Produktdaten-Verwaltung in heterogenen Workflow-Umgebungen

Interner Bericht

Markus Bon, Theo Härder, Norbert Ritter

Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
{bon | haerder | ritter}@informatik.uni-kl.de

Zusammenfassung: In heutigen Unternehmen finden wir immer häufiger die Situation vor, dass verschiedene organisatorische Einheiten unterschiedliche, heterogene Workflow-Management-Systeme (WfMS) nutzen. Organisationsübergreifende Abläufe sind somit durch mehrere unabhängige, heterogene Workflows zu kontrollieren und abzuwickeln. Nicht selten müssen dabei Abhängigkeiten zwischen unterschiedlichen Abläufen von Hand gewartet werden, obwohl Wissen über die sich daraus ergebenden globalen Zusammenhänge vorliegt. Daher liegt der Schwerpunkt eines unserer Projekte, das in Zusammenarbeit mit einem Industriepartner durchgeführt wird, auf der Interoperabilität heterogener WfMS. In diesem Beitrag stellen wir zum einen allgemein vor, welche Unterstützung unser Ansatz zum Einbinden heterogener Workflows bietet, zum anderen gehen wir detailliert auf das Problem ein, wie zwischen verschiedenen, möglicherweise heterogenen Workflow-Umgebungen Produktdaten automatisiert ausgetauscht werden können. Bisher lag die Datenversorgung von Aktivitäten/Applikationen, beispielsweise mit Produktdaten, nicht oder nur sehr eingeschränkt im Aufgabengebiet eines WfMS. Sie wurde vielmehr von Experten übernommen, die manuell dafür sorgten, dass den ausgeführten Applikationen rechtzeitig die zu ihrer Ausführung benötigten Daten vorlagen. In einem „Multi-WfMS“ sind daher zusätzliche Vorkehrungen notwendig, um die Versorgung mit nicht-lokalen Daten sicherzustellen und einen gemeinsamen Datenzugriff zwischen verschiedenen Systemen zu unterstützen. Wir stellen verschiedene Möglichkeiten einer automatisierten Produktdatenkontrolle in einer heterogenen WfMS-Landschaft vor und bewerten diese entsprechend. Dabei wird sich zeigen, dass sich globale Datenflussabhängigkeiten zwischen Workflows verschiedener Systeme gut nachbilden lassen, in dem die lokalen Workflows um Aktivitäten zur transparenten Bereitstellung der Daten erweitert werden. Dies bedarf in den meisten Fällen nur einer geringen Modifikation der betroffenen lokalen Workflow-Schemata.

1. Motivation

Moderne Vorgehensweisen in der Abwicklung von sowohl klassischen Geschäftsprozessen [Sche94] als auch von Ingenieursprozessen, z. B. Entwicklungsvorgängen [BDS98], führten in den letzten Jahren verstärkt zum Einsatz von Workflow-Management-Systemen (WfMS). Da

sich jedoch Unternehmen in vielen Fällen aus mehreren, möglicherweise organisatorisch weitgehend selbstständigen Unternehmensteilen zusammensetzen, kam es häufig vor, dass sich innerhalb eines Unternehmens eine heterogene WfMS-Landschaft entwickelte, d. h., verschiedenartige WfMS in verschiedenen Unternehmensteilen derselben Unternehmung im Einsatz sind. Als Konsequenz sind organisationsübergreifende Abläufe durch mehrere unabhängige und heterogene Workflows abzuwickeln. Diese Situation stellt solange kein größeres Problem dar, solange zwischen diesen Workflows (Teilprozessen) keine komplexen Abhängigkeiten bestehen. Nach neueren Erkenntnissen gibt es jedoch solche Abhängigkeiten sehr häufig. Diese rühren oft daher, dass alle verschiedenen Unternehmensteile zu einem gemeinsamen Unternehmensziel beitragen müssen. Da die Wartung solcher Abhängigkeiten jedoch die Grenzen von Workflow-Umgebungen überschreitet, wurden entsprechende Arbeiten bisher ausschließlich von Hand erledigt. Ein einfaches Beispiel hierfür ist das explizite Versenden von vorläufigen oder auch stabilen Entwicklungsdaten von einer Organisationseinheit zu einer anderen, die diese Daten wiederum als Ausgangspunkt für weitere Entwicklungsvorgänge nutzt.

Die allgemeine Problemstellung, die in unserem Projekt bearbeitet wird, besteht nun darin, eine Automatisierung der Wartung dieser Abhängigkeiten zwischen heterogenen Unternehmensprozessen zu entwickeln. Dabei betrachten wir verschiedene, mit möglicherweise heterogenen WfMS ausgestattete Unternehmensteile als sogenannte Workflow-Inseln (kurz *Wf-Insel*). Eine Wf-Insel umfasst dabei nicht nur das installierte WfMS mit zugehörigen Workflow-Beschreibungen (Wf-Schemata), sondern auch die vorhandenen Ressourcen wie Benutzer (repräsentiert durch Rollen), Applikationen, Datenbanken und Produktdatenverwaltungssysteme (PDMS). Es müssen Mittel und Wege gefunden werden, bisher nur implizit erfasste, d. h. ausschließlich in den Köpfen bestimmter Mitarbeiter gegenwärtige Abhängigkeiten zwischen den Wf-Inseln explizit zu erfassen und ihre bisherige manuelle Wartung durch eine entsprechende Automatisierung, zumindest jedoch Teilautomatisierung, zu ersetzen. Die heterogenen Wf-Inseln werden also in gewissem Sinne einer globalen Steuerung ‘unterworfen’, die für die Abwicklung von Inter-Workflow-Abhängigkeiten verantwortlich ist. Es sollen nicht (neue) globale Workflows spezifiziert und mit Hilfe der vorhandenen Wf-Inseln abgewickelt werden (Top-down-Vorgehensweise), sondern ausgehend von den vorhandenen lokalen Workflows sollen Abhängigkeiten zwischen diesen erkannt, explizit beschrieben und automatisiert abgewickelt werden (Bottom-up-Vorgehensweise!).

Einer der wesentlichen Aspekte dieser allgemeinen Problemstellung ist der Datenflussaspekt, d. h. die Behandlung von Datenflüssen über Wf-Insel-Grenzen hinaus. Da der Datenflussaspekt in unseren weiteren Betrachtungen eine zentrale Rolle spielt, werden wir zunächst diesen im nachfolgenden Abschnitt näher motivieren. In Abschnitt 3 werden die verschiedenen logischen Aspekte der Problemlösung herausgearbeitet. Ihre Realisierung wird in den Abschnitten 4-6 erörtert. Dabei werden zunächst in Abschnitt 4 Lösungen betrachtet, die durch eine Anpassung der beteiligten Workflows die für die Datenversorgung erforderliche Interoperabilität gewährleisten. In Abschnitt 5 werden dagegen Lösungen diskutiert, die auf einer Erweiterung der beteiligten PDMS beruhen. Schließlich untersuchen wir in Abschnitt 6 Mechanismen zur Datenübertragung, bevor wir unseren Ansatz nocheinmal zusammenfassen.

2. Kontrolle von Datenflussabhängigkeiten

Nach einer Beschreibung, wie herkömmlicherweise die Datenversorgung zwischen heterogenen Workflows gehandhabt wird, betrachten wir die Parameter, die für die automatisierte Datenversorgung und die Überwachung eines solchen Datenflusses zu spezifizieren sind. Eine effektive Kontrolle von Datenflussabhängigkeiten erfordert eine zentrale Lösung in Form eines Koordinators, dessen Aufgaben in den nachfolgenden Abschnitten detailliert werden.

2.1 Datenflussproblematik

Ohne Beschränkung der Allgemeinheit wollen wir Beispiele aus dem Entwicklungsbereich untersuchen. Ein hierfür typisches Szenario besteht darin, dass auf verschiedenen Wf-Inseln Teile eines Produktes entwickelt werden, die natürlich später in irgendeiner Weise zu integrieren sind. Beispielsweise macht es wenig Sinn, einen Kotflügel zu entwerfen, wenn sich der vorgesehene Scheinwerfer nicht mehr einpassen lässt. Wir betrachten also zwei auf unterschiedlichen Wf-Inseln ablaufende Workflows und nehmen an, dass eine Abhängigkeit dadurch entsteht, indem die durch eine Applikation des einen Workflows erzeugte Kotflügel-Geometrie auf die andere Wf-Insel übertragen wird, um dort als (weitere) Eingabe eines Schrittes in der Entwicklung des Scheinwerfers genutzt zu werden. Abbildung 1 illustriert diese Situation sowie den Vorgang einer manuellen Wartung der entsprechenden Abhängigkeit.

Auf Insel 2 benötigt ein Ingenieur zum Ausführen seiner Tätigkeit Daten, die im $PDMS_1$ auf Insel 1 gespeichert sind (z. B. die Geometrie des Kotflügels). Diese stehen allerdings nicht direkt zur Verfügung, d. h., die Informationen müssen zuerst aufbereitet und bereitgestellt werden. Im ersten Schritt muss er die Daten anfordern, in dem er sich beispielsweise per Mail oder Telefon mit dem Verantwortlichen auf der Gegenseite in Verbindung setzt. Dieser prüft, ob er die gewünschten Informationen weitergeben darf, und extrahiert diese aus dem PDMS. Das Transportformat muss beiden Seiten bekannt sein. Nach der Extraktion muss die Transportdatei von Insel 1 (der *Quellinsel*) zu Insel 2 (der *Zielinsel*) übertragen werden, was auf verschiedene Art und Weise möglich ist. Der schnellste Weg ist sicherlich die Nutzung des Internets zur Übertragung, beispielsweise per E-Mail oder FTP. Denkbar ist auch eine Freigabe der Datei, um dem Empfänger ein „Download“ zu ermöglichen. Ist dies alles nicht möglich, weil z. B. Firewalls den Transport behindern, bleibt noch der klassische Weg, einen Datenträger per Post zu verschicken. Nach Ankunft der Daten auf der Zielinsel werden sie vom Systemverantwortlichen in das dortige $PDMS_2$ eingespielt, was wiederum mit Hilfe angebotener Import-Funktionen oder spezieller Import-Programme geschieht. Anschließend kann der Ingenieur bzw. die von ihm zur Erfüllung seiner Aufgabe zu nutzende Ziel-Applikation auf Insel 2 lokal darauf zugreifen. Werden dabei Änderungen vorgenommen, so müssen diese wieder zurück zu Insel 1 fließen, und konsistent in $PDMS_1$ eingebracht werden.

Wie bereits angesprochen, besteht unser Ziel in einer Automatisierung der Wartung solcher Datenflussabhängigkeiten. Dies erfordert im allgemeinen Fall grob zwei Schritte, die anhand von Abbildung 2 erläutert werden sollen. Der erste Schritt besteht in der Erkennung und Spezifikation der eigentlichen Abhängigkeit auf der Typebene (siehe Abbildung 2, Teil A), der zweite in der Realisierung dieser Datenflussabhängigkeit auf Ausprägungsebene. Auf Typebene sind die den beiden unterschiedlichen WF-Inseln zugeordneten Workflow-Typen zu betrachten, hier WfT_{11} und WfT_{21}. Jede Art von Datenflussabhängigkeit kann als Datenflussbeziehung über

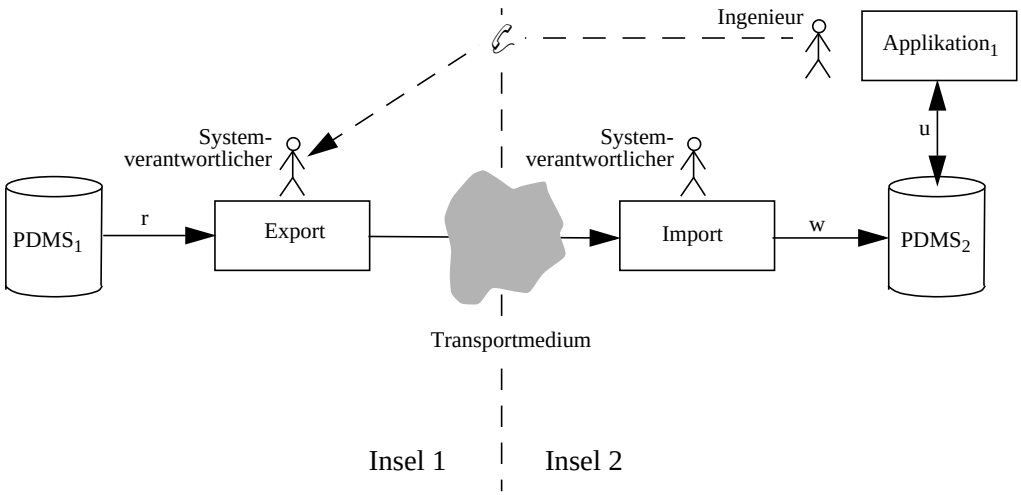


Abbildung 1 Manuelles Übertragen von Kooperationsdaten

Workflow-Grenzen hinaus betrachtet werden. So verbindet beispielsweise die in Abbildung 2 eingezeichnete Datenflussabhängigkeit zwei Workflow-Aktionen der beiden betroffenen Workflow-Typen. Die zugehörige Semantik besagt, dass die Ausgabe von WfA_{115} als zusätzliche Eingabe von WfA_{213} benötigt wird. 'Zusätzlich' bedeutet hier, dass diese Daten, im Folgenden auch als Kooperationsdaten bezeichnet, von der sich hinter WfA_{115} verbergenden Applikation als vorhanden vorausgesetzt werden, obwohl sie nicht direkt von den WfT_{21}-internen Datenflussspezifikationen erfasst sind. Für die Spezifikation der Datenflussabhängigkeit brauchen wir im wesentlichen folgende Teile:

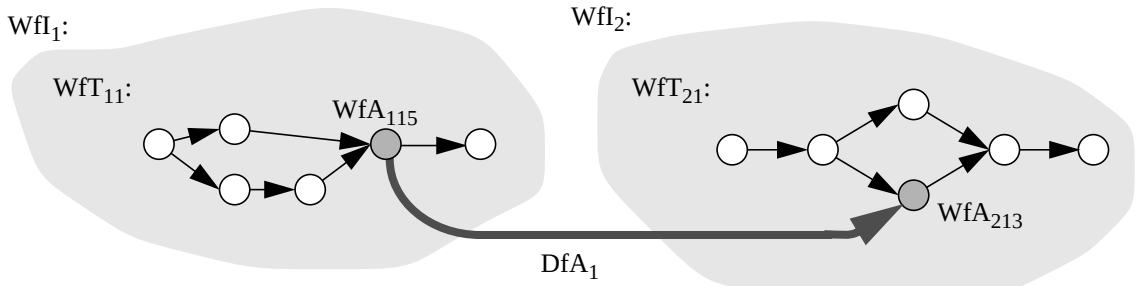
- die beiden betroffenen Workflow-Typen,
- die beiden durch die Datenflussabhängigkeit verbundenen Workflow-Aktionen,
- eine geeignete Spezifikation der Menge der Kooperationsdaten,
- die Datenquelle (i. Allg. ein PDMS) und die Art der Extraktion der Kooperationsdaten aus dieser Datenquelle,
- die Art der Übertragung der Kooperationsdaten von der Quell- zur Zielinsel sowie
- die Datensenke (i. Allg. ein PDMS) und die Art der Integration der Kooperationsdaten in diese Datensenke.

Die Ausprägungsebene soll anhand von Abbildung 2, Teil B skizziert werden. Als Ausprägungen betrachten wir zwei konkrete Instanzen (Prozesse, Workflows) Wf_1 und Wf_2 der beiden in Teil A betrachteten Workflow-Typen. Die Abbildung ist hier weitestgehend selbsterklärend. Wir wollen daher an dieser Stelle lediglich weiter motivieren, dass es zur Abwicklung des eingezeichneten Inter-Workflow-Datenflusses eine ganze Reihe von Optionen gibt, deren Nutzung sehr stark von den zugrunde liegenden Systemlandschaften (im konkreten Anwendungsfall) abhängig sind. Daraus ergeben sich im einzelnen die im Abschnitt 3 beschriebenen Dimensionen.

