Diplomarbeit

Analyse und Monitoring eines verteilten Datenmanagements

Clemens Decker
Fachhochschule Heidelberg

DLR

Fachhochschule
Heidelberg

Analyse und Monitoring eines verteilten Datenmanagements

Diplomarbeit

zur Erhaltung des akademischen Grades

“Diplom – Informatiker (FH)“

im Diplomstudiengang Medieninformatik

an der

Fakultät für Informatik der Fachhochschule Heidelberg

September 2008

	
Themensteller	Deutsches Zentrum für Luft- und Raumfahrt Einrichtung für Simulations- und Softwaretechnik (SISTEC) Porz - Wahnheide Linder Höhe 51147 Köln Germany
Betreuer DLR	Dipl. - Math. Andreas Schreiber Dipl. - Ing. (BA) Thijs Metsch
Betreuer FH Heidelberg	Prof. Dr. Dieter Homeister
Autor	Clemens Alexander Decker Hermann - Löns - Straße 28a 53721 Siegburg Germany Clemens.Decker@gmail.com Gruppen - Nr. : 21720502 Martrikel - Nr. : 0058515

Inhaltsverzeichnis

1 Einleitung.....1
1.1 Motivation.....1
1.2 Aufgabenstellung.....2
1.3 Überblick.....3
2 Datenmanagement.....5
2.1 Aufgaben des Datenmanagements.....5
2.2 Operationen im Datenmanagement.....7
2.3 Mögliche Risiken bei der Datenverwaltung.....8
3 Struktur des Schiffentwurfs- und Simulationssystems (“SESIS“).....13
3.1 Schichtenmodell 14
3.2 Basissystem 14
3.3 Verwendung des Analyse- und Monitoring-Konzeptes im Basissystem 18
3.4 Datenmanagement des SESIS-Projekts.....19
3.4.1 Auf-/Abwärtskompatibilität bei SESIS.....21
3.4.2 Beispiel für ein SESIS-Datenmanagement-Szenario.....22
3.5 Anbindung an den Daten-Speicherung-Ressourcen 23
4 Anforderungen an das SESIS-Datenmanagement.....24
4.1 Definition der Benutzergruppe.....25
4.2 Ziele des Datenmanagements.....27
4.3 Umsetzung des Datenmanagements.....28
4.4 Beispiel für eine Datenreferenz im SESIS-Datenmanagement.....29
4.5 Benutzungsmerkmale des Datenmanagements.....29
4.6 Momentaner Stand.....30
4.7 Nächste Schritte.....30
5 Konzept zur Ermittlung von alten und nicht mehr gebrauchten Daten.....31
5.1 Lokale automatische Ermittlung von Abhängigkeiten.....33
5.1.1 Startauslöser der lokalen Ermittlung von Abhängigkeiten.....34
5.1.1.1 Erster Startauslöser (Commit / Save).....35
5.1.1.2 Zweiter Startauslöser (Branch).....36
5.1.1.3 Dritter Startauslöser (Merge).....37
5.1.2 Lokale Ermittlung von Abhängigkeiten.....39
5.2 Globale automatische Ermittlung von Abhängigkeiten.....47
5.3 Mögliche Probleme der lokalen und globalen Ermittlung.....54
5.4 Mögliche Problemlösung der lokalen und globalen Ermittlung.....58
5.5 Manuelles Starten der globalen Ermittlung von Abhängigkeiten.....59
5.6 Überblick der Faktoren die für die Ermittlung wichtig sind 60

6 Konzept zur Fehleraufdeckung im verteilten SESIS-Datenmanagement.....62
6.1 Prüfungsmöglichkeiten zur Fehlerermittlung im SESIS-Datenmanagement.....62
6.2 Mögliche Fehlerquellen im SESIS-Datenmanagement.....69
6.2.1 Definition der Fehlertypen bei Daten.....69
6.2.2 Beschreibung der möglichen Fehler im SESIS-Datenmanagement.....70
6.2.3 Überblick der Fehlerszenarien im SESIS-Datenmanagement.....76
6.3 Lösungsansätze für Fehlerquellen im SESIS-Datenmanagement.....78
6.4 Fehlerermittlung im SESIS-Datenmanagement.....80
6.5 Benachrichtigung über Pull-Aktion bei der Fehlerermittlung.....91
7 Konzept zur Lokalisierung der Daten und Optimierung von Speicherorten.....97
7.1 Konzept zur Lokalisierung der Daten.....97
7.1.1 Broadcast-Problem.....100
7.1.2 Lösung des Broadcast-Problems.....101
7.1.3 Versenden und Empfangen der Broadcast.....103
7.1.4 Analyse der erhaltenen Datenkataloge zur Lokalisierung der Daten.....107
7.2 Konzept zur Optimierung von Speicherorten.....112
8 Konzept zur Statusermittlung des Systems.....116
8.1 Benötigte Informationen zur Statusermittlung.....116
8.2 Überblick- und Statusdarstellung anhand einer GUI.....118
8.3 Ermittlung der Statuswerte.....121
9 Fazit / Ausblick 128

Ehrenwörtliche Erklärung.....130
Zusammenfassung.....131
Abstract.....132
Literaturverzeichnis.....133
Abbildungsverzeichnis.....137
Tabellenverzeichnis.....139

Anhang.....140
A Überblick über elementare Operationen.....140
A.1 Begriffe und Notation 140
A.2 Dimensionen des Branching.....144
A.3 Verzweigungmuster.....145
A.3.1 Einleitung.....145
A.3.2 Branch per Release.....146
A.3.3 Branch per Promotion.....147
A.3.4 Branch per Task.....148
A.3.5 Branch per Component.....149
A.3.6 Branch per Technology.....150
A.3.7 Zusammenfassung.....150
A.4 Risikofaktoren beim Branching und parallelem Entwickeln 153

B Die deutschen und internationalen Normen....162
C Datenschutzgesetze bezüglich personenbezogener Datenspeicherung....165
C.1 Vorbemerkung....166
C.2 Datenschutzmaßnahmen....166
C.3 Datenschutzbeauftragter....169
C.4 Zusammenfassung....172
D Mögliche Alternative zur Löschung von nicht mehr gebrauchten Daten....173
D.1 Hierarchisches Speichermanagement....173
D.1.1 HSM unter z/OS....173
D.1.2 HSM unter Unix....174
D.1.3 Vorteile von HSM....174
D.2 Informationslebenszyklusmanagement....175
D.2.1 Definition....176
D.2.2 Funktionalität von ILM - Lösungen....176
E Existierende Systeme....178
E.1 Bazaar....178
E.1.1 Eigenschaften von Bazaar....178
E.1.2 Geschichte von Bazaar....179
E.1.3 Vorgängerprojekte....179
E.2 GIT....180
E.2.1 Vorteile von GIT....180
E.2.2 Nachteile von GIT....181
E.2.3 Unterschied zwischen GIT und Bazaar....181

1 Einleitung

In diesem einleitenden Kapitel wird zunächst ein Überblick über die allgemeine und persönliche Motivation, die Aufgabenstellung und das gesamt Vorgehen bezüglich der Diplomarbeit gegeben.
Im Rahmen dieser Diplomarbeit wurde das Thema “Analyse und Monitoring eines verteilten Datenmanagements“ erarbeitet. Dieses Thema unterteilt sich des weiteren in vier Unterthemen, auf die im Überblick genauer eingegangen wird. Des weiteren werden im Überblick weitere Rahmenpunkte beschrieben die, die gesamte Diplomarbeit abrunden. In dem hieran folgenden Unterkapiteln werden nun die allgemeine und persönliche Motivation bezüglich des Diplomarbeitsthemas, die Beschreibung der Aufgabenstellung und ein kurzer Überblick über diese Diplomarbeit gezeigt.
Das erste Unterkapitel zeigt nun beginnend die allgemeine und persönliche Motivation.

1.1 Motivation

In den letzten Jahren hat der Einsatz verteilter Systeme sehr stark zugenommen. Ein Grund hierfür ist, dass aufgrund von einzelnen Rechnerausfällen das Arbeiten im Gesamtsystem zwar in eingeschränktem Masse weiterhin möglich ist, während dies jedoch bei einem zentralen System zum Totalausfall führen kann. Eine Hauptaufgabe von verteilten Systemen ist die Speicherung und Zurverfügungstellung von Daten, wodurch Benutzer gleichzeitig von verschiedenen Orten bzw. Systemen auf die Daten der einzelnen Systeme zugreifen können. Um diese Funktion zu gewährleisten, besitzen die einzelnen Systeme diverse Schnittstellen wie zum Beispiel im Bereich Kommunikation und Datenmanagement.
Neben dem zuvor beschriebenen Sicherheitsfaktor entsteht durch den Einsatz verteilter Systeme auch ein Kostenfaktor und teilweise sogar ein Performancefaktor. Durch den Einsatz eines solchen Systems benötigt man nicht mehr einen teuren, zentralen Server, sondern verteilt die Rechenlast auf die existierende Netzwerkinfrastruktur.

Durch diesen Faktor werden zwar Kosten eingespart und sogar teilweise Performance gewonnen, es können aber hierdurch andere Probleme auftreten. Zu diesen Problemen zählen vor allem die Überschaubarkeit bzw. Wartbarkeit eines solchen verteilten Systems.
In dieser Diplomarbeit geht es um die Entwicklung eines Konzeptes, das die Analyse und das Monitoring eines verteilten Datenmanagement ermöglicht bzw. die gerade beschriebene Überschaubarkeit gewährleisten kann. Eine genauere Aufteilung des Diplomarbeitsthemas ist unter dem nachfolgenden Kapitel beschrieben.

Meine persönliche Motivation für diese Arbeit ist vor allem durch das breite Spektrum des Themengebietes begründet. Es umfasst durch die Einarbeitung in das SESIS Projekt, die Strategien bzw. Konzepte der Revisionsverwaltung und verteilten Systemen, die existierenden Systeme und die allgemeinen und rechtlichen Hintergründe dieser. Des weiteren stellte die selbständige Entwicklung eines Konzeptes, welches die Analyse und das Monitoring eines verteilten Datenmanagements ermöglichen soll, eine weitere persönliche Motivation dar.

1.2 Aufgabenstellung

Im Rahmen dieser Diplomarbeit ist es Aufgabe, ein Konzept zu entwickeln, das die Analyse und Monitoring eines verteilten Datenmanagements gewährleistet.
Diese Diplomarbeit unterteilt sich in vier Unterthemen, auf die an dieser Stelle kurz eingegangen wird. Der erste Aspekt der Arbeit ist, den Status des Systems zu ermitteln. Der ermittelte Status setzt sich zum Beispiel aus der Anzahl der fehlerhaften, nicht vollständigen und alten oder nicht mehr benötigten Daten zusammen, und wird anhand des Status Explorers kenntlich gemacht.
Der zweite Aspekt der Arbeit ist die Konzeption von Strategien, die das aufdecken von Fehlern im System ermöglichen sollen. Der dritte Aspekt der Arbeit ist die Konzeption von Algorithmen, die alte und nicht mehr gebrauchte Daten ermitteln sollen. Der vierte und letzte Aspekt der Arbeit ist es, Methoden zu entwickeln, die einen Überblick darüber liefern, wo Daten liegen und ob man durch Verlagerung des Speicherortes die Zugriffsgeschwindigkeit optimieren kann.

Zusätzlich zu den genannten Aspekten wird in dieser Arbeit auch auf Begriffsdefinitionen, Notationen, bestehende Merge-Branch Muster, Normen, Risiken, mögliche Fehlerquellen, Zielgruppendefinition und bestehende Vorgaben eingegangen.
Im Einzelnen sind in der Diplomarbeit folgende Teilaufgaben zu erfüllen :

1. Ermittlung des Systemstatus
2. Aufdecken von Fehlern im System
3. Ermittlung von alten und nicht mehr gebrauchten Daten
4. Überblick darüber, wo Daten liegen und ob man durch Verlagerung des Speicherortes die Geschwindigkeit optimieren kann

1.3 Überblick

Im Rahmen dieser Diplomarbeit wird eine Analyse und ein Monitoring eines verteilten Datenmanagements konzeptioniert.
Des weiteren wurde diese Diplomarbeit in einem Zeitraum von vier Monaten im Institut für Simulations- und Softwaretechnik (SISTEC) des DLR
(Deutsches Zentrum für Luft- und Raumfahrt) bearbeitet.
Das Konzept unterteilte sich in vier Unterthemen. Diese Unterthemen sind, erstens das Ermitteln von alten und nicht mehr benötigten Daten, zweitens das Ermitteln von Fehlern im SESIS-Datenmanagement, drittens einen Überblick darüber zu erhalten wo Daten liegen und ob durch Speicherort Verlagerung die Zugriffsgeschwindigkeit optimiert werden kann und viertens den Status des Systems ermitteln und darstellen. In dieser Reihenfolge werden diese Unterthemen auch in dieser Diplomarbeit erarbeitet. Die entwickelten Konzepte bzw. deren Basen sind an sich allgemeiner Natur, sind aber in diesem speziellen Fall an die Gegebenheiten des SESIS-Projekts (SESIS “Schiffsentwurfs- und Simulationssystem“) angepasst. Bevor diese Konzepte gezeigt werden, wird zuerst auf das Datenmanagement, das bestehende SESIS-Projekt bzw. dessen relevanten Bestandteile und die Anforderungen an das SESIS-Datenmanagement eingegangen. Des weiteren wird in diesem Rahmen das bestehende verteilte Datenmanagement-Konzept beschrieben.

Neben den behandelten Unterthemen wird auf die Definition bzw. Beschreibung von Branch- und Merge-Aktionen eingegangen. Des weiteren wird diesbezüglich im Anhang, unter Kapitel A, die möglichen, existierenden Branch und Merge Strategien beschrieben.
Weitere zusätzliche Punkte, die im Anhang beschreiben werden, sind einmal die Datenschutzbestimmungen (Kapitel C), da mit Personen bezogenen Daten gearbeitet wird, und bestehende internationale und deutsche Normen (Kapitel B), die für die ordnungsgemäße Umsetzung elementar wichtig sind.
Bezüglich der Fehlererkennung in einem verteilten Datenmanagement, wird des weiteren auf bestehende und allgemein gültige Fehlererkennungsalgorithmen eingegangen.
Diese zusätzlich genannten Punkte sind für die Rahmenbedingung bzw. das Verständnis dieser Diplomarbeit sehr wichtig.
Das Ergebnis dieser wissenschaftlich ausgearbeiteten Diplomarbeit zeigt ein Konzept, welches eine Analyse und Monitoring des verteilten Datenmanagements ermöglicht und in das bestehende SESIS-Projekt implementiert werden kann.

2 Datenmanagement

In diesem Kapitel werden die nötigen Anforderungen, die das Datenmanagement betreffen, beschrieben. Darüber hinaus wird auf mögliche Risiken bezüglich der Datenverwaltung eingegangen. Diese beiden Punkte stellen den theoretischen Hintergrund für die darauf folgende Anforderungsanalyse dar.

2.1 Aufgaben des Datenmanagements

Als erstes muss definiert werden, welche Anforderungen ein Datenmanagement bzw. dessen Daten erfüllen müssen. Die Daten eines solchen Systems sollten Anforderungen wie Belegbarkeit, Vollständigkeit, Richtigkeit, Zeitgerechtigkeit, Klarheit, Verfügbarkeit und Nachvollziehbarkeit gewährleisten [DVR08]. Diese genannten Anforderungen werden auch durch deutsche und internationale Normen wie DIN und ISO gestützt [DVR08,ISO108,ISO208]. Nun aber zuerst zu den zuvor genannten Anforderungen und deren Beschreibungen. Bezüglich der existierenden Normen wird unter dem Anhangspunkt B “Die deutschen und internationalen Normen“ eine genauere Beschreibung gegeben. Diese stellen eine elementare Basis dar.

- **Belegbarkeit**
  Die Belegbarkeit beinhaltet, dass für alle erzeugten Daten Informationen vorhanden sein müssen, die als Nachweis für deren Inhalte herangezogen werden können. Solche Informationen können zum Beispiel anhand von Log- oder Metadaten bei der Erzeugung der tatsächlichen Daten generiert werden.

- **Vollständigkeit**
  Sowie die erzeugten Daten als auch die damit verbundenen Datenreferenzen, Metadaten und die sonst erforderlichen Daten müssen vollständig sein, das heißt vollzählig und lückenlos.

- **Richtigkeit**
  Die im vorhergehenden erwähnten Daten und Metadaten müssen darüber hinaus richtig sein. Neben der Anforderung, objektiv zutreffend sein zu müssen, wird hierin auch die korrekte, sachliche Zuordnung der Daten, Datenreferenz und Metadaten gesehen. Hierbei ist zu beachten, dass die Datenreferenz korrekt auf die ihm zugeordneten Daten als auch damit zusammenhängenden Metadaten zeigt. Des weiteren müssen die Metadaten valide sein, um bei einer späteren Auswertung jeglicher Art von Nutzen sein zu können.

- **Zeitgerechtigkeit**
  Auch diese Anforderung an die Daten, speziell die Metadaten, ist von größter Relevanz. Hierbei ist es sehr wichtig, dass sowohl das Erstellungsdatum als auch das Änderungsdatum zeitnah, also ohne nennenswerten Zeitverzug, in die Metadaten der betreffenden Datenreferenz eingetragen werden. Diese Informationen könnten zum Beispiel bei einer späteren Fehlererkennung, mittels einer Historie, sehr hilfreich sein.

- **Klarheit**
  Bei der Klarheit handelt es sich um eine Ableitung der bestehenden Anforderungen an eine geordnete Datenstruktur. Die sachliche Ordnung ist, wie bereits schon bei der Richtigkeit angesprochen, ebenfalls mit dem Prinzip der Datenreferenz in Verbindung zu bringen, da eine geordnete Datenreferenz auch zur Klarheit bezüglich der zugrunde liegenden Sachverhalte beiträgt.

- **Verfügbarkeit / Sicherheit**
  Die Anforderung der Verfügbarkeit der Daten und Metadaten ist ebenfalls ein sehr wichtiger Punkt, da hierdurch größere Ausfälle bezüglich des Datenzugriffs verhindert werden können. Hierdurch wird gefordert, dass die Daten während der Dauer der Aufbewahrungsfrist verfügbar sind und jederzeit innerhalb angemessener Frist lesbar gemacht werden können.

Da die zeitnahe Verfügbarkeit ebenfalls auf die möglichen Regelungen hin sichergestellt sein muss, wird das Prinzip der Verfügbarkeit häufig auch als Anforderung an die Sicherheit interpretiert und aus den bestehenden Vorschriften abgeleitet. Diesbezüglich wird unter dem Anhangspunkt C “Datenschutzgesetze bezüglich personenbezogener Datenspeicherung“ eine genauere Beschreibung gegeben.

- **Nachvollziehbarkeit**
  Die Anforderung der Nachvollziehbarkeit beschreibt, dass man Daten in ihrer Erzeugung und Änderung verfolgen kann und ihr ursprünglicher Inhalt auch später noch feststellbar sein muss [DVR08]. Hieraus lässt sich unschwer die Anforderung der Unveränderbarkeit der tatsächlichen Revisionsdaten, ihrer Nachvollziehbarkeit und nachträglichen Prüfbarkeit ableiten. Bezüglich der Nachvollziehbarkeit ist die Validität der Metadaten von elementarer Wichtigkeit, da anhand ihnen die Nachvollziehbarkeit der Datenentwicklung möglich wird.

2.2 Operationen im Datenmanagement

In diesem Unterkapitel werden nun die Operationen im Datenmanagement beschrieben. Diese Operationen bzw. Komponenten dienen zur Flexibilisierung des Datenmanagement und damit zur schnelleren Reaktion auf Änderungswünsche. Hierzu sind Techniken wie das Branching und Merging notwendig, die das Abweichen von einer streng sequentiellen Entwicklung ermöglichen. Der Branch ist eine Verzweigung beim Erzeugen von Varianten einer Datenreferenz. Das heißt, dass nach einem Branch zwei Varianten einer Datenreferenz existieren, an denen unabhängig voneinander Änderungen vorgenommen werden können.
In der Regel kommt ein Branch bei der Entwicklung von Quellcodes zum Einsatz. Notwendig wird er, wenn beispielsweise ein Programm auf der einen Seite weiterentwickelt wird und auf der anderen Seite Fehlerkorrekturen an der alten Version notwendig sind [Def08]. Ist die neuere Version des Programms, bzw. in diesem Fall die Datenreferenz, fertig gestellt, müssen die behobenen Fehler oder Änderungen aus der “älteren“ Version in die “neue“ Version übernommen werden.

Dieses Mischen der beiden Varianten wird als Merge bezeichnet [Def08]. Branch und Merge können somit als die beiden Enden einer verzweigten Entwicklung gesehen werden. Beim Merge wird die Verzweigung geschlossen, welche beim Branch erzeugt wurde.

2.3 Mögliche Risiken bei der Datenverwaltung

Bevor auf mögliche Risiken eingegangen wird, werden an dieser Stelle drei mögliche Gruppen von Risiken definiert. Jedes Risiko kann einer der drei Gruppen zugeordnet werden. Die drei Gruppen setzen sich zusammen aus Immanenten Risiken, Kontrollrisiken und Erkennungsrisiken [DVR08].

- **Immanente Risiken**
  Darunter versteht man alle Risiken, die sich durch den Einsatz von DV an sich im Betrieb ergeben.

- **Kontrollrisiken**
  Darunter versteht man alle Risiken, die sich dadurch ergeben, dass Kontrollmaßnahmen nur unzureichend vorhanden oder wirksam sind.

- **Erkennungsrisiken**
  Darunter versteht man alle Risiken, die sich dadurch ergeben, dass bei Prüfungen Risiken nicht oder nicht vollständig erkannt werden.

Neben den drei Risikogruppen, gibt es auch drei Kontrollgruppen mit deren Hilfe versucht wird, die Risiken zu minimieren. Die folgende Abbildung zeigt das Zusammenspiel dieser Gruppen.

