produktdaten (in unserem Fall einer `Artifact`-Version) anbieten muss. Es ist natürlich vorstellbar, dass aufgrund des gewählten Objekttyps einige Attribute des CVs automatisch gefüllt werden können. Sobald eine Beziehung zwischen RDX und der ausgewählten `Artifact`-Version hergestellt wurde, überwacht das IM die Abhängigkeiten.

Damit Produktdaten über die EAPI sichtbar gemacht werden können, müssen Formatgeneratoren bereitgestellt werden, die natürlich PDM-spezifisch arbeiten. In unserem Beispiel muss ein solcher Formatgenerator “wissen”, dass ausgehend von einer bestimmten `Artifact`-Version alle dieser `Artifact`-Version (bzgl. Aggregations-/Kompositionsbeziehungen) untergeordneten `Artifact`-Versionen bei der Erzeugung der Repräsentation berücksichtigt werden müssen. Hier ist also sehr wohl zwischen ‘einfachen’ Beziehungen (Assoziationen) und Aggregations-/Kompositionsbeziehungen zu unterscheiden. Beispiel einer Aggregationsbeziehung ist `contains` (siehe Abbildung 7), wohingegen `references` als einfache Beziehung zu betrachten ist.

Im Folgenden werden die Besonderheiten und Abhängigkeiten erläutert, die sich aus den verschiedenen Zugriffsmöglichkeiten auf die Produktdaten zum einen direkt über die PAPI und zum anderen über das Web-Interface (bzw. allgemein der EAPI) der EDB ergeben. Die Abhängigkeiten werden durch das IM gewartet.

7.1 Manipulation der Produktdaten über die PAPI

Nachdem bestimmte Produktdaten (siehe Kapitel 6) in der in Kapitel 5 beschriebenen Weise durch Assoziation mit RDX ‘als Erfahrungen ausgewiesen worden sind’, können diese Daten natürlich weiterhin über die PAPI manipuliert werden. Einige der an der PAPI zur Verfügung stehenden Operationen haben Auswirkungen auf die Erfahrungsdatenverwaltung, die in diesem Abschnitt diskutiert werden sollen.

Dieser Diskussion ist zunächst folgendes vorauszuschicken. Wie bereits oben angedeutet, ist zwar ein RDX mit genau einer `Artifact`-Version assoziiert, die eigentliche Extension des zugehörigen Erfahrungsdateneintrags ergibt sich jedoch aus der von dieser Version ausgehenden Aggregations-/Kompositionshierarchie, die daher auch vollständig von den Formatgeneratoren berücksichtigt wird. Aus diesem Grund können nicht nur Manipulationen der mit einem RDX assoziierten `Artifact`-Version sondern auch der (direkten oder transitiven) ‘Teile’ dieser Versionen Auswirkungen auf die Erfahrungsdatenverwaltung haben. Bei solchen Manipulationen handelt es sich um nicht-versionsfortschreibende Änderungen sowie um Löschungen. Soll also eine `Artifact`-Version (nicht-versionsfortschreibend) geändert bzw. gelöscht werden ist folglich zunächst festzustellen, ob diese Version direkt mit einem RDX verbunden ist oder ein direktes oder transitives Teil einer mit einem RDX verbundenen Artifact-Version ist. Ist dies der Fall, sind entsprechende Zusatzmaßnahmen durchzuführen, die im Folgenden diskutiert werden sollen.

Eine **nicht-versionsfortschreibende Änderung** einer `Artifact`-Version liegt genau dann vor, wenn eine Aggregations-/Kompositionsbeziehung hergestellt bzw. aufgelöst werden soll. Damit ändert sich die Extension des zugehörigen Erfahrungsdateneintrags, da hier die gesamte Aggregations-/Kompositionshierarchie berücksichtigt werden muss, wie oben beschrieben wurde. Mit einer solchen Änderung können der zugehörige CV sowie alle früher durch Anwendung von Formatgeneratoren angelegten Repräsentationen invalidiert werden, da die geforderte semantische Äquivalenz nicht mehr sichergestellt werden kann.