2.2 Der Koordinator

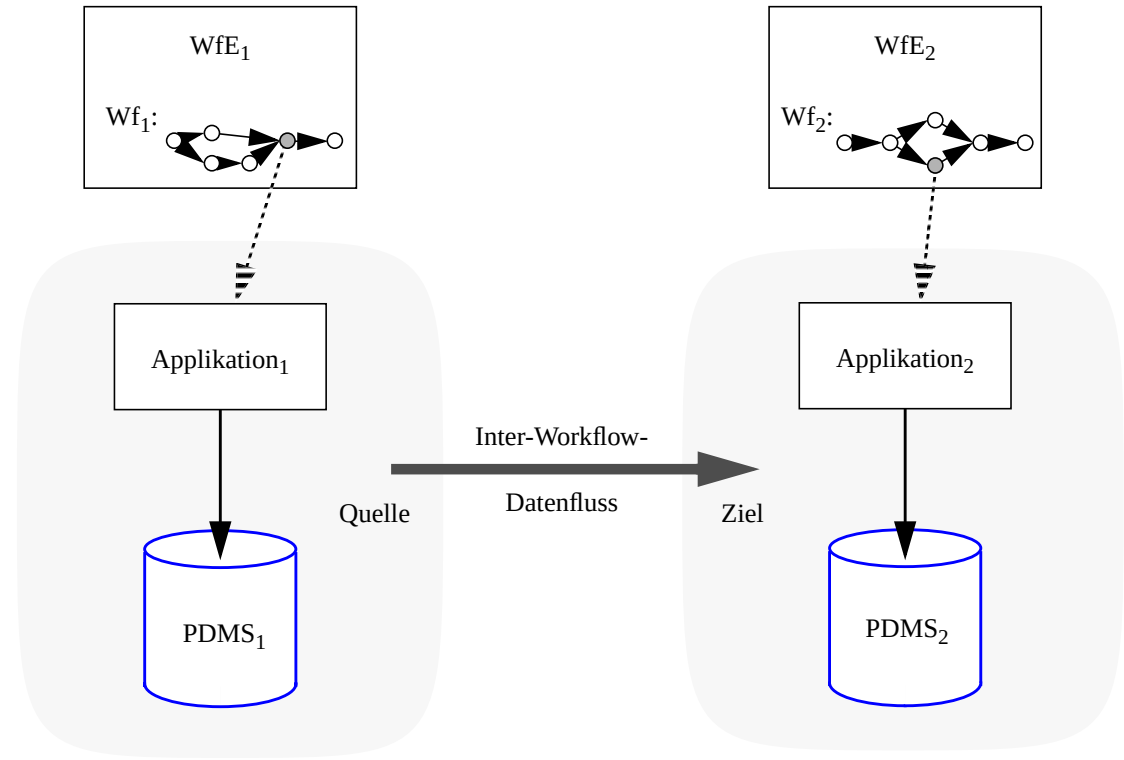
Um Workflow-übergreifende Prozesse durchzuführen, müssen wir globales Wissen über den Gesamtprozess nutzen können. Das auf den Inseln lokal vorhandene Wissen reicht dazu nicht

A. Typebene



WfI_i: Workflow-Insel i
WfT_{ij}: Workflow-Typ j auf Insel i
WfA_{ijk}: Workflow-Aktion k in Workflow-Typ j auf Insel i
DfA_i: Datenfluss-Abhängigkeit i

B. Ausprägungsebene



WfE_i: Workflow-Engine auf Insel i
Wf_i: Workflow auf Insel i (jeweils Instanz des entsprechenden Workflow-Typs in A.)
$Applikation_i$: in der Instanz der zugehörigen Workflow-Aktion aufgerufene Applikation
$PDMS_i$: von Applikation i genutztes Produktdatenverwaltungssystem

Abbildung 2 Datenflussabhängigkeiten

aus. Weiterhin wollen wir die lokalen Systeme so wenig wie möglich ändern. Eine verteilte Realisierung der Koordinatorfunktionalität würde nicht nur komplexe Protokolle zur Behandlung globaler Information und zur Abstimmung zwischen den lokalen WfMS (siehe nachfolgende Koordinationsaufgaben) erfordern ([Schu99]), sondern auch massive Änderungen in den beteiligten lokalen WfMS verlangen. Deshalb dient es der Reduzierung der Komplexität und der Be-

grenzung der notwendigen Systemanpassungen, eine zentrale Komponente zu benutzen, welche die Abhängigkeiten zwischen den lokalen Workflows kennt und die daraus resultierende Interaktion zwischen den Inselsystemen unterstützt bzw. steuert. In diesem Koordinator wird folgendes globale Wissen zur Verfügung gestellt:

- globaler Kontrollfluss: *zeitliche* Abhängigkeiten (auf Typ-Ebene) zwischen Wf-Aktivitäten auf verschiedenen Inseln;
- globaler Datenfluss: Abhängigkeiten zwischen Workflow-Typen, die beschreiben, welche Aktivitäten Daten produzieren, die als Eingabe für Aktivitäten auf anderen Inseln benötigt werden;
- Ausführungsorte: bestimmte Wf-Typen können nur auf bestimmten Systemen ausgeführt werden;
- Ausführungsfortschritt: Monitoring-Information über den aktuellen Status jeder Workflow-Instanz, aber auch über den Gesamtprozess;
- Zeitvorgaben / Exception Handling: zur Ausführungszeit können im Zusammenspiel zwischen den lokalen Workflows Fristen bestehen; weiterhin müssen entsprechende Maßnahmen vorgesehen sein, die z. B. im Konfliktfall geeignete Recovery-Schritte einleiten;
- Authentifizierung: eine weitere wichtige Rolle spielt die Authentifizierung der Systeme untereinander; bei Datenzugriff auf fremde Systeme muss gewährleistet werden, dass diese nicht in falsche Hände geraten.

All dieses Wissen muss dem Koordinator zur Verfügung stehen, um als vertrauenswürdiger, neutraler Vermittler in Erscheinung treten zu können. Er ist somit verantwortlich für:

- die Registrierung und Verwaltung lokaler Workflows, die den globalen Prozess abbilden;
- die inselübergreifende Koordination der lokalen Workflows, beispielsweise Identifikation der Partner eines Datenaustauschs oder Überwachung des notwendigen Kommunikationsvorgangs;
- das Überwachen (‘Monitoring’) des globalen Workflows durch Überwachen jedes teilnehmenden lokalen Workflows;
- die Statusanpassung (suspend/resume) der lokalen Workflow-Instanzen, um (inselübergreifende) Kontrollflussabhängigkeiten einzuhalten;
- die Durchführung eines (globalen) ‘Exception Handling’ im Fehlerfall.

2.3 Koordination von Datenflussabhängigkeiten

In Abbildung 2 wird zwischen Wf-Typebene und Wf-Ausprägungsebene unterschieden. Die Definition der Datenflussabhängigkeit findet auf der Typebene statt. Sobald Instanzen der beteiligten Workflows erzeugt worden sind, ist eine solche Datenflussabhängigkeit auf Ausprägungsebene den entsprechenden Instanzen zuzuordnen. Wir benötigen folglich eine Einrichtung, die bei Erzeugung von an Datenflussabhängigkeiten beteiligten Wf-Instanzen benachrichtigt wird, um ausgehend von den zugehörigen Identifikatoren Instanzenpaare zu verwalten und somit die Kommunikation zwischen den richtigen Partnern zu ermöglichen. Eine Möglichkeit, dies zu realisieren, ist die Nutzung des Koordinators, der das Wissen über die globalen Zusammenhänge verwaltet ([BRZ00]).

Neben globalen Kontrollflussinformationen werden hier auch die globalen Datenflüsse zwischen den Workflows beschrieben. Die Definition auf Typebene muss in eine Abhängigkeit zwischen Instanzen dieser Typen abgebildet werden. Da jedoch Datenflussabhängigkeiten zwischen mehreren Workflows bzw. zwischen mehreren Workflow-Aktivitäten bestehen können, reicht die Identifikation der Instanz alleine auch nicht aus. Es wird darüber hinaus die Information benötigt, welche Aktivitäten betroffen sind.

Folgende Vorgehensweise erscheint hier erforderlich. Durch Einführen von Bezeichnern ('label') ist es möglich, bestimmte Stellen im lokalen Workflow für den Koordinator erkennbar zu machen. Versehen wir z. B. WfA_{115} auf Insel WfI_1 und WfA_{213} auf Insel WfI_2 jeweils mit dem Bezeichner 'S_1', dann lässt sich DfA_i im Koordinator (in Anlehnung an das Interoperabilitätsmodell der WfMC) als SYNC ($WfT_{11}.S_1$, $WfT_{21}.S_1$) beschreiben. Nach Instanzieren der Workflows zu 'Wf_1' und 'Wf_2' wird diese Paarzuordnung entsprechend in ($Wf_1.S_1$, $Wf_2.S_1$) umgesetzt. Dabei besteht jedes Paar aus dem Tupel (Quellinstanz, Zielinstanz). Diese Paarbildung ist nur schwer zu automatisieren. Existieren beispielsweise mehrere Instanzen des gleichen Workflow-Typs (z. B. von WfT_{11}), so kann das System nicht eigenständig entscheiden, welche dieser Instanzen mit einem synchronisationswilligen Workflow vom Zieltyp (im Beispiel WfT_{21}) zusammenarbeiten soll. Vielmehr werden beim Initialisieren des Workflows die durch den Workflow-Typ festgelegten Abhängigkeiten geprüft und dem (menschlichen) Initiator potentielle Partner-Instanzen angeboten. Ist der entsprechende Partner-Workflow vorhanden, wird das Paar gebildet, andernfalls ein *offenes Paar* gebildet. Dieses kann wiederum als potentieller Kandidat für neu initialisierte Workflows (von entsprechendem Typ) angeboten werden.

Der Koordinator überwacht die Ausführung der beiden Workflow-Instanzen. Werden die durch die Definition der Datenflussabhängigkeit festgelegten Stellen in den Workflows erreicht, unterstützt der Koordinator auch die Kommunikation zwischen den Inseln. Wie diese Kommunikation aussieht, hängt jedoch stark davon ab, welcher Mechanismus für die Bereitstellung der Daten auf der Ziel-Insel gewählt wird. Wir werden in den entsprechenden Abschnitten noch näher darauf eingehen.

3. Dimensionen der Problemlösung

Bei der Lösungsfindung können für das vorgestellte Problem der Datenversorgung von Aktivitäten verschiedener heterogener Workflows folgende wesentlichen Fragestellungen (Dimensionen) unterschieden werden:

- Wie lassen sich die auszutauschenden Datengranulate spezifizieren?
- Welche Probleme sind zwischen den beteiligten PDMS hinsichtlich Datenzugriff, Datenbereitstellung und Zugriffskontrolle zu lösen?

Diese beiden Fragen wollen wir in den nachfolgenden Abschnitten diskutieren.

3.1 Spezifikation von Datengranulaten

In diesem Abschnitt beschäftigen wir uns mit den oben angesprochenen Kooperationsdaten. Im Allgemeinen hat sich zur Beschreibung von Produktdaten eine hierarchische Struktur bewährt: Die Wurzel repräsentiert eine Art Handle des Gesamtprodukts und kann sich in zusammenge-

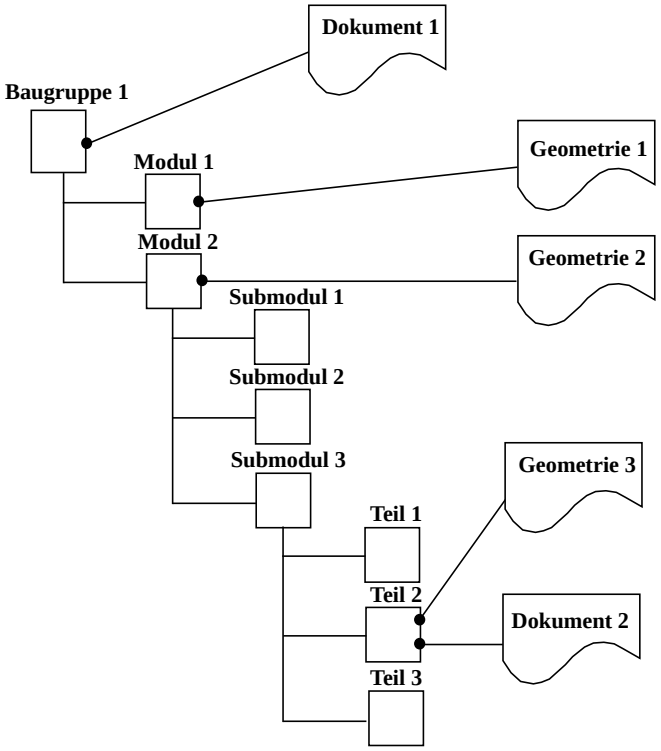
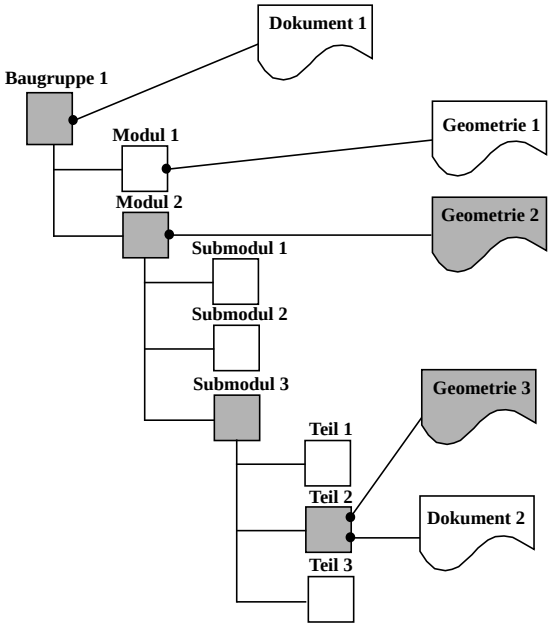


Abbildung 3 Hierarchische Darstellung der Datenstrukturen eines PDMS

setzte Bauteile (Assemblies oder Composite Parts [Step92]) unterteilen, die wiederum aus einfachen Bauteilen (Components oder Atomic Parts) oder zusammengesetzten Teilen aufgebaut sein können (Abbildung 3). Es entsteht eine Baumstruktur (*Produktbaum*), deren innere Knoten zusammengesetzte Teile und deren Blätter einfache Bauteile beschreiben. Auf jeder Stufe können den (einfachen oder zusammengesetzten) Bauteilen weitere Objekte zugeordnet werden. Dabei kann es sich beispielsweise um beschreibende Dokumente oder Geometrie-Dateien, die zur Bearbeitung des Bauteils mit Hilfe eines CAD-Werkzeugs benötigt werden, handeln. Solche Objekte bezeichnen wir als *Anhänge*.

Nehmen wir nun an, dass die in einem Quell-PDMS (siehe $PDMS_1$ in Abbildung 2, Teil B) gehaltenen Produktdaten in der soeben beschriebenen Weise verwaltet werden. Üblicherweise ist ein ganz bestimmter Teil dieser im Quell-PDMS verwalteten Daten (Produktbaum und Anhänge) für das Ziel-PDMS (siehe $PDMS_2$ in Abbildung 2, Teil B) relevant, so dass die Kooperationsdaten ‘nur’ einen ganz bestimmten Teil des Produktbaums ausmachen.

Natürlich ist aus Aufwandsgründen darauf zu achten, dass tatsächlich genau die Kooperationsdaten, also weder zu wenige noch zu viele Daten, erfasst werden. Die Identifikation der Kooperationsdaten in einem möglicherweise sehr komplexen Produktbaum ist eine sehr anspruchsvolle Aufgabe und kann nur intellektuell von einem Experten durchgeführt werden. Um diese Aufgabe zu erfüllen, ist detailliertes Wissen mindestens über die Produktdatenstrukturen der Quell- und Ziel-PDMS sowie über die Vorgehensweise der Ziel-Applikation (siehe $Applikation_2$ in Abbildung 2, Teil B) notwendig. Nach Identifikation der Menge der Kooperationsdaten ist dem System eine entsprechende Spezifikation (dieser Kooperationsdaten) bekanntzumachen. Diese ist offensichtlich eine Voraussetzung für die automatisierte Behandlung der zugehörigen Datenflussabhängigkeit und erfordert wiederum eine Spezifikationssprache




```
<part>
   <name>Baugruppe1</name>
   <subparts>
      <part>
         <name>Modul2</name>
         <geometry mode="materialized">
            <uri>"Geometrie2"</uri>
         </geometry>
         <subparts>
            <part>
               <name>Submodul3</name>
               <subparts>
                  <part>
                     <name>Teil2</name>
                     <geometry mode="referenced">
                        <uri>"Geometrie3"</uri>
                     </geometry>
                  </part>
               </subparts>
            </part>
         </subparts>
      </part>
   </subparts>
</part>
```

Abbildung 4 Identifikation und Spezifikation von Kooperationsdaten

ausreichender Mächtigkeit. Betrachten wir ein Beispiel (siehe Abbildung 4).