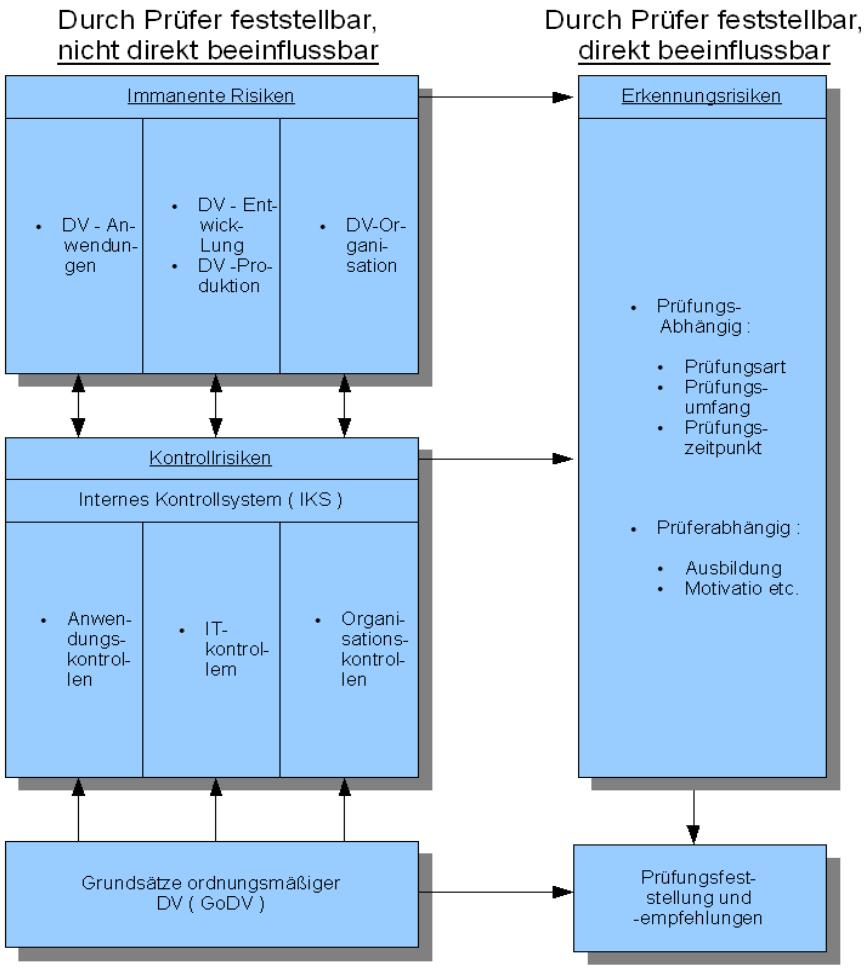

Abbildung 2.3-1: Risiken aus Sicht der DV-Revision [DVR08]

Diese Gruppen werden wie folgt in Anwendungskontrollen, Organisationskontrollen und IT-Kontrollen unterteilt [DVR08]. Kontrollen, die in DV-Anwendungen realisiert sind, bezeichnen man als Anwendungskontrollen,

die sich wie folgt zusammensetzten :

- Kontrolle bezüglich Berechtigungen
  - Kompetenzsysteme mit Regelungen für
    - Zugang zur Anwendungen
    - Zugang zu einzelnen Funktionsbereichen / Funktionen
    - Zugriff zu Datenbereiche / Daten
- Kontrollen bezüglich Dateiein- / -ausgang bzw. - übernahme / -übergabe
  - Schnittstellen
  - Plausibilitätsüberprüfungen
  - Abstimmsummen
  - Prüfziffern
- Kontrollen bezüglich Verarbeitung und Datenhaltung
  - Verfahrensdokumentation
  - Verarbeitungsprotokolle
  - Plausibilitätsprüfungen
  - Abstimmsummen
  - Journale
- Kontrollen bezüglich Benutzung
  - Verarbeitungsprotokolle
  - Sitzungsprotokolle
  - Journale
  - Prüfspur
  - Protokolle bezüglich Ablehnung von Zugriffen durch das Kompetenzsystem

Neben den Anwendungskontrollen kann man Kontrollen definieren, die im Umfeld des Einsatzes von DV-Anwendungen vorzufinden sind.

Diese so genannten Organisationskontrollen teilen sich wie folgt auf :

- Arbeitsanweisungen
  - Arbeitsvorbereitung
  - Arbeitsnachbereitung
  - Prüf- und Kontrollverfahren
  - Aufgabentrennung von Verarbeitung und Kontrollen
- Belegwesen
  - Ablage
  - Inhalt
  - Aufbereitung
  - Verarbeitung
- Abstimmung
  - Summen- / Saldenlisten
  - Auswertungen
  - Abgrenzungen
- Änderungsmanagement
- Abnehmeverfahren
- Übernahmeverfahren in Produktion

Als dritte und letzte Gruppe werden nun die “IT-Kontrollen“ betrachtet, die aus Maßnahmen im Bereich der DV-Entwicklung sowie DV-Produktion bestehen. Hierzu gehören :

- Kontrollen auf Basis der Qualitätssicherung
  - Einhaltung von Standards und Vorgaben
  - Reviews von Dokumenten, Codeinspektionen u.ä.
  - Testverfahren
- Zugangsbeschränkungen für das Rechenzentrum
- Überprüfung und Wartung der technischen Ausstattung
- Überprüfung des Datensicherungs- und -auslagerungsverfahren
- Überprüfung Sicherheitsstandards und deren Umsetzung

Das IKS muss an Veränderungen angepasst werden und seine Wirksamkeit ohne Unterbrechung bewahren. Ob dies in ausreichendem Maße der Fall ist, sollte regelmäßig zu festgelegten Zeiten überprüft werden (zum Beispiel im Rahmen von Jahresabschlussprüfungen). Bei einer solchen Überprüfung werden sowohl die immanenten Risiken als auch die IKS selbst zum Prüfungsgegenstand. Wie zuvor erwähnt werden nun mögliche Risiken aufgelistet, die bei einem Datenmanagement entstehen könnten [DVR08]. Jedes der nun folgenden Risiken kann einer der oben genannten Risikogruppen zugeordnet werden. Im späterem Verlauf wird zusätzlich auf mögliche Fehlerquellen eines Datenmanagements eingegangen.

- Administratoren sind in Konfliktsituationen nicht erreichbar bzw. verfügbar.
- Zugriffskontrolle und Zugangssicherung : Dem Benutzer wurden zu viele oder zu wenige Rechte erteilt.
- Durch ehemalige Mitarbeiter, deren Rechte nicht geändert wurden, kann es zu Sabotage bzw. Manipulation der Daten kommen.
- Fehler in den Daten können zu Folgefehlern oder sogar zum Stopp der Entwicklung führen.
- Fehler in den Datenreferenzen können zu Folgefehlern oder sogar zum Stopp der Entwicklung führen.
- Fehler in den Metadaten können zu Folgefehlern, zu Fehlern in der Statusauswertung oder sogar zum Stopp der Entwicklung führen.
- Die Hardwareausstattung des Clients oder Servers ist für das System nicht ausreichend.
- Das Netzwerk ist nicht in einer geeigneten Weise ausgebaut und verfügt nicht über eine angemessene Verbindungs- und Datensicherheit .
- Die Übersicht über Daten, Datenreferenzen, Speicherorte der Daten und Branch- oder Merge-Aktionen geht verloren.
- Nicht mehr benötigte Daten nehmen Speicherplatz in Anspruch.
- Fehler bei Daten können nicht rekonstruiert oder behoben werden.
- Großer Zeitaufwand bei der Fehlerbehebung von Daten.
- Geschwindigkeitseinbußen durch große Datenmengen, die ständig verschoben werden.

3 Struktur des Schiffentwurfs- und Simulationssystems (“SESIS“)

In diesem einleitenden Kapitel wird nun zunächst das SESIS-Projekt bzw. dessen Struktur beschrieben. Dies ist wichtig, da das Basissystem und dessen Struktur die Grundlage für die in dieser Diplomarbeit zu bearbeitenden Themen darstellt. Im speziellen beschreiben die bestehenden Plug-Ins und Schnittstellen die Struktur die bezüglich des zu entwickelnden Konzeptes benötigt werden. Ziel des Systems ist eine es Simulations- und Entwurfsumgebung zu erstellen, welche in verteilten Umgebungen einsetzbar und durch klar definierte Schnittstellen offen für Erweiterungen ist. Durch einfache Integration existierender Anwendungen soll sie leicht um beliebige fachliche Funktionalitäten erweiterbar sein.
Sämtlich Informationen bezüglich des SESIS-Projektes, welche hier im folgenden gezeigt werden, wurden der Anforderungsanalyse entnommen. Die diesbezügliche Quelle ist unter [SESIS108] beschrieben. Folgendes Grundkonzept liegt hinter der Systemarchitektur :

“Auf jedem teilnehmenden Rechner im verteilten (oder lokalen) System soll eine Instanz der Software installiert werden. Je nach Aufgabe des Systems (Client, Server, etc.) soll diese Instanz eine andere Ausprägung erhalten“ [SESIS108,S8].

Zusätzlich soll auf jedem Rechner ein immer gleiches Basissystem installiert werden, welches durch Nachladen verschiedener Plug-Ins an spezielle Aufgaben (Datenserver, GUI-Client, Methoden-Server) angepasst wird.
Vorteile dieser Lösung sind :

- Es besteht eine Gleichartigkeit aller Systeme, das heißt jede Instanz des Systems kann grundsätzlich durch das Nachladen entsprechender Plug-Ins zur Laufzeit für jegliche Aufgabe konfiguriert und erweitert werden
- Die Installation und der Update der Software sind immer einheitlich
- Die Kommunikationsprotokolle zwischen den Instanzen können exakt vorgegeben und an die Rechner- und Netzwerk-Infrastruktur angepasst werden
- Es sind keine zentralen Services zum Betrieb notwendig

3.1 Schichtenmodell

Das Schichtenmodell ist die Grundlage der gesamten Softwarearchitektur. Daten, Design und Logik sind bei diesem Modell komplett getrennt. Die Schichten repräsentieren dabei die logischen Grenzen im System.

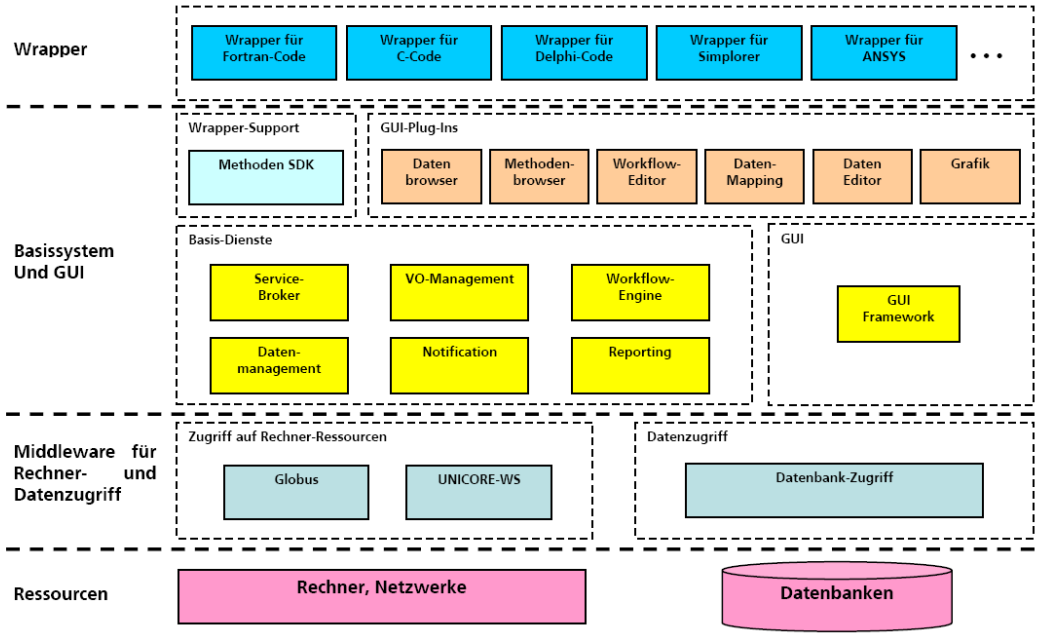

Abbildung 3.1-1: Schichtenmodell [SESIS108]

3.2 Basissystem

Das Basissystem soll auf einer Bundle Architektur aufbauen. Mittels dieser Bundles kann das System erweitert werden. Je nach Anzahl und Art der Bundles erhält das System eine andere Ausprägung. Um eine Gleichartigkeit aller Systeme zu erhalten, gehören gewisse Plug-Ins zu dem Basissystem. Das Basissystem wird nach den Spezifikationen des OSGi (Open Service Gateway Initiative) [OSGi08] realisiert.

Diese Spezifikationen wurden von der OSGi Alliance veröffentlicht und beschreiben eine Java Softwareplattform, auf der Dienste ausgeführt werden können.
Eigentlich angedacht, um auf mobilen Devices mehrere Applikationen gleichzeitig laufen zu lassen, bilden die OSGi-Spezifikationen heute die Grundlage vieler moderner Implementierungen.
Die Spezifikationen des OSGi-Frameworks definieren eine auf Java basierende Laufzeitumgebung für Services. Dieses Framework erlaubt es zur Laufzeit Services zu installieren oder zu entfernen. Die Services sind in so genannten Bundles implementiert. Diese Bundles sollen durch Remote-Zugriff verwaltet werden können. Das Starten und Stoppen von verteilten Bundles soll es erleichtern, verschiedenste Geräte zu koppeln. Diese Bundles können direkt vom Benutzer angefragt werden, oder sie exportieren ihre Services an andere Bundles. Neben der Beschreibung eines Frameworks beinhaltet die OSGi Spezifikation auch noch Beschreibungen von einem Satz Standard-Services / Bundles. Die verschiedenen Bundles laufen alle in einer JVM. Das Framework verwaltet den Lebenszyklus der einzelnen Bundles.
Da die Bundles keine direkten Einsprungspunkte für die verschiedenen Applikationen und Dienste bieten, wird die OSGi Umgebung um eine Plug-In Abstraktions-Schicht erweitert. Hierfür wird die von Eclipse Equinox bereitgestellte Plattform genutzt [Equi08]. Diese Implementierung ist gegenüber dem OSGi erweitert worden. So nutzt sie Extensions und Extension-Points. Mittels dieser Technik ist es möglich, die Bundles genauer zu definieren. So wird genau spezifiziert, an welcher Stelle ein Bundle andocken kann.
Ein Bundle bildet somit die Grundlage für die verschiedenen Dienste und Applikationen. Um Sie dem System bekannt zu machen, werden Extensions und Extension-Points genutzt. Innerhalb eines Bundles können verschiedene Einstiegspunkte definiert werden.
Jeder dieser Einstiegspunkte beschreibt ein Plug-In. Alle Erweiterungen gegenüber der OSGi Implementierung, die für das zu entwickelnden System nötig sind, werden in einer zusätzlichen Schicht bereitgestellt. Diese Schicht heißt RCE-Schicht (Reconfigurable Computing Environment) und liegt direkt über der Eclipse Equinox-Schicht.

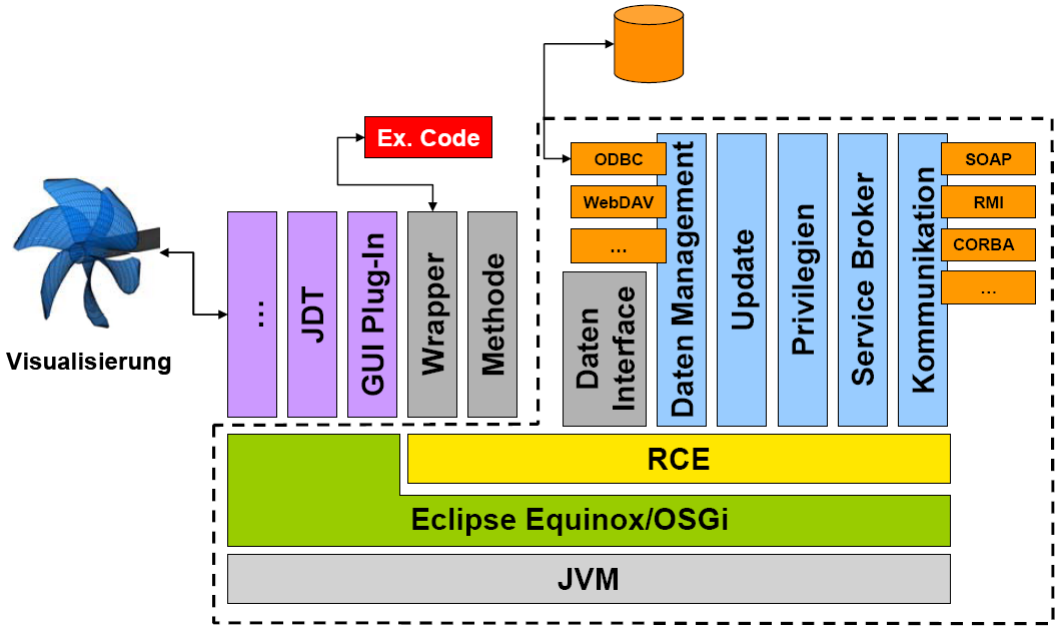

Abbildung 3.2-1: Basissystem [SESIS108]

Alle Bundles sind gemäß der OSGi-Spezifikation realisiert. Jedoch werden innerhalb des Projekts SESIS diese Begriffe wie folgt unterschieden :

- **SESIS-Plug-In :**
  Das SESIS-Plug-In ist ein Bundle, welches auf der RCE-Schicht aufbaut. Es nutzt Funktionalitäten des gesamten Systems und ist Schiffbau-spezifisch. Es kann verschiedene schiffbauliche Funktionen / Einsprungspunkte (also verschiedene Plug-Ins) bereitstellen.
- **Eclipse-Plug-In :**
  Das Eclipse-Plug-In ist ein Bundle, welches auf Eclipse aufbaut und keine Funktionalitäten des Systems braucht. Somit können vorhandene Plug-Ins für Eclipse in das System eingepflegt werden. So kann die Entwicklungsumgebung von Eclipse übernommen werden um neue Bundles / Plug-Ins für das System zu entwickeln.

- **RCE-Plug-In :**
  Das RCE-Plug-In gehört zum dem Basissystem und wird zur Laufzeit nur einmal gestartet. Es baut auf der RCE Schicht auf und ist nicht Schiffsbau-spezifisch.

Die Aufgaben der verschiedenen Komponenten im Basissystem, sind :

- **JVM :**
  Die Java Virtual Machine bildet die Schnittstelle zwischen der Software und der Hardware.
- **Eclipse / Equinox :**
  Diese Implementierung der OSGi verwaltet die lokalen Bundles des Systems. Die verschiedene Stadien, in denen sich ein Bundle befinden kann und Informationen über vorhandenen Bundles werden hier gehalten.
- **RCE-Schicht :**
  Diese Schicht verwaltet die Bundles in einem verteilten System. Neben dieser Verwaltungsaufgabe ermöglicht es auch die Bereitstellung einer Sicherheitsinfrastruktur.
- **Datenmanagement und Daten-Interface-Bundle** :
  Diese Bundles sind verantwortlich für das Datenmanagement. Es gibt pro Benutzer ein Daten-Interface, aber nur ein gemeinsames Datenmanagement, welches den Zugriff auf die Daten Speicher-Ressourcen steuert.
- **Privilegien-Bundle :**
  Dieses Bundle ist zur Rechte- und Privilegien-Verwaltung vorgesehen. Zum Systemstart wird der Benutzer authentifiziert und während der Laufzeit werden über diese Bundle seine Privilegien verwaltet.
- **Service Broker Bundle :**
  Die Referenz zu den einzelnen Bundles wird über dieses Bundle erfragt. Der Service Broker ist ähnlich dem DNS System des Internets aufgebaut.

- **Update Bundle :**
  Neue Versionen von Teilen des Basissystems oder von anderen Bundles werden mittels dieses Bundles ermittelt. Es bietet die Möglichkeit, das Basissystem und ihre Erweiterungen zu aktualisieren.
- **Kommunikations-Bundle :**
  Die Kommunikation in einem verteilten System wird über dieses Bundle realisiert. Alle nötigen Kommunikationspfade werden hier verwaltet. Verschiedene Protokolle sind dafür nötig (zum Beispiel CORBA, RMI, SOAP) und werden durch RCE-Plug-Ins bereitgestellt.

Die JVM, die Eclipse Equinox- und die RCE-Schicht bilden die Basis für das Gesamtsystem. Das Datenmanagement und Daten-Interface-Bundles des Basissystems werden im Folgenden beschrieben. Diese bilden neben dem Basissystem die Grundlage für das Verständnis dieser Diplomarbeit.

3.3 Verwendung des Analyse- und Monitoring-Konzeptes im Basissystem

Nachdem das bestehende Basissystem bzw. dessen Plug-Ins / Bundles beschrieben wurde, wird nun die Verwendung des zu entwickelnden Konzeptes im Basissystem gezeigt. Das zu entwickelnde Konzept ermöglicht die Analyse und das Monitoring eines verteilten Datenmanagements. Des weiteren unterteilt sich dieses Konzept in die vier zuvor beschriebenen Unterthemen. Eine genauere Beschreibung der Unterthemen kann im Kapitel 1.2 “Aufgabenstellung“ entnommen werden. Anhand der folgenden Abbildung wird nun die mögliche Verwendung des Konzeptes gezeigt.