Es bestehen nun grundsätzlich zwei Möglichkeiten, auf die Invalidierung der Repräsentationen zu reagieren. Einerseits könnte man mit einer solchen Änderung die Verbindung der (Wurzel-) Version mit dem RDX auflösen, was jedoch unserer eigentlichen Intension der Integration widerspricht. Die andere, aus unserer Sicht angemessenere Lösung besteht darin, alle Repräsentationen zu löschen. Dies sollte einhergehen mit entweder einer Benachrichtigung des entsprechenden Benutzers, so dass dieser durch expliziten Aufruf von Formatgeneratoren aktualisierte Repräsentationen anlegen kann, oder mit einem automatischen Aufruf der Formatgeneratoren. Wir halten i. Allg. die erstgenannte Möglichkeit (Benutzernotifikation) für angemessener, da der Benutzer so die Auswirkungen seiner Manipulationsoperation besser einschätzen kann. Im Falle von häufigen Änderungen geringer Tragweite kann jedoch auch die automatische Propagierung als Option angeboten werden. Hierbei sollte der von VirtualMedia unterstützte asynchrone Operationsmodus (Entkopplung von Aufruf und Ausführung) genutzt werden, um unnötig das System belastende Formatgenerierungsläufe zu vermeiden (z. B., wenn zwischen zwei Änderungen kein Lesezugriff erfolgt).

Die Invalidierung des CVs kann in keinem Fall mit automatischen Mechanismen behandelt werden, da die Beschreibung immer durch einen Benutzer vorgenommen werden muss. Mit der Invalidierung muss der gesamte EDE der allgemeinen Sichtbarkeit entzogen werden und der zuständige Benutzer muss benachrichtigt werden und zur Anpassung (und der daraus resultierenden Wiederfreigabe) des CVs aufgefordert werden. In Ausnahmefällen kann (in Abhängigkeit von der speziellen Applikationssemantik) auch so verfahren werden, dass vorab vermerkt wird, dass nicht-versionsfortschreibende Änderungen von `Artifact`-Version den CV nicht invalidieren können und demnach eine entsprechende Behandlung nicht vom System veranlasst werden muss.

Hinsichtlich des **Löschens** einer `Artifact`-Version ist zunächst zu unterscheiden, ob es sich um die Wurzel (d. h. die mit einem RDX verbundene `Artifact`-Version) oder eine indirekt zu einer Erfahrungsdatenextension gehörende `Artifact`-Version handelt. Im letzteren Fall kann genauso reagiert werden, wie für den Fall der nicht-versionsfortschreibenden Änderung beschrieben (siehe oben). Im ersteren Fall wird eine (zusätzliche) Spezialbehandlung notwendig, da der Handle der gesamten Erfahrungsdatenextension gelöscht werden soll. Darauf kann einerseits mit einer automatischen Löschung des gesamten Erfahrungsdateneintrags reagiert werden, oder andererseits ein Eingreifen eines zuständigen Benutzers durch entsprechende Notifikation herbeigeführt werden.

7.2 Manipulation von Erfahrungsdaten über die EAPI

Der bisherige (vor der PDM-Integration) verfügbare EDB-Prototyp stellt eine Reihe von Funktionen zur Erfahrungsdatenverwaltung zur Verfügung, wie in [2, 3] beschrieben. Diese Funktionen berück-

sichtigen jedoch noch nicht die Betrachtung von Produktdaten als Erfahrungen, d. h. die Einbindung von Produktdaten in EDEs über RDXs (vgl. Abbildung 6) und die (automatische) Ableitung weiterer Repräsentationen durch Formatgeneratoren. In diesem Abschnitt wollen wir kurz diskutieren, inwieweit sich die im bisherigen Ansatz realisierten Operationen auch auf Produktdaten-basierte EDEs (kurz PEDE; siehe Abbildung 9, die den entsprechenden Auszug aus Abbildung 6 darstellt) anwenden lassen bzw. welche Zusatzmaßnahmen erforderlich werden.

Grundsätzlich gilt, dass Produktdaten nur explizit über die zu diesem Zweck zur Verfügung gestellte PAPI manipuliert werden können. Auch nach Anlegen eines PEDE, d. h. nach der Aufwertung von (bestimmten) Produktdaten in den Status ‘Erfahrung’, sollen diese Produktdaten nur über die PAPI und nicht über die EAPI geändert werden können. Die Auswirkungen solcher Änderungen wurden im vorangegangenen Abschnitt diskutiert.