Der dargestellte Produktbaum zeigt die Gesamtstruktur einer Produkt-Baugruppe, beispielsweise des Antriebssystems eines Kraftfahrzeugs. Die Baugruppe lässt sich in 2 Module unterteilen (Modul 1, beispielsweise der Motor, und Modul 2, das Getriebe). Diese beiden Module sollen nun in verschiedenen Abteilungen entwickelt werden (Modul 1 auf Insel 1, Modul 2 entsprechend auf Insel 2). Der Entwickler von Modul 2 muss natürlich wissen, wie die Kopplung mit Modul 1 aussieht. In unserem Motorenbeispiel muss er die Beschaffenheit und Lage der Welle kennen und muss wissen, wieviel Platz ihm zur Verfügung steht und welche Bauteile angrenzen.[1] Deshalb werden dem Entwickler von Modul 1 ausreichende Informationen über Modul 2 zur Verfügung gestellt. Es ist dabei jedoch nicht notwendig, alle Daten zu übertragen. Der relevante Teilbaum ist in Abbildung 4 grau markiert. Es werden eine CAD-Geometrie des Gesamtmoduls (beispielsweise ein Volumenmodell des Motors) und ein Ausschnitt aus der Struktur des Moduls 2 übertragen (Submodul 3 könnte in unserem Beispiel die Kraftübertragung des Motors sein, Teil 2 die Welle zum Anschluss an das Getriebe). Braucht der Entwickler auf Insel 1 nun detailliertere Informationen zu Teil 2, kann er auf die zugehörige Geometrie zugreifen. Die für ihn nicht interessanten Submodule 1 und 2 sowie die weiteren Bauteile Teil 1 und Teil 3 von Submodul 3 sind für ihn nicht von Bedeutung und werden ihm daher auch nicht zur Verfügung gestellt.

Die Aufgabe des Kooperationsdatenspezifizierers besteht nun darin, den grau markierten Teilbaum zu identifizieren und in einer dem System verständlichen Sprache zu beschreiben, um die Automatisierung der Übertragung zu ermöglichen. Zur Motivation nutzen wir an dieser Stelle eine XML-basierte Spezifikationssprache, da sich auf diese Art und Weise die hierarchische

1. Die Rückkehr von Porsche in die Formel 1 beim britischen Arrows-Team wurde 1991 beispielsweise dadurch verhindert, dass beim Einbau des Motors dieser schlicht und ergreifend nicht passte. Die notwendigen Änderungen führten dazu, dass der Motor zu schwer wurde und zu wenig Leistung erbrachte.

Struktur der Produktdaten sehr gut abbilden lässt. In Abbildung 4 ist rechts der Ausschnitt aus einem XML-Dokument zu sehen, das den markierten Teilbaum beschreibt. Jede Hierarchie-Ebene wird mit Hilfe eines <part>-Tags beschrieben. Attribute (nicht im Sinne von XML) sind beispielsweise der Name des Bauteils, zugehörige Geometrien und Dokumente, und, falls es sich um ein zusammengesetztes Bauteil (oder Composite Part) handelt, eine Liste der untergeordneten Teile. Diese Liste wird mit Hilfe des <subparts>-Tags identifiziert, die Elemente sind wiederum part-Elemente.

Eine Besonderheit weisen die <geometry>- und <document>-Tags auf: sie besitzen ein Attribut „mode“, mit dem festgelegt werden kann, wann der Anhang übertragen werden soll. Es besteht die Möglichkeit, einen Anhang materialisiert oder referenziert zu übertragen. Materialisiert bedeutet dabei, dass der Anhang sofort auf die Zielinsel übertragen wird. Da Anhänge sehr groß werden können, macht dies nur Sinn, wenn mit großer Wahrscheinlichkeit auf die Daten zugegriffen wird. Bei referenzierter Übertragung wird hingegen nur eine Referenz angeboten, mit deren Hilfe der zugehörige Anhang bei Bedarf übertragen werden kann. So lässt sich überflüssiges Übertragen von Dokumenten, die nur mit geringerer Wahrscheinlichkeit wirklich benötigt werden, verhindern.

In unserem Beispiel ist es sicherlich sinnvoll, die Gesamt-Geometrie zu Modul 2 direkt zu übertragen, da der Entwickler auf Insel 1 diese Informationen unmittelbar braucht. Zur Geometrie von Teil 2 wird hingegen nur eine Referenz angeboten. Erst, wenn der Entwickler feststellt, dass er darauf zugreifen muss, wird die Geometrie auch tatsächlich angefordert und übertragen.
Obwohl wir in diesem Abschnitt eine XML-basierte Spezifikationssprache zu Motivationszwecken genutzt haben, soll dies nicht bedeuten, dass bereits eine Entscheidung für eine solche Sprache gefallen ist. Weitere Sprachansätze müssen betrachtet und für unsere Zwecke evaluiert werden, bevor eine endgültige Entscheidung getroffen werden kann. Bereits jetzt lassen sich jedoch folgende Mindestanforderungen an eine Spezifikationssprache formulieren:

- Deskriptive Spezifikation: Die Sprache muss hinreichende Mittel zur deklarativen Beschreibung der relevanten Daten, bzw. des gewünschten Ausschnitts aus dem Produktbaum, zur Verfügung stellen.
- Hierarchische aufgebaute Datenstrukturen: Produktdaten sind hierarchisch aufgebaut Die Spezifikationssprache muss diesem Umstand Rechnung tragen, in dem sie zum einen die Beschreibung jeder Stufe für sich zulässt, zum anderen aber auch das Abbilden der hierarchischen Gesamtstruktur ermöglicht.
- Erweiterbare Funktionalität: Die Sprache soll auf verschiedenen, heterogenen Systemen zum Einsatz kommen. Zwar unterstützen alle PDMS ähnliche Modelle zum Speichern von Produktdaten, es können jedoch Anpassungen notwendig werden. Sollen beispielsweise zusammengesetzte Bauteile explizit von normalen Teilen unterscheidbar sein, so ist in unserem Beispiel die Einführung eines neuen Tags <assembly> möglich.
- Einfache Syntax: Da die Verarbeitung automatisiert erfolgen soll, muss die Sprache vom System leicht „verstanden“ und effizient verarbeitet werden können.

Da es dem Spezifizierer im Allgemeinen wohl nicht zumutbar ist, die Kooperationsdaten direkt in der Spezifikationssprache zu beschreiben, müssen geeignete Tools mit entsprechenden grafischen Bedienungselementen angeboten werden. Aufgrund der sehr einfachen Funktionalität,

die nur in der Darstellung der Produktstruktur und der Auswahl des relevanten Teilbaums besteht, sollte dies jedoch keine größeren Probleme bereiten.

3.2 Datenflussinteroperabilität

In Abschnitt 2.1 wurde beschrieben, welche Teile die Spezifikation einer Datenflussabhängigkeit umfasst. Im vorangegangenen Abschnitt wurde deutlich, dass die Spezifikation der Kooperationsdaten ein äußerst sensibler Teil der Gesamtspezifikation ist, da die Effektivität des Gesamtsystems doch sehr auf eine möglichst genaue Eingrenzung der Kooperationsdaten angewiesen ist. Daneben kommt der Art und Weise, wie der Zugriff auf diese Kooperationsdaten durch die Ziel-Applikation (siehe Applikation_2 in Abbildung 2, Teil B) ermöglicht wird, offensichtlich eine besondere Bedeutung bei der Systemgestaltung zu. Mit dieser Problematik wollen wir uns in diesem Abschnitt beschäftigen. In Tabelle 1 sind Kriterien angeführt, welche die Möglichkeiten der Systemunterstützung einer solchen Datenflussinteroperabilität mitbestimmen.

PDMS	gleich	verschieden
Zugriff	nur lesend	lesend und schreibend
Datenbereitstellung	Zugriffsumleitung	Datenübertragung

Tabelle 1: Klassifizierung des Zugriffs

So ist zunächst zu unterscheiden, ob es sich bei Quell- und Ziel-PDMS um dasselbe[2] oder unterschiedliche Systeme handelt. Man beachte, dass der Fall, bei dem viele verschiedene Workflow-Applikationen auf dasselbe Werkzeug zur Produktdatenverwaltung zurückgreifen, in realen Umgebungen sehr häufig vorzufinden ist. Das zweite wichtige Kriterium ist die Art des Zugriffs auf die Kooperationsdaten durch die Ziel-Applikation. In der Mehrzahl der in realen Umgebungen auftretenden Fälle wird man sich hier auf einen lesenden Zugriff beschränken. Trotzdem sollte auch ein schreibender Zugriff in die Betrachtung miteinbezogen werden. Ein dritter, zu den beiden vorgenannten orthogonaler Aspekt ist die Art der Datenbereitstellung für die Ziel-Applikation. Je nach Umfang der Kooperationsdaten und Zugriffscharakteristika (z. B. Lokalität der Referenzen!) ist zu entscheiden, ob eine Datenübertragung von dem Quell- in das Ziel-PDMS oder eine Umleitung der Zugriffe der Ziel-Applikation auf das (entfernte) Quell-PDMS vorzuziehen ist. Es ist offensichtlich, dass es sich hierbei um ein Analogon zu den bereits bekannten Prinzipien des *Data Shipping* bzw. *Function Request Shipping* handelt ([Sel00]). Im Folgenden wollen wir die angesprochenen Kriterien und ihre Ausprägungen etwas näher betrachten.

3.2.1 PDMS

Im günstigsten Fall wird auf Ziel- und Quellinsel das gleiche PDMS eingesetzt. Dies hat den Vorteil, dass beide Systeme ein identisches Datenmodell verwenden und über die gleiche API-Funktionalität verfügen. Damit würde im Falle einer Datenübertragung eine Konvertierung der

2. Hiermit ist gemeint, dass dasselbe System zur Produktdatenverwaltung auf beiden Seiten genutzt wird. Wir gehen jedoch bei unseren Betrachtungen davon aus, dass es sich dabei um verschiedene Installationen handelt und damit (vor Kooperation) auch die Datenbestände differieren.

Kooperationsdaten entfallen. Eine eventuelle Zugriffsumleitung würde sich ebenfalls einfacher gestalten, da die in der Ziel-Applikation enthaltenen Datenzugriffe lediglich hinsichtlich des Zugriffsorts, nicht jedoch bzgl. der Art des Zugriffs mit Hilfe eines Wrappers umgesetzt werden müssten, da dasselbe PDMS-API auf Quell- und Zielseite zur Verfügung steht.

Etwas anders sieht es aus, wenn verschiedene Systeme zum Einsatz kommen. Hier müssen die Kooperationsdaten vom Datenmodell des Quellsystems auf das des Zielsystems abgebildet werden. Eine solche Abbildung ist natürlich insbesondere im Falle einer Datenübertragung notwendig, aber auch zur Vorbereitung einer geeigneten Funktionsumleitung äußerst sinnvoll. Während sie im Falle der Übertragung der Kooperationsdaten deren Konvertierung unterstützt, dient sie im Falle der Zugriffsumleitung der Konvertierung der Zugriffsparameter und hilft darüber hinaus bei der Entscheidung, wie die Zugriffe der Ziel-Applikation mit Hilfe der Quell-PDMS-API abgewickelt werden können. Näheres, auch welche Vorgehensweisen zur Konvertierung herangezogen werden können, wird in Abschnitt 4 behandelt.

3.2.2 Zugriff

Wie bereits oben angesprochen, unterscheiden wir *rein lesenden* von *lesendem und schreibendem* Zugriff. Der erste Fall ist relativ einfach handhabbar, da auf der Zielseite keine Aktionen durchgeführt werden, deren Auswirkungen auf die Quellseite zu propagieren sind. Es ist lediglich darauf zu achten, dass auf der Zielseite die richtige Version der Kooperationsdaten sichtbar/zugreifbar gemacht wird und bei der Konversion keine Informationen verloren gehen. Dies ist auch bei der Entscheidung über die Art der Datenbereitstellung zu beachten. Wenn beispielsweise nach Herstellung des für die Zielseite relevanten Zustands der Kooperationsdaten auf der Quellseite weitere Veränderungen an den Daten vorgenommen werden können, so ist natürlich eine Zugriffsumleitung nur dann sinnvoll, wenn der relevante Zustand durch Nutzung der von PDMS üblicherweise unterstützten Versionsverwaltung erhalten werden kann, d. h., sich eventuelle weitere Änderungen auf der Quellseite als neue Versionen niederschlagen und die ‘alten’ Versionen erhalten bleiben.

Wesentlich schwieriger gestalten sich die Aufgaben der Datenkontrolle, sobald auf der Zielseite auch schreibender Zugriff ermöglicht werden soll und die Auswirkungen dieser Zugriffe auf der Zielseite ins Ursprungs-PDMS propagiert werden müssen. Dies wird insbesondere dann schwierig, wenn die Daten auf beiden Seiten schreibend manipuliert werden sollen. Da nicht davon auszugehen ist, dass die derzeit in realen Umgebungen verwendeten PDMS mit vertretbarem Aufwand um eine Systemgrenzen überschreitende Kontrolle durch Synchronisationsverfahren erweitert werden können, klammern wir diesen Fall zunächst aus und beschränken uns auch hier auf Verarbeitungscharakteristika, die mit Hilfe der von PDMS unterstützen Versionierungsmechanismen kontrolliert werden können.

Wir nehmen also erstens an, dass im Falle eines schreibenden Zugriffs auf der Zielseite keine simultanen Veränderungen derselben Daten (Versionen) auf der Quellseite stattfinden, die mit den Manipulationen der Zielseite abgeglichen werden müssen, sondern sich entsprechende Änderungen auf der Quellseite als neue Versionen der entsprechenden Daten niederschlagen. Die zweite Annahme besteht darin, dass ein eventuell notwendig werdendes Propagieren der Manipulationen durch die Ziel-Applikation durch das Anlegen neuer Versionen im Quell-PDMS geschehen können.

3.2.3 Datenbereitstellung

Bezüglich der Datenbereitstellung gehen wir zunächst davon aus, dass der Handle der Kooperationsdaten, wie in Abschnitt 2.1 beschrieben, existiert und der Identifikator dieses Handle entweder als Datenflussparameter in den Ziel-Workflow eingespeist wird oder durch einen an der Abarbeitung des Ziel-Workflows beteiligten Mitarbeiter interaktiv der Ziel-Applikation mitgeteilt wird, so dass diese Ziel-Applikation an die Kooperationsdaten gelangen kann. In aktuellen Multi-Workflow-Umgebungen, in denen es noch keine Automatisierung der Wartung von Datenflussabhängigkeiten gibt, ist wohl der zweite Fall die Regel. Für uns erscheint es jedoch erstrebenswert, den Identifikator der Kooperationsdaten automatisch über einen (zusätzlichen) Datenflussparameter in den Ziel-Workflow einfliessen zu lassen, da diese Vorgehensweise eine weitergehende Automatisierung und damit eine Entlastung von Mitarbeitern darstellt.

Wir haben bezüglich der Datenbereitstellung bereits die Datenübertragung und die Zugriffsumleitung unterschieden. Diese beiden Möglichkeiten können wie nachfolgend beschrieben weiter unterteilt werden (siehe Abbildungen 5 und 6).