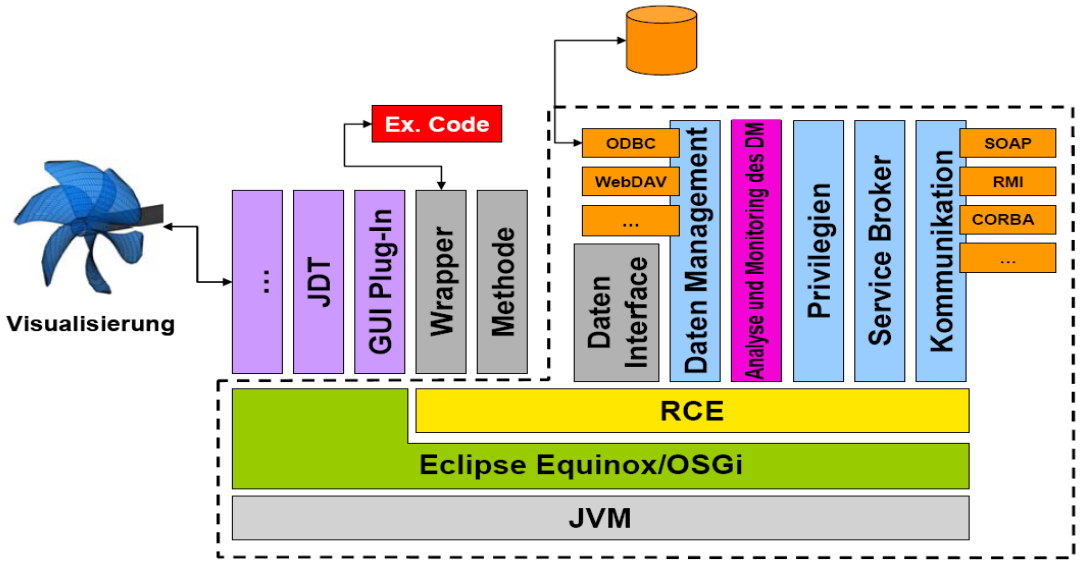

Abbildung 3.3-1: Verwendung des Analyse- und Monitoring-Konzeptes im Basissystem
[SESIS108]/Eigen Entwurf

Die Verwendungsstelle des Analyse- und Monitoring-Konzeptes im Basissystem ist innerhalb der gezeigten Abbildung als Hellmagenta farbiges Kästchen gekennzeichnet. Durch diese Verwendungsstelle ist erkennbar, dass das zu entwickelnde Konzept ein Bestandteil des Basissystem darstellt und dadurch bei jeder installierten Version vorhanden ist.

3.4 Datenmanagement des SESIS-Projekts

Das im SESIS-Projekt eingebundene Datenmanagement bildet eines der wichtigen Bestandteile des Basissystems. Dieses wird durch zwei wichtige Bundles repräsentiert. Das Datenmanagement- und Dateninterface-Bundles sind die besagten zwei Bundles die für das Datenmanagement verantwortlich sind. Es gibt pro Benutzer ein Daten-Interface, aber nur ein Datenmanagement, welches den Zugriff auf die Daten-Speicher-Ressourcen steuert. Rechte und Privilegien werden über das Privilegien Bundle realisiert.

Das Datenmanagement-Bundle bildet eine Schnittstelle zwischen dem Daten-Interface und den eigentlichen Speicher-Ressourcen. Somit abstrahiert es von den einzelnen Protokollen und kann je nach Datentyp (Nutzung von Daten-Java Klassen Mapper wie JDO, Schema, JAXB) entscheiden, welche Speicher-Ressource angesprochen wird. Auch abstrahiert es von den verschiedenen Protokollen, die benötigt werden, um die unterschiedlichsten Speicher-Systeme anzusprechen. Das Daten-Interface bildet die Schnittstelle zwischen einem Plug-In und dem Datenmanagement. Die Hauptaufgabe ist die eigentliche Persisierung von Daten. Hierfür wird das Datenmanagement angesprochen. Jedes Plug-In hat sein eigenes Daten-Interface und beinhaltet somit die Rechte und Einstellungen des eigentlichen Benutzers. Es wird im Basissystem nicht unterschieden zwischen Dateien und Daten-Modellen. Im Allgemeinen werden die Daten über eine Daten-Referenz angesprochen. Diese Daten-Referenz ist absolut eindeutig in einer RCE-Umgebung definiert. Jedes Datenmanagement hält sich einen so genannten Katalog, wo alle Daten-Referenzen aufgelistet sind, welche in dem Datenmanagement vorhanden sind. Suchanfragen nach Daten können somit realisiert werden. Fordert ein Plug-In nun Daten an, kann dieses über das Daten-Interface die Daten referenzieren und abholen. Die nächste Abbildung (Datenmanagement und Daten-Katalog) zeigt dieses Zusammenspiel.

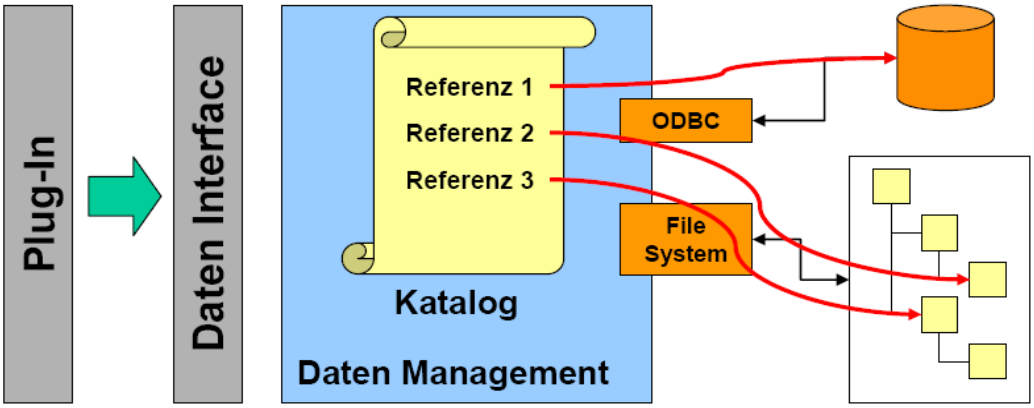

Abbildung 3.4-1: Datenmanagement und Daten-Katalog [SESIS108]

Somit sind alle Daten unabhängig von ihren Typ im System auffindbar und ansprechbar. Auf höheren Ebenen kann man eine Ordnung erstellen mit Hilfen von Projekten. Ein Projekt beinhaltet mehrere Daten-Referenzen und gruppiert somit Daten. Eine Daten-Referenz kann zu verschiedenen Projekten gehören. Eine Daten-Referenz kann auch auf andere Daten-Referenzen zeigen. Somit kann ein verteiltes Datenmanagement realisiert werden. Eine weitere Möglichkeit ist, das Daten-Referenzen Teile von anderen Daten darstellen.

3.4.1 Auf-/Abwärtskompatibilität bei SESIS

Um die Auf- / Abwärtskompatibilität zwischen Methoden und Datenmodellen zu gewährleisten, wurde vereinbart, dass das Datenmodell zunächst nur um Attribute und / oder Tabellen erweitert werden darf. Diese Änderungen am Datenmodell können einfach durch Hinzufügen der entsprechenden Informationen dokumentiert werden. Reduzierungen des Datenmodells (Löschen von Attributen) werden nach zeitlich festzulegenden Reviews des Datenmodells durchgeführt. Hierzu ist es notwendig, dass vorgesehene Reduzierungen des Datenmodells in ausreichender Zeit angekündigt werden, um dem Methoden-Entwickler (bzw. dem Benutzer) die Gelegenheit zu geben, seine Methoden an das veränderte Modell anzupassen bzw. der Reduzierung des Datenmodells zu widersprechen. Um eine ausreichende Sicherheit zu erlangen, keine Attribute aus dem Datenmodell zu entfernen, die noch benötigt werden, wurde folgendes Vorgehen vorgeschlagen: Im Methoden-SDK wird eine „Warning“ inklusive Beschreibung erzeugt, sobald als „deprecated“ markierte Attribute in einer Methode verwendet werden.

3.4.2 Beispiel für ein SESIS-Datenmanagement-Szenario

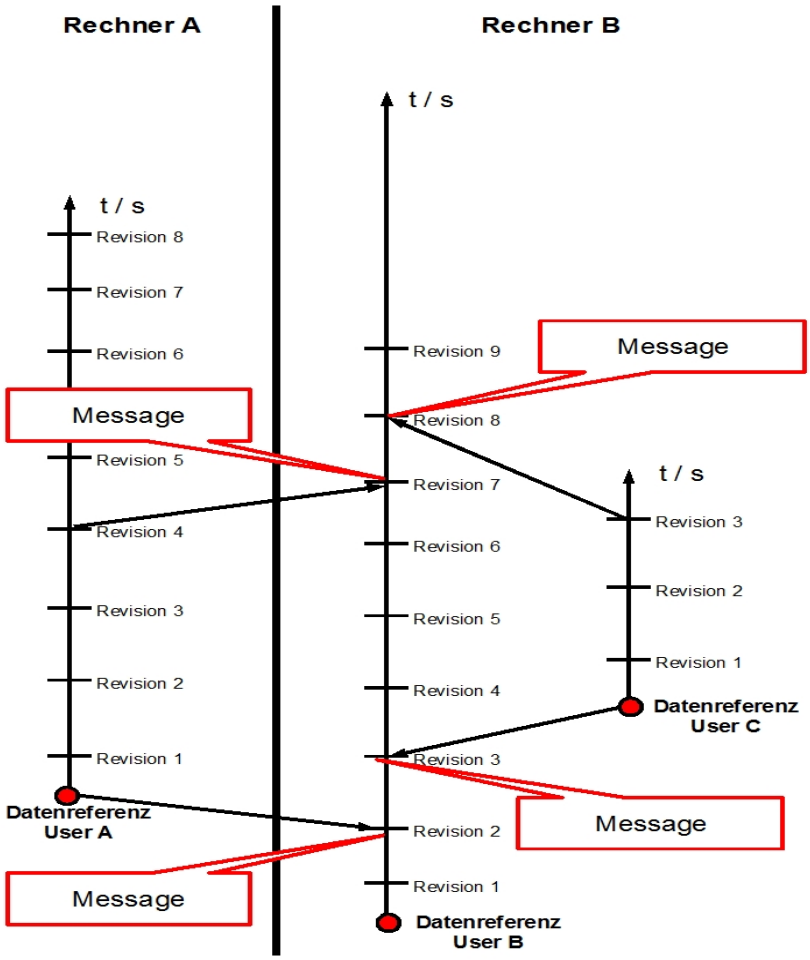

Abbildung 3.4.2-1: Beispiel für ein SESIS-Datenmanagement-Szenario

(Eigener Entwurf)

Diese Abbildung zeigt ein Beispiel, wie ein Szenario in dem Datenmanagement aussehen könnte. Hierbei werden Branch- und Merge-Aktionen von verschiedenen stellen gezeigt.

3.5 Anbindung an den Daten-Speicherung-Ressourcen

Dateien werden im Dateisystem oder mittels WebDAV abgelegt. Wesentliche Bestandteile des Systems aus datentechnischer Sicht sind :

- In jedem Basissystem wird eine lokale Datenbasis bereitgestellt. Dadurch wird gewährleistet, dass Clients auch im Offline-Betrieb arbeiten können und Methoden direkt auf Daten, die zuvor in die lokale Datenbasis des Servers transferiert wurden, zugreifen können.
- Zwischen der lokalen und der zentralen Datenbank gibt es einen Abgleich. Ebenso können Datensätze aus den Datenbasen exportiert und importiert werden. Dieser Abgleich wird mit Hilfe der Daten-Referenzen realisiert, welche aufeinander zeigen.
- In den einzelnen Installationen wird voraussichtlich ein zentraler Datenserver betrieben werden, auf dessen Daten alle Benutzer mit den entsprechenden Rechten zugreifen können. Die Hoheit über die Daten verbleibt dabei beim Eigentümer, und die Autorisierung der Zugriffe wird über das Privilegien-Bundle und mit Hilfe der einzelnen Datenspeicherungsdienste realisiert.
- Die Modellierung der schiffbaulichen Daten erfolgt durch die Anwendungspartner in einer Anwendungsschicht oberhalb des Basissystems in Form von Java Klassen. Hierzu wird dem Methodenentwickler eine Mapping-Tabelle zur Verfügung gestellt, aus der das aktuelle Datenmodell abgeleitet werden kann.
- Das Basissystem hat keinerlei Kenntnisse über diese Klassenstrukturen und ist somit in der Lage, beliebige Datenstrukturen zu verarbeiten. Durch das Schichtenkonzept wird erreicht, dass es keine Abhängigkeit zwischen dem Datenmanagement und der darunter liegenden Datenbank existiert, sodass die physikalische Abspeicherung der Daten durch verschiedene Systeme realisiert und später ggf. ausgetauscht werden kann.

4 Anforderungen an das SESIS-Datenmanagement

Das folgende Kapitel zeigt die Anforderungen an das SESIS-Projekt. Teile dieser Anforderungen wurden bereits umgesetzt. Zunächst wird aber an dieser Stelle kurz allgemein auf das Arbeiten mit verteilten Daten eingegangen. Projekte in der Wissenschaft und in der Wirtschaft werden heutzutage meistens in Teams bearbeitet. Häufig ist es dabei der Fall, dass die Teammitglieder an verteilten Orten arbeiten. Ein Problem stellt oft der Austausch und der gemeinsame und schnelle Zugriff auf Daten wie Messergebnisse, Auswertungen, Softwareprogramme und Literatur dar. Die Organisation der Daten bietet die Möglichkeit, Zusammenhänge zwischen diesen darzustellen. Einzelne Informationsquellen können so angeordnet werden, dass ein Benutzer eine Beziehung zwischen ihnen wahrnimmt. Dieser Ansatz der räumlichen Informationsorganisation nutzt die menschliche Fähigkeit, visuelle Reize schnell und effektiv zu verarbeiten. Haben verschiedene Benutzer zeitgleich Zugriff auf die Informationsquellenorganisation, können Konflikte bezüglich der Positionierung und Existenz einzelner Informationsquellen auftreten [Kasp08]. Synchronisation von Informationsquellen ist ein Problem, das unter anderem aus dem Bereich der Softwareentwicklung bekannt ist. Bekannte Verfahren, wie zum Beispiel das verbreitete Concurrent Version System (CVS)', lassen sich auch in anderen Kontexten anwenden, insbesondere auch im Bereich der kollaborativen Datenorganisation [CVS08]. In diesem Kapitel wird nun auf die bestehenden Vorgaben bzw. Anforderungen bezüglich des Datenmanagement eingegangen. Das darauf folgende Unterkapitel bezieht sich auf die Definition der Zielgruppe. Hierbei wird vor allem auf die Zielgruppe der Administratoren eingegangen, da diese von größter Wichtigkeit für das Datenmanagement sind. Sie koordinieren, überprüfen und verwalten das gesamte Datenmanagement. Als nächstes wird auf die Ziele des Datenmanagement eingegangen. Eines dieser Ziele ist zum Beispiel das parallele Arbeiten mit verteilten Daten. Ein weiter Unterkapitel betrifft die Umsetzung des Datenmanagement. Hier wird unter anderem auf die Kommunikation zwischen den verschiedenen, verteilten RCE-Installationen eingegangen.

Des weiteren wird auf das Privilegien-Bundle eingegangen, welches den Zugriff auf die Daten bzw. die Rechteverwaltung mittels Zertifikaten steuert. In den weiteren Folgekapiteln wird auf die Grundlagen, die Benutzung, der momentane Stand und nächste Schritte des Datenmanagements eingegangen. Informationen bezüglich der Anforderungen an das SESIS-Projekt selbst, wurden der Quelle [SESIS208] entnommen.

4.1 Definition der Benutzergruppe

Bevor die betreffende Zielgruppe definiert bzw. gezeigt wird, wird zunächst ein Überblick über alle Zielgruppen des SESIS-Projekts gegeben. Hierbei werden kurz die Zielgruppe der Entwurfsingenieure, der Power-User, der Softwareentwickler und der Administratoren gezeigt. Des weiteren wird in diesem Zusammenhang jeweils auf die Geforderten Kenntnisse, die Fertigkeiten, die Erfahrungen und die Einschränkungen eingegangen. Dies ist wichtig, da hieran die einzelnen Aufgabenverteilungen ersichtlich werden.

Überblick Zielgruppen :

Nutzergruppe	Entwurfsingenieur	Power-User	Software-entwickler	Administrator
Geforderte Kenntnisse	Simulationscodes, Berechnungsmethoden	Details der Methoden	Systeminterna, Schnittstellen der Simulationscodes	Installation und Konfiguration des Systems
Fertigkeiten	Durchführung von Simulation mit dem beteiligten Methoden	Durchführung von Simulationen mit den beteiligten Methoden, Erstellung von prototypischen Methoden	Erstellung neuer Methoden, Erweiterung des Systems	Konfiguration des Systems
Erfahrungen	Arbeit mit den Methoden	Arbeit mit den Methoden, Programmier-erfahrung (z.B. Fortran oder C)	Programmiererfahrung (Java, Python, Fortran, C/C++)	Betriebssystem-kenntnisse
Einschränkungen	Kann das System nur zum Durchführen von Simulationen (Entwerfen) nutzen	Kann keine neuen Methoden in die zentrale Version einbringen		

Tabelle 4.1-1: Überblick Zielgruppen [SESIS208]

Nachdem nun ein Gesamtüberblick gezeigt wurde, wird nun auf die betroffene Zielgruppe eingegangen. Diese betreffende Zielgruppe stellt die Gruppe der Administratoren dar. Die in dieser Definition beschriebenen Merkmale sollen eine Bedingung bzw. Verständnis des entwickelten Konzeptes gewährleisten.

Administrator:

Ein Administrator bzw. Daten-Administrator konfiguriert die lokale Installation des Systems und vergibt Zugriffs und Nutzungsrechte auf Daten und Methoden [SESIS208].
Des weiteren hat ein Daten-Administrator die Möglichkeit, den Status der Daten anhand des „Status Explorer“ einzusehen. Mittels des Explorers hat der Daten-Administrator die Möglichkeit fehlerhafte oder selten genutzte Daten zu identifizieren und gegebenenfalls zu löschen oder zu archivieren. Der Daten-Administrator muss darüber hinaus gute IT-Kenntnisse im Bereich der Revisionsverwaltung, im Datenmanagement und Kentnisse der verteilten und komponentenbasierten Systeme besitzen.
Neben diesen Kenntnissen muss der Daten-Administrator sich mit diversen Betriebssystemen, wie zum Beispiel Windows 2000/XP/Vista und Linux, auskennen, da die Systeme auf verschiedenen Betriebssystemen laufen könnten.

4.2 Ziele des Datenmanagements

Neben der Zielgruppendefinition wird hier nun auf die Ziele des SESIS-Datenmanagements eingegangen bzw. definiert. Das erste Ziel, ist der Zugriff auf entfernte Daten. Genauer gesagt beschreibt dieses Ziel das verteilte Arbeiten bzw. den Einsatz eines Fileservers. Das verteilte bzw. parallele Arbeiten mit Daten kann sowohl rein lokal als auch in einem Netzwerk geschehen. Das zweite Ziel ist die Unterstützung für das gemeinsame Arbeiten. Dies bedeutet im Einzelnen den Schutz vor ungewollten bzw. gewollten Zerstörung von Daten bzw. in diesem Zusammenhang eine Wiederherstellung und Nachvollziehbarkeit der Daten. Ein weiterer Bestandteil dieses Zieles sind die Zustandsinformationen wie zum Beispiel wer, wann und warum Änderungen an den Daten vorgenommen hat. Diese Informationen sollen mittels Metadaten verfügbar gemacht werden.

Da in diesem Zusammenhang Personenbezogene Daten gespeichert werden und hierdurch Rechtliche Konflikte entstehen können, wird im Anhang unter Kapitel C eine genauere Beschreibung der bestehenden Datenschutzgesetze gegeben.
Der dritte Punkt bezüglich der Ziele beschreibt die Sicherheit des Datenmanagements. Hierbei müssen bestimmte Rechte für Anwender und Methoden definiert und umgesetzt werden. Des weiteren muss die Möglichkeit einer Versionierung bzw. Speicherung einer Historie gewährleistet werden, damit der Zugriff (zum Beispiel aus Testzwecken bzw. zur Fehlerbehebung) auf ältere Versionen ermöglicht wird. Diese Ziele stellen die grobe Anforderungsanalyse an das Datenmanagement dar. Im Rahmen dieser Diplomarbeit wird speziell auf die folgenden Punkte eingegangen.

- Ermittlung des Systemstatus
- Aufdecken von Fehlern im System
- Ermittlung von alten und nicht mehr gebrauchten Daten
- Überblick darüber, wo Daten liegen und ob man durch Verlagerung des Speicherortes die Geschwindigkeit optimieren kann

4.3 Umsetzung des Datenmanagements

Im Zusammenhang mit der Umsetzung sind die folgenden drei Punkte zu beachten : Als erstes wird auf die Kommunikation zwischen den verschiedenen, verteilten RCE-Installationen eingegangen. Die verschiedenen Datenmanagement-Bundles der einzelne RCE-Installationen kommunizieren über ein Communication-Bundle [SESIS108].
Des weiteren wurde bei der Umsetzung geklärt, wie das gemeinsame, parallele Arbeiten ermöglicht wird. In diesem Zusammenhang und wie zuvor auch schon beschrieben existieren zwei typische Ansätze. Diese Ansätze wären das Sperren (Sequenzielles Arbeiten) oder Mergen (paralleles Arbeiten) der Daten. Aufgrund der Anforderung, dass eine paralleles Arbeiten mit den Daten möglich sein soll, wurde der Zugriff auf die Daten mittels Branch / Merge plus Notification realisiert. Diesbezüglich wird unter dem Anhangspunkt A eine genauere Beschreibung der elementaren Operationen gegeben.

Diese Realisierung ermöglicht zwar auf der einen Seite das parallele Arbeiten mit den Daten, aber auf der anderen Seite kann es hierdurch zu Konfliktsituationen, zum Beispiel beim Mergen zweier Revisionen, die in der selben Zeile unterschiedliche Änderungen haben, kommen. Weitere Konfliktsituationen werden unter dem Kapitel 6 “Konzept zur Fehleraufdeckung im verteilten SESIS-Datenmanagement“ dargestellt.
Der dritte und letzte Punkt der Umsetzung bezieht sich auf die Sicherheit. Hierbei ist die Definition und Umsetzung der Regeln von größter Wichtigkeit. In diesem Zusammenhang wurde das Privilegien-Bundlet erweitert. Dieses Bundlet steuert den Zugriff auf die Daten. Die Versionierung wurde ähnlich einem RCS (Revision-Control-System) umgesetzt [GNU08].