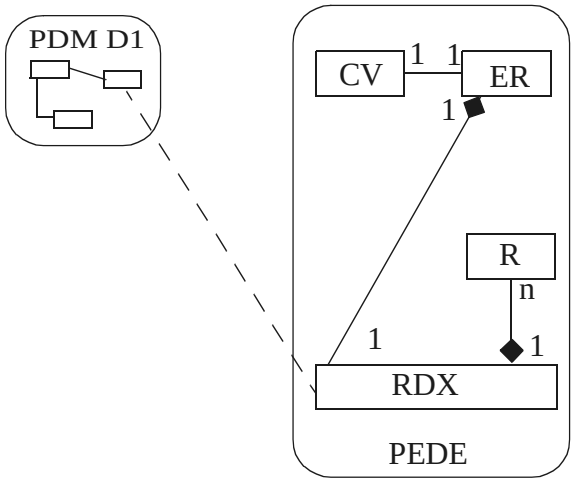


Abbildung 9: Aufbau eines Produktdaten-basierten Erfahrungsdateneintrags (PEDE)

Aufgrund der Abhängigkeit zwischen einem PEDE und den zugehörigen Produktdaten (siehe gestrichelt dargestellte Beziehung in Abbildung 9), kann sich jedoch die Wirkung von an der EAPI angebotenen Manipulationsoperationen in Abhängigkeit davon, ob sie auf einen PEDE oder einen nicht-Produktdaten-basierten EDE angewendet werden, unterscheiden. Die (EAPI-) Operationen, bei denen eine solche Unterscheidung zu treffen ist, sollen im Folgenden angesprochen werden.

Die bisher zur Verfügung stehenden Operation zur Manipulation von nicht-Produktdaten-basierten EDE umfassen solche zur Änderung einzelner bzw. aller Repräsentationen eines EDE. Dabei wird, soweit dies nicht automatisch überprüft werden kann, der Benutzer für die Einhaltung der semantischen Äquivalenz zwischen den verschiedenen Repräsentationen eines EDE verantwortlich gemacht. Wie in diesem Bericht vorgeschlagen, sollen die Repräsentationen eines PEDE jedoch in erster Linie automatisch, d. h. durch Anwendung von Formatgeneratoren erzeugt werden. Dennoch wollen wir es dem Benutzer erlauben, auch hier eigenständig Repräsentationen hinzuzufügen; er ist jedoch in diesem Falle auch für die Gewährleistung der semantischen Äquivalenz verantwortlich zu machen. Unter Berücksichtigung dieser Aspekte ist der Fall, dass ein Benutzer **Repräsentationen eines PEDE ändern** will, folgendermaßen zu behandeln. Möchte er eine Repräsentation ändern, die nicht durch einen Formatgenerator automatisch erzeugt worden ist, so ist diese Änderung erlaubt. Möchte er hingegen eine durch einen Formatgenerator erzeugte Repräsentation ändern, so wird ihm mitgeteilt, dass eine Änderung dieser Repräsentationen nur durch Änderung der zugrundeliegenden Produktdaten (über die PAPI) und nachfolgende Propagierung dieser (Produktdaten-)Änderungen durch Neuberechnung mit Hilfe eines der Formatgeneratoren geschehen kann. Das heißt, auf die in Kapitel 4 skizzierten und bereits dort als problematisch charakterisierten Check-in-Filter wird bewusst verzichtet.

Eine **Löschung einzelner bzw. aller Repräsentationen** eines PEDE ist dagegen erlaubt, da diese nur in solchen Zeiträumen explizit materialisiert gehalten werden müssen, in denen sie auch wirklich von der Anwendung benötigt werden. Entsprechend können auch die Formatgeneratoren zur **Erzeugung von PEDE-Repräsentationen** jederzeit explizit über die EAPI aufgerufen werden.

Die vorangegangene Beschreibung macht deutlich, dass wir keine vollkommene Transparenz hinsichtlich der Nutzung der beiden EDB-Schnittstellen herstellen. Dies wird dem Benutzer beispielsweise dadurch deutlich, dass er eine durch einen Formatgenerator erzeugte Repräsentation eines EDE nicht gleichermaßen bearbeiten kann wie eine selbst explizit eingefügte. Diese Ungleichbehandlung muten wir dem Benutzer jedoch aus folgenden Gründen bewusst zu. Produktdaten sollten lediglich im Kontext eines Entwicklungsprozesses (Nutzung der PAPI) verändert werden dürfen, d. h. in einem Umfeld, in dem die Projektziele bekannt sind. Die Nutzung von Erfahrungsdaten ist jedoch vollkommen losgelöst von den speziellen Projekten, in denen diese Erfahrungen ursprünglich als Produktdaten erzeugt wurden. Sie verlangt vielmehr einen projektunabhängigen Überblick über die Erfahrungsdaten, um ein neues Projekt zu unterstützen. Aus diesem Grund unterstützen wir keine Mechanismen der automatischen Propagierung von über die EAPI initiierten Repräsentationsänderungen auf die Ebene der Produktdaten. Diese Propagierung wäre sowieso nur in Ausnahmefällen realisierbar (vgl. Beschreibung der Check-in-Filter in Abschnitt 4.2).