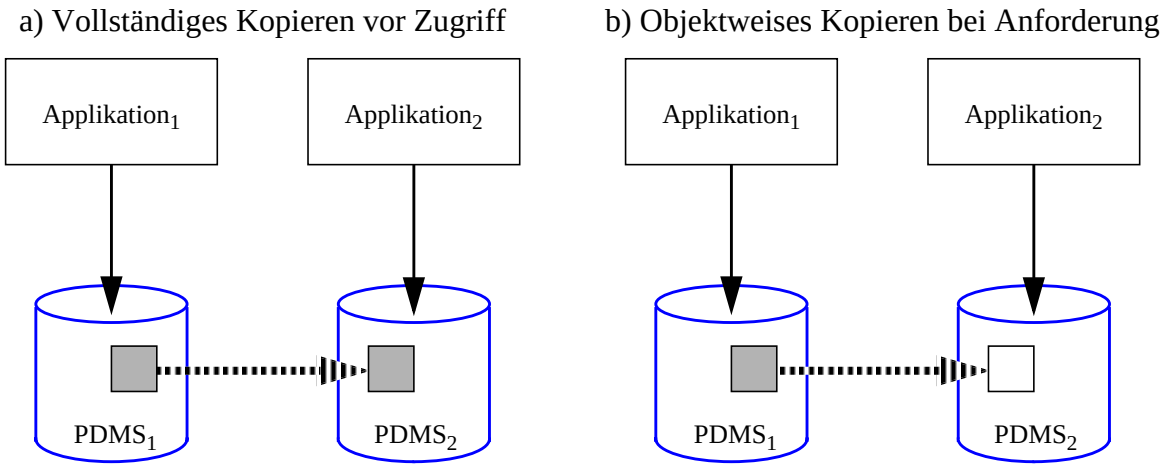


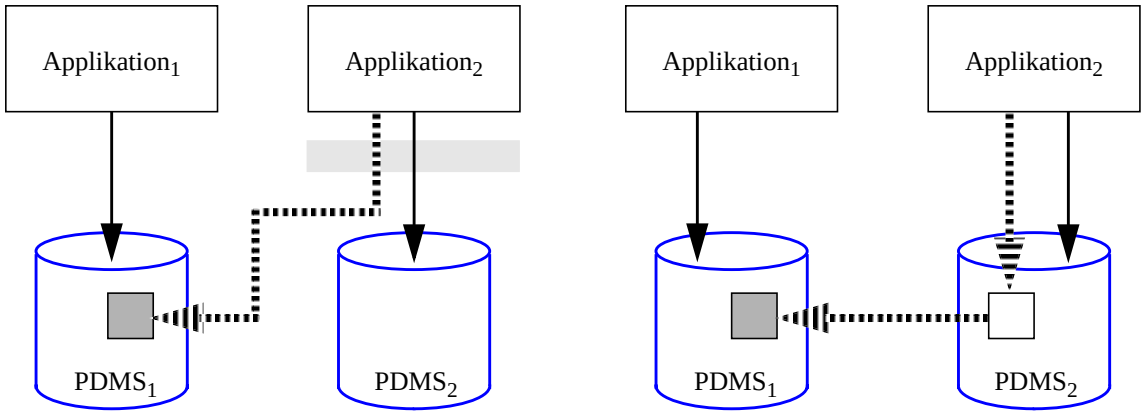
Abbildung 5 Ansätze zur Datenübertragung

Bereits bei der Diskussion von Einbringstrategien in anwendungsnahe Objektpuffer von Client/Server-DBS wurden verschiedene mögliche Zeitpunkte der Datenübertragung untersucht ([Sel00]). So zeigt Abbildung 5a das vollständige Übertragen der Menge der Kooperationsdaten vom Quell- ins Ziel-PDMS vor der eigentlichen Verarbeitung, während der in Abbildung 5b illustrierte Ansatz davon ausgeht, dass zunächst nur ein Handle für die Kooperationsdaten im Ziel-PDMS eingelagert wird und die eigentlichen Objekte erst während der Verarbeitung durch die Ziel-Applikation sukzessive auf Anforderung vom Quell- ins Ziel-PDMS übertragen werden. Diese zunächst sehr schlüssig klingenden Alternativen der Datenübertragung müssen jedoch bzgl. ihrer Anwendbarkeit in unserem Workflow-Szenario weiter diskutiert werden. Folgende Voraussetzungen gelten für beide Alternativen:

- Hinsichtlich einer automatischen Abwicklung der Datenübertragungsaktionen (Übertragung der gesamten Menge der Kooperationsdaten im Falle von A1 bzw. des Handle im

Falle von A2) sind mindestens Aktionen zum Bestimmen und Auslesen der Kooperationsdaten aus dem Quell-PDMS, zum Senden der Kooperationsdaten zur Ziel-Umgebung, ggf. zum Konvertieren sowie zum Einlagern in das Ziel-PDMS bereitzustellen. Ob es Aufgabe eines Experten sein muss, diese Aktionen von Hand zu programmieren oder ob es möglich ist, zumindest Teile dieser Aktivitäten durch Generierung bereitzustellen, soll im nachfolgenden Kapitel dieses Berichts untersucht werden.

- Die soeben angesprochenen, für eine Datenübertragung bereitzustellenden Aktivitäten müssen zum richtigen Zeitpunkt aufgerufen werden. So kann eine Datenübertragung erst angestoßen werden, wenn die Quell-Applikation erfolgreich beendet werden konnte. Eine weitere zeitliche Restriktion ist die Forderung, dass das Einlagern der Kooperationsdaten erfolgreich abgeschlossen sein muss, bevor die Ziel-Applikation gestartet werden kann. Eine Möglichkeit, diese Probleme zu lösen, ist die Transformation der betroffenen Workflow-Schemata durch Einbringen der im ersten Punkt angesprochenen Aktivitäten (an den richtigen Stellen). Jedoch auch diese Möglichkeiten und eine möglicherweise darüber hinaus gehende zeitliche Koordination der betroffenen Workflows sollen im nächsten Kapitel diskutiert werden.



a) Direkte Umleitung der Zugriffe b) Indirekte Umleitung der Zugriffe

Abbildung 6 Ansätze zur Zugriffsumleitung

- Das objektweise Kopieren bei Anforderung muss vom eingesetzten PDMS unterstützt werden. Hierzu ist ein spezielles Datenobjekt als Referenz notwendig, das alle Informationen zum Lokalisieren der eigentlichen Produktdaten enthält. Bei Zugriff auf dieses Objekt wird ein Kopiermechanismus in Gang gesetzt, durch den die Referenz expandiert wird. Das bei der Kooperationsdatenspezifikation beschriebene referenzierte Übertragen von Anhängen ist nur bei PDMS möglich, die objektweises Kopieren bei Anforderung unterstützen, d. h., das Datenmodell muss sich um einen entsprechenden Referenztyp erweitern lassen. Andernfalls muss vollständig kopiert werden.

Bei den Ansätzen der Zugriffsumleitung (siehe Abbildung 6) gehen wir davon aus, dass die Kooperationsdaten im Quell-PDMS verbleiben, dem Ziel-PDMS jedoch Wege eröffnet werden, auf diese Daten im entfernten PDMS zuzugreifen [AGL98]. Abbildung 6a illustriert den Ansatz

der direkten Umleitung der Zugriffe der Ziel-Applikation auf das entfernte Ziel-PDMS. Man beachte, dass hier lediglich die Zugriffe auf Kooperationsdaten umgeleitet werden müssen; eventuelle Zugriffe auf lokale Daten können selbstverständlich im selben Verarbeitungsvorgang auch auftreten. Bei dem in Abbildung 6b angedeuteten Ansatz gehen alle Zugriffe der Ziel-Applikation zunächst auf das Ziel-PDMS. Dort wurden jedoch vor Verarbeitungsbeginn sogenannte Proxy-Objekte für bestimmte Teile der Kooperationsdaten angelegt, die bei Zugriff eine entsprechende Weiterleitung zum Quell-PDMS vornehmen. Vorteil dieser Lösung ist natürlich, dass die Zugriffe der Ziel-Applikation in gewohnter Weise an das Ziel-PDMS abgesetzt werden können, während bei der direkten Umleitung der Zugriffe eine (in Abbildung 6a durch den grauen Balken angedeutete) Zusatzschicht zwischen Applikation und PDMS eingezogen werden muss, welche die einzelnen Zugriffe korrekt routet.

Natürlich bringen auch diese Ansätze eine Reihe von Problemen mit sich, die im Folgenden kurz angesprochen und im nachfolgenden Abschnitt weiter detailliert werden sollen.

- Es müssen Mechanismen der Umsetzung von Zugriffen über die API des Ziel-PDMS in semantisch äquivalente Zugriffe über die API des Quell-PDMS transformiert werden.
- Die aus dieser Transformation resultierenden Zugriffe müssen auf dem Quell-PDMS initiiert werden. Bei diesen Aufrufen müssen also Grenzen von Systemumgebungen überschritten werden, was bei heute üblichen Sicherheitsvorkehrungen (z. B. Firewalls) sehr schwierig sein kann.
- Da die Kooperationsdaten auf dem Quellsystem verbleiben, sind alle Anhänge quasi referenziert. CAD-Geometrien beispielsweise werden erst als Ergebnis eines entsprechenden (umgeleiteten) Zugriffs zum Zielsystem übertragen, was zu langen Wartezeiten auf Benutzerseite führen kann.

3.2.4 Zugriffskontrolle

Da Produktdaten meist sicherheitskritische Information beinhalten, darf natürlich deren Schutz gegen unerlaubten Zugriff nicht außer Acht gelassen werden. Zum einen dürfen die vergebenen Rechte durch den entfernten Leser nicht verletzt werden. Weiterhin muss sichergestellt werden, dass der Empfänger der Daten auch wirklich derjenige ist, für den die Daten bestimmt sind. Darüber hinaus ist die Übertragung durch eine entsprechende Kodierung abzusichern.

Nachdem wir nun in diesem Abschnitt die eigentlichen Dimensionen des Problems der automatisierten Handhabung von Datenflussabhängigkeiten näher erläutert haben, wollen wir in den nachfolgenden Abschnitten detaillierter diskutieren, inwieweit die Workflow-Schicht zur Steuerung der Datenbereitstellung eingesetzt werden kann.

4. Interoperabilität durch Workflow-Anpassung

Wie wir schon in Abschnitt 2 beschrieben haben, bestehen unsere Inselsysteme aus mehreren Schichten, einer PDMS-Schicht, einer Aktivitätenschicht und einer Wf-Schicht. Nachdem wir im letzten Abschnitt die PDMS-Schicht näher betrachtet haben, wollen wir nun untersuchen, welche Automatisierungsmaßnahmen darauf aufbauend in der Wf-Schicht vorgenommen werden können. Dabei gehen wir davon aus, dass an den bisher eingesetzten Workflow-Aktivitäten keine Veränderungen vorgenommen werden sollen. Durch den Workflow wird nur das initiale

Bereitstellen der Kooperationsdaten gesteuert, d. h., die in der Kooperationsdatenspezifikation beschriebenen Daten werden durch Datenübertragung bereitgestellt. Die weitere Behandlung, etwa das Auflösen von Referenzen, liegt wie oben beschrieben in der Verantwortung des eingesetzten PDMS.

4.1 Koordination von Datenflussabhängigkeiten

Wir haben bereits in Abschnitt 2.3 die Rolle des Koordinators bei der Behandlung von Datenflussabhängigkeiten kennengelernt. Um die Kooperationsdaten zu übertragen, muss die Quellseite um eine Sende- und die Zielseite um eine Empfangskomponente erweitert werden, die wir als *Import*- bzw. *Export*-Aktivität bezeichnen. Diese Aktivitäten bestehen jeweils aus zwei Teilen: einer Kommunikationsapplikation (Sender/Empfänger), die für die Datenübertragung zuständig ist, und der eigentlichen Import- bzw. Export-Applikation, die mit dem lokal genutzen PDMS zusammenarbeitet. Dabei wird durch den Koordinator sichergestellt, dass die richtigen Wf-Instanzen miteinander kommunizieren, indem der Koordinator der Export-Aktivität (Sender) die Adresse der zugehörigen Import-Aktivität (Empfänger) mitteilt. Im synchronen Fall der Datenübertragung stellt der Sender anschließend eine Verbindung zum Empfänger her, die Kooperationsdaten können dann (ohne weiteres Eingreifen des Koordinators) übertragen werden (Abbildung 7). Ein Problem dieser synchronen Übertragung entsteht jedoch dadurch, dass beide Workflow-Instanzen die jeweilige Aktivität schon erreicht haben müssen, der schnellere muss folglich auf sein Gegenstück warten. Um das Blockieren des gesamten Workflows bei dieser notwendigen Synchronisation zumindest auf Quellseite zu verhindern, gibt es verschiedene Möglichkeiten. Die Export-Aktivität kann in dem lokalen Workflow in einem Parallelzweig modelliert werden. Alternativ kann sie aber auch in einen eigenen Mini-Workflow ausgelagert werden, der nur aus dieser Aktivität besteht. Dieser Workflow wird vom Koordinator separat gestartet und kann blockieren, ohne die weitere Ausführung von WF1 zu behindern. Auf Zielseite sind solche Optimierungen nicht möglich, da der Workflow bis zum Bereitstehen der Daten und der anschließenden Ausführung der zugehörigen Applikation auf jeden Fall blockiert werden muss.

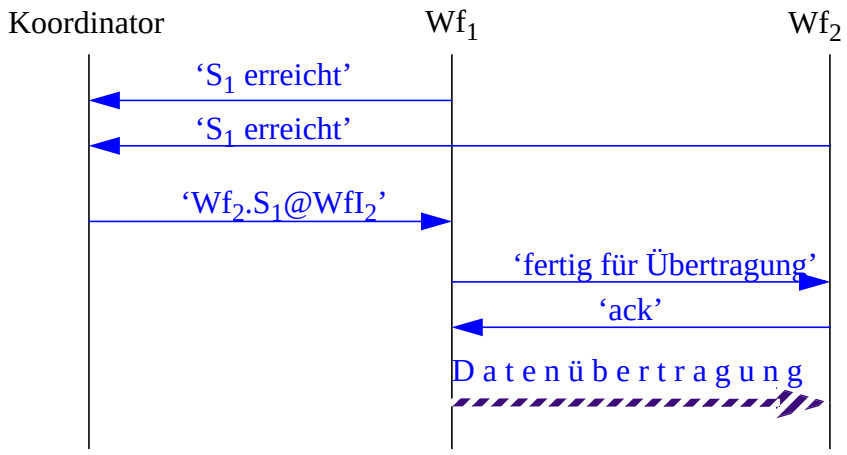


Abbildung 7 Verbindungsaufbau zwischen Wf-Instanzen mit Hilfe des Koordinators

Etwas einfacher gestaltet sich der asynchrone Fall. Sobald WF_1 den die Export-Aktivität erreicht hat, wird dem Sender die Zieladresse mitgeteilt, beispielsweise die Adresse einer zugeordneten Ausgangs-Nachrichtenschlange. Die Kooperationsdaten werden sofort dorthin übermittelt. Durch die Entkopplung von Sender und Empfänger ist kein Warten auf den Kommunikationspartner notwendig. Auf der Empfängerseite sieht dies natürlich anders aus. Erreicht Wf_2 die Importaktivität vor Eintreffen der Kooperationsdaten, wird die Empfangsaktivität blockiert, bis die Daten bereit liegen. Liegen diese vor, teilt der Koordinator dem Empfänger mit, in welcher Eingangs-Nachrichtenschlange er sie findet.

Welche Anforderungen an die eigentlichen Import-/Export-Aktivitäten gestellt werden, ist nun Gegenstand der folgenden Abschnitte. Diese unterscheiden ausgehend von den oben eingeführten Dimensionen Fälle unterschiedlicher Komplexität.

4.2 Lesender Zugriff bei gleichartigen PDMS

Zunächst wollen wir uns auf den einfachsten Fall beschränken, in dem sich auf dem Quell- und dem Zielsystem das gleiche PDMS befindet. Da auf beiden Seiten die Daten in gleicher Art und Weise modelliert sind (gleiches Schema), ist für die Integration ins Zielsystem keine Konvertierung vorzunehmen. Anderenfalls wäre eine Konversion und Abbildung gemäß dem Zielschema (für eine homogene Systemumgebung) notwendig.