4.4 Beispiel für eine Datenreferenz im SESIS-Datenmanagement

Eine Datenreferenz ist bildlich gesehen wie eine Aktenschublade. Aus diesem Grund existiert für jede Datei eine eigene Schublade. Des weiteren existiert für jede Revision einer Datei ein Hängeordner in der entsprechenden Schublade. In einem Hängeordner werden dann die Datei / Datensatz und Zusatzinformationen (Metadaten) gespeichert. Zusätzlich werden in einem Hängeordner auch Metainformationen, zum Beispiel wann und wer ein Branch der Datenreferenz durchgeführt hat, über die Parant-Referenz gespeichert [SESIS108]. Ein Beispiel für das Datenmanagement kann unter der Abbildung “3.3.2-1 Beispiel für ein SESIS-Datenmanagement-Szenario“ eingesehen werden. Die Abbildung zeigt ein mögliches Szenario bezüglich Branch und Merge-Aktivitäten.

4.5 Benutzungsmerkmale des Datenmanagements

Ein wichtiger Punkt bei den Benutzungsmerkmalen ist der, dass der Anwender keinen direkten Kontakt mit dem Datenmanagement hat. Es besteht nur die Möglichkeit, indirekt über den Dataexplorer, Projektbrowser oder die Methoden auf das Datenmanagement zuzugreifen. Eine weitere Möglichkeit bezüglich des Zugriffes auf das Datenmanagement wird mittels des später beschrieben “Status Explorer“ ermöglicht.

Der Dataexplorer hat die Aufgabe, Datenreferenzen und ihre Metadaten anzuzeigen [SESIS108]. Informationen können nicht nur für die lokale RCE-Installation, sondern auch für entfernte RCE-Installationen gezeigt werden. Des weiteren können mittels des Dataexplorers Datenreferenzen zu einem Projekt hinzugefügt werden [SESIS108]. Der Projektbrowser stellt das gesamte Projekt mit seinen Abhängigkeiten dar. Im Projektbrowser wird bei jedem Speichern einer Datei (Datenreferenz) automatisch eine neue Revision erstellt [SESIS108]. Methoden bieten die Möglichkeit, mit den Datenreferenzen, Daten und Metadaten zu arbeiten bzw. zuzugreifen. Die Methoden sollten per DM-SDK mit Datenreferenzen hantieren.

4.6 Momentaner Stand

Ein Einsatz des Datenmanagements ist zur Zeit lokal und remote möglich. Darüber hinaus lassen sich Datenreferenzen versionieren und branchen. Eine weitere implementierte Funktionalität ist es, mittels Suchanfragen bestimmte Datenreferenzen bzw. Metadaten ausfindig zu machen. Der Speicherort der Daten ist das Eclipse File System. Des weiteren steht das SDK für die Methodenprogrammierer zur Verfügung [SESIS108].

4.7 Nächste Schritte

Es soll zukünftig möglich sein, revisionsunabhängig Metadaten zu speichern bzw. zu benutzen. Weiterhin soll der Projektbrowser und der Dataexplorer durch weitere GUI-Elemente erweitert werden. Außerdem sollen Integrationstests implementiert werden. Diese Integrationstests sollen die Performance prüfen und optimieren. Außerdem sollen die Integrationstests das Handling bei dem gemeinsamen Arbeiten prüfen und optimieren (Out-of-sync, Notifikation).
Ein weiterer Schritt ist das Mergen für die ausgewählten Daten [SESIS108;SESIS208]. Des weiteren stellen die weiteren Punkte dieser Diplomarbeit ebenfalls mögliche weitere Schritte dar. Der abschließende Schritt umfasst Junit-Tests und Dokumentation, die im Laufe des Entwicklungsprozesses vervollständigt werden.

5 Konzept zur Ermittlung von alten und nicht mehr gebrauchten Daten

Die Ermittlung von alten oder nicht mehr gebrauchten Daten muss aus drei Hauptgründen in zwei automatisierte Ermittlungskonzepte vollzogen werden. Das erste Ermittlungskonzepte stellt die lokale Ermittlung von Abhängigkeiten im SESIS-Datenmanagement dar. Hierbei werden nur, wie durch den Namen beschrieben, die lokal Daten überprüft, die von einer Aktion betroffenen sind. Das zweite Ermittlungskonzept stellt die globale Ermittlung von Abhängigkeiten dar. Bei diesem Konzept wird der gesamte Kontext der Daten betrachtet und überprüft. Neben diesen zwei automatisierten Ermittlungskonzepte sollte es aber auch die Möglichkeit existieren, dass ein Daten-Administrator eine Ermittlung auch manuell starten kann. Hierzu wird unter Kapitel 5.5 “Manuelles Starten der globalen Ermittlung von Abhängigkeiten“ genauer auf den Sachverhalt eingegangen. Bevor auf die zwei automatisierten Ermittlungskonzepte eingegangen wird, werden nun an dieser Stelle die besagten drei Gründe beschrieben.

Der erste Grund dieser Unterteilung ist die Performance der einzelnen Systeme. Dies bedeutet, da gegebenenfalls mit sehr vielen unterschiedlichen Daten arbeiten wird, dass gewährleistet werden muss, dass eine derartige Ermittlung schnell und komplett durchführbar ist. Hierdurch wird es zudem ermöglicht, mehrere Ermittlungen sequentiell durchzuführen.

Der zweite Grund für dieses Vorgehen ist der, dass durch so eine Ermittlung gewährleistet werden muss, dass hierdurch nicht der Arbeitsprozess verlangsamt oder gar verhindert wird. Dies kann zum Beispiel geschehen, wenn eine Ermittlung von alten oder nicht mehr gebrauchten Daten automatisch gestartet wird, während mehrere Benutzer mit den betroffenen Daten arbeiten. Dies kann abhängig vom Umfang der Daten bedeuten, dass die Ermittlung so viel Zeit und Performance in Anspruch nimmt, dass die reguläre Arbeit an dem betroffenen System bzw. Daten komplett für den Zeitraum der Ermittlung zum Erliegen kommt.

Der dritte Grund ist eine mögliche Verhinderung entstehender Kosten, die zum Beispiel durch die zuvor beschriebene Blockade des Arbeitsprozesses entstehen könnte. Diese Kosten können aber durch eine geeignete Ermittlungsstrategie verhindert werden, da die Ermittlung der Daten auf einen Zeitpunkt gesetzt wird, in dem nicht mit den betroffenen Daten gearbeitet wird.

Ein weiterer sehr wichtiger Punkt ist der, dass Lösch- oder Archivierungsmaßnahme niemals automatisch durchgeführt werden dürfen. Unter den folgenden Kapiteln wird zwar von Lösch- oder Archivierungsmaßnahme gesprochen, aber tatsächlich werden alle gefundenen Daten bzw. deren Metadaten, die alt sind oder nicht mehr gebraucht werden, in einer XML-Datei oder “Apache Derby“-Datenbank gespeichert. Diese Datei steht dem Daten-Administrator jederzeit zur Verfügung. Wie zuerst vermutet, hat diese Datei kein Aktualitätsproblem. Ein Beispiel hierfür ist, wenn zwar eine globale Ermittlung von Abhängigkeiten durchgeführt wurde, aber kurz danach auf eine der gefundenen Revisionen bzw. Datenreferenzen eine Rückwertsreferenz zeigt. In diesem Fall würde man bei einer manuellen Durchführung durch den Daten-Administrator eine tatsächliche Lösch- oder Archivierungsmaßnahme an einer Revision oder Datenreferenz durchführen, die fatale Folgen hätte. Durch diese Branch-Aktion wird aber ein Teilbereich der lokalen Ermittlung von Abhängigkeiten gestartet (siehe Kapitel 5.1.1 “Startauslöser der lokalen Ermittlung von Abhängigkeiten“) der vor der Überprüfung in der XML-Datei oder
“Apache Derby“-Datenbank überprüft, ob die Revision oder Datenreferenz, auf die ein Branch durchgeführt wurde, schon eingetragen ist. Falls dies zutreffend sein sollte, wird diese aus der XML-Datei oder “Apache Derby“-Datenbank entfernt. Der
Daten-Administrator hat nun jederzeit die Möglichkeit mittels der XML-Datei oder “Apache Derby“-Datenbank bzw. anhand des “Status Explorers“, welcher im späterem Verlauf beschrieben wird, alte und nicht mehr benötigte Daten zu identifizieren und eine Lösch- oder Archivierungsmaßnahme durchzuführen. Neben der Archivierung muss es des weiteren die Möglichkeit der Löschung von Revisionen oder Datenreferenzen geben. Der Grund hierfür ist rechtlicher Natur, da es vorkommen kann, dass Systembestandteile weitergegeben werden, aber die Revisionen oder Datenreferenzen unter das Datenschutzgesetz fallen und nicht weitergegeben werden dürfen. In diesem Fall müssen die Daten gelöscht werden, bevor eine Weitergabe möglich ist. Die besagte Löschung muss zudem explizit durchgeführt werden. Des weiteren wird parallel zu dem lokal gespeicherten Datenkatalog, eine XML-Datei bzw. ein “Apache Derby“-Datenbank gespeichert, die die komplette Branch- und Merge-Abhängigkeit darstellt.

Mit komplett ist gemeint, dass jede Plattform bzw. jedes SESIS-System, dass auf die lokal liegenden Revisionen oder Datenreferenzen ein Branch durchgeführt hat, erfasst wird, aber auch die Systeme, die von den besagten Systemen branchen werden ebenfalls erfasst. Die Tiefe ist hierbei beliebig tief. Eine genauere Beschreibung diesbezüglich wird unter dem Kapitel 8 (“Konzept zur Statusermittlung des System“) gegeben. Die Ermittlung der Abhängigkeiten wird mittels einer Broadcast-Anfrage ermöglicht. Die Revisionen bzw. Datenreferenzen, die eine direkte bzw. indirekte Abhängigkeit besitzen, antworten auf diesem Broadcast mit den entsprechenden Metadaten für die Revision bzw. Datenreferenz.
Man kann diese Datei als ein erweiterten Datenkatalog betrachten, da er nicht nur die lokalen Daten beinhaltet, sondern auch die Plattform übergreifenden Abhängigkeiten, die durch eine Branch-Aktion entstehen. Zusätzlich zu den Metadaten wäre es denkbar einen “Counter“ zu speicheren, der die Anzahl der Rückwertsreferenz darstellt. Ist der “Counter“ null, so kann in Erwägung gezogen werden, die Daten zu löschen oder zu archivieren. Dieses Vorgehen ähnelt dem eines “Journaling Filesystem“ [JFS08]. Bezüglich der Archivierung bzw. der Alternative einer Löschaktion wird unter dem Anhangspunkt D (“Mögliche Alternative zur Löschung von nicht mehr gebrauchten Daten“) eine genauere Beschreibung gegeben.

5.1 Lokale automatische Ermittlung von Abhängigkeiten

Die lokale Ermittlung von Abhängigkeiten nutzt bzw. überprüft nicht das gesamte / globale SESIS-Datenmanagement, sondern nur die lokalen Daten. Diese Ermittlung wird anhand einer Aktivierung durch einer der hieran folgenden Startauslöser automatisch gestartet. Des weiteren nutzt die lokale Ermittlung, bezüglich der Merk-Funktion, die gleiche XML-Datei oder “Apache Derby“-Datenbank wie die globale Ermittlung, welche im späterem Verlauf beschrieben wird. Dies ist wichtig, da die ermittelten Revisionen bzw. Datenreferenz nicht doppelt in die zuvor beschriebene Datei oder Datenbank eingetragen werden. Ein weiterer Punkt ist der, dass die lokale Ermittlung bei einer Branch-Aktion prüfen muss, ob die Revision, auf die eine Branch-Aktion durchgeführt wurde, in der XML-Datei oder “Apache Derby“-Datenbank eingetragen ist.

Falls ja, muss dieser Eintrag durch das lokale Ermittlungskonzept entfernt werden, da diese Revision durch den Branch wieder benötigt wird.
Die lokale Ermittlung untersucht lediglich die lokalen Daten die durch die Aktion betroffen sind. Hierbei wird untersucht, wie viele Rückwertsreferenzen bzw. Branches auf die vorige Revision zeigen. Falls diese null sein sollte und eine Folgerevision existiert, kann in Erwägung gezogen werden diese zu löschen oder zu archivieren. Auf jeden Fall wird eine entsprechende Information im “Status Explorer“ gezeigt, um dem Daten-Administrator eine Übersicht über die alten oder nicht mehr gebrauchten Daten zu geben. Bei sämtlichen Änderungen am Datenmodell muss der Datenkatalog angepasst werden. Bevor die lokale Ermittlung gezeigt wird, wird unter dem folgenden Unterkapitel zunächst die definierten Startauslöser gezeigt.

5.1.1 Startauslöser der lokalen Ermittlung von Abhängigkeiten

Nachdem nun die Gründe erläutert worden sind, wird nun auf die Startauslöser der lokalen Ermittlung von Abhängigkeiten eingegangen. Die lokale Ermittlung beinhaltet das kontinuierliche Ermitteln von alten oder nicht mehr gebrauchten Daten. Unter einer kontinuierlichen Ermittlung ist gemeint, dass die lokale Ermittlung immer dann automatisch gestartet wird, wenn eine der drei folgenden Aktionen durch den Benutzer durchgeführt wird. Hierdurch kann man die lokale Ermittlung als einen Dienst bezeichnen, der auf bestimmte definierte Aktionen wartet. Die zweite Aktion (Branch-Aktion) hat neben dem Starten der lokalen Ermittlung von Abhängigkeiten eine weitere sehr wichtige Aufgabe. Sie überprüft, ob die betreffende Revision bzw. Datenreferenz schon ich der zuvor beschriebenen
XML-Datei oder “Apache Derby“-Datenbank eingetragen ist. Falls ja, muss diese aus der Liste der alten und nicht mehr benötigten Daten entfernt werden, da sie durch diese Aktion wieder benötigt wird. Die folgende Aufzählung zeigt nun die drei möglichen Startauslöser die im weiteren Verlauf genauer beschrieben werden.

1. Speichern einer Änderung und dadurch Erzeugung einer neuen Revision
2. Branch-Aktion
3. Merge-Aktion

5.1.1.1 Erster Startauslöser (Commit / Save)

Die im Folgenden dargelegten drei Startauslöser wurden aus den folgenden Gründen gewählt : Der Grund für den ersten Startauslöser ist der, dass durch das Speichern ein Änderung und das daraus resultierende Erzeugen einer neuen Revision ältere Revision nicht mehr benötigt werden könnten. Der Nutzen einer älteren Revision könnte aber dennoch bezüglich einer möglichen Fehleranalyse, Ursachenanalyse oder bezüglich der Nachvollziehbarkeit bestehen. Aus diesem Grund ist von einer Löschung der betroffenen Revisionen abzuraten. Alternativ zu einer Löschung kann eine Archivierung der Revision in Betracht gezogen werden (siehe Kapitel D “Mögliche Alternative zur Löschung von nicht mehr gebrauchten Daten“). Natürlich ist zu beachten, dass durch das Löschen oder Archivieren der Revision große Probleme, wie zum Beispiel dass auf die zu löschende oder zu archivierende Revision eine Rückwertsreferenz zeigt, auftreten können. Auf diese Probleme wird unter der Beschreibung der lokalen Ermittlung von Abhängigkeiten genauer eingegangen. Anhand der folgenden Abbildung wird ein Beispiel gezeigt, wie der gerade beschriebene Startauslöser aussehen könnte.

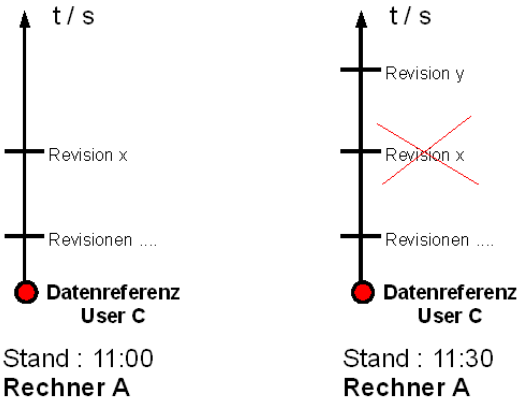

Abbildung 5.1.1.1-1: Beispiel für ersten Startauslöser: Speichern einer Änderung und dadurch Erzeugung einer neuen Revision

(Eigener Entwurf)

5.1.1.2 Zweiter Startauslöser (Branch)

Der Grund für den zweiten Startauslöser ist der, dass durch ein Branch einer Revision zu beachten ist, ob zuvor schon von einer früheren Revision der selben Datenreferenz eine Branch-Aktion durchgeführt wurde, ohne jemals von dem selben Benutzer eine Merge-Aktion erfahren zu haben. Mit dieser Situation ist ein Branch gemeint, bei dem nicht mehr das Ziel einer Merge-Aktion verfolgt wird. In diesem Fall kann die Entwicklung in diesem Zweig bei einer auslösenden Branch-Aktion gelöscht bzw. archiviert werden, da diese quasi so betrachtet werden kann, als wäre sie überschrieben worden. Aber auch hier ist zu beachten, dass Probleme bei der Löschung bzw. Archivierung auftreten können. Wie zuvor existiert auch hier das Problem einer bestehenden Rückwertsreferenz auf eine der Revisionen des Astes welches gelöscht oder archiviert werden soll. Beim Entfernen dieses Astes müsste diese Rückwertsreferenz entweder eine Löschung oder eine Archivierung verhindern oder angepasst werden. Darauf wird später noch eingegangen werden. Bezüglich einer Löschung ist aber eine Archivierung auch hier zu bevorzugen, da die Nachvollziehbarkeit eine sehr wichtige Rolle in einem Entwicklungsprozess spielt. Eine genauere Beschreibung der Alternative wird im Anhang unter Kapitel D gegeben. Anhand der folgenden Abbildung wird ein Beispiel gezeigt, wie der gerade beschriebene Startauslöser aussehen könnte.

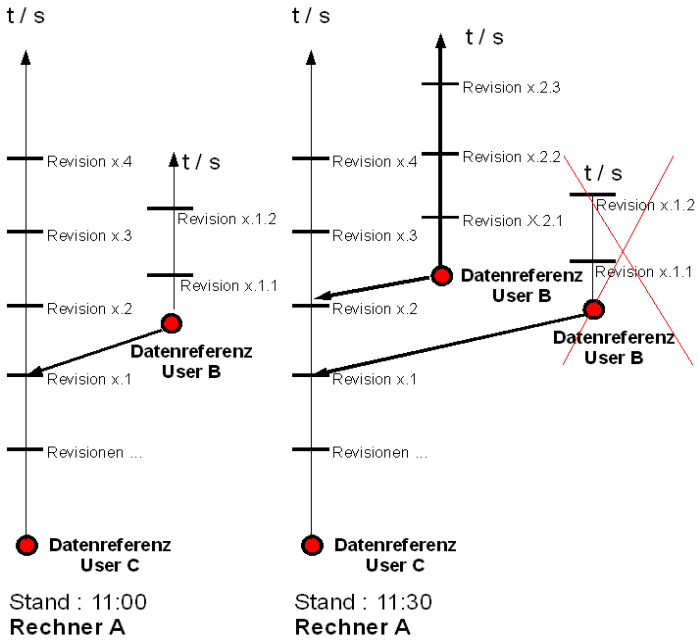

Abbildung 5.1.1.2-1: Beispiel für zweiten Startauslöser: Branch-Aktion

(Eigener Entwurf)

5.1.1.3 Dritter Startauslöser (Merge)

Der Grund für den dritten Startauslöser ist der, dass durch eine Merge-Aktion einer Revision, bei der zu beachten ist dass der Entwicklungszweig nach der Aktion nicht weitergeführt wird, der Zweig, aus dem die Merge-Aktion gestartet wurde, nicht mehr benötigt wird. Dies kommt daher, dass die Entwicklung in den Zweig, auf dem zuvor eine Branch-Aktion durchgeführt wurde, zurückgeführt wird und ab diesem Zeitpunkt in diesem verfügbar ist. Falls aber der Entwicklungszweig doch fortgeführt werden soll, kann dies erreicht werden, in dem man direkt nach der Merge-Aktion einen neuen Branch durchführt. Hierdurch erhält man eine neue Datenreferenz mit der man nun weiter arbeiten kann.

Auch wie bei den Aktionen zuvor muss beachtet werden, dass sich eine Rückwertsreferenz auf eine oder mehrere betreffende Revisionen zeigen könnte. Aber auch wie bei den zwei Startauslösern zuvor sollte auch in diesem Fall von einer Löschung abgeraten werden, da auch hier durch eine Archivierung die Nachvollziehbarkeit bzw. eine Fehleranalyse möglich wird.
Anhand der folgenden Abbildung wird ein Beispiel gezeigt, wie der gerade beschriebene Startauslöser aussehen könnte.

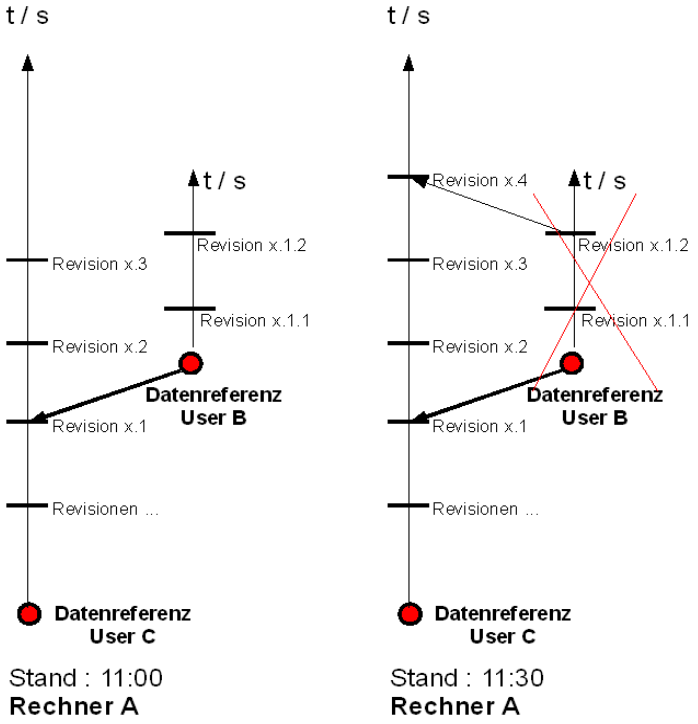

Abbildung 5.1.1.3-1: Beispiel für dritten Startauslöser: Merge-Aktion

(Eigener Entwurf)

5.1.2 Lokale Ermittlung von Abhängigkeiten

An dieser Stelle wird nun auf die lokale automatische Ermittlung von Abhängigkeiten eingegangen. Hierbei wird jeweils zuerst das entsprechende Flussdiagramm des Startauslösers gezeigt und anschließend beschrieben. In der folgenden Abbildung wird nun das Flussdiagramm des ersten Startauslösers gezeigt.