8 Beispielhafter Prozess

Der im Folgenden beschriebene beispielhafte Prozess ist an die im SFB durchgeführte Fallstudie CS3/2 [12] angelehnt. Hierbei handelte es sich um eine Iteration der vorangegangenen Fallstudie CS3/1 [5]. In dieser ersten Fallstudie wurde als Baseline für weitere Fallstudien eine Lichtsteuerung des Flurs 32/4 der Universität Kaiserslautern entwickelt. Die Problembeschreibung von CS3/1 wurde für die anschließende Iteration CS3/2 um eine Heizungssteuerung erweitert. Ziel von CS3/2 war es, das erweiterte System (Licht und Heizung) unter möglichst umfassender Wiederverwendung der Entwurfsobjekte von CS3/1 zu entwickeln.

Die wiederzuverwendenden Entwurfsobjekte des Baselining wurden nach Projektabschluss in den experimentübergreifenden Teil der EDB eingebracht und als Erfahrung ausgewiesen. Dies galt insbesondere für alle in CS3/1 entwickelten Sensoren und Aktuatoren (im Weiteren mit S/A abgekürzt). Nach ‘Anhebung’ dieser Entwurfsobjekte in den Zustand ‘Erfahrung’ standen sie allen weiteren Projekten - und damit auch der Iteration CS3/2 - zur Verfügung.

S/A sind für die Entwicklung eingebetteter Systeme von großer Bedeutung, da sie die Schnittstelle zur Umgebung darstellen. Darüber hinaus eignen sie sich in hohem Maße für die Wiederverwendung. Neu entworfene S/A sollten so schnell wie möglich anderen Projekten zur Verfügung gestellt werden, wo sie direkt übernommen und modifiziert werden können. Dieses Vorgehen wurde einerseits bei der Definition des Entwurfsprozesses von CS3/2 berücksichtigt und andererseits in der Projektdurchführung auch ausgeführt. Soweit wie möglich wurden die S/A für den Lichtsteuerungsteil von CS3/2 direkt übernommen. Zusätzlich dienten einige S/A als Grundlage der Entwicklung neuer, noch nicht in der EDB enthaltener S/A der Heizungsregelung. Wiederverwendete (d. h. aus der EDB ausgelesene) und neu entwickelte S/A waren dann Teil des weiteren Systementwurfs von CS3/2.

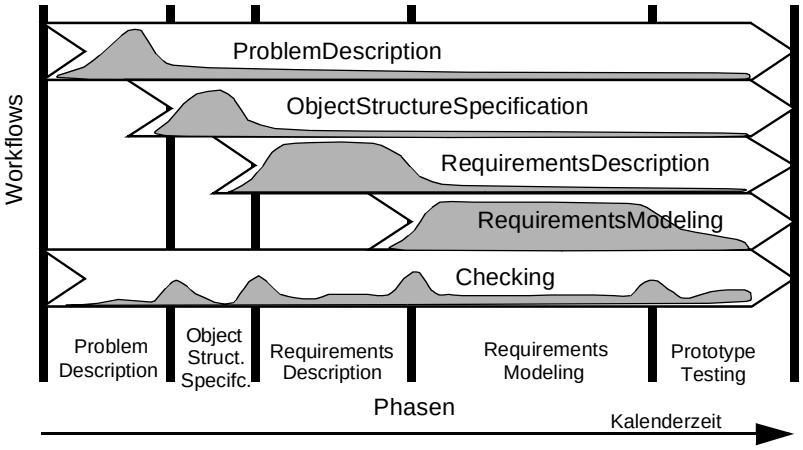


Abbildung 10: Workflows und Phasen des Analyseprozesses
Die grauen Flächen zeigen die Aktivitäten der Workflows als Funktion der Zeit.