Bei der Datenübertragung werden die Kooperationsdaten vom Quell- zum Zielsystem kopiert. Die Verantwortung für das Anstoßen der Übertragung liegt dabei beim Quell-WfMS. Um dies zu erreichen, wird die Datenflussabhängigkeit zwischen den Wf-Aktivitäten explizit durch Erweitern der vorhandenen Wf-Schemata durch einen expliziten Export- (WfA_{11E}) bzw. Import-Schritt (WfA_{21I}) modelliert (Abbildung 8).

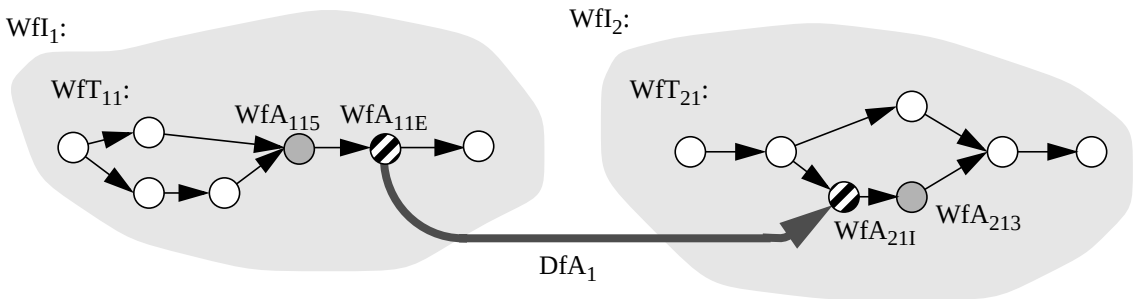


Abbildung 8 Anpassung der Workflows auf Typebene

Auf Ausprägungsebene ergibt sich daraus die in Abbildung 9 gezeigte Situation. Der Export-Schritt des Quell-Workflows Wf_1 (WfA_{11E}) besteht im Aufruf der Applikation *Export*, im Ziel-Workflow Wf_2 in WfA_{11I} entsprechend im Aufruf der Applikation *Import*. Diese Applikationen sind sehr einfach, da sie die PDMS-API direkt zum Erzeugen bzw. Importieren einer Transportdatei einsetzen können. Dabei benutzt *Export* die Kooperationsdatenspezifikation, um relevante Daten zu identifizieren. Durch den Einsatz der Workflows wird auf beiden Inseln der Datentransfer durch rechtzeitiges Starten der beiden Aktivitäten eingeleitet. Auf Seite von WfI_1 wird von *Export* eine Transportdatei in proprietärem Format erzeugt, wie schon beschrieben mit Hil-

fe der Sender- und Empfänger-Applikationen zu WfI_2 übertragen und dort von *Import* in das lokale $PDMS_2$ eingefügt. Erst danach wird von Wf_2 $Applikation_2$ zur Bearbeitung vorgesehen, die nun lokal auf die Kooperationsdaten zugreifen kann.

Export kann erfreulicherweise auch sehr generisch gestaltet werden. Da die Zugriffe sehr einfach sind und immer in gleicher Form erfolgen, genügt die Kooperationsdatenspezifikation (plus Identifikationsparameter zur Auswahl des gewünschten Produkts, der von der erzeugenden Aktivität als Kontrolldatum bereitgestellt wird) als Eingabe, um eine geeignete Transportdatei zu erzeugen. Durch den rein lesenden Zugriff sind auf WfI_1 nach dem Kopieren keine weiteren Synchronisationsmaßnahmen notwendig. Auch Anhänge in Form von referenzierten Objekten bereiten hier, auf Grund der von den PDMS unterstützten Versionsverwaltung, keine Probleme, da die lokal gültige Version auf dem Quell-System gegen Veränderungen geschützt werden kann.

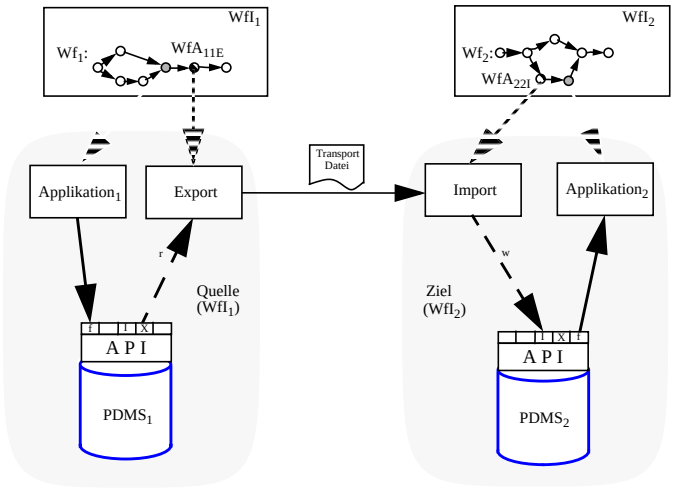


Abbildung 9 Datenübertragung bei gleichartigen PDMS

4.3 Statische und dynamische Kooperationsdatenspezifikation

Wir haben bereits erörtert, wie Kooperationsdaten durch eine entsprechende Spezifikation beschrieben werden können. Dabei sind wir stillschweigend davon ausgegangen, dass diese Spezifikation nur einmal erstellt werden muss und damit das Granulat der Kommunikation zwischen den Aktivitäten hinreichend beschrieben ist. Leider ist dieser *statische* Fall in der Praxis eher unwahrscheinlich. Vielmehr müssen wir davon ausgehen, dass sich auf dem Quellsystem nicht nur die Produktdaten ändern, sondern auch die Anforderungen an den Umfang der zu übertragenden Daten. Beispielsweise werden in einer frühen Entwicklungsphase nur sehr grobe Daten verlangt, mit steigender Iterationsanzahl werden jedoch auch detailliertere und ausführlichere Informationen benötigt. Dies wiederum bedeutet, dass erst kurz vor Übertragen der Daten der genaue Umfang der Kooperationsdaten bekannt ist und die Spezifikation erstellt werden kann. Wir bezeichnen dies als *dynamische Kooperationsdatenspezifikation*. Anpassungen können bis zum Starten des Exportschrittes vorgenommen werden, müssen aber auch spätestens dann abgeschlossen sein. Leider ist an dieser Stelle aus den schon zuvor beschriebenen Gründen keine Automatisierung möglich, das Anpassen der Spezifikation muss manuell durchgeführt werden.

4.4 Import- und Exportapplikationen

Wie wir schon in Abschnitt 3.1 erläutert haben, wird der Umfang der zu übertragenden Kooperationsdaten durch eine vorzugebende Spezifikation beschrieben. Diese dient auch als Ausgangspunkt für die Erzeugung der Transportdatei. Die in der Spezifikation identifizierten Objekte müssen dabei aus dem PDMS ausgelesen und in ein Transportformat umgewandelt werden. Es sind dabei verschiedene Vorgehensweisen möglich:

- Viele PDMS unterstützen schon von Haus aus ein Exportieren von Objekten in einem proprietären Format. Dies kann in einer Exportaktivität ausgenutzt werden, die aus 3 Einzelschritten besteht:
  1. Erzeugen einer (proprietären) Extraktionsdatei mit dem gesamten Produktteilbaum (ohne Anhänge);
  2. Aufruf eines Filters, der die entstandene Datei mit den Spezifikationsdaten abgleicht. Die Ergebnisdatei wird in dem für die Übertragung vereinbarten Format (beispielsweise XML) erzeugt;
  3. Kontaktaufnahme eines Sendeprogramms mit dem Koordinator und Verschicken der Transportdatei (siehe 2) an den entsprechenden Empfänger.

  Der kritische Teil dieser ersten Lösungsmöglichkeit ist der im zweiten Schritt eingesetzte Filter. Dieser muss die Fähigkeit haben, zwischen drei verschiedenen Formaten zu übersetzen, dem Exportformat des PDMS, dem Kooperationsdatenspezifikationsformat und dem Ausgabeformat. Ändert sich eines dieser Formate, muss der Filter neu geschrieben werden. Die Information, wie Exportdatei und Spezifikation miteinander strukturell abgeglichen werden können, steckt fest im Programmcode. Ein großer Vorteil dieser Lösung ist natürlich, dass das Filterprogramm direkt auf den Eingabedateien arbeiten kann und somit nicht für jede Datenflussbeziehung ein eigenes Programm generiert werden muss.

- Es kann auch auf die SQL-Schnittstelle zugegriffen werden. Da fast alle PDMS zur Datenhaltung auf relationale Datenbanksysteme zurückgreifen, ist auch dieser Weg meistens möglich. Basierend auf der in der Kooperationsdatenspezifikation beschriebenen Struktur kann eine Reihe von SQL-Statements generiert werden, welche die gewünschten Objekte aus der PDMS-Datenbank ausliest (beispielsweise „SELECT PartNumber, Revision, ... FROM part WHERE Name = ‘Baugruppe1’;“, um das Wurzelobjekt aus Abbildung 3 auszulesen).
  Das Datenbankschema muss dem Programmierer bekannt sein, um ein Navigieren in den Strukturen vornehmen zu können. In der Spezifikation aus dem Beispiel wird Modul 2 beschrieben, das Teil von Baugruppe 1 ist. Diese Information fließt entsprechend in das generierte SQL-Statement ein, z. B.:

  ```
  SELECT P.PartNumber, ...
  FROM PART P, ASSEMBLY A
  WHERE P.Name=’Modul 2’
  AND A.SubPartNumber=P.SubPartNumber
  AND A.SuperPartNo=:SuperNumber
  ```

- wobei für ‘:SuperNumber’ die PartNumber von ‘Baugruppe 1’ eingesetzt wird, die schon aus der Auswertung der vorherigen Ebene bekannt ist. Da sich die Struktur rekursiv auf jeder Ebene wiederholt, sind die zu generierenden SQL-Statements immer gleich aufgebaut. Das Wissen über das eingesetzte Schema wird im Programm fest abgebildet (z. B., welches XML-Tag in der Spezifikation welche Tabelle widerspiegelt oder wie *assemblies* abgebildet werden, um die Produktstruktur weiter zu verfolgen).
- Eine weitere Möglichkeit besteht aus einer Mischung der beiden vorgenannten Ansätze. Mit Hilfe der SQL-Schnittstelle wird eine Darstellung des kompletten Strukturbaums erzeugt (z. B. unter Zuhilfenahme des Document Object Models (DOM) [W3C01]). Diese kann sehr effizient bearbeitet und auf den zu übertragenden Umfang der Daten reduziert werden. Die SQL-Anfragen zum Erzeugen des Produktbaums müssen nur einmal erzeugt werden, das entstandene Programm kann anschließend für verschiedene Spezifikationen (und somit Datenflusskanten) eingesetzt werden. Das Wissen über das zugrunde gelegte Schema steckt in den fest codierten SQL-Anweisungen.

Die Erstellung eines Programms auf Seite des Ziel-Systems für das Importieren der Daten wird durch die Tatsache erleichtert, dass kein weiteres Filtern der Daten notwendig wird. Die Import-Applikation muss ‘lediglich’ die übertragene Transportdatei auf die Bedürfnisse des lokalen PDMS abbilden. Dies kann wiederum über die PDMS-API geschehen oder über SQL direkt auf der Datenbank. Im zweiten Fall muss die Programmlogik dazu in der Lage sein, die Struktur-Informationen und XML-Tags mit Hilfe entsprechend ‘on the fly’ generierter INSERT-Statements abzubilden. Gelingt dies, so kann dieses Import-Programm ohne weitere Anpassung für alle Datenflussabhängigkeiten (zu diesem speziellen PDMS) benutzt werden. Falls mehrere PDMS auf dem Zielsystem vorhanden sind, muss *Import* weiterhin das richtige PDMS auswählen können (beispielsweise über Parameter).

4.5 Zugriffskontrolle

Beim entfernten Zugriff durch das Zielsystem ist dafür zu sorgen, dass die Zugriffsrechte auf den Quelldaten nicht verletzt werden. Die zur Datenübertragung eingesetzten Kommunikationskomponenten für Im- und Export müssen dabei nicht nur selbst mit entsprechenden Rechten ausgerüstet werden, sie müssen auch dafür sorgen, dass nach der physischen Übertragung die Rechte im Zielsystem korrekt gesetzt sind. Durch die Isolation der Inseln gibt der Besitzer der Daten die Verantwortung für die weitere Verwendung der Kooperationsdaten komplett an das Zielsystem ab, er hat nach dem Übertragen praktisch keine weitere Möglichkeit zu kontrollieren, von wem die Daten denn nun tatsächlich gelesen werden können. Dies bedeutet, dass zwischen den Partnern ein großes Maß an Vertrauen herrschen muss. Die erzeugende Applikation_1 auf dem Quellsystem hat natürlich volle Lese- und Schreibrechte. *Export* benötigt nur Leserechte, um die Kooperationsdaten zur weiteren Bearbeitung (wie im letzten Abschnitt beschrieben) auszulesen. Die von *Export* erzeugte Transportdatei wird anschließend kodiert und zum Empfänger übertragen.

Zur Authentifizierung/Kodierung bieten sich Verfahren mit asymmetrischen Schlüsseln wie beispielsweise ‘Pretty Good Privacy’ (PGP) an ([Zim95]) . Denkbar wäre die Generierung eines ‘projektweiten Schlüsselbundes’. Jede Empfängerinsel verwaltet ihren Privatschlüssel selbst, die Verwaltung der öffentlichen Schlüssel kann als weitere Funktionalität des Koordinators an-

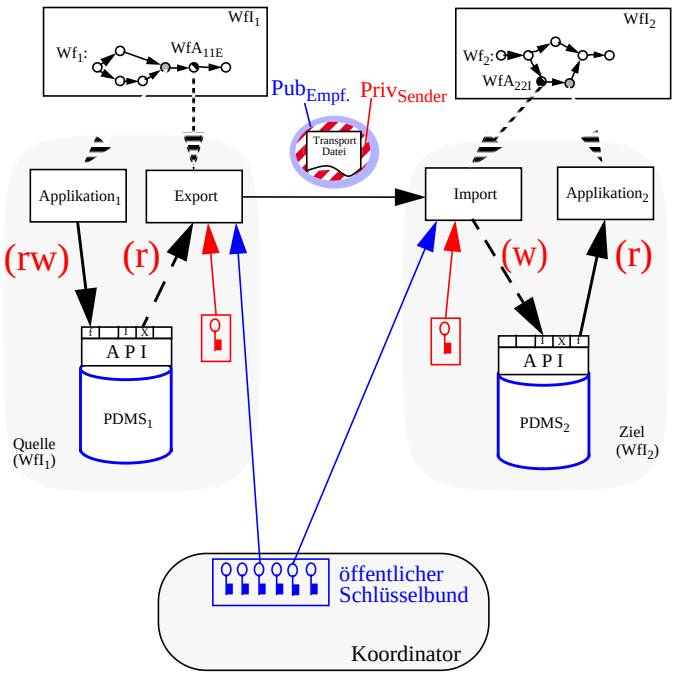


Abbildung 10 Sicherheitskonzept

geboten werden. Vor dem Versenden der Kooperationsdaten erfragt der Sender vom Koordinator den öffentlichen Schlüssel des Empfängers. Nach Verschlüsseln der Transportdatei ist nur der Empfänger dazu in der Lage, diese wieder zu entschlüsseln. Soll auch der Sender eindeutig identifiziert werden können, wird die Transportdatei mit dem privaten Schlüssel des Senders signiert. Der Empfänger kann diese mit dem (vom Koordinator gelieferten) öffentlichen Schlüssel des Senders entschlüsseln und somit die Herkunft der Daten verifizieren (Abbildung 10).

Der private Schlüssel muss ebenfalls geschützt werden. Hier kann jedoch ausgenutzt werden, dass nur die Sender- bzw. Empfängerkomponenten darauf zugreifen müssen, nicht jedoch die Projektmitarbeiter selbst. Der Schlüsselzugriff kann somit sehr restriktiv gehandhabt werden.

Ein Nachteil dieses Verfahrens ist sicherlich der Zeitaufwand. Gerade bei großen Dateien entsteht durch das zusätzliche Kodieren ein nicht zu unterschätzender Zeitbedarf. Daher ist je nach Sicherheitsrelevanz der Daten zu prüfen, ob wirklich die volle Kodierung notwendig ist oder ob nur Teile davon eine Verschlüsselung benötigen.