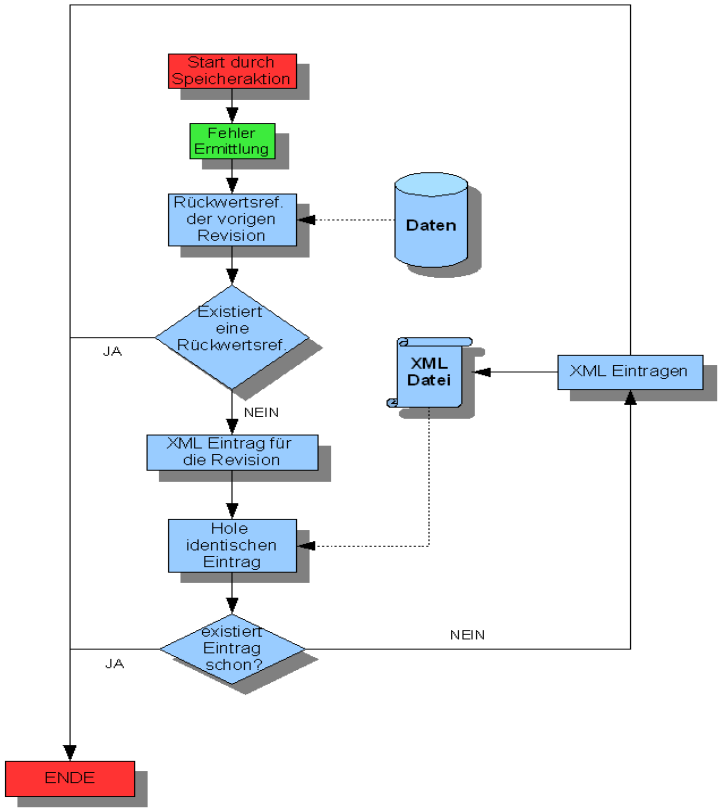

Abbildung 5.1.2-1: Flussdiagramm des ersten Startauslösers bei der lokalen Ermittlung von Abhängigkeiten

(Eigener Entwurf)

Nachdem nun das Flussdiagramm gezeigt wurde, wird nun dieses beschrieben.
Die rot gekennzeichneten Kästchen stellen den Startauslöser bzw. das Ende der Ermittlung dar. Das grün gekennzeichnete Kästchen hingegen stellt die Fehlerermittlung dar. Auf die konkrete Fehlerermittlung wird im späterem Verlauf eingegangen. Dennoch ist an dieser Stelle zu erwähnen, dass die Ermittlung der Fehler bei jedem Startauslöser durchgeführt wird. Als erstes wird nun auf den ersten Startauslöser eingegangen. Der erste Startauslöser wird, wie zuvor schon beschrieben, durch die Speicherung und damit durch die Neu-Erzeugung einer Revision ausgelöst. Nachdem diese Aktion ausgelöst wurde, wird versucht, die mögliche Rückwertsreferenz der vorherigen Revision zu bekommen. Die vorherige Revision ist die Revision, in der die Änderungen durchgeführt und gespeichert wurden. Wenn nun eine Rückwertsreferenz auf die vorherige Revision zeigt, darf diese unter keinen Umständen gelöscht oder archiviert werden. Aus diesem Grund wird die Ermittlung hiernach beendet. Existiert aber keine Rückwertsreferenz auf die vorige Revision, so wird ein entsprechender XML- oder “Apache Derby“-Eintrag erzeugt. Anschließend wird geschaut ob dieser Eintrag schon in der XML-Datei oder “Apache Derby“-Datenbank eingetragen wurde. Diese XML-Datei oder
“Apache Derby“-Datenbank beinhaltet alle Revisionen und Datenreferenzen die gelöscht oder archiviert werden können. Ein entsprechendes Beispiel für eine solche XML-Datei wird am Ende dieses Kapitels gegeben. Wenn dieser Eintrag schon in der Datei vorhanden sein sollte, so wird diese Ermittlung ebenso beendet. Falls der Eintrag aber nicht vorhanden sein sollte, so wird dieser erzeugte
XML- oder “Apache Derby“-Eintrag in die XML-Datei oder
“Apache Derby“-Datenbank eingetragen, und anschließend wird aber auch diese Ermittlung beendet. Nachdem nun das Vorgehen nach dem ersten Startauslöser beschrieben wurde, wird nun auf den zweiten Startauslöser eingegangen. In der folgenden Abbildung wird nun das Flussdiagramm des zweiten Startauslösers gezeigt.

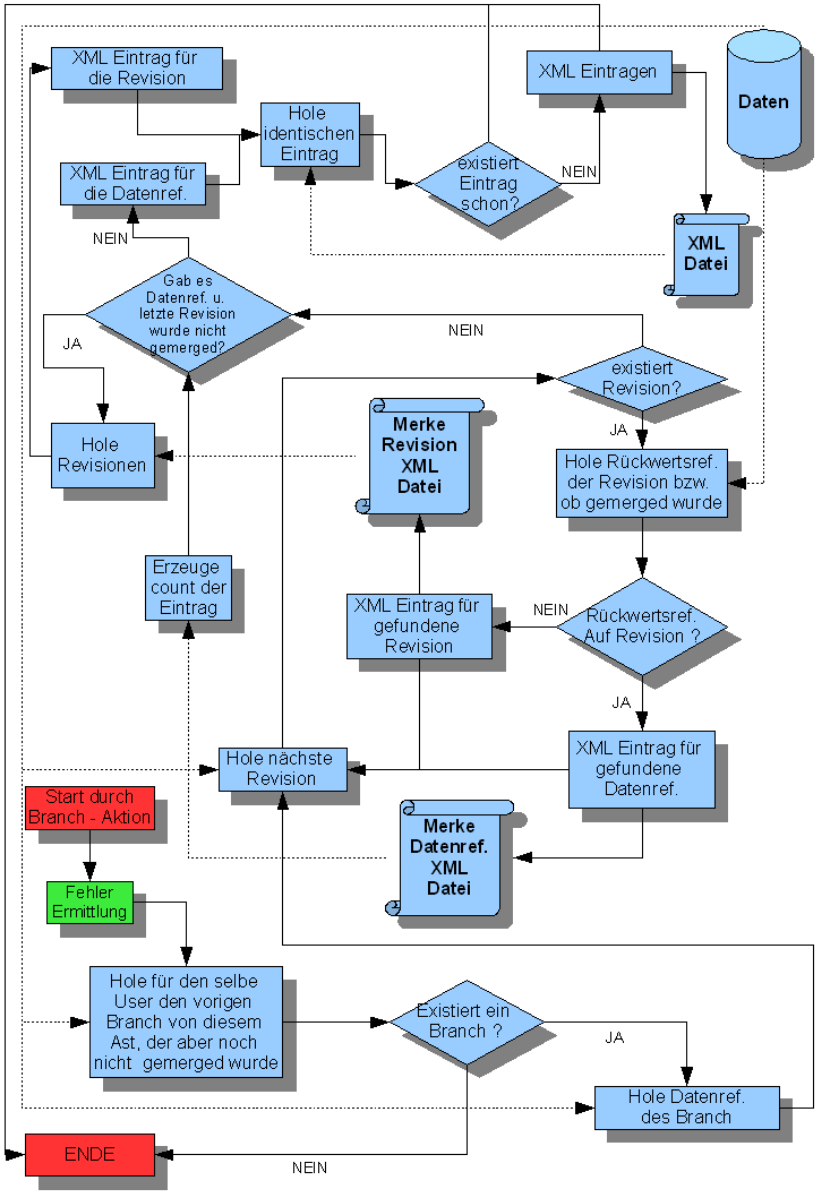

Abbildung 5.1.2-2: Flussdiagramm des zweiten Startauslösers bei der lokalen Ermittlung von Abhängigkeiten

(Eigener Entwurf)

Nachdem nun das Flussdiagramm gezeigt wurde, wird nun dieses beschrieben. Der zweite Startauslöser wird durch eine Branch-Aktion ausgelöst. Wenn diese Aktion gestartet wurde, wird als erstes geschaut, ob der selbe Benutzer schon einmal auf diesen Ast eine Branch-Aktion durchgeführt hat, aber noch nicht eine entsprechende Merge-Aktion. Falls dies der Fall sein sollte, wird die Datenreferenz, die durch den vorigen Branche entstanden ist, geholt. Falls dies aber nicht der Fall sein sollte, wird die Ermittlung beendet. Wenn nun aber eine Datenreferenz existieren sollte, so wird beim ersten Start die erste Revision, ansonsten immer die nächste Revision dieser Datenreferenz geholt. Wenn nun eine Revision existiert, wird geschaut, ob diese eine Rückwertsreferenz besitzt. Falls diese eine solche Referenz besitzt, wird lediglich ein Vermerk über diese Datenreferenz in eine XML-Datei oder eine “Apache Derby“-Datenbank vorgenommen, damit diese später nicht gelöscht oder archiviert werden kann. Diese XML-Datei ähnelt sehr der zuvor beschriebenen XML-Datei, die am Ende dieses Kapitels anhand eines Beispiels gezeigt wird. Nach dem XML- oder “Apache Derby“-Eintrag für die Datenreferenz wird die nächste Revision derselben geholt. Falls keine Rückwertsreferenz auf die gefundene Revision zeigen sollte, so wird diese Revision in einer XML-Datei oder einer “Apache Derby“-Datenbank gemerkt. Anschließend wird ebenso die nächste Revision geholt. Diese Schleife wird erst beendet, wenn die Datenreferenz keine Revisionen mehr hat, die überprüft werden können. Wenn nun keine Revision mehr vorhanden sein sollte, wird geschaut, ob es eine Revision in dieser Datenreferenz gab, die eine Rückwertsreferenz hatte. Falls dies der Fall sein sollte, so existiert ein entsprechender Eintrag für diese Datenreferenz in der XML-Datei oder “Apache Derby“-Datenbank. Dieser Eintrag soll verhindern, dass diese Datenreferenz gelöscht wird. Falls nun kein Eintrag für die Datenreferenz existiert und die letzte Revision dieser Datenreferenz durch eine Merge-Aktion verschmolzen wurde, wird ein entsprechender XML- oder “Apache Derby“-Eintrag für die Datenreferenz erzeugt. Nach dieser Erzeugung wird nun geschaut, ob diese Datenreferenz schon in der XML-Datei oder “Apache Derby“-Datenbank vorhanden ist. Falls der Eintrag schon vorhanden sein sollte, so wird die Ermittlung beendet, ohne eine Maßnahme durchzuführen.

Falls der Eintrag aber noch nicht vorhanden sein sollte, so wird dieser XML- oder “Apache Derby“-Eintrag in die XML-Datei oder “Apache Derby“-Datenbank durchgeführt und anschließend wird die Ermittlung beendet.
Dies hat nun die Folge, dass der Daten-Administrator nun im “Status Explorer“ diese Datenreferenz als löschbar bzw. archivierbar erkennen kann und entsprechende Maßnahmen einleiten könnte. Falls aber nun eine Datenreferenzeintrag in der XML-Datei oder “Apache Derby“-Datenbank existieren sollte, so darf unter keinem Umständen die Datenreferenz gelöscht werden. Daher werden in diesem Fall alle Revisionen, die keine Rückwertsreferenz besitzen, geholt, die der betreffenden Datenreferenz gehören. Diese Revisionen wurden zuvor schon, beim Schleifendurchlauf in eine entsprechende XML-Datei oder eine “Apache Derby“-Datenbank gespeichert und müssen nun nur noch aus der XML-Datei oder “Apache Derby“-Datenbank abgefragt werden. Ebenso wie zuvor, gleicht auch hier diese XML-Datei dem am Ende folgenden Beispiel. Bei jeder Ermittelten Revision wird aber auch hier geprüft, ob diese schon in der Merk-XML-Datei oder “Apache Derby“-Datenbank vorhanden ist. Falls ja, so wird die nächste Revision, die keine Rückwertsreferenz besitzt, geholt und überprüft. Falls aber diese Revision noch nicht in der XML-Datei oder “Apache Derby“-Datenbank vorhanden sein sollte, so wird der erzeugte Eintrag in die XML-Datei oder “Apache Derby“-Datenbank geschrieben. Dies wird nun für alle in der Merk-XML-Datei oder “Apache Derby“-Datenbank gespeicherten Revisionen durchgeführt. Wenn nun keine Revisionen mehr gefunden werden, so wird die Ermittlung ohne weitere Maßnahme beendet. Nachdem nun das Vorgehen nach dem zweiten Startauslöser beschrieben wurde, wird nun auf den dritten Startauslöser eingegangen. In der folgenden Abbildung wird nun das Flussdiagramm des dritten und letzten Startauslösers gezeigt.

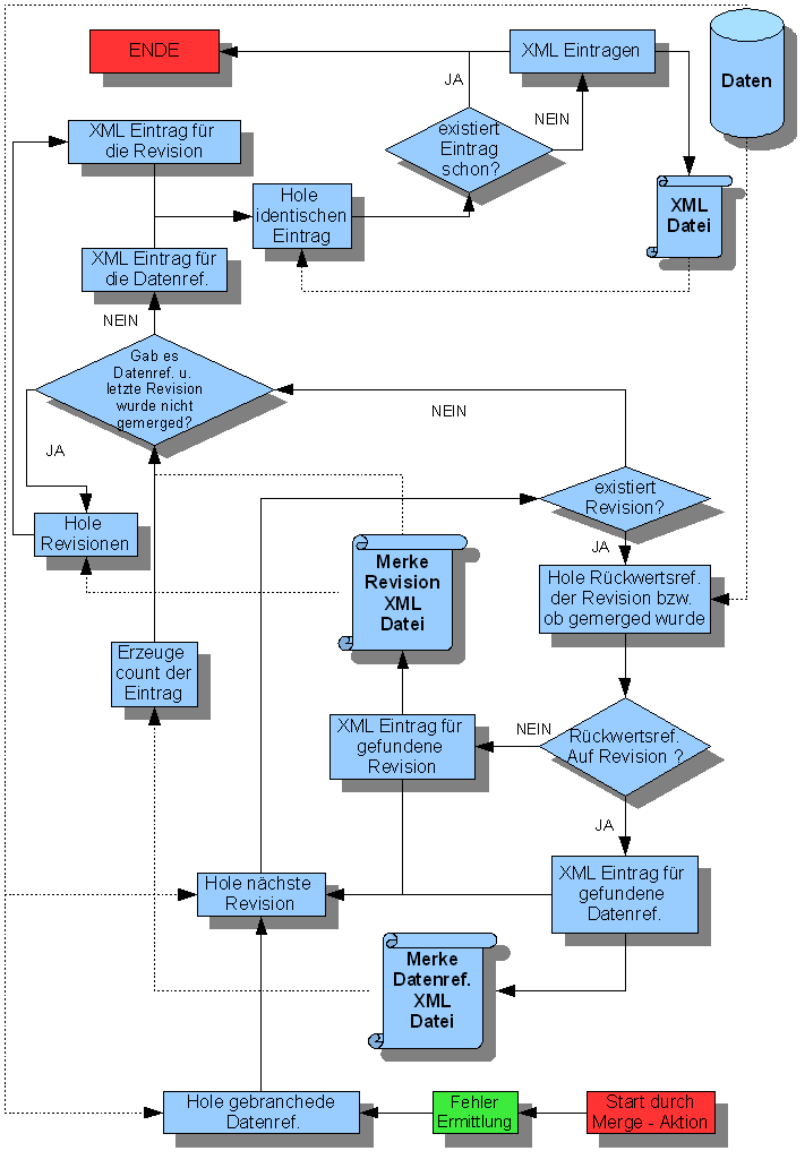

Abbildung 5.1.2-3: Flussdiagramm des dritten Startauslösers bei der lokalen Ermittlung von Abhängigkeiten

(Eigener Entwurf)

Nachdem nun das Flussdiagramm gezeigt wurde, wird nun dieses beschrieben. Als letztes wird nun auf den dritten Startauslöser eingegangen, Bei welcher es sich um die Merge-Aktion handelt. Bei dieser Aktion wird als erstes die Datenreferenz geholt, deren Revision gemerged werden soll. Wenn nun diese ermittelt wurde, wird wie folgt vorgegangen: Wenn nun eine Datenreferenz existieren sollte, so wird beim ersten Start die erste Revision, ansonsten immer die nächste Revision geholt. Wenn nun eine Revision existiert, wird geschaut, ob diese eine Rückwertsreferenz besitzt. Falls diese eine solche Referenz besitzt, wird lediglich ein Vermerk über diese Datenreferenz in eine XML-Datei oder eine “Apache Derby“-Datenbank vorgenommen, damit diese später nicht gelöscht oder archiviert werden kann. Nach dem XML- oder “Apache Derby“-Eintrag für die Datenreferenz wird die nächste Revision derselben, wie auch zuvor, geholt. Falls keine Rückwertsreferenz auf die gefundene Revision zeigen sollte, so wird dies Revision in einer XML-Datei oder einer “Apache Derby“-Datenbank gemerkt. Anschließend wird ebenso die nächste Revision geholt. Diese Schleife wird wie auch zuvor erst beendet, wenn die Datenreferenz keine Revisionen mehr hat. Wenn nun keine Revision mehr vorhanden sein sollte, wird geschaut ob, es eine Revision in dieser Datenreferenz gab, die eine Rückwertsreferenz hatte. Falls dies der Fall sein sollte, so existiert ein entsprechender Eintrag für diese Datenreferenz in der XML-Datei oder “Apache Derby“-Datenbank. Dieser Eintrag soll verhindern, dass diese Datenreferenz gelöscht wird. Falls nun kein Eintrag für die Datenreferenz existiert und die letzte Revision dieser Datenreferenz durch eine Merge-Aktion verschmolzen wurde, wird ein entsprechender XML- oder “Apache Derby“-Eintrag erzeugt. Nach dieser Erzeugung wird nun geschaut, ob diese Datenreferenz schon in der
XML-Datei oder “Apache Derby“-Datenbank vorhanden ist. Falls der Eintrag schon vorhanden sein sollte, so wird die Ermittlung beendet, ohne eine Maßnahme durchzuführen. Falls der Eintrag aber noch nicht vorhanden sein sollte, so wird dieser XML- oder “Apache Derby“-Eintrag in die XML-Datei oder
“Apache Derby“-Datenbank durchgeführt und anschließend wird die Ermittlung beendet. Dies hat nun die Folge, dass der Daten-Administrator nun im “Status Explorer“ diese Datenreferenz als löschbar bzw. archivierbar erkennen und entsprechende Maßnahmen einleiten kann.

Falls aber nun eine Datenreferenzeintrag in der XML-Datei oder
“Apache Derby“-Datenbank existieren sollte, so darf unter keinem Umständen die Datenreferenz gelöscht werden. Daher werden in diesem Fall alle Revisionen, die keine Rückwertsreferenz besitzen, geholt die der betreffenden Datenreferenz gehören. Diese Revisionen wurden zuvor schon, beim Schleifendurchlauf, in eine entsprechende XML-Datei oder eine “Apache Derby“-Datenbank gespeichert und müssen nun nur noch aus der XML-Datei oder “Apache Derby“-Datenbank abgefragt werden. Bei jeder Ermittelten Revision wird aber auch hier geprüft, ob diese schon in der Merk-XML-Datei oder “Apache Derby“-Datenbank vorhanden ist. Falls ja, so wird die nächste Revision, die keine Rückwertsreferenz besitzt, geholt, und überprüft. Falls aber diese Revision noch nicht in der XML-Datei oder “Apache Derby“-Datenbank vorhanden sein sollte, so wird der erzeugte Eintrag in die
XML-Datei oder “Apache Derby“-Datenbank geschrieben. Dies wird nun für alle in der Merk-XML-Datei oder “Apache Derby“-Datenbank gespeicherten Revisionen durchgeführt. Wenn nun keine Revisionen mehr gefunden werden, so wird die Ermittlung ohne weitere Maßnahme beendet. Wichtig zu beachten ist noch, dass eine Datenreferenz nur gelöscht oder archiviert wird, wenn diese durch eine
Merge-Aktion verschmolzen und nach dem Merge nicht weiter geführt wurde.
Des weiteren darf von keiner Revision dieser Datenreferenz gebranched worden sein. Bei der Löschung und Archivierung von Revisionen ist zu beachten, dass die letzte Revision einer Datenreferenz niemals gelöscht oder archiviert wird, egal ob auf diese eine Rückwertsreferenz zeigt oder nicht. Diese Revision stellt den letzten Stand dar. Diese wichtigen Punkte gelten ausnahmslos für alle Startauslöser.
Das nun hieran folgende XML-Beispiel gilt sowohl für die beiden Merk-XML-Datei als auch für die XML-Datei, die alle alten und nicht mehr benötigten Daten speichert. Wie zuvor schon beschrieben wurde, ist die Speicherung in diesem Konzept mittels XML realisiert, kann aber ohne weiteres auch als “Apache Derby“-Lösung realisiert werden.

```
<Notused>
        <Revision Key="xyz1">
                <Author>Otto</Author>
                <Date>11.01.2008</Date>
                <Extension>abc</Extension>
                <Size>123123</Size>
                <Name>Tatjana</Name>
        </Revision>
        <Datareference Key="xyz2">
                <Author>Hans</Author>
                <Date>11.10.2008</Date>
                <Extension>cba</Extension>
                <Size>321321</Size>
                <Name>Flo</Name>
        </Datareference>
</Notused>
```

5.2 Globale automatische Ermittlung von Abhängigkeiten

Die globale Ermittlung von Abhängigkeiten enthält die Optimierung der gesamten / globalen Daten. Diese Optimierung sollte aus Performancegründen automatisiert an Tagen wie zum Beispiel am Wochenende, an Feiertagen oder zu Uhrzeiten wie zum Beispiel von 00:00 bis 05:00 durchgeführt werden. Allgemein sollte der Zeitpunkt so gewählt werden, dass zum Zeitpunkt der Ermittlung niemand mit den betroffenen Daten arbeitet. Falls die Entscheidung für den Durchführungszeitpunkt Täglich um 01:00 ist, muss aber das Problem der Zeitzonen beachtet werden, da auch Systeme in einer Zeitzone laufen könnten, die eine Zeitverschiebung von zum Beispiel sechs Stunden haben. In diesem Fall wird die Ermittlung, in der entsprechenden Zeitzone, nicht um 01:00, sondern um 07:00 durchgeführt. Dies könnte zu der zuvor beschriebenen Performance-Problematik führen, da bei dieser Uhrzeit damit zu rechnen ist, dass Benutzer parallel zur Datenermittlung mit den Daten arbeiten.