Die Wiederverwendung von S/A beschränkt sich nicht nur auf die beiden erwähnten Fallstudien CS3/1 und CS3/2. Jedes weitere Entwicklungsprojekt, das möglicherweise zeitgleich zu CS3/2 läuft, ist davon betroffen. Beispielsweise könnte ein drittes Projekt einen der in CS3/2 entwickelten S/A benötigen, bevor CS3/2 abgeschlossen ist. Es ist daher sinnvoll, die in CS3/2 entwickelten S/A bereits während der Projektlaufzeit in der Status 'Erfahrung' zu erheben und sie damit anderen Projekten zur Verfügung zu stellen. Voraussetzung hierfür ist allerdings ein flexibler Entwurfsprozess, wie er bei CS3/2 gegeben war. Der Prozess muss es erlauben, einige Objekte des zu entwerfenden Systems (hier die S/A) vorab sehr weit zu entwickeln und als Erfahrung weiterzugeben, während sich der restliche Entwurf noch in einer relativ frühen Phase befindet.

Der in CS3/2 zugrunde liegende SE-Prozess lehnte sich an den in [7] beschriebenen „Unified Software Development Process" an. Hierbei wird der Entwicklungsprozess eines Systems in Workflows (Zusammenfassung zusammengehöriger Aktivitäten) unterteilt. Im Gegensatz zu anderen SE-Prozessen wird die Systementwicklung durch die Workflows nicht in aufeinanderfolgende Phasen partitioniert; Workflows sind vielmehr gleichzeitig aktiv, was für die Bereitstellung von Entwurfsobjekten zur Wiederverwendung aus laufenden Projekten ausgenutzt werden kann. Die entsprechenden managementorientierten Entwicklungsphasen können den Zeiten, zu denen die entsprechenden Workflows vorwiegend aktiv sind, zugeordnet werden. Der zeitliche Ablauf der Workflows der Anforderungsanalyse ist in Abbildung 10 dargestellt.

Durch die Flexibilität des Entwicklungsprozesses konnten in der Fallstudie CS3/2 einige Objekte - in unserem Fall die Sensoren und Aktuatoren - bereits sehr weit entworfen werden, während sich der eigentliche Systementwurf noch in den ersten Entwicklungsphasen befand. Zu jedem Zeitpunkt konnten Entwurfsdokumente beliebig neu angelegt, verfeinert und verändert (versioniert) werden. Der Zustand eines Entwurfsobjekts ist durch die Konfiguration aus beliebigen Dokumenten beschrieben. Die EDB-Schnittstelle zum Produktmodell (PAPI) muss folgende vier Aktivitätsklassen auf allen Dokumenten des PDM unterstützen:

- Erzeugung (`create`, `create alternative`)
- Verfeinerung (`refine`, `extend`)
- Verbesserung (`change`, `correct`)
- Überprüfung (`review`, `test`).

Die Aktivitäten der Klasse *Erzeugung* legen eine initiale Version eines neuen Entwurfsobjekts an. Besonders erwähnenswert ist, dass neben den hier angeführten Operationen auch die Wiederverwendungsinfrastruktur der EDB zur Erzeugung von Entwurfsobjekten genutzt werden kann. Soll ein neues Objekt durch Wiederverwendung existierender Entwurfsdaten erzeugt werden, so wird der gesamte, durch die EDB unterstützte [2], Wiederverwendungsprozess (beginnend mit der Suche nach wiederzuverwendenden Daten über die EAPI) angestoßen. Aktivitäten der Klassen *Verfeinerung* und *Verbesserung* führen zu neuen Objektversionen, wobei die Verfeinerungsaktivitäten zu einer Weiterentwicklung der Entwurfsdaten führen, während Verbesserungsaktivitäten als Reaktionen auf in einer *Überprüfung*saktivität erkannte Fehler oder Änderungen in anderen Dokumenten angesehen werden können. Eine detailliertere Beschreibung der aufgelisteten Aktivitäten befindet sich in [12].

Die Anwendung dieser Aktivitäten führt zu einem Versionsbaum für alle Instanzen der Klassen des in Kapitel 6 beschriebenen Produktmodells. Ein- und Ausgaben aller Aktivitäten sind jeweils mehr als eine Dokumentversion, da bei komplexeren Objekten deren Aggregationshierarchie betrachtet werden muss. Als Beispiel sei die Aktivität `extend` für einen Temperatursensor im Folgenden beschrieben.