Nach der Übertragung in das Ziel-PDMS müssen noch die Rechte auf die Kooperationsdaten angepasst werden. Hierzu braucht *Import* Informationen über die Anwendung, die auf die Kooperationsdaten zugreift, insbesondere über die zugeordnete Rolle, die zur Identifikation der ausführenden organisatorischen Einheiten (OE) dient. Da wir hier vorerst nur den lesenden Fall betrachten, reicht es, Lesezugriff für die entsprechenden OEs einzurichten.

Eine kritischer Punkt ist das Besitzrecht für die Daten. Hierzu gibt es im wesentlichen zwei Möglichkeiten. Zum einen kann für alle importierten Daten grundsätzlich der gleiche, speziell hierfür definierte Benutzer (z. B. ‘Importeur’) eingetragen werden. Diese Lösung ist sehr einfach umzusetzen, aber auch entsprechend unflexibel. Zum anderen kann das Besitzrecht an den

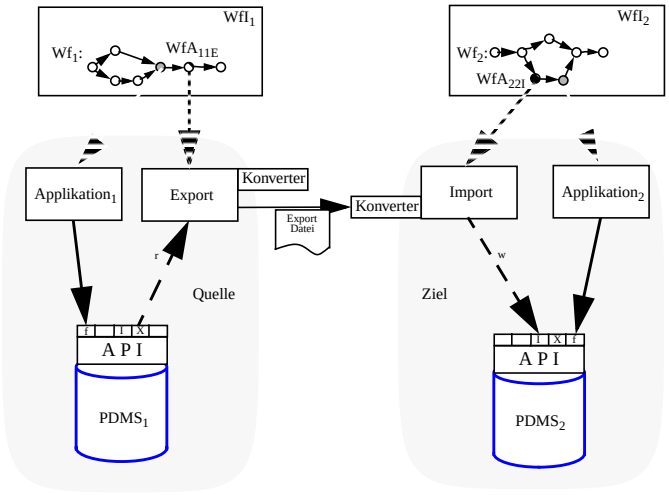


Abbildung 11 Datenübertragung in proprietärem Format mit Konvertierung

Benutzer (oder Rolle) übergehen, der die anschließend auf die Kooperationsdaten zugreifende Aktivität ausführt. Dazu muss frühzeitig eine Zuordnung des Benutzers zu der Aktivität stattfinden (vor Durchführen des Imports) und der Organisationsmanager des eingesetzten WfMS entsprechende Anfragen unterstützen.

4.6 Lesender Zugriff bei verschiedenartigen PDMS

Als nächstes wollen wir uns dem Fall zuwenden, dass auf den beteiligten Inseln verschiedene PDMS zum Einsatz kommen, sich somit die PDMS-Produkte, Datenmodelle und Schemata unterscheiden. Durch die unterschiedlichen APIs müssen die Kooperationsdaten in angemessener Form konvertiert werden. Wie schon beim Fall gleichartiger PDMS liegt auch hier das Durchführen des Datentransfers in der Verantwortung des WfMS. Die Anpassungen auf Typ-Ebene entsprechen dabei den schon in Abbildung 8 dargestellten Erweiterungen.

Etwas komplizierter sieht es auf der Ausprägungsebene aus. Da Quell- und Ziel-PDMS verschiedene Datenmodelle benutzen, muss die Struktur der übertragenen Daten in eine vom Zielsystem $PDMS_2$ akzeptierte Form konvertiert werden. Nach dem Einspeichern in $PDMS_2$ stehen die Daten als inseleigene Kopien zur Verfügung und Veränderungen auf dem Quellsystem $PDMS_1$ haben nach dem Kopieren keinerlei Auswirkungen mehr. Da keine Änderungen vorgenommen werden sollen, brauchen für $PDMS_1$ auch keine weiteren Maßnahmen vorgesehen zu werden. Die Konvertierung der Daten kann auf verschiedene Art und Weise geschehen. Zum einen können die APIs der Systeme benutzt werden. Falls die von $PDMS_1$ angebotene Exportfunktionalität nur ein proprietäres Format liefert, muss dieses in ein für $PDMS_2$ verständliches Format konvertiert werden. Dieses Importformat kann ebenfalls wieder proprietär sein (Abbildung 11). Die Umwandlung vom Quellformat zum Zielformat kann dabei wahlweise auf der Quell- oder der Zielinsel durchgeführt werden. Falls entsprechende Konvertierungsroutinen nicht von den Herstellern vorgesehen sind, sollte auf diese Vorgehensweise verzichtet werden. Die Anzahl der benötigten Konvertierungswerkzeuge wächst quadratisch mit der Anzahl der Systeme, so dass eine Eigenentwicklung viel zu aufwendig ist.

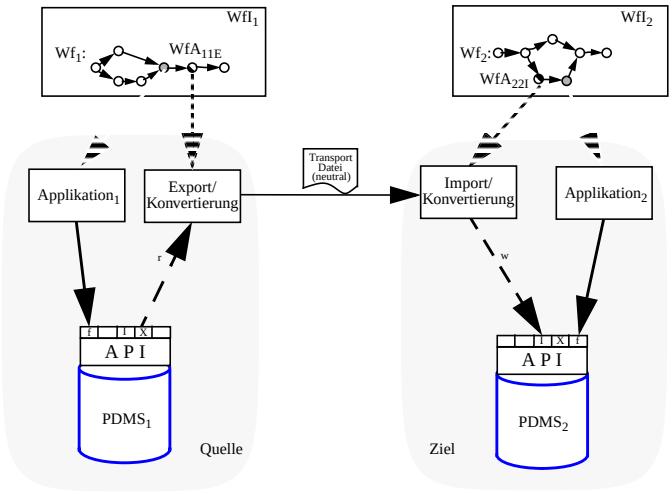


Abbildung 12 Datenübertragung mit Transportdatei in neutralem Zwischenformat

Besser verhält sich hier der in Abbildung 12 illustrierte Ansatz. Dabei werden die Kooperationsdaten in ein neutrales Zwischenformat abgebildet, das beispielsweise XML-basiert sein kann. Beim PDM-Enablers-Ansatz der OMG ist der Export in ein STEP-basiertes Zwischenformat schon in der API vorgesehen. Falls das DBMS dies von Haus aus jedoch nicht unterstützt, muss wieder ein Konverter eingesetzt werden. Wird dieser vom Hersteller nicht angeboten, muss auf eine Eigenentwicklung zurückgegriffen werden. Da dies jedoch pro System nur einmal zu erfolgen hat, ist der erforderliche Aufwand vertretbar.

Ein in diese Richtung zielender Ansatz ist beispielsweise PDML (Product Data Markup Language), das für das amerikanische Verteidigungsministerium (Departement of Defense, DoD) entwickelt wurde [Bur99]. Hierbei wird für jedes beteiligte PDMS ein *Transaction-Set*-Vokabular erstellt, mit dessen Hilfe die Produktdaten in ein XML-Dokument abgebildet werden können. Dieses XML-Dokument wird über eine definierte Abbildung in ein neutrales *Integration Schema* überführt. Für das Zielsystem existiert ebenfalls ein solches XML-Vokabular, in das wiederum vom *Integration Schema* aus abgebildet werden kann.

Eine dritte Möglichkeit ist der Einsatz spezieller Applikationen, die (z. B. aus Geschwindigkeitsgründen [MDJF01]) die API des PDMS umgehen und direkt auf das zugrunde liegende DBMS zugreifen (Abbildung 13). Dadurch ergeben sich auch Möglichkeiten, die Kooperationsdaten vor der Übertragung auf ein anderes System zu manipulieren. So können beispielsweise sicherheitskritische Teile verdeckt werden, ohne relevante Informationen zu zerstören [Naw01]. Fast allen PDMS liegt ein relationales DBMS zugrunde. Auf die gespeicherten Produktdaten kann somit, bei bekanntem Schema, direkt über die SQL-Schnittstelle zugegriffen werden, wodurch ein deutlicher Geschwindigkeitsgewinn erreicht werden kann. Als neutrales Format für die Transportdatei bietet sich XML an, da sich Produktdaten durch die ebenfalls baumartige Struktur eines XML-Dokuments gut beschreiben lassen. In [W3C97] wird ein Verfahren zur Abbildung zumindest einfacher relationaler Schemata in eine XML-Struktur vorgestellt. Mit Hilfe solcher Abbildungsverfahren lässt sich die Transportdatei ohne Zuhilfenahme der PDMS-API erstellen.

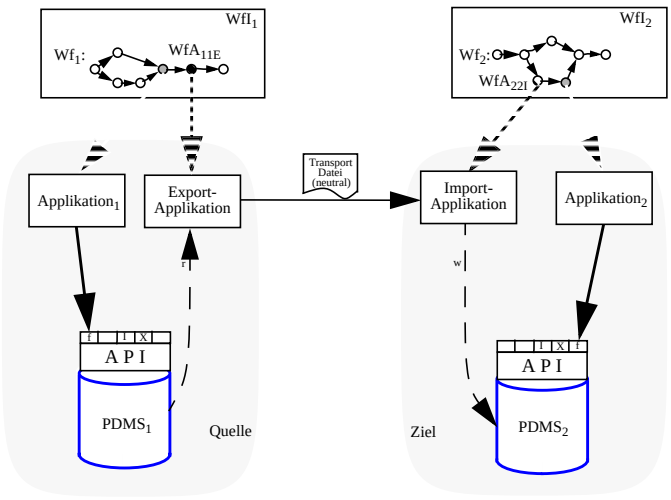


Abbildung 13 Datenübertragung in neutralem Zwischenformat ohne Nutzung der PDMS-API

4.7 Lesender und schreibender Zugriff bei gleichartigen PDMS

Nachdem wir uns mit dem rein lesenden Zugriff beschäftigt haben, soll nun betrachtet werden, wie Änderungen auf der Zielinsel ins Quellsystem zurückpropagiert werden können. Wir beschränken uns dabei zunächst auf den Fall, dass die Kooperationsdaten auf dem Quellsystem nicht gleichzeitig verändert werden.

Interessant sind hier vor allem die möglichen und notwendigen Änderungen auf Wf-Typebene. Auf Ausprägungsebene werden die Import-/Export-Mechanismen analog zum Lesefall eingesetzt. Es sind hier lediglich beide Richtungen zu berücksichtigen, d. h., neben der Übertragung der Daten von Quell- auf die Ziel-Insel ist nach Beendigung der Ziel-Applikation eine Übertragung in der umgekehrten Richtung notwendig.

Ein erster naiver Ansatz für die notwendigen Erweiterungen auf Typebene ist in Abbildung 14 zu sehen. Auf beiden Inseln werden die Workflows um zwei Schritte erweitert, die jeweils notwendige Übertragung der Kooperationsdaten durchführen. Vorteilhaft bei diesem Ansatz ist sicherlich, dass die veränderten Daten möglichst schnell wieder in das für sie „verantwortliche“ System WfI_1 zurückgeführt werden. Allerdings ist dies mit einer unter Umständen langen Blockierung von Workflow WfT_{11} zu bezahlen, da erst das Ende der Wf-Aktion WfA_{213} abgewartet werden muss.

Um diese Blockade-Situation zu verhindern, kann WfA_{11I}, die für den Rückimport zuständige Wf-Aktion, in einen Parallelzweig ausgelagert werden, so dass beispielsweise, wie in Abbildung 15 dargestellt, WfA_{116} und WfA_{117} nicht blockiert werden. Erst WfA_{118} benötigt die aktuellen Daten und muss warten, bis diese in der neuen Version vorliegen.

Bei dieser Art der Modellierung entsteht das Problem der „wechselnden Zuständigkeit“. Obwohl die Kooperationsdaten eigentlich zu WfI_1 „gehören“, übernimmt plötzlich WfI_2 die Aufgabe der Bereitstellung.

Beide Lösungen sind offensichtlich nicht sonderlich zufriedenstellend. In der Praxis wird dieses

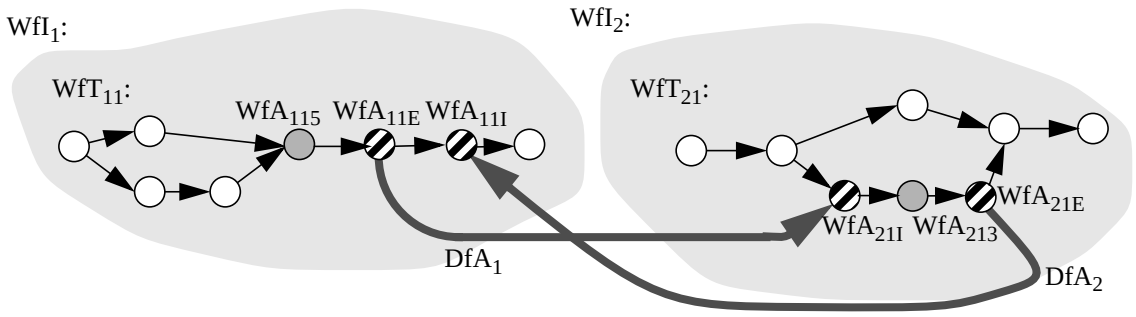


Abbildung 14 Sofortige Propagierung der Änderung an das Quellsystem

Problem meist umgangen, indem direkte Änderungen nicht zugelassen werden. Die Datenzuständigkeiten sind fest zugeordnet, und nur der für sie Verantwortliche darf Änderungen durchführen. Änderungswünsche müssen als Anforderungen an diesen gerichtet werden. Nach (lokaler) Durchführung der Änderung wird dann die neue Version auf der Quellinsel freigegeben.

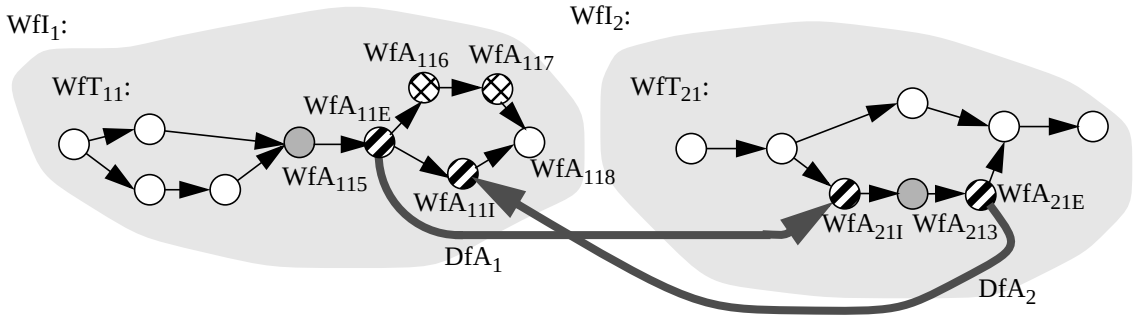


Abbildung 15 Auflösen der Blockadesituation durch Einführen eines Zusatzzweiges

Wir wollen trotzdem kurz skizzieren, welche Vorkehrungen getroffen werden müssen, um beim Propagieren von Änderungen die betroffenen Produktdaten konsistent zu halten. Im wesentlichen gibt es zwei Möglichkeiten: einen globalen Synchronisationsmechanismus und den Einsatz einer Versionsverwaltung. Bei letzterer kann weiter verfeinert werden in lineares Fortschreiben von Versionen und graphartige Verwaltung des Ableitungsbaums.

Obwohl der Einsatz von Sperren zum Vermeiden von Anomalien bei Datenbanken üblich ist, scheint dieser Ansatz in unserem Fall jedoch wenig erfolgversprechend zu sein. Zum einen bräuchten wir ein nicht ganz einfach zu realisierendes globales Sperrprotokoll. Alle beteiligten PDMS müssten in der Lage sein, an diesem Protokoll teilzunehmen, was bei den aktuellen Systemen nicht möglich ist. Zum anderen betrachten wir Zugriffe, die sehr viel Zeit benötigen. Das exklusive Sperren von Daten, die auch von anderen Entwicklern zumindest lesend benötigt werden, über einen so langen Zeitraum ist schlicht und ergreifend in der Praxis nicht tragbar.