Ein weiteres Problem können Feiertage sein, da diese ebenfalls variieren und so zu Überschneidungen führen können. Bei dieser Optimierung werden neben der zuvor genannten Zählung der Referenzen Durchschnittswerte wie das Alter einer Revision bzw. Datenreferenz genommen und verglichen, ob sie älter als das durchschnittliche Alter aller anderen sind. Wie unter dem Startauslöser drei bereits beschrieben ist ein weiterer Punkt bei dieser Optimierung, wenn von einer Revision eine Branch-Aktion durchgeführt wurde und auch schon die entsprechende Merge-Aktion resultierte und des weiteren von den Revisionen, bis einschließlich der welche durch die
Merge-Aktion verschmolzen wurde, kein Branch durchgeführt wurde, kann ebenso in Erwägung gezogen werden diese zu Löschen oder zu Archivieren. Ebenso wie die Merge-Aktion sind die Branch und Speicherungsaktionen wichtige Bestandteile dieser Optimierung. Grundsätzlich lässt sich sagen, dass die globale Ermittlung von Abhängigkeiten neben der Berechnung der Durchschnittswerte alle Faktoren der lokalen Ermittlung enthält. Die Ermittlung der Durchschnittswerte wurde aus Performancegründen in die globale Ermittlung integriert. Der Grund hierfür ist der, dass die Ermittlung der Durchschnittswerte alle Daten benötigt und hierdurch sehr viel Zeit und Performance in Anspruch nehmen kann. Es ist aber denkbar, dass eine “abgespeckte“ Version der Durchschnittswertermittlung ebenfalls in die lokalen Ermittlung von Abhängigkeiten integriert werden kann. Mit der abgespeckten Version ist gemeint, dass die Ermittlung der Durchschnittswerte nur innerhalb der betreffenden Datenreferenz stattfindet, in der ein Startauslöser registriert wurde, durchgeführt wird. Hierdurch ist der Zeit- und Performanceaufwand, abhängig von der Anzahl der Daten, minimierbar und erhöht auf der anderen Seite die Zuverlässigkeit, mit der alte oder nicht mehr gebrauchte Daten gefundenen werden. Jegliche Optimierung, wie unter den Startauslösern beschrieben, erzeugt lediglich eine Protokoll in Form von XML oder “Apache Derby“, dass anschließend vom Daten-Administrator genutzt werden kann, um gezielt durch eine entsprechende Bestätigung die Löschung bzw. Archivierung durchzuführen. In den folgenden Abbildungen werden nun die entsprechenden Flussdiagramme gezeigt.

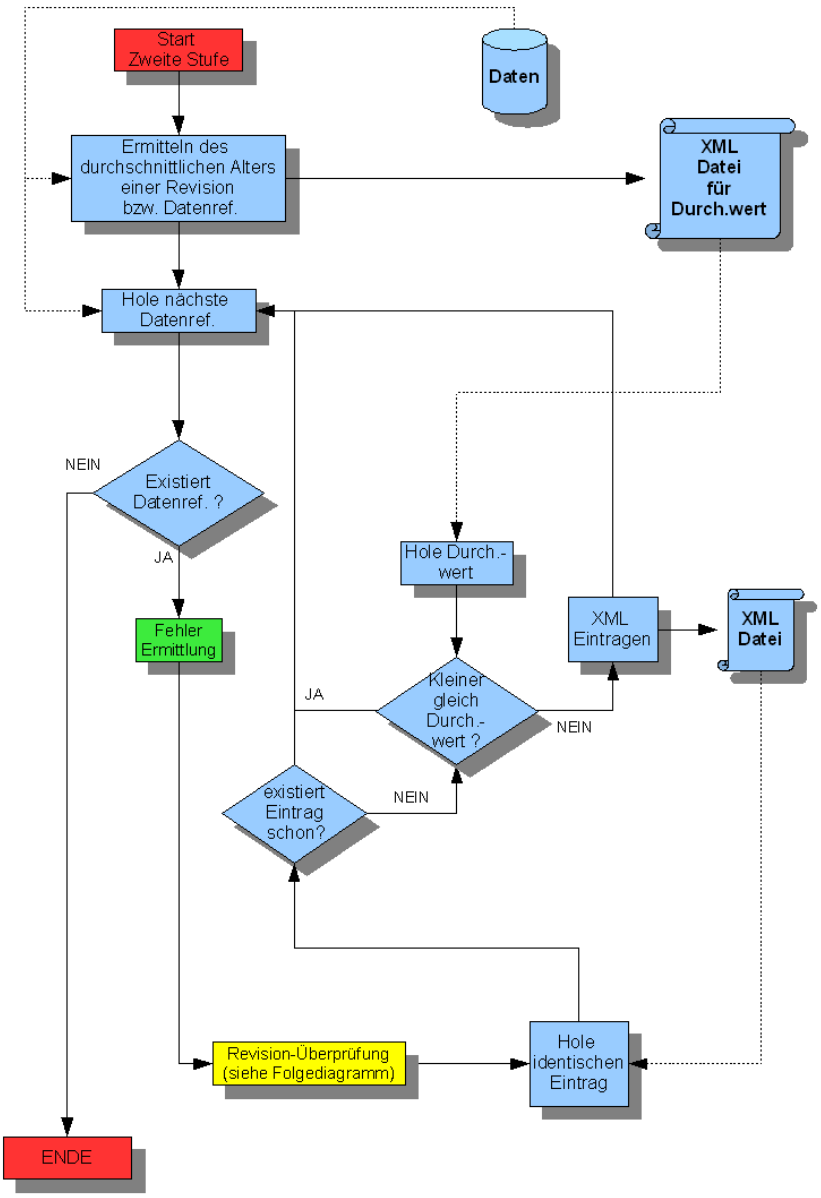

Abbildung 5.2-1: Flussdiagramm der globalen Ermittlung von Abhängigkeiten (1/2)

(Eigener Entwurf)

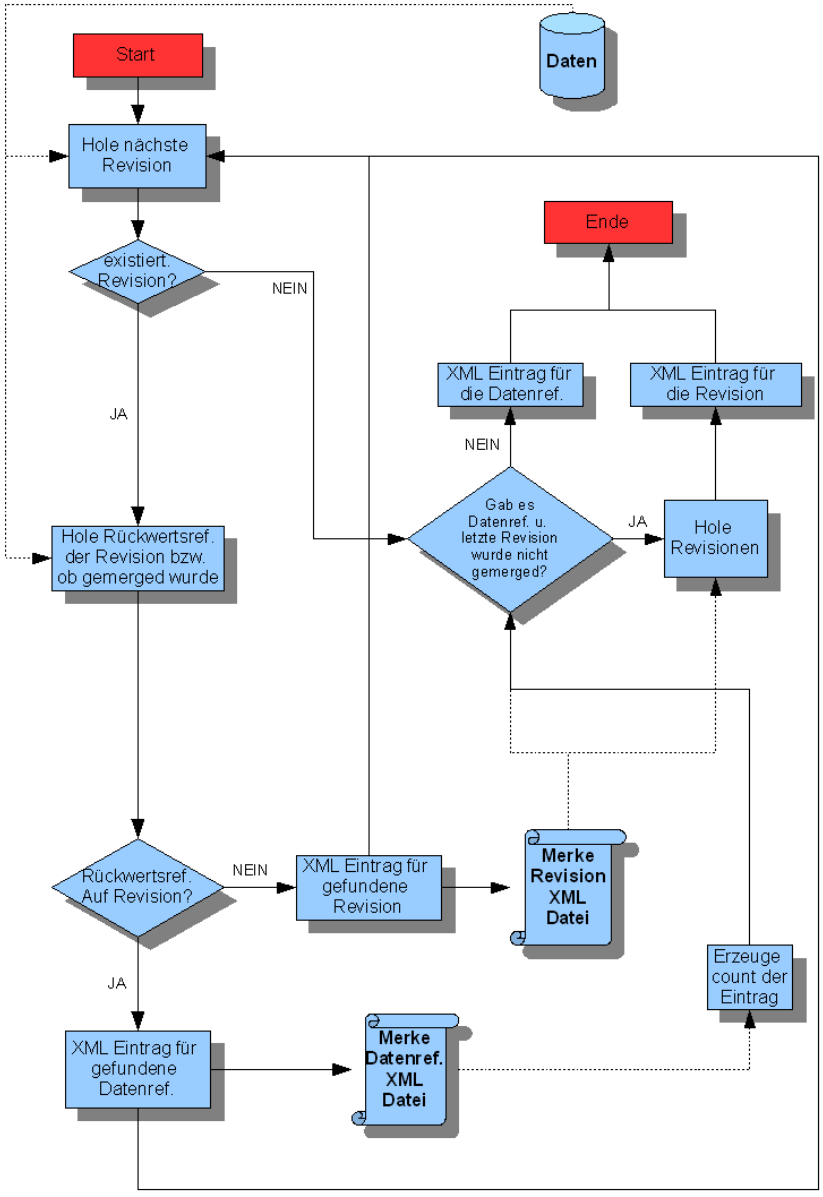

Abbildung 5.2-2: Flussdiagramm der globalen Ermittlung von Abhängigkeiten (2/2)

(Eigener Entwurf)

An dieser Stelle werden nun die zuvor gezeigten Flussdiagramme (5.2-1 und 5.2-2) beschrieben: Das grün gekennzeichnete Kästchen stellt die Fehlerermittlung dar und wird im späteren Verlauf genauer beschrieben. Wie auch zuvor in der lokalen Ermittlung von Abhängigkeiten, wird die Fehlerermittlung auch hier jedes mal mit gestartet. Die globale Ermittlung von Abhängigkeiten wird, wie zuvor schon beschrieben, automatisch oder manuell zu einem Zeitpunkt gestartet, an denen möglichst niemand mit den Daten arbeitet. Bevor aber eine Ermittlung durchgeführt wird, wird zuerst ein Durchschnittswert ermittelt. Dieser Durchschnittswert ist das durchschnittliche Alter aller Revisionen. Aber auch das durchschnittliche Alter einer Datenreferenz wird hier ermittelt. Sämtliche ermittelten Werte werden in einer XML-Datei oder einer “Apache Derby“-Datenbank gesichert und stehen bei der Auswertung der ermittelten Daten zu Verfügung. Ein entsprechendes XML-Beispiel wird hieran folgend gegeben. Bei der Durchschnittswertermittlung könnte man zur Optimierung des Algorithmus zusätzlich den durchschnittlichen Zeitraum in dem eine Änderung durchgeführt wird ermitteln und auswerten. Nachdem dieser Wert ermittelt wurde, wird nun die erste Datenreferenz (bei erstmaligen Ausführen), geholt (ansonsten wird immer die nächste Datenreferenz) geholt. Falls keine Datenreferenz mehr existieren sollten, wird die Ermittlung beendet. Falls aber eine Datenreferenz existieren sollte, so wird beim ersten Durchlauf derselben die erste Revision geholt. Ansonsten wird immer die nächste Revision geholt und überprüft. Dieser nun folgende Ablauf ist in der Abbildung 5.2-2 einsehbar. Wenn nun eine Revision gefunden wurde, wird dessen mögliche Rückwertsreferenz geholt. Ist nun eine Rückwertsreferenz auf diese Revision vorhanden, so wird die momentane Datenreferenz mittels einer XML-Datei oder einer “Apache Derby“-Datenbank gemerkt. Dies ist nötig, da diese Datenreferenz unter keinen Umständen gelöscht oder archiviert werden darf. Falls dies der Fall sein sollte, dürfen lediglich mögliche Revisionen gelöscht oder archiviert werden. Nach der Eintragung der Datenreferenz in die XML-Datei oder “Apache Derby“-Datenbank, wird jetzt die nächste Revision geholt. Falls keine Rückwertsreferenz auf die Revision zeigt, wird die Revision in der XML-Datei oder “Apache Derby“-Datenbank gemerkt. Auch hier wird nach der Eintragung die nächste Revision geholt. Wenn es nun keine Revision dieser Datenreferenz, die überprüft werden können, mehr existieren, so wird die Schleife unterbrochen und geschaut, ob eine Datenreferenz gefunden wurde.

Falls nun eine Datenreferenz gemerkt wurde, bedeutet dies, dass diese Referenz nicht gelöscht werden darf, da diese Datenreferenz eine Revision enthält, auf die eine Branch-Aktion durchgeführt wurde. Das heißt, falls die Datenreferenz gelöscht wird, würde die Rückwertsreferenz des Astes, welcher durch den Branch erzeugt wurde, ins Leere zeigen. Dies muss unter allen Umständen verhindert werden. Aus diesem Grund werden bei einer gemerkten Datenreferenz alle gemerkten Revisionen geholt, die keine Rückwertsreferenzen besitzen. Für jede dieser Revisionen wird nun ein XML- oder “Apache Derby“-Eintrag erzeugt. Anschließend wird für jeden erzeugten Eintrag geschaut, ob dieser schon in der Datei ist. Dieser Sachverhalt wird in der Abbildung 5.2-1 beschrieben. Bei der Datei handelt es sich um eine XML-Datei oder eine “Apache Derby“-Datenbank, die die Aufgabe hat, die zu löschenden oder archivierenden Revisionen oder Datenreferenz dauerhaft zu merken. Falls also der Eintrag noch nicht in der Datei sein sollte und der entsprechende Wert größer als der am Anfang ermittelte Durchschnittswert ist, so wird der erzeugte Eintrag in die XML-Datei oder “Apache Derby“-Datenbank geschrieben. Nach dieser Aktion wird dann die nächste Datenreferenz geholt. Falls nun aber keine Datenreferenz gemerkt wurde und die letzte Revision dieser Datenreferenz durch ein Merge verschmolzen wurde, wird ein entsprechender XML- oder “Apache Derby“-Eintrag für diese Datenreferenz erzeugt. Dies ist möglich, da von keiner Revision dieser Datenreferenz ein Branch durchgeführt wurde bzw. Abhängigkeit besteht. Des weiteren wurde der Ast durch die Merge-Aktion zurückgeführt. Nach der Erzeugung des XML- oder “Apache Derby“-Eintrags wird nun überprüft, ob dieser Datenreferenzeintrag schon in der Datei vorhanden ist. Falls ja, wird einfach die nächste Datenreferenz bezüglich einer Überprüfung geholt. Falls aber noch kein entsprechender Eintrag in der Datei vorhanden sein sollte, wird nun der Wert der Datenreferenz mit dem zuvor ermittelten Durchschnittswert verglichen. Wenn nun die Werte der Datenreferenz größer als der Durchschnitt sein sollten, wird der erzeugte XML- oder “Apache Derby“-Eintrag in die XML-Datei oder “Apache Derby“-Datenbank gespeichert. Auch nach dieser Aktion wird die nächste Datenreferenz geholt und überprüft. Falls aber keine Datenreferenz mehr vorhanden sein sollte, wird die Ermittlung beendet. Wichtig zu beachten ist noch, dass eine Datenreferenz nur gelöscht oder archiviert werden darf, wenn diese durch einen Merge zurück geführt wurde.

Des weiteren darf diese Datenreferenz nach dem Merge nicht weiter geführt worden sein. Ein weiterer Punkt ist der, dass von keiner Revision dieser Datenreferenz ein Branch durchgeführt worden sein darf. Bei der Löschung und Archivierung von Revisionen ist zu beachten, dass die letzte Revision einer Datenreferenz niemals gelöscht oder archiviert wird. Egal ob auf diese eine Rückwertsreferenz zeigt oder nicht. Diese Revision stellt den letzten Stand dar.
Das nun hieran folgende XML-Beispiel gilt sowohl für die beiden Merk-XML-Datei, als auch für die XML-Datei die alle alten und nicht mehr benötigten Daten speichert.
Wie zuvor schon erwähnt wurde, ist die Speicherung in diesem Konzept mittels XML realisiert, kann aber ohne weiteres auch als “Apache Derby“-Lösung realisiert werden.

```
<Notused>
        <Revision Key=“ddd1“>
                <Author>Otto</Author>
                <Date>09.05.2008</Date>
                <Extension>rrr</Extension>
                <Size>123123</Size>
                <Name>Tatjana</Name>
        </Revision>
        <Datareference Key=“ddd2“>
                <Author>Hans</Author>
                <Date>12.12.2008</Date>
                <Extension>fff</Extension>
                <Size>123123</Size>
                <Name>Flo</Name>
        </Datareference>
</Notused>
```

Das entsprechende XML-Beispiel bezüglich der Speicherung der Durchschnittswerte wird nun ebenfalls an dieser Stelle gezeigt.

```
<average>
        <averageage Type="Revision">123</averageage>
        <averageage Type="Datareference">135</averageage>
</average>
```

5.3 Mögliche Probleme der lokalen und globalen Ermittlung

An dieser Stelle wird nun auf die möglichen Probleme der lokalen und globalen Ermittlung von Abhängigkeiten eingegangen. Unter Kapitel 5.4 werden dann mögliche Lösungsansätze für die zuvor beschriebenen Probleme gezeigt.
Wie zuvor schon beschrieben können durch das Löschen oder Archivieren von Revisionen oder ganzen Datenreferenzen große Probleme auftreten. Ein Problem, dass zur Inkonsistenz der Datenstruktur führt, ist zum Beispiel eine oder mehrere bestehende Rückwertsreferenzen auf eine Revision, die gelöscht oder archiviert werden soll. Das exakte Problem ist nun, dass die einzelnen Rückwertsreferenzen bei einer erfolgten Entfernung ins Leere zeigen würden. Wenn nun durch den Benutzer versucht wird eine Merge-Aktion eines betreffenden Astes durchzuführen, wird ihm dies nicht gelingen, da das Ziel dieser Aktion durch die leere Rückwertsreferenz nicht bekannt ist. Dieses Zuordnungsproblem darf unter keinen Umständen auftreten, da hierdurch die komplette Entwicklung ebenso wie die zuvor entfernte Revision verworfen werden können, da diese nicht zuzuordnen sind. Durch dieses Problem kann ein enormer Kostenfaktor entstehen.
Ebenso kann aber auch durch die Löschung oder Archivierung einer ganzen Datenreferenz dieses Problem auftreten, wobei sogar noch viel mehr Branches betroffen sein können. Anhand der folgenden Abbildung wird ein Beispiel gezeigt, wie die gerade beschriebene Problematik aussehen könnte.

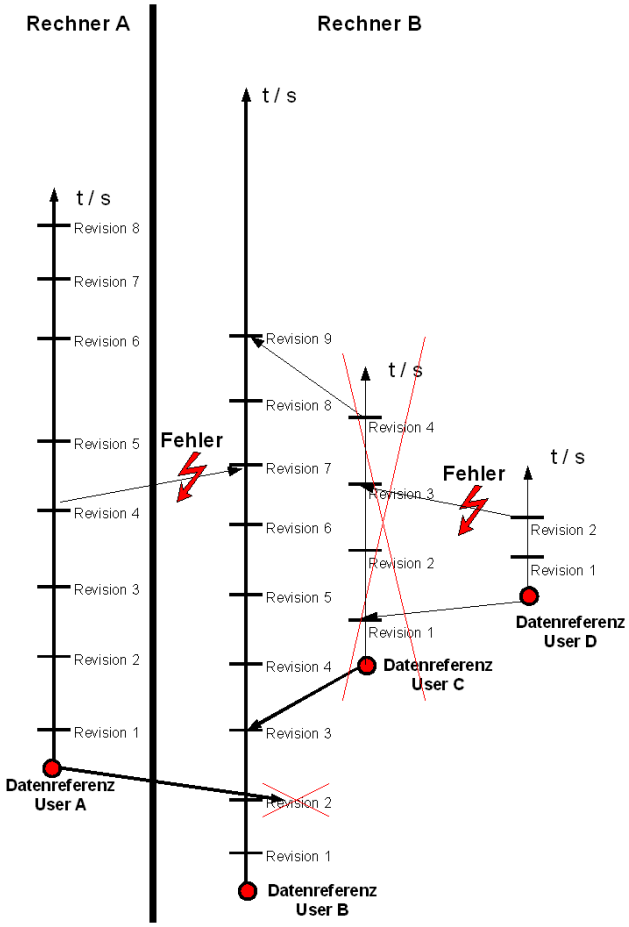

Abbildung 5.3-1: Beispiel für Zuordnungsproblematik

(Eigener Entwurf)

Ein weiteres Problem, welches speziell die globale Ermittlung betrifft, ist der Durchführungszeitpunkt der Datenermittlung. Grundsätzlich sollte es so gewählt werden, dass zum Zeitpunkt der Ermittlung niemand mit den Daten arbeitet. Dies sollte auch bei einer automatischen Intervall-Durchführung der globalen Ermittlung von Abhängigkeiten zutreffen.

Aber auch bei einem manuellen Starten der Datenermittlung durch den Daten-Administrator, muss eine Warnmeldung über die bevorstehende Aktion an alle betreffenden Benutzer zum Beispiel mittels E-Mail und Warnmeldung im Tool selbst geschehen. Die Warnmeldungen sollten optional auf verschiedenen Arten durchgeführt werden, da hierdurch eine Sicherheit der Kommunikationswege gewährleistet wird. Ein Beispiel hierfür ist der Ausfall des E-Mail Servers. In diesem Fall bekommt der Benutzer immer noch durch das RCE-System eine entsprechende Warnmeldung. Anderes herum gilt das selbe. Hat der Benutzer zum Beispiel seine SESIS-Anwendung nicht gestartet, erhält er eine entsprechende Warnmeldung mittels E-Mail. Anhand der folgenden Abbildung wird ein Beispiel gezeigt, wie die gerade beschriebene Problematik aussehen könnte.