Die typische Struktur eines allgemeinen, bustauglichen Sensors ist in Abbildung 11 skizziert. Ein solcher Sensor aggregiert die eigentliche analoge Messhardware (bei einem Temperatursensor z. B. ein temperaturabhängiger Widerstand mit entsprechender Beschaltung), einen A/D-Wandler und eine Busschnittstelle. Die Funktionalität des Sensor-Controllers (Klasse `ControlObjectType`, vgl. Abbildung 8) enthält Funktionen wie etwa die Normierung eines digitalen Spannungswerts auf eine Temperaturskala, Fehlerbehandlungsroutinen oder Filter.

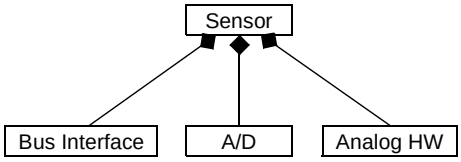


Abbildung 11: Aufbau eines bustauglichen Sensors

Ein solcher Temperatursensor wird in diesem Beispiel um die Aufgabe der Fehlererkennung und -weitergabe erweitert. Der aktuelle Entwurfsschritt ist die Erweiterung des entsprechenden Sensorobjekts der Klasse `ControlObjectType` um die entsprechende Aufgabe „error detection and propagation“ (Klasse `Task`) und eine geeignete Lösungsstrategie (Klasse `Strategy`). Die Strategie könnte lauten „generate a signal ‘newMalFct’ if the output value of the D/A converter leaves the specified temperature interval”. Im Fehlerfall wird demnach ein Signal (Klasse `Signal`) erzeugt, das entsprechend referenziert wird. `Task` und `Strategy` werden als informelle HTML-Beschreibungen erzeugt. Zur Bearbeitung der beschriebenen Entwurfsaufgabe benötigt der Entwickler u. a. die HTML-Beschreibungen des A/D-Wandlers und der Analoghardware (neben der bisher vorliegenden HTML-Beschreibung des Sensors selbst). Diese müssen dem Entwickler aus der EDB zur Verfügung gestellt werden.

Ist ein solcher Temperatursensor in mehreren Entwurfsschritten soweit entworfen und getestet, dass er auch von anderen Projekten übernommen werden kann, so müssen die entsprechenden Daten in der EDB in den Zustand ‘Erfahrung’ angehoben und den anderen Projekten über den projektübergreifenden Teil der EDB zur Verfügung gestellt werden. Hierzu wird für den betrachteten Temperatursensor

ein neuer Erfahrungsdateneintrag (EDE, vgl. Abbildung 6) angelegt (die Verknüpfung wird intern durch eine IM-überwachte Beziehung zwischen dem RDX-Objekt des EDE und dem Wurzelobjekt des Temperatursensors realisiert).

Mit diesem neuen EDE steht nun der Temperatursensor allen Projekten des SFB zur Wiederverwendung zur Verfügung, während das aktuelle Entwicklungsprojekt (Fallstudie CS3/2) noch aktiv ist und andere Entwurfsobjekte des Kontrollsystems bearbeitet. Zur Erzeugung einer 'Erfahrungsrepräsentation' des Temperatursensors wird die EAPI unter Angabe des gewünschten externen Formats, z. B. HTML, aufgerufen. Zu diesem Zweck muss im System mindestens ein Formatgenerator existieren, der das entsprechende PDM 'kennt' und folglich 'weiß', wie dort ein Temperatursensor modelliert ist. In der Regel wird man hierfür einen speziell angepassten, also PDM-spezifischen Generator bereitstellen[4]. Im Idealfall stellt dieser PDM-spezifische Generator die Repräsentationen in einem Universalformat wie beispielsweise XML zur Verfügung, welches dann mit Standardfiltern in das vom Benutzer gewünschte Format umgewandelt wird ('Universal Media Server', vgl. [9]).

Wenn wir nun annehmen, dass im weiteren Entwurfsverlauf der in CS3/2 entwickelte und zur Wiederverwendung 'freigegebene' Temperatursensor geändert werden muss, so hat das Einfluss auf das aktuelle Projekt, aber auch auf die über die EDB als Erfahrung weitergegebenen Daten. Eine Änderung des Sensors kann dann notwendig werden, wenn beispielsweise der am A/D-Ausgang zur Verfügung stehende Messwert kein ausreichend lineares Verhalten aufweist. Ist der genaue Zusammenhang zwischen Messwert und Temperatur ermittelt, kann das Verhalten des Sensors entsprechend angepasst werden. Hierzu muss zunächst die HTML-Beschreibung und anschließend die formale SDL-Modellierung und die Implementierung (falls eine solche bereits vorhanden ist) geändert und über die PAPI in der EDB aktualisiert werden. Wir erhalten dadurch neue Versionen der entsprechenden Entwurfsdokumente, die die bisherigen Versionen ersetzen.