Vielversprechender scheint der Einsatz von Versionierungsmechanismen zu sein. Falls eine oder auch mehrere entfernte Aktivitäten verändernd auf Produktdaten zugreifen sollen, so wird die gelesene Version „eingefroren“ (z. B. als „Version 1“), steht jedoch weiterhin zum Lesen zur Verfügung. Änderungen auf der Quell-Insel müssten nun als neue Version (Nachfolgerversion der eingefrorenen Version) eingebracht werden. Hier ist nun zu beachten, ob ein lineares oder ein baum- bzw. graphartiges Fortschreiben von Versionen vorgesehen ist. Im Falle einer linearen Fortschreibung muss das Einfügen einer Nachfolgerversion auf der Quellseite verboten werden, damit nach Beendigung der Ziel-Applikation die dort erzeugten Daten als neue Version auf der Quellseite eingebracht werden können. Andernfalls müssten die auf Quell- und Zielseite unabhängig voneinander erzeugten Versionen vor dem Eintragen in das Quell-PDMS miteinander abgeglichen und zu einer Version verschmolzen werden, was im Allgemeinen nicht automatisch möglich ist.

Zum Funktionsumfang der meisten PDMS gehört jedoch auch das Verwalten verschiedener Versionen eines Produkts (Produktstruktur- und Konfigurationsmanagement, [VDI99]). Es können somit auch Unterversionen erzeugt werden, wodurch ein Versionsbaum entsteht. Nach Übertragen der Kooperationsdaten können diese lokal manipuliert und anschließend zurücktransferiert werden. Das Re-Importieren besteht dann aus dem Anlegen einer neuen Unterversion (beispielsweise „Version 1.1“). Werden Änderungen durch eine weitere Insel-Aktivität vorgenommen, so liest diese ebenfalls die ursprüngliche Version (1.0) und propagiert ihre Änderungen durch Anlegen einer weiteren Unterversion (1.2). Ein automatisches Zusammenführen der Änderungen wäre wünschenswert, doch auch hier ist wohl menschliches Eingreifen von Nöten. Zum einen muss auch hier beurteilt werden, welche der entfernt durchgeführten Änderungen wirklich in das Produkt einfließen sollen. So ist evtl. eine Nachbesserung durch einen Ingenieur auf Seite des Quellsystems notwendig. Weiterhin können die bearbeiteten Teilmengen nicht disjunkt sein, d. h., das gleiche Teil wurde von verschiedenen Stellen verändert. Derartige Konflikte müssen sinnvoll aufgelöst werden. Wurden alle neuen Versionen zu einem neuen Produkt zusammengefügt, kann eine neue Hauptversion freigegeben werden (2.0), die wiederum exportiert und auf gleiche Art und Weise bearbeitet werden kann.

Die zuletzt vorgestellte Variante scheint den gangbarsten Weg darzustellen. Durch das Bilden von Versionen kann eine Blockadesituation zumindest für Leser, die mit der letzen gültigen Version zufrieden sind, verhindert werden. Weiterhin können entfernt vorgenommene Änderungen zunächst als Vorschlag betrachtet werden, der sich in einer separaten Unterversion widerspiegelt. Leider ist das Ableiten einer neuen Version aus den gesammelten Änderungen auch in diesem Fall nur schlecht zu automatisieren und erfordert den Einsatz von Experten. Der Zeitpunkt für das Erstellen einer neuen Version ist auch mit Bedacht zu wählen. Wird zu oft integriert, nähern wir uns dem linearen Fall an. Werden die Änderungen zu selten nachgezogen, so steigt zum einen der Integrationsaufwand, zum anderen spiegeln die freigegebenen Versionen nicht den jeweils aktuellen Stand wider.

5. Interoperabilität durch PDMS-Erweiterung

Wie bereits erläutert, sieht die Zugriffsumleitung (sowohl der direkten Umleitung als auch beim Proxy-Objekt-Ansatz) vor, Funktionsaufrufe von der Ziel-Insel an das Quell-PDMS weiter zu leiten. Bei der direkten Umleitung ergibt sich jedoch das Problem, dass für den Zugriff auf ent-

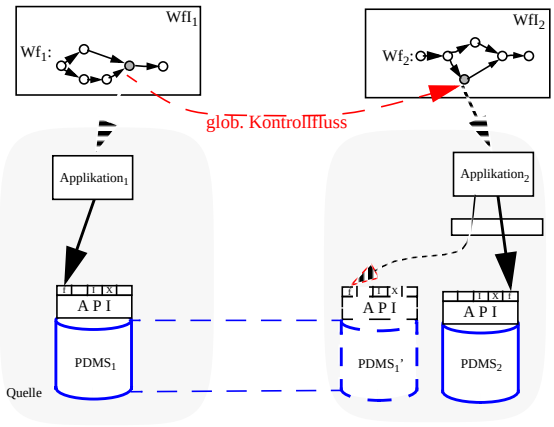


Abbildung 16 Datenbereitstellung bei direkter Zugriffsumleitung

fernte Kooperationsdaten zwischen $PDMS_1$ und $PDMS_2$ „umgeschaltet“ werden muss, d. h., die Wf-Aktion wird, wie in Abbildung 6 gezeigt, um eine Schicht erweitert, die den Zugriff für entfernte Objekte entsprechend umleitet.

Etwas anders sieht es bei der PDMS-internen Umleitung aus (Abbildung 16). Die Kooperationsdaten verbleiben auf Seite der Quellinsel, d. h., die Ziel-Applikation muss vor Lese-Anomalien geschützt werden, die durch Bearbeiten der Kooperationsdaten auf der Quellinsel entstehen könnten. Da die Funktionsaufrufe durch das Zielsystem auf Quellseite jedoch wie lokale Aufrufe behandelt werden, können an dieser Stelle die vom Hersteller vorgesehenen Synchronisationsmechanismen/Versionskontrollen greifen.

Die Spezifikation der Kooperationsdaten spiegelt sich bei diesem Ansatz in den Proxy-Objekten wider [Cha98]. Für das Anlegen dieser Proxy-Objekte gibt es im wesentlichen zwei Möglichkeiten. Zum einen können sie direkt beim Anlegen des lokalen Produktbaums vorgesehen werden, d. h. jedoch, die Umsetzung der Kooperationsdaten liegt in der Verantwortung des PDM-Systemadministrators. Das WfMS hat in diesem Fall keinerlei Eingriffsmöglichkeit, dementsprechend sind auch keinerlei Änderungen an den Workflows notwendig. Die Datenflussabhängigkeit wird nur implizit durch Vorhandensein/Nichtvorhandensein der erforderlichen Daten abgebildet. Eine explizite zeitliche Modellierung ist nur über Workflow-übergreifende Kontrollflussstrukturen möglich (Abbildung 17). Dieser Ansatz erfordert jedoch zuviel manuellen Einsatz und vernachlässigt unser Ziel der Automatisierung.

Vielversprechender erscheint es, mit Export- und Import-Aktivitäten zu arbeiten. Auf Typebene sind die Workflows ebenfalls, wie in Abbildung 8 illustriert, zu erweitern. Erst auf der Ausprägungsebene ergibt sich ein gewichtiger Unterschied: die Export-Aktivität extrahiert nur die Struktur der Kooperationsdaten, die Import-Aktivität erzeugt daraus (in Verbindung mit den über das Quellsystem vorhandenen Informationen) im Zielsystem die entsprechenden Proxy-Objekte. Beim anschließenden Zugriff durch die Applikation auf ein Proxy-Objekt kann dieses als entferntes Objekt identifiziert und der Zugriff (unter Kontrolle des PDMS) entsprechend weitergeleitet werden.

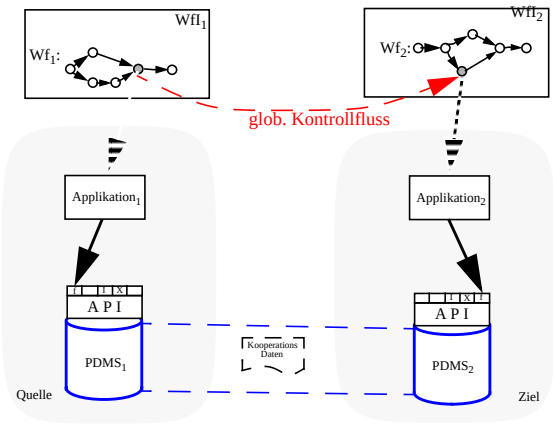


Abbildung 17 Zugriff durch PDMS-interne Funktionsweiterleitung (Proxy-Objekte)

Ein weiterer inzwischen recht weit verbreiterter Ansatz besteht in einer gemeinsamen Datenhaltung. Der Datenzugriff geschieht über ein zentrales PDMS. PDM-Enablers beispielsweise bietet die Möglichkeit, mittels CORBA auf die so verwalteten Daten zuzugreifen, Dokumente zu suchen und sich innerhalb der Produktstruktur navigierend weiterzubewegen. Soll jedoch weiterhin, wie in Abbildung 18 zu sehen, auf der Ziel-Insel Applikation$_2$ auf das lokale PDMS$_2$ zugreifen, so muss wie bei der direkten Zugriffsumleitung explizit zwischen Zugriffen auf entfernte und lokale Objekte unterschieden werden.

Allen drei Ansätzen ist gemeinsam, dass das Bereitstellen der Daten unter Kontrolle der beteiligten PDMS stattfindet. Dadurch hat jedoch das übergeordnete WfMS keine Möglichkeit, direkt einzugreifen. Um die aber immer noch vorhandene zeitliche Abhängigkeit zu modellieren, muss in diesem Fall eine Kontrollflussabhängigkeit auf Typebene eingeführt werden.

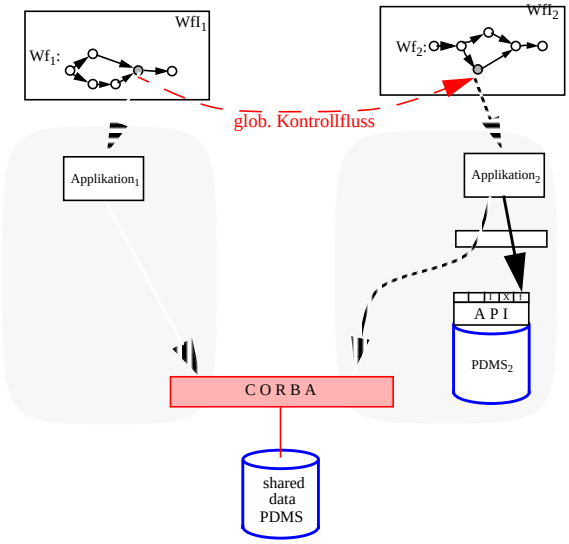


Abbildung 18 Zugriff auf Kooperationsdaten bei gemeinsamer Datenhaltung

6. Übertragungsprotokolle

Bei den im letzten Abschnitt vorgestellten Lösungsvorschlägen wird im Falle des Datentransfers ein physisches Übertragen der Kooperationsdaten von der Quell- zur Zielinsel notwendig. Nachfolgend werden dafür verschiedene Vorgehensweisen vorgestellt. Dabei gehen wir davon aus, dass die einzelnen Inselsysteme, wie inzwischen allgemein üblich, durch eine Firewall gegen Missbrauch geschützt sind.

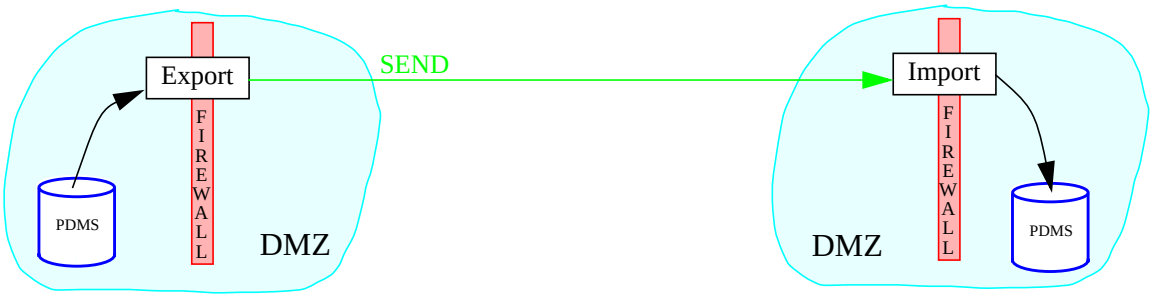


Abbildung 19 Übertragen von Daten mit einfachem Senden/Empfangen

6.1 Binäres Protokoll

Die einfachste Möglichkeit besteht im Einsatz eines einfachen Nachrichtenprotokolls. Im Fall der Übertragung über das Internet bietet sich dabei beispielsweise TCP/IP oder (eher ungewöhnlich) UDP an. Die Export-Applikation fungiert dabei als Sender, die Import-Applikation auf der Zielinsel als Empfänger (Abbildung 19). Der Aufwand, der dabei entsteht, ist jedoch nicht zu unterschätzen. Die Kommunikation findet auf sehr niedrigem Niveau statt. Die Applikationen sind dafür zuständig, sich um die Verbindung zu kümmern, sich auf ein Nachrichtenformat zu einigen, die Kooperationsdaten sendefähig aufzubereiten und sie schließlich in Nachrichten zu ver- bzw. aus Nachrichten zu entpacken. Die Wartung einer solchen Sende-/Empfangsapplikation ist alles andere als einfach. Zusätzlich müssen auf der Firewall Ports freigeschaltet werden, um die Kommunikation zu ermöglichen, was nur in Ausnahmefällen möglich ist. Weiterhin kann auch nicht davon ausgegangen werden, dass beide Systeme rund um die Uhr dazu bereit sind, synchron miteinander Daten auszutauschen. Dieser Ansatz ist daher insgesamt für unsere Zwecke nicht besonders geeignet.

Als weiteres Protokoll kommt das File Transfer Protocol (FTP) in Frage (Abbildung 20). Die Export-Applikation bereitet die Kooperationsdaten zu einer Export-Datei auf, die von außen zugänglich beispielsweise in der DMZ (*demilitarized zone*) gespeichert wird. Die Import-Applikation der Zielinsel kann diese anschließend über FTP transferieren und weiterverarbeiten. Dieses Freigeben der Kooperationsdaten in der DMZ widerspricht jedoch völlig dem Sicherheitskonzept, das hinter dem Einsatz einer Firewall steht und ist somit ebenfalls kaum praktikabel.

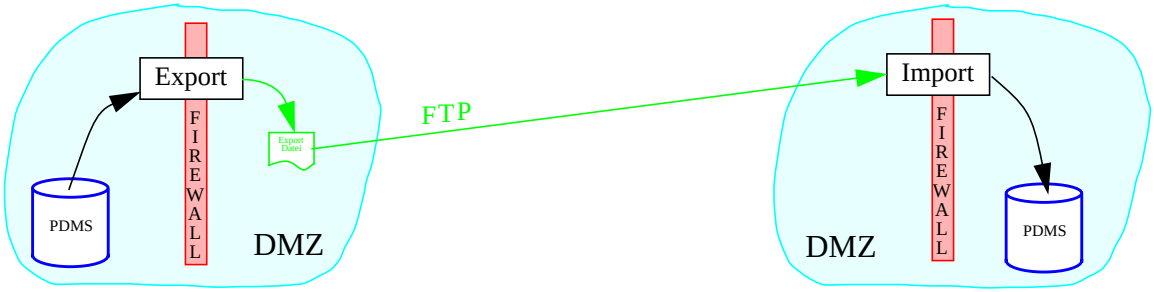


Abbildung 20 Übertragen der Kooperationsdaten mittels FTP

6.2 Nachrichten-Warteschlangen

Um Quell- und Zielinsel voneinander zu entkoppeln, bietet sich der Einsatz von persistenten Warteschlangen ([Jot97], [SZ98]) an. Die Export-Applikation bereitet die Kooperationsdaten auf und stellt diese in eine beiden Systemen bekannte Warteschlange. Aus dieser können sie von der Import-Applikation wieder ausgelesen werden. Beide Systeme sind (abgesehen von den im Workflow definierten Abhängigkeiten) zeitlich voneinander unabhängig. Die Warteschlangen-API ist im Allgemeinen recht komfortabel, zusätzlich gewinnt das Gesamtsystem durch die persistente Speicherung der übertragenen Daten auch an Robustheit. Einstellen und Auslesen in die Warteschlange können (zumindest bei „nicht paranoid“ eingestellten Firewalls) durch Applikationen innerhalb der Sicherheitszone ausgeführt werden. Von Nachteil ist lediglich, dass mit dem Warteschlangen-Service ein weiteres System ins Spiel kommt, das hohe Leistung und Verfügbarkeit gewährleisten muss. Insgesamt scheint dieser Ansatz jedoch vielversprechend.