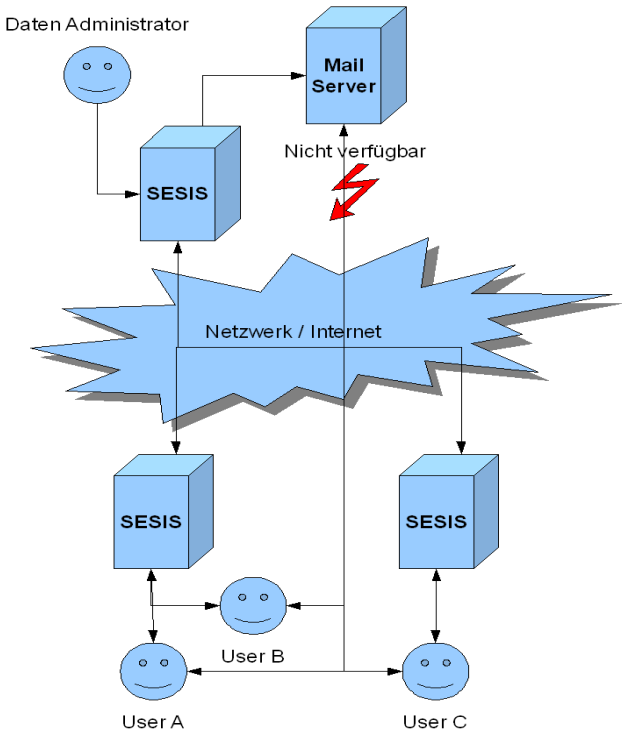

Abbildung 5.3-2: Beispiel für mehrere, parallele Mitteilungswege für eine Warnmeldung

(Eigener Entwurf)

Falls die Entscheidung für den Durchführungszeitpunkt zum Beispiel täglich um 01:00 ist, muss aber das Problem der Zeitzonen beachtet werden, da zum Beispiel auch Systeme in einer Zeitzone laufen könnten, die eine Zeitverschiebung von beispielsweise sechs Stunden haben. In diesem Fall wird die Ermittlung, in der entsprechenden Zeitzone nicht um 01:00, sondern um 07:00 durchgeführt. Dies könnte zu der zuvor beschriebenen Performance-Problematik führen, da bei dieser Uhrzeit damit zu rechnen ist, dass Benutzer parallel zur Datenermittlung mit den Daten arbeiten. Ein weiteres Problem können Feiertage sein, da diese ebenfalls variieren, was zu Überschneidungen führen könnte.

5.4 Mögliche Problemlösung der lokalen und globalen Ermittlung

Nachdem nun die Probleme beschrieben wurden, wird nun auf zwei mögliche Lösungsansätze eingegangen. Die besagten Lösungsansätze sind :

1. Verhindern der Aktion
2. Anpassen aufgrund der Aktion

Unter dem ersten Lösungsansatz ist zu verstehen, dass bei einer Lösch- oder Archivierungsmaßnahme einer Revision oder Datenreferenz, bei denen eine oder mehrere Rückwertsreferenzen existieren, diese zwar entgegen genommen, aber nicht umgesetzt wird. Diese Maßnahme muss natürlich mittels einer Fehlermeldung mitgeteilt werden. Der zweite Lösungsansatz ist etwas komplizierter und könnte als Alternative zum ersten Lösungsansatz mit der beschriebenen Fehlermeldung angeboten werden. Unter dem zweiten Lösungsansatz ist zu verstehen, dass bei einer Lösch- oder Archivierungsmaßnahme einer Revision oder Datenreferenz, bei denen eine oder mehrere Rückwertsreferenzen existieren, dass die Maßnahme durchgeführt wird. Da nun, wie schon beschrieben, das Zuordnungsproblem auftritt, muss die Rückwertsreferenz ein neues Ziel bekommen, damit eine Merge-Aktion wieder möglich und dadurch die Datenstruktur wieder konsistent wird. Ein potentielles Ziel bei der Löschung oder Archivierung einer Datenreferenz ist in diesem Fall die Revision, von der die gelöschte Datenreferenz den Branch gemacht hat. Ist die Maßnahme aber nicht auf eine Datenreferenz durchgeführt worden, sondern auf eine einzelne Revision, so wäre das potentielle Ziel die Vorgänger-Revision. Falls aber keine derartige Revision existiert, darf die Lösch- oder Archivierungsmaßnahme nicht durchgeführt werden. Diese Verhinderung der Maßnahme muss auch bei der Löschung oder Archivierung einer Datenreferenz gelten. Anhand der folgenden Abbildung wird ein Beispiel gezeigt, wie die gerade beschriebene Problematik aussehen könnte. Diese Abbildung baut auf der Abbildung 5.3-1 „Beispiel für Zuordnungsproblematik“ auf.

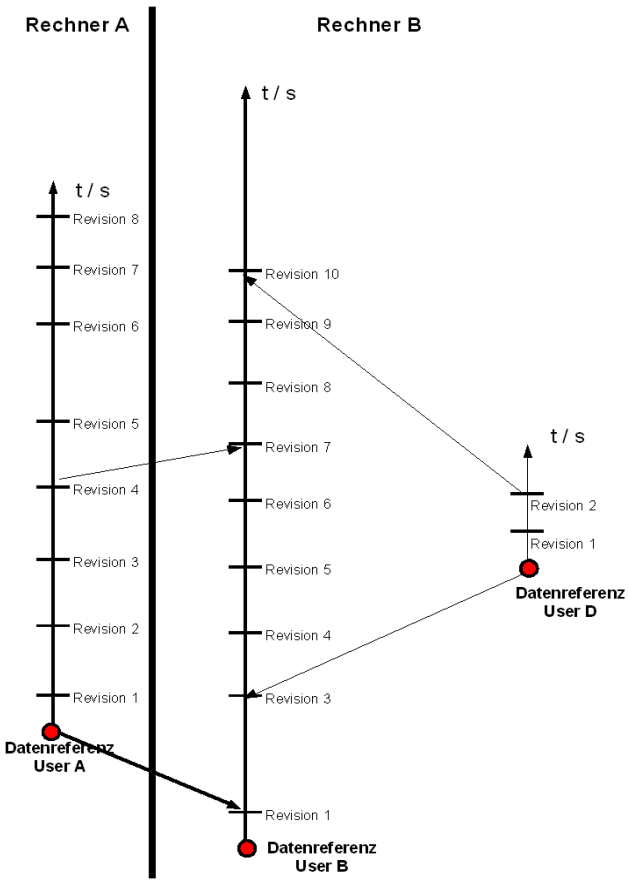

Abbildung 5.4-1: Ergebnis der Lösungsansätze

(Eigener Entwurf)

5.5 Manuelles Starten der globalen Ermittlung von Abhängigkeiten

Neben der automatisierten Ermittlung anhand der lokalen und globalen Ermittlung von Abhängigkeiten muss es aber auch die Möglichkeit der manuellen Ermittlung durch den Daten-Administrator geben. Hierdurch hat der Daten-Administrator die Möglichkeit vor einer Lösch- oder Archivierungsmaßnahme die XML-Datei oder “Apache Derby“-Datenbank auf den aktuellen Stand zu bringen.

Es sollte sogar eine Ermittlung vor der Lösch- oder Archivierungsmaßnahme durchgeführt werden, da so auf jeden Fall sichergestellt werden kann, dass die Daten wirklich nicht mehr benötigt werden. Mit dieser Art der Ermittlung ist aber auch, wie zuvor beschrieben, ein Performance-Problem verbunden. Dessen sollte sich der Daten-Administrator bewusst sein, da das manuelle Starten der Ermittlung die Arbeit mit den betroffenen Daten unmöglich machen kann. Die manuelle Ermittlung nutzt bzw. startet die zuvor beschriebene globale Ermittlung von Abhängigkeiten.
Aus diesem Grund sollte einerseits ein Zeitpunkt gewählt werden, bei dem niemand mit den Daten arbeitet. Des weiteren muss eine Benachrichtigung über die bevorstehende Ermittlung an alle betroffene Benutzer gehen. Diese Benachrichtigung sollte auf verschiedenen Wegen, wie zum Beispiel per
E-Mail und per Warnmeldung im RCE-System, geschehen. Diese redundanten Benachrichtigungen gewährleistet die Sicherheit des Kommunikationsweges.

5.6 Überblick der Faktoren die für die Ermittlung wichtig sind

Die hieran folgende Aufzählung zeigt kurz zusammengefasst die möglichen Faktoren bei der Ermittlung von alten und nicht mehr benötigten Daten.

- Keine automatische Löschung oder Archivierung bei alten oder nicht mehr benötigten Daten. Die Maßnahme wird erst durch eine Bestätigung mittels eines Dialoges mit dem Daten-Administrator durchgeführt
- Gefundene Daten werden in XML-Datei oder “Apache Derby“-Datenbank gespeichert
- Mögliche Faktorenkombination :
  - Alter der Datenreferenz oder Revision (Feststellung anhand der Metadaten) älter als das durchschnittliche Alter aller Datenreferenzen oder Revisionen
  - Löschen oder Archivierung einer Revision oder Datenreferenz darf nur durchgeführt werden, wenn keine Rückwertsreferenzen auf diese zeigen
  - Wenn keine Rückwertsreferenzen auf eine Revision zeigen, muss eine Folgerevision existieren. Diese stellt den letzten Stand dar

- Wenn keine Rückwertsreferenzen auf eine Datenreferenz zeigen, muss die letzte Revision dieser eine Merge-Aktion durchgeführt haben
- Wenn ein Merge erfolgreich durchgeführt wurde, es die letzte Revision war die diesen Merge durchgeführt hat und kein Branch von einer Revision dieser Datenreferenz gemacht wurde, könnte man die Datenreferenz bzw. alle alten Revisionen löschen. Folgende Möglichkeiten bzw. Alternativen stehen zur Verfügung :
  - Zuerst mergen um dann neu branchen und alte Datenreferenzen löschen, um weiter zu arbeiten
  - oder Datenreferenz weiterführen, um weiter zu arbeiten

6 Konzept zur Fehleraufdeckung im verteilten SESIS-Datenmanagement

Dieses Kapitel stellt einen weiteren Teilbereich dieser Diplomarbeit dar und befasst sich mit den Prüfungsmöglichkeiten, den möglichen Fehlerquellen und den Lokalisierungsmöglichkeiten von Fehlern. Bevor die möglichen Fehlerquellen des Datenmanagement, die zum Beispiel durch einen Branch oder Merge entstehen könnten, aufgezeigt werden, wird auf die Prüfungsmöglichkeiten eingegangen. Anschließend werden Möglichkeiten aufgezeigt, die das Aufdecken von Fehlern ermöglichen.

6.1 Prüfungsmöglichkeiten zur Fehlerermittlung im SESIS-Datenmanagement

Im Bereich DV-Systemprüfung wird das Augenmark insbesondere auf den Überblick und das Zusammenwirken der DV-Anwendungen, Organisation, IKS usw. gerichtet.
Aufgrund der zentralen Rolle der DV-Anwendungen im Unternehmen kann es sinnvoll sein, die DV-Systemprüfung durch weitere Prüfungsverfahren zu ergänzen, die sich mit den DV-Anwendungen detailliert befassen.
Je nach Zielrichtung der Prüfung unterscheidet man folgende zwei Verfahren [DVR08] :

- **Programmprüfung :**
  Verfahren zur Prüfung der Funktionalität von DV-Anwendungen

- **Einzelfallprüfung :**
  Verfahren zur Datenprüfung

Die folgende Abbildung zeigt das Zusammenwirken der Prüfungsverfahren, insbesondere die teilweise Überlappung von Programmprüfung und Einzelfallprüfung. Dies ist insbesondere bedingt durch die Analyse der Daten, die sich bei beiden Verfahren ergibt.

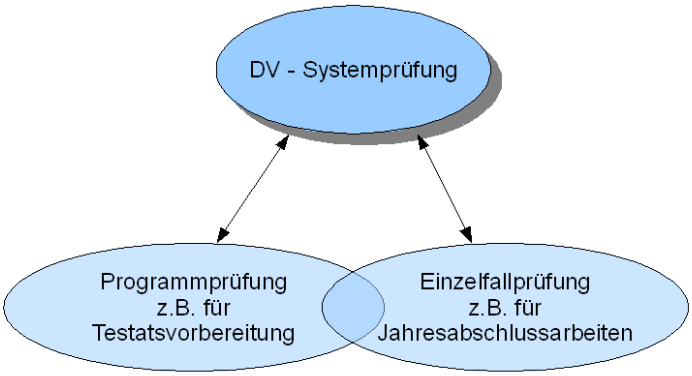

Abbildung 6.1-1: Zusammenwirken der Prüfungsverfahren [DVR08]

Bei der Programmprüfung steht die DV-Anwendung mit ihrer Funktionalität im Mittelpunkt. Man untersucht deren Verhalten in verschiedensten Situationen, beginnend bei der Eingabe über die Verarbeitung bis hin zur Ausgabe, wobei sowohl fiktive als auch reale Geschäftsvorfälle herangezogen werden. Bei der Prüfung kann sich der DV-Revisor aller Methoden und Verfahren bedienen, wie sie auch im Rahmen der Qualitätssicherung angewendet werden. Insbesondere sind hierbei die Testmethoden hervorzuheben, wobei allerdings die folgende Erweiterung berücksichtigt werden muss. Soll beim Test einer Anwendung zunächst nur der Nachweis erbracht werden, dass die zuvor festgelegten Anforderungen
(zum Beispiel aus Pflichtenheften oder Fachkonzepten) im vollem Umfang erfüllt sind, so muss der Test im Rahmen der Programmprüfung demgegenüber auch stets die Anforderungen der GoDV mit einbeziehen, selbst wenn diese nicht explizit Bestandteil der Anwendungsspezifikation sind. Wegen der nahen Verwandtschaft von Programmprüfung und Qualitätssicherung wird nun nicht genauer auf Letzteres eingegangen. Stattdessen steht im Folgenden vor allem die Einzelprüfung im Mittelpunkt der Untersuchung.

Bei der Einzelprüfung tritt die Funktionalität der DV-Anwendung in den Hintergrund [DVR08].
Betrachtet werden lediglich die von der Anwendung erzeugten und bereitgestellten Daten. Aus der Beschaffenheit der Daten lassen sich jedoch unter Umständen zusätzlich auch Rückschlüsse auf die Funktionalität der Anwendung ziehen, so zum Beispiel im Bereich der Prüfungen und Kontrollen. Trotzdem ist Folgendes zu beachten :

"Werden bei Einzelfallprüfung keine Abweichungen oder Fehler entdeckt, ergibt sich daraus noch nicht zwingend die Ordnungsmäßigkeit der Verarbeitung" [DVR08,S].

Einzelfallprüfungen werden zum Beispiel im Rahmen von Jahresabschlussprüfungen benötigt oder aber auch dann eingesetzt, wenn sich gezielte Fragestellung ergeben, wie zum Beispiel Ursachen von Differenzen in Abstimmkreisen. Einzelfallprüfungen stellen ein hilfreiches Instrument dar für die Untersuchung von

- **Abstimmungen**
- **Verprobungen**
- **Folgeprüfungen**
- **Kontroll- und Bestätigungsmeldungen**

In der folgenden Abbildung wurde ein möglicher Ablauf einer Einzelfallprüfung dargestellt.
Zuerst müssen Prüfungsbereich und -gegenstand festgelegt werden. Diese Festlegungen werden beeinflusst von

- **Prüfungsergebnissen**
- **Prüfungsstrategie**
- **Verdachtsmomenten**
- **Zeitpunkt der letzten Prüfung usw.**

Die Anforderungen an Einzelfallprüfungen ergeben sich zum Beispiel aus Vorgaben des Wirtschaftsprüfers an den DV-Prüfer. Die Erstellung des Anforderungskatalogs ist mit dem Vorgehen bei einer Anforderungsanalyse für DV-Projekte vergleichbar, nur dass hier das Ziel keine DV-Anwendung ist, sondern ein Katalog von Prüfungshandlungen und bestimmten erwarteten Ergebnissen. Inwieweit gestellte Anforderungen sinnvoll sind, ist von verschiedenen Einflussfaktoren abhängig. Es ist deshalb entscheidend, ob die Zielsetzung der Prüfung klar definiert ist, des weiteren ob die gewünschten Daten mit vertretbarem Aufwand aus DV-Produktion gewonnen werden können, ob die vorhandene Datenbasis ausreichend ist und mit welcher Methode bei großen Datenmengen Stichproben gezogen werden sollen. Die Definition der Anforderungen und der erwarteten Ergebnissen stellen die Voraussetzung für eine erfolgreiche Einzelfallprüfung dar. Fehlt diese Vorbereitung, so kann hierdurch eine große Enttäuschungen resultieren. Zudem kann sogar im schlimmsten Fall auch die Prüfungsergebnisse selbst nicht verwendbar sein.

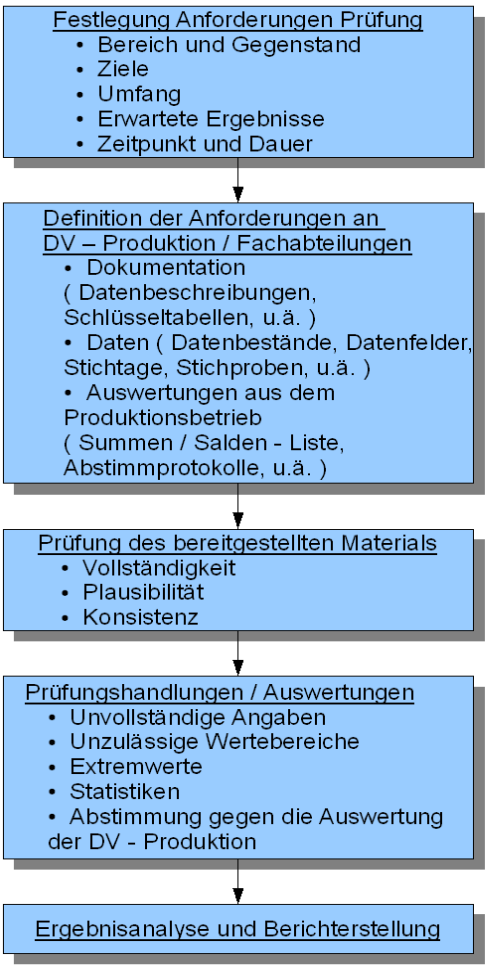

Abbildung 6.1-2: Ablauf Einzelfallprüfung [DVR08]

Aus den Anforderungen des Wirtschaftsprüfers oder des DV-Prüfers ergeben sich wiederum auch Anforderungen an die Mitarbeiter der DV / Org.-Abteilung oder der Fachabteilung. Der DV-Prüfer muss seine Anforderungen frühzeitig sowie verständlich, eindeutig und vollständig formulieren, damit die zu untersuchenden Daten, Dokumentationen und Auswertungen bereitgestellt werden können [Oden94].

Manchmal ist es sogar erforderlich, dass für den Erhalt bestimmter Daten besondere Aktivitäten in der DV-Produktionsumgebung eingeleitet werden müssen. Bei der Einzelfallprüfung können verschiedene Datenbestände einbezogen werden [DVR08] :

- **Bewegungsdaten**
  Zum Beispiel Buchungsjournale, Lagerzu- / -abgänge
- **Stammdaten**
  Zum Beispiel Konten, Artikelstämme
- **Verwaltungsdaten**
  Zum Beispiel Benutzerverzeichnisse, Berechtigungstabellen
- **Protokolldaten**
  Zum Beispiel Loggingdaten, Verarbeitungsprotokolle
- **Schnittstellendaten**
  Zum Beispiel Lohndaten für Fibu, Arbeitszeiten aus BDE für Lohnabrechnung

Auch wenn viele Prüfer sich hier überfordert fühlen dürften, ist es unabdingbar, genau zu definieren, welcher Datenausschnitt (zum Beispiel Felder, Views, Auswahlkriterien), welcher Datenumfang (zum Beispiel vollständiger Datenbestand, Stichproben) und welcher Zeitrahmen (zum Beispiel Zeitraum von - bis, bestimmter Stichtag) benötigt wird. Der Prüfer muss weiterhin berücksichtigen, dass auch verbundene Daten mit einbezogen werden. Weiterhin kann nur der Prüfer definieren, welche Informationen und Maßnahmen er zur Abstimmung benötigt bzw. vorsehen will [DVR08]. Bei Einzelfallprüfungen in einer DV-Umgebung der Revision sollte stets mit Kopieren der originalen Daten gearbeitet werden. Für die Auswertung steht dem Prüfer so genannte Prüfsoftware zur Verfügung.

Prüfsoftware muss mindestens folgende Leistungen bezüglich Daten erbringen können [Minz87] :

- **Sortieren**
- **Verknüpfen**
- **Mischen**
- **Selektieren und Extrahieren**
- **Verdichten**
- **Kopieren**
- **Vergleichen**
- **Berechnen**
- **Statistiken erstellen**
- **Berichte und Reports verfassen**

Neben der besagten Prüfsoftware, welche speziell für Hosts sind, wie zum Beispiel CULPRIT / EDP-AUDITOR von der Firma CA Computer Associates und SIROS von der Firma Ton Beller GmbH, existiert auch eine Reihe leistungsfähiger Prüfsoftware für PC's. Hierzu gehören zum Beispiel ACL (ACL Services Ltd.), REVEX (IDW-Verlag) oder KONAUDIT (Kontext GmbH). Der Vorteil von solchen PC-basierten Prüfsoftware liegt darin, dass der Prüfer seine Prüfungshandlungen unabhängig von der ursprünglichen Systemumgebung durchführen kann [DVR08]. Neben den zuvor beschriebenen Funktionalen Anforderungen an die Prüfsoftware ergeben sich weitere, wichtige Anforderungen aus Sicht der Praxis. Hierzu gehören

- **Benutzerfreundlichkeit**
  - Erstellungen von Auswertungen ohne großen Schulungsaufwand
  - Unterstützen von komplexen Prüfungshandlungen
- **Effizienz**
  - Auswertung großer Datenmengen
- **Weiterverarbeitungsmöglichkeit der Prüfergebnisse**
  - Hinzufügen von Erläuterungen mit Textverarbeitung
  - Integration in Prüfungsberichte usw.