Bezüglich der in der EDB abgelegten Erfahrungsdaten hat eine solche Änderung folgende Auswirkung: Das IM registriert eine nicht-versionsfortschreibende Änderung eines mit einem EDE verknüpften Artefakts. Daraufhin invalidiert es alle evtl. vorhandenen, durch Formatgeneratoren erzeugten Repräsentationen für diesen (Temperatursensor-) EDE. Ferner werden der EDE für 'normale' EDB-Benutzer unsichtbar gemacht und eine entsprechende Nachricht an den für den betroffenen EDB-Bereich zuständigen EDB-Manager gesendet. Dieser kann nach Prüfung und eventueller Änderung des EDE diesen wieder für die 'Allgemeinheit' freigeben.

9 Zusammenfassung und Ausblick

Durch den in diesem Bericht vorgeschlagenen, auf die SFB-501-EDB zugeschnittenen Ansatz der Integration von Erfahrungs- und Produktdatenverwaltung konnten folgende Ziele erreicht werden:

- Produktdaten können bereits während der Laufzeit des zugehörigen Projektes in den Status 'Erfah-

4. Theoretisch wäre auch ein generischer Formatgenerator vorstellbar, der (im Prinzip) jedes beliebige PDM als Parameter akzeptiert. Um damit akzeptable Resultate zu erzielen, müssten allerdings die PDMs in einer weit über den normalen Systemkatalog hinausgehenden Weise in der EDB beschrieben sein, etwa mit Hilfe semantisch angereicherter Beziehungen [15].

rung' versetzt werden und damit zur Wiederverwendung bereitgestellt werden.

- Produktdaten können aufgrund der integrierten Verwaltung mit Erfahrungsdaten in Beziehung gesetzt werden, was sowohl ihre Nutzung im Projekt als auch die anschließende Auswertung bzgl. ihres Wiederverwendungspotentials (QIP-Phasen *Analyzing* und *Packaging*) erheblich vereinfacht.
- Zur Bearbeitung von Produktdaten wird eine zugeschnittene API zur Verfügung gestellt (PAPI), über die einerseits Entwurfswerkzeuge auf die Produktdaten zugreifen können, die aber auch Ad-hoc-Zugriffe durch den Benutzer unterstützt.
- Auch auf Produktdaten basierende Erfahrungsdateneinträge können über die speziell zum Erfahrungsdatenzugriff bereitgestellte API (EAPI) bearbeitet werden. Hier wird dem Benutzer allerdings zugemutet, dass im Falle von Produktdaten-basierten Erfahrungen keine verändernden Zugriffe über die EAPI möglich sind.
- Alle Produktdatenänderungen sind demnach über die PAPI vorzunehmen. Zur Propagierung von Produktdatenänderungen, die die Grundlage von Erfahrungsdateneinträgen bilden, sehen wir einen praktikablen, extensional orientierten Lösungsansatz vor.

Als repräsentatives Produktdatenmodell wurde eine stark vereinfachte Version des Produktdatenmodells des SFB-Teilprojekts D1 untersucht. Die prinzipielle, in diesem Bericht beschriebene Vorgehensweise bei der Integration ist zunächst unabhängig von dem speziellen Produktdatenmodell. Daher ist es noch immer eine offene Frage, ob ein SFB-weites, globales Produktdatenmodell in der EDB berücksichtigt werden soll, oder jedem Teilprojekt seine spezifischen Produktdatenstrukturen zur Verfügung gestellt werden sollten. Wir werden in sich anschließenden Arbeiten versuchen, diese Frage zu beantworten. Es besteht jedoch bereits eine leichte Tendenz zur zweiten Lösung, da sie mehr Praktikabilität und eine höhere Akzeptanz verspricht. In diesem Zusammenhang ist auch die Gestaltung der PAPI(s) noch näher zu analysieren. Auch hier könnte es eine globale Schnittstelle oder mehrere Teilprojekt-spezifische Schnittstellen geben, wobei ersteres wenig sinnvoll in Verbindung mit mehreren PDMs erscheint, letzteres jedoch ohne weiteres auch für ein globales PDM Sinn macht. Grundsätzlich stehen zur Realisierung der PAPI(s) alle Möglichkeiten vom Rückgriff auf die Standardschnittstelle des ORDBMS (SQL) bis hin zu einem 'fetten' Applikationsserver offen.