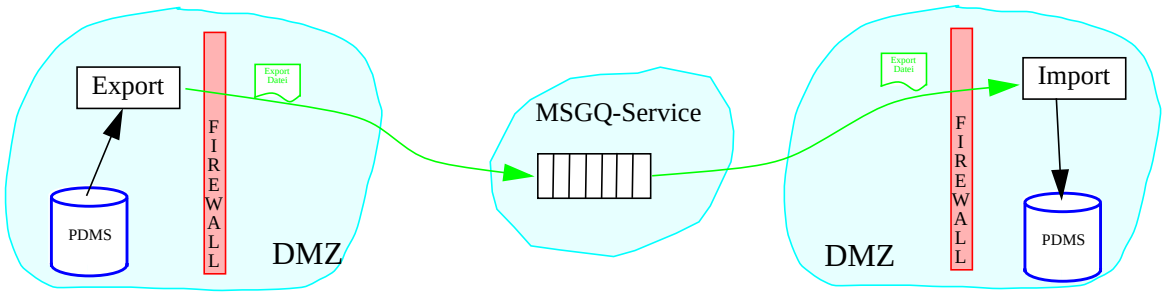


Abbildung 21 Einsatz einer Nachrichtenwarteschlange zur asynchronen Übertragung

6.3 Remote Procedure Call

Ein inzwischen oft benutzter Mechanismus zur Kommunikation ist auch der Einsatz von Remote Procedure Calls (RPC), dem Aufruf von Methoden, die an entfernter Stelle angeboten werden. In diesem Zusammenhang wird oft die CORBA-Middleware eingesetzt; im JAVA-Umfeld wird mit RMI ein etwas schlankerer Dienst angeboten. Immer mehr Anklang findet auch das auf XML basierende *Simple Object Access Protocol* (SOAP), mit dessen Hilfe die Firewall-Problematik, wie später noch beschrieben wird, elegant umgangen werden kann.

6.3.1 CORBA/RMI

Beim Einsatz von RPCs werden lokal Methoden aufgerufen, die auf einem entfernten System ausgeführt werden (Abbildung 22). Dies wird (grob vereinfacht) möglich, in dem auf dem rufenden System nur Stellvertreter der eigentlichen Methoden (Stubs) aufgerufen werden. Die Parameter werden von der Middleware verpackt und übertragen, auf dem Zielsystem von einer entsprechenden Methode (Skeleton) entpackt und aufbereitet, bevor die eigentliche Methode ausgeführt wird. Das Ergebnis wird dann auf gleiche Art und Weise zurückgeliefert. Leider erfordert der Einsatz dieser Middleware wieder „Lücken“ in der Firewall, was den Einsatz in der Praxis erschwert, wenn nicht sogar unmöglich macht.

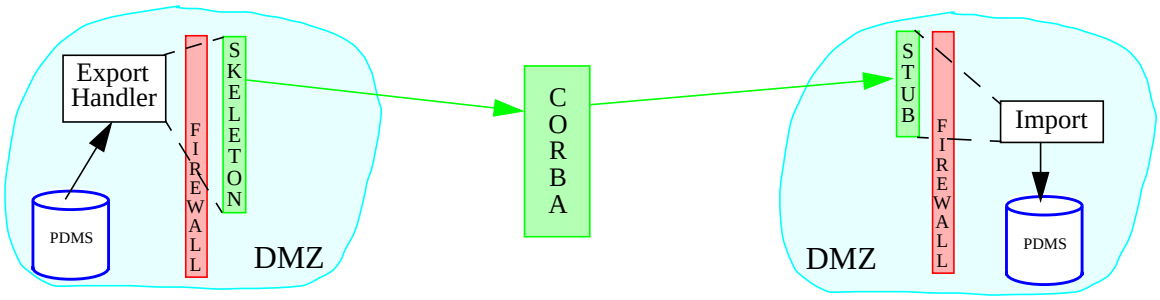


Abbildung 22 Datenübertragung mit Hilfe der CORBA-Middleware

6.3.2 SOAP

SOAP geht einen etwas anderen Weg. SOAP-Nachrichten werden in der Regel mittels HTTP übertragen. Die Verarbeitung geschieht über ein auf einem Applikationsserver bereitgestelltes Servlet. Der Applikationsserver kann sich an einen gängigen Web-Server angliedern (beispielsweise lässt sich TOMCAT über ein Erweiterungsmodul leicht an den APACHE-Web-Server koppeln). Die SOAP-Nachricht kann somit über den auf allen Firewalls üblicherweise freigeschalteten Port 80 das System erreichen und wird entsprechend an den Applikationsserver bzw. das zuständige Servlet (den *SOAP Message Handler*) weitergeleitet.

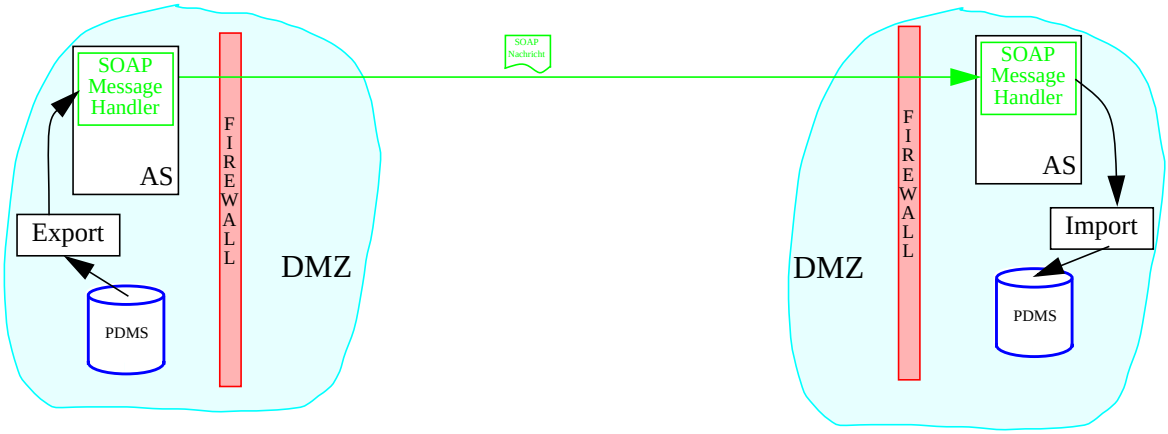


Abbildung 23 Datenübertragung durch SOAP-Nachrichten

Im Beispiel bereitet die Export-Applikation die Kooperationsdaten für den Transport vor. Diese werden von einem geeigneten Servlet als „Nutzfracht“ in eine SOAP-Nachricht verpackt und an die Zielinsel verschickt. Hier wird die Nachricht wiederum von einem Servlet zerlegt, die Kooperationsdaten können anschließend weiterverarbeitet werden. In unserem Fall bedeutet dies das Einspielen der Daten durch die Import-Applikation.

Das Verschicken von Nachrichten mit SOAP ist deutlich langsamer als der Einsatz von CORBA oder RMI [GSC+00]. SOAP hat jedoch den großen Vorteil, dass es beim Einsatz von Firewalls die wenigsten Probleme bereitet, da es den im Allgemeinen freigeschalteten Port 80 benutzen kann.

7. Zusammenfassung und Ausblick

In diesem Bericht haben wir die Notwendigkeit von globalen Datenflüssen zwischen verschiedenen Wf-Systemen („Wf-Inseln“) motiviert, wie sie in Entwicklungsprozessen häufig anzutreffen sind. Da in diesem Umfeld PDMS immer stärker an Bedeutung gewinnen[3], spielen diese Systeme beim Datenaustausch zwischen Wf-Inseln eine wichtige Rolle. Unter Beachtung der Voraussetzung, dass die lokalen Anwendungen weiterhin auf ihre Daten über ein lokales PDMS zugreifen sollen, ergeben sich verschiedene Möglichkeiten, wie mit Hilfe von Workflows die Bereitstellung der Daten zum richtigen Zeitpunkt gewährleistet werden kann. Dazu müssen die zu übertragenden Daten, die wir als Kooperationsdaten bezeichnen, spezifiziert werden. Durch das genaue Spezifizieren läßt sich auch die Übertragung von Daten auf das Notwendigste reduzieren. Für Anhänge (Dokumente oder CAD-Geometrien) haben wir die Möglichkeit der materialisierten (d. h., im Voraus stattfindenden) der referenzierten Übertragung (d. h. der Übertragung zum Zugriffszeitpunkt, um das anfallende Datenvolumen zu reduzieren) gegenübergestellt. Referenzierte Übertragung ist dabei jedoch nur möglich, wenn das zugrunde liegende PDMS ein objektweises Kopieren bei Zugriff unterstützt. Der entfernte Zugriff kann mittels Datenübertragung oder Zugriffsumleitung durchgeführt werden. Bei der Datenübertragung werden die Kooperationsdaten physisch vom Quell-PDMS zum Ziel-PDMS übertragen, bei der Zugriffsumleitung hingegen werden die Zugriffsanfragen an das Quell-System weitergeleitet, die Kooperationsdaten verbleiben physisch weiterhin auf dem Quellsystem.

Bei der Automatisierung der inselübergreifenden Datenbereitstellung können Workflows unterstützend eingesetzt werden. Erfreulicherweise müssen dabei an den bestehenden lokalen Workflows nur verhältnismäßig kleine Änderungen vorgenommen werden. Das Hauptproblem besteht vielmehr darin, geeignete Export- und Import-Applikationen zu entwickeln, die für den eigentlichen Datenaustausch zuständig sind. Ein Übertragen von Produktdaten zwischen verschiedenen PDMS ist im Allgemeinen gut möglich, da alle Systeme ähnliche, hierarchische Datenstrukturen verwenden. Um die Anzahl der Konvertierungen möglichst gering zu halten, bietet sich der Einsatz eines neutralen Zwischenformats an. Aufgrund der Baumstruktur der Produktdaten ist eine Abbildung in XML eine naheliegende Wahl. Beim schreibendem Zugriff durch das Zielsystem können weitere Probleme entstehen. Der Einsatz eines globalen Sperrprotokolls scheitert dabei an den Einschränkungen der beteiligten Systeme. Eine Versionsverwal-

3. Laut Computer-Zeitung soll sich ihre Verbreitung bis in zwei Jahren auf ein Drittel der eingesetzten Systeme erhöhen.

tung beim Zurückschreiben von Änderungen erscheint als gangbarer Weg, erfordert jedoch ein hohes Maß an manuellen Eingriffen, um verschiedene Versionen eines Produkts zu integrieren.

Die eigentliche Übertragung der Daten findet häufig zwischen Systemen statt, die zwar durch Firewalls gegen unerlaubten Zugriff geschützt sind, dadurch aber auch die normale Kommunikation erschweren. Vielversprechend erscheint hier der Einsatz von Nachrichtenwarteschlangen oder das Verschicken der Daten per SOAP-Nachrichten. Letztere können über den meist freigeschalteten HTTP-Port 80 den Firewall passieren.

Als nächster Schritt ist die Praxistauglichkeit unserer Ideen zu überprüfen. Insbesondere die Ansätze zur Datenbereitstellung bieten interessante Möglichkeiten, Workflow-Techniken zur Automatisierung von Datenabhängigkeiten zu nutzen, um für den Anwender, der auf nicht lokale Daten angewiesen ist, optimierte Zugriffszeiten zu erzielen. Daher soll zunächst in einer realistischen Umgebung praktisch untersucht werden, welcher Aufwand mit dem Erstellen geeigneter Export- und Import-Applikationen verbunden ist, inwieweit diese mit Hilfe eines Generators erzeugt werden können und welche Probleme beim Anpassen der ursprünglichen Workflow-Typen auftreten. Weiterhin muss die Komponente des Koordinators noch weiter verfeinert werden. Trotz des zentralen Ansatzes soll diese Komponente keineswegs als monolithisches System aufgebaut werden. Hierzu müssen noch geeignete Entwurfskonzepte entwickelt und geprüft werden. Weiterhin verlangen auch die skizzierten Übertragungswege noch weitere Untersuchungen und Messungen. Insbesondere scheinen hier die Ansätze mit Warteschlangen im asynchronen Fall bzw. SOAP-basierte RPCs im synchronen Fall als vielversprechend.

8. Literatur

[AGL98] Abramovici, M., Gerhard, D., Langenberg, L.: *Supporting Distributed Product Development Processes with PDM*, in: Krause, Heimann, Raupach (Hrsg.), New Tools and Workflows for Product Development - Proc. Seminar STC Design, Berlin, Fraunhofer IRB Verlag, 1998: 1-11

[BDS98] Beuter, T., Peter Dadam, Schneider, P.: *The WEP Model: Adequate Workflow-Management for Engineering Processes*, Proc. European Concurrent Engineering Conf. (ECEC) Erlangen, 1998

[BRZ00] Bon, M., Ritter, N., Zimmermann, J.: *Interoperabilität heterogener Workflows*, Proc. Grundlagen von Datenbanken, 2000: 11-15

[Bur99] Burkett, W.: *PDML - Product Data Markup Language - A New Paradigm for Product Data Exchange and Integration*, 30.04.1999, www.pdml.org/whitepap.pdf

[Cha98] Chadha, B.: *A Federated Architecture to Support Supply Chains*, Acquisition and Logistics Initiative MTS Conference, November 1998 www.atl.external.lmco.com/overview/papers/962-9937.pdf

[GSC+00] Govindaraju, M., Slominski, A., Choppella, V., Bramley, R., Gannon, D.: *Requirements for and Evaluation of RMI Protocols for Scientific Computing*, Proc. Supercomputing 2000 (SC2000), Dallas, 2000

[Jot97] Jotzo, M.: *JPMQ - Ein verteilter, objektorientierter Kommunikationsdienst für asynchrone Transaktionsverarbeitung*, Diplomarbeit, Universität Kaiserslautern, 1997

[MDJF01] Müller, E., Dadam, P., Enderle, J., Feltes, M.: *Tuning an SQL-Based PDM System in a Worldwide Client/Server Environment*, International Conference on Data Engineering (ICDE), Heidelberg, 2001: 99-108

[Naw01] Nawotki, A.: *Eine selektive Methode zur Verschlüsselung von Konstruktionsdaten mit Wavelets*, Dissertation, Kaiserslautern, 2001

[Sche94] Scheer, A.-W.: *Business Process Engineering: Reference Models for Industrial Enterprises*, 2nd ed., Springer, 1994

[Schu99] Schulze, W.: *Workflow-Management für CORBA-basierte Anwendungen*, Springer-Verlag, Berlin Heidelberg, 2000

[Sel00] Sellentin, J.: *Konzepte und Techniken der Datenversorgung fuer Informationssysteme*, Informatik Forsch. Entw. 15(2), 2000: 92-109

[Step92] Subcommittee 4 of ISO Technical Committee 184, *Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual*, ISO Dokument, ISO DIS 10303-11, August 1992

[SZ98] Steiert, H.-P., Zimmermann, J.: *JPMQ - An Advanced Persistent Message Queuing Service, in: Advances in Data-bases*, Proc. 16th Nat. British Conf. on Databases (BNCOD16), LNCS 1405, Springer, 1998: 1-18

[VDI99] Verein Deutscher Ingenieure, VDI-Richtlinien VDI2219, *Datenverarbeitung in der Konstruktion – Einführung und Wirtschaftlichkeit von EDM/PDM Systemen*, November 1999, Beuth Verlag GmbH, 10772 Berlin

[W3C97] W3C, *XML representation of a relational database*, Online Document, www.w3.org/XML/RDB.html

[W3C01] W3C, *Document Object Model (DOM) Level 3 Core Specification*, W3C Working Draft 1, 3 September 2001, www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913/

[Zim95] Zimmermann, P.: *The Official PGP User's Guide*, MIT Press, 1995