6.2 Mögliche Fehlerquellen im SESIS-Datenmanagement

Unter diesem Kapitel wird nun auf die möglichen Fehlerquellen eingegangen. Dies umfasst als erstes die Fehlertypendefinition, als zweites die möglichen Fehlerquellen bzw. Fehlerbeschreibung und zuletzt die Vorgehensweise für die Fehlerermittlung bzw. mögliche Lösungsansätze.

6.2.1 Definition der Fehlertypen bei Daten

Unter diesem Unterkapitel wird nun zuerst auf die möglichen, allgemein gültigen Fehlertypen bei Daten eingegangen. Die Art des Fehlers lässt sich in vier unterschiedlichen Kategorien unterteilen. In der folgenden Aufzählung werden diese Kategorien beschrieben [DVR08].

- **Inkonsistenz Fehler :**
  Referenzen sind falsch oder nicht vorhanden
- **Lexikalische Fehler :**
  Fehlerhafte Literale
- **Grammatikalische Fehler :**
  Fehlende Zeichen
- **Statisch semantische Fehler :**
  Fehlende Deklaration oder Initialisierung von Variablen (fehlerhafte Typen, Missachtung der Zugriffsrechten)

Mit Inkonsistenz Fehler sind Fehler gemeint die zum Beispiel durch die Löschung der Branch-Quelle entstehen können. In diesem Fall würde die Rückwertsreferenz ins Leere zeigen und bei einer Merge-Aktion einen Fehler verursachen. Dieser Fehler entsteht dadurch, dass die fehlerhafte Rückwertsreferenz zum Beispiel auf eine nicht mehr vorhandene Revision bzw. dessen Ast zeigt. Hierdurch ist es nicht mehr möglich eine Merge-Aktion durchzuführen. Unter Lexikalische Fehler sind Fehler zu verstehen, die durch Fehlerhafte Literale entstehen können. Ein Beispiel hierfür wäre “2args“. Ein weiterer Fehlertyp sind grammatikalische Fehler.

Diese können entstehen, wenn Werte nicht vollständig sind bzw. wichtige, identifizierende Zeichen fehlen. Der letzte Fehlertyp sind statisch semantische Fehler. Diese Fehler können auftreten, wenn zum Beispiel Metadatenvariablen nicht initialisiert wurden, die aber für die Ermittlung von alten oder nicht mehr benötigten Daten elementar wichtig sind. Unter den folgenden Kapiteln wird nun genauer auf die möglichen Fehler bzw. auf eine mögliche Fehlererkennung eingegangen.

6.2.2 Beschreibung der möglichen Fehler im SESIS-Datenmanagement

Nachdem die Fehlertypen ausgiebig beschrieben wurden, wird nun auf die möglichen Fehlerquellen eingegangen.
Einer der wichtigsten Fehler, die vermieden werden müssen, betrifft die Rückwertsreferenzen die in der “Apache Derby“-Datenbank (Datenkatalog) gespeichert sind. Hierbei können folgende Fehlerquellen auftreten:
Die erste Fehlerquelle entsteht, wenn das Ziel nicht mehr existiert bzw. die Rückwertsreferenz ins Leere zeigt. Dieser Fehler entsteht dann, wenn das Ziel durch eine Löschung nicht mehr vorhanden ist und der Benutzer eine Merge-Aktion startet. Die selbe Problematik entsteht aber auch, wenn das Ziel, also die Revision oder die Datenreferenz, archiviert wurde. Des weiteren kann aber auch durch Verschieben der Revision oder Datenreferenz dieser Fehler auftreten. Rückwertsreferenzen bergen aber noch eine weitere Gefahr, und zwar wenn diese nach der Initialisierung auf eine falsche Revision oder Datenreferenz zeigen. Dies führt zu katastrophalen Folgen, wenn nun eine Merge-Aktion durchgeführt wird. Im schlimmsten Fall werden dadurch zum Beispiel zwei Revisionen miteinander verschmolzen, die nichts miteinander zu tun haben. Dieser Fehler ist zudem schwer zu identifizieren, da an sich alles richtig erscheint, also eine funktionierende Rückwertsreferenz vorhanden ist. Ein weiteres großes Problem stellt die Änderung der IP-Adresse bzw. das Umbenennen des Rechnernamens da. Dies ist ebenso wie das offline oder nicht mehr vorhanden sein eines Rechners ein sehr problematischer Punkt, da es hier bei einer Merge-Aktion zu einem Fehler kommt, da der Rechner bzw. die Datenreferenz nicht gefunden werden kann. Das Selbe Problem betrifft nicht nur das Umbenennen des Rechners, sondern auch das Umbenennen einer Datenreferenz bzw. Revision, da hierdurch eine Identifizierungsproblematik auftritt.

Eine mögliche Lösung für das Problem ist einerseits das Verhindern von IP- oder Namensänderungen und andererseits das Anpassen der IP oder der Namen bei den betreffenden Systemen. Mit Anpassung sind Änderungen der Datenreferenzpfade, die in der “Apache Derby“-Datenbank enthalten sind, gemeint. Auch wenn die Rückwertsreferenz komplett richtig ist, kann es zu Problemen kommen. Dieses Problem entsteht, wenn die Rückwertsreferenz auf eine Revision bzw. Datenreferenz zeigt, die einem Benutzer gehört, der nur Leserechte auf den Ast hat, von dem er selber einen Branch durchgeführt hat. Kurz gesagt, sind es die Benutzer, die zwar branchen dürfen, aber nicht das Ziel eines Mergens verfolgen, da sie auch nicht die benötigten Rechte dafür haben. Wenn nun von so einem Ast gebranched wird, muss eine Strategie her, die dieses Problem löst. Zwar besteht die Möglichkeit, direkt in den Uhrsprungsast zu mergen, aber dadurch durchbricht man die Hierarchiestruktur des Datenmanagements, da der zweite Benutzer übersprungen wird. Eine bessere Merge-Strategie ist die, dass man mit dem Read-only Benutzer merged.
Da dieser aber jetzt nicht die Möglichkeit hat in dem von ihm gebrancheden Ast zu mergen, muss dies mittels einer Pull-Aktion durch den Besitzer des Hauptastes durchgeführt werden. Voraussetzung hierfür ist, dass der Besitzer des gebrancheden Astes Read-Rechte auf den besagten Ast hat. Diese Strategie ist empfehlenswert, da in diesem Fall die Hierarchiestruktur des Datenmanagements nicht durchbrochen wird. In der folgenden Abbildung sind nun beide Ansätze zu erkennen. Im späteren Verlauf wird genauer auf die Lösungsansätze eingegangen.

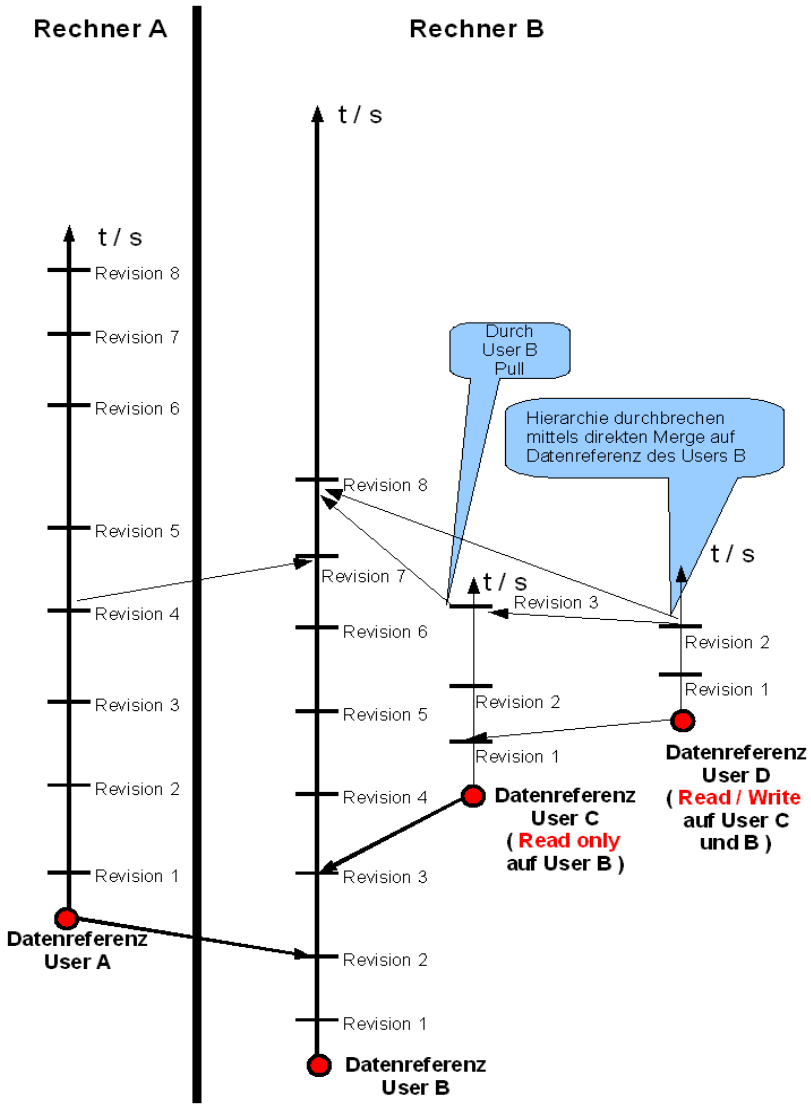

Abbildung 6.2.2-1: Problematik bezüglich Lese/Schreibrechten und Merge-Aktion

(Eigener Entwurf)

Neben der Rechte-Problematik, die zuvor beschrieben wurde, existiert auch ein Problem bei einem Merge eines Astes, von dessen Revision ein Branch durchgeführt wurde, aber noch nicht ein Merge. In diesem Fall muss, wie auch zuvor, die Hierarchie des Datenmanagements eingehalten werden. Dies bedeutet, dass nach dem Fertigstellen bzw. Speicherung der endgültigen Revision des Astes, welcher durch den Branch erzeugt wurde, in den ursprünglichen Ast gemerged wird. Hierbei wird eine neue Revision erzeugt. Dass Problem nun ist, dass dieser Ast schon eine Merge-Aktion erfahren hat und darüber hinaus nicht weiter geführt wird. Damit aber nun die Änderungen, die gemerged wurden, nicht verloren gehen, muss eine Pull-Aktion aus dem Hauptast erfolgen. Anhand der folgenden Abbildung ist diese Problematik beschrieben.

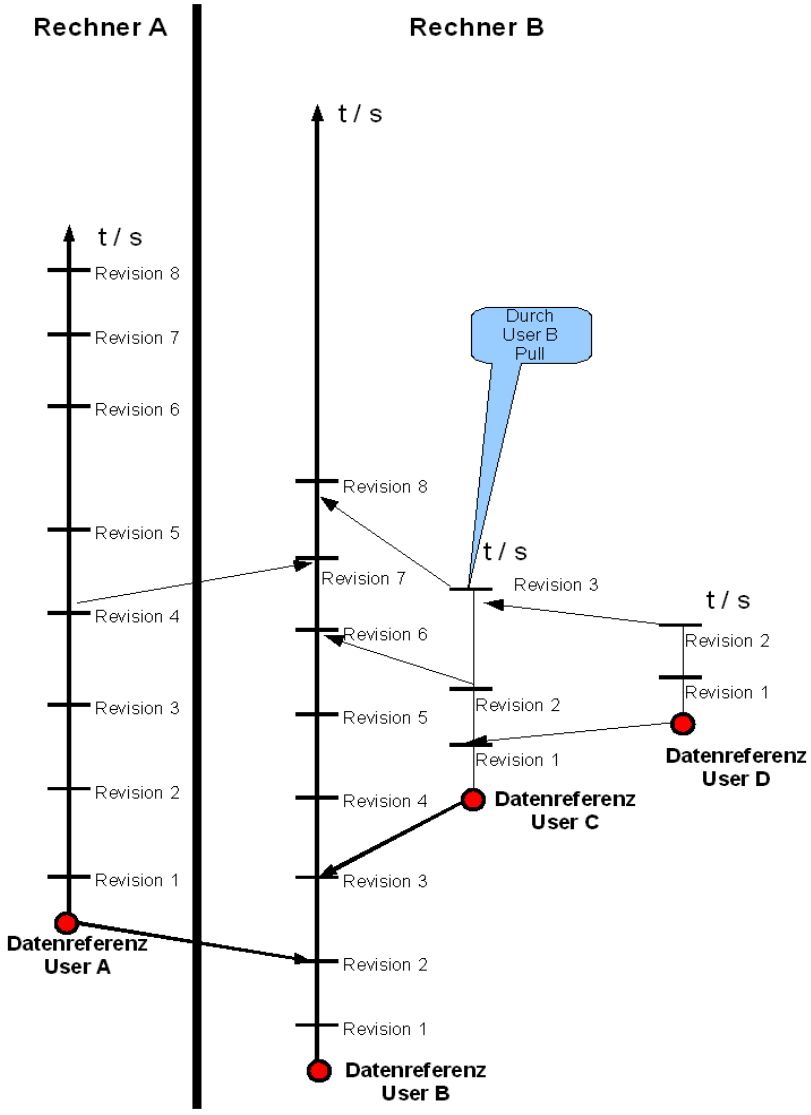

Abbildung 6.2.2-2: Problematik bezüglich dem verloren gehen einer Merge-Aktion

(Eigener Entwurf)

Eine weitere Fehlerquelle ist der Datenkatalog, der die lokal gespeicherten Datenreferenzen und Revisionen beinhaltet.
Hierbei ist zu beachten, dass nach einer Änderung an den Referenzen oder den Metadaten (hinzufügen, löschen und ändern) die tatsächlichen, lokalen Daten mit denen aus der “Apache Derby“-Datenbank (Datenkatalog) übereinstimmen müssen.
Aber auch die Werte der Metadaten müssen korrekt sein. So müssen zum Beispiel die folgenden Variablen deklariert und mit richtigen Werten initialisiert sein.

- Author
- Date
- Extension
- Size
- Name
- Key

Diese Variablen müssen mit Werten initialisiert werden, die dem jeweiligen Datentyp entsprechen. Des weiteren dürfen die Variablen nicht leer oder gleich null sein oder falsche Werte enthalten.
Eine sehr wichtige Rolle spielt die Key Variable, da diese genutzt wird, um Datenreferenzen oder Revisionen eindeutig identifizieren zu können. Falls ein Key mit einem Wert initialisiert wird, der schon existiert, entsteht hierdurch ein Fehler, da der Key wegen der eindeutigen Identifizierung nur einmal existieren darf.
Neben den Fehlern, welche durch falsche oder Nichtinitialisierung von Variablen entstehen können, kann aber auch ein Fehler aufgrund von falschen oder nicht erteilten Lese- oder Schreibrechte entstehen. Hierbei ist allgemein zu beachten, dass eine Schreibberechtigung eine höhere Priorität besitzt als Leseberechtigungen. Ein Beispiel hierfür wäre, wenn ein Benutzer zwar aufgrund von bestehender Leseberechtigung von einer Revision branchen kann, aber das mergen aufgrund von fehlenden Schreibberechtigung zu einem Fehler führt bzw. nicht durchführbar ist. Hierbei ist aber zu beachten, dass dieser Fall teilweise gewollt sein kann.

Neben den fehlerhaften Rückwertsreferenz und Metadaten können auch die tatsächlichen Daten fehlerhaft sein. Hierbei können die Daten falsch oder sogar beschädigt sein. In diesem Fall könnte der Einsatz einer Hashfunktion, wie zum Beispiel MD5, hilfreich sein. Durch diese Hashfunktion könnte man eine Prüfsumme der jeweiligen Daten erzeugen. Falls nun ein unerlaubter Zugriff, Sabotage oder Beschädigung an den jeweiligen Daten durchgeführt worden ist, so existiert keine Übereinstimmung der alten gespeicherten Prüfsumme mit der neu erzeugten. Durch dieses Vorgehen ist eine mögliche Fehlererkennung möglich.

6.2.3 Überblick der Fehlerszenarien im SESIS-Datenmanagement

Hieran folgend wird nun abschließend eine Auflistung der möglichen Fehlerszenarien gezeigt.

- Die Rückwertsreferenz zeigt ins Leere, da die Eltern (Datenreferenz bzw. Revision) gelöscht wurden (“Apache Derby“-Datenbank wurden nicht aktualisiert)
- Die Rückwertsreferenz zeigt ins Leere, da die Eltern (Datenreferenz bzw. Revision) archiviert wurde (“Apache Derby“-Datenbank wurden nicht aktualisiert)
- Die Rückwertsreferenz zeigt ins Leere, da die Eltern (Datenreferenz bzw. Revision) verschoben wurde (“Apache Derby“-Datenbank wurden nicht aktualisiert)
- Die Rückwertsreferenz zeigt ins Leere, da die Eltern (Datenreferenz bzw. Revision) umbenannt wurde (“Apache Derby“-Datenbank wurden nicht aktualisiert)
- Nach der Löschung, Verschiebung oder Umbenennung einer Revision oder Datenreferenz, auf die von einer anderen Revision bzw. Datenreferenz eine Rückwertsreferenz zeigt, entsteht ein Fehler bei mergen dieser oder einer Folgerevision in den Ast, auf dem der Branch durchgeführt wurde

- Name oder IP-Adresse eines Rechners dürfen sich nicht ändern, da sonst das Ziel der Rückwertsreferenz nicht mehr stimmt bzw. auffindbar ist. Falls IP-Adresse oder Name doch geändert wird, müssen Anpassungen an allen betreffenden
  "Apache Derby"-Datenbanken vorgenommen werden.
- Datenkatalog stimmt nach einer Aktion (Speichern, Löschen, Archivieren) an den Daten bzw. Dateien nicht mehr mit diesen überein. Es wird zum Beispiel auf Revisionen referenziert die nicht mehr existieren oder geändert wurden
- Falsche Berechtigung beim branchen, mergen bzw. Zugriff auf Revisionen, Datenreferenzen, Dateien
- Metadaten sind nicht vollständig bzw. fehlerhaft
  - Gründe :
    - Nicht initialisiert
    - Falscher Datentyp
    - Doppelt auftretende Unique-Werte (Hashkollision)
- Rückwertsreferenz zeigt auf falsche Revision bzw. Datenreferenz
- Ein Fehler kann entstehen wenn ein Rechner / Server bzw. deren Datenreferenz offline ist, da diese Situation dem IP-Adressen- bzw. Namensänderungsproblem ähnlich ist
- Ebenso wie das Löschen, Verschieben oder Archivieren ist auch das Umbenennen bzw. ein Identitätswechsel einer Datenreferenz eine Fehlerquelle, da die Metadaten nicht mehr konsistent bzw. richtig sind und hierdurch zum Beispiel ein mergen mit einer falschen Datenreferenz geschehen kann
- Entwicklungen bzw. Revisionen können verloren gehen, wenn ein Benutzer, der nur Leserechte hat, aus einem Ast branched und Revisionen erzeugt. Dieser Benutzer darf aber nicht zurück mergen, da er keine Schreibrechte hat. Nun branched aber ein zweiter Benutzer von diesem Benutzer. Dieser zweite Benutzer hat
  Lese- und Schreibrechte. Was passiert nun wenn der zweite Benutzer mergen möchte ?

Fragen : Wo geht der Merge hin ? Zu dem Benutzer, der nur Leserechte hat oder zu dem ursprünglichen Ast aus dem gebranched wurde ? Falls zu dem Benutzer, der nur Leserechte hat, der Merge durchgeführt wird, muss der Benutzer des Hauptastes ein Merge mittels Pull durchführen. Letzteres ist empfehlenswert, da keine Hierarchiesprünge gemacht werden.

- Entwicklungen bzw. Revisionen können verloren gehen, wenn in einen Ast gemerged wird, der nicht mehr weiter verfolgt wird

6.3 Lösungsansätze für Fehlerquellen im SESIS-Datenmanagement

Beim Lösungsansatz bzw. bei tatsächlichen Beseitigen der Fehler muss immer vor Augen gehalten werden, dass durch das Beheben eines Fehlers ein neuer Folgefehler auftreten kann. Diese Folgefehler kann entsteht, wenn ein Fehler zwar lokal behoben wird aber dieser auch global vorhanden ist. Ein Beispiel hierfür ist die Behebung einer falschen Kennung einer Revision. Wenn nun von dieser Revision ein Branch durchgeführt wurde, entsteht bei einer Merge-Aktion ein Fehler da die neue Kennung nicht bekannt ist.
Eine mögliche Lösung, die die Nicht-Initialisierung von wichtigen Metadaten verhindert, ist die Definition der betreffenden Metadaten Keys als Pflichtfelder, da diese Felder bei der Ermittlung von alten oder nicht mehr benötigten Daten elementar wichtig sind.
Aber auch bezüglich der Nachvollziehbarkeit sind diese Daten sehr wichtig. Die betreffenden Metadatenvariablen sind erneut :

- Author
- Date
- Extension
- Size
- Name
- Key

Dies bedeutet, dass diese Metadatenvariablen auf jeden Fall initialisiert werden müssen, um eine Datenreferenz bzw. Revision anlegen zu können. Die Ermittlung und Initialisierung der Werte geschieht automatisch. Im Falle, dass Werte nicht initialisiert werden, wird dem Benutzer eine entsprechende Fehlermeldung gezeigt, die ihn auffordert, die entsprechende Werte anzugeben. Bezüglich der Werte muss aber auch eine Validitätsprüfung durchgeführt werden, um sicher zu stellen, dass die ermittelten oder angegebenen Werte den richtigen Datentyp haben oder eindeutig sind. Ebenso wird beim offline gehen, bei IP-Adressen- oder Namensänderung der Rechner, und auch beim Umbenennen, Verschieben, Löschen oder Archivierung einer Revision oder Datenreferenz, auf die eine Rückwertsreferenz zeigt, dem Benutzer eine Warnmeldung gezeigt. Diese Warnmeldung macht den Benutzer ebenso darauf aufmerksam, dass ein mergen durch die Aktion nicht mehr möglich wird. Soll aber eine der