Weitere zukünftige Arbeiten werden die Integration der (Produkt- und Erfahrungs-)Datenverwaltung mit der Ablaufsteuerung MILOS [11] umfassen. Damit wird das Ziel eines umfassenden SFB-501-Repositories angestrebt. Nachdem diese Integrationsfragen dann auf konzeptioneller Ebene gelöst sein werden, soll anschließend der Schwerpunkt auf der Erarbeitung eines geeigneten Architektur- und Realisierungsansatzes gelegt werden. Hier wird es insbesondere darum gehen, wie objekt-relationale Datenbanktechnologie noch effektiver als bisher genutzt werden kann, und ob ein zentralisiertes oder ein verteiltes, d. h., durch Kopplung von bestehenden Produktdatenverwaltungskomponenten mit der Erfahrungsdatenverwaltungskomponente entstehendes, System angemessener ist.

10 Literatur

[1] Basili, V. R., Rombach, H. D.: Support for comprehensive reuse. In *IEE Software Engineering Journal*, 6(5):303–316, September 1991.

[2] Feldmann, R. L., Geppert, B., Mahnke, W., Ritter, N., Rößler, F.: An ORDBMS-based Reuse Repository Supporting the Quality Improvement Paradigm - Exemplified by the SDL-Pattern Approach, in: Proc. TOOLS USA 2000, 34th International Conference & Exhibition, Santa Barbara, CA, July, 30 - August, 3, 2000.

[3] Feldmann, R. L., Geppert, B., Mahnke, W., Ritter, N., Rößler, F.: The ORDBMS-based SFB 501 Experience Base - Exemplified by the SDL-Pattern Approach, SFB-Report 08/99, SFB 501, University of Kaiserslautern, 1999.

[4] Feldmann, R. L., Mahnke, W., Ritter, N.: (OR)DBMS-Support for the SFB 501 Experience Base, Technical Report SFB 501-12/98, Fachbereich Informatik, Universität Kaiserslautern, 1998.

[5] Feldmann, R. L., Münch, J., Queins, S., Vorwieger, S., Zimmermann, G.: Baselining a Domain-Specific Software Development Process. SFB 501 Technical Report No. 02/99, University of Kaiserslautern, 1999

[6] Härder, T., Sauter, G., Thomas, J.: The Intrinsic Problems of Structural Heterogeneity and an Approach to their Solution, in: The VLDB Journal 8:1, 1999, pp. 25-43.

[7] Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process, Addison-Wesley, 1999.

[8] Marder, U.: Medienspezifische Datentypen für objekt-relationale DBMS: Abstraktionen und Konzepte, in: Tagungsband der GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW'99), A. Buchmann (Hrsg.), Informatik aktuell, Freiburg, März 1999, Springer-Verlag, S. 210-231.

[9] Marder, U.: Towards a Universal Media Server, SFB-Report 03/2000, SFB 501, University of Kaiserslautern, Feb. 2000

[10] Mili, A., Mili, R., Mittermeir, R. T.: A survey of software reuse libraries. In: *Annals of Software Engineering* 5 (1998), 349-141.

[11] Maurer, F., Dellen, B., Bendeck, F., Goldmann, S., Holz, H., Kötting, B., Schaaf, M.: Merging project planning and web-enabled dynamic workflow for software development, IEEE Internet Computing, Special Issue on Internet-Based Workflow - May/June 2000.

[12] Queins, S., Zimmermann, G.: A First Iteration of a Reuse-Driven, Domain-Specific System Requirements Analysis Process. SFB 501 Technical Report No. 13/99, University of Kaiserslautern, 1999

[13] Ritter, N., Steiert, H.-P., Mahnke, W., Feldmann, R. L.: An Object-Relational SE-Repository with Generated Services, in: Proc. Managing Information Technology Resources in Organizations in the Next Millenium (Computer-Aided Software Engineering Track of IRMA’99), IDEA Group Publ., May 1999, pp. 986-990.

[14] SFB-501-EDB, http://donau.informatik.uni-kl.de:18070/cgi-bin/cgiwrap/expfact/webdriver.

[15] Zhang, N., Härder, T.: On a Buzzword "Extensibility" - What we have learned from the ORIENT Project, in: Proc. Int. Database Engineering and Applications Symposium (IDEAS'99), Montreal, Canada, Aug. 1999, pp. 360-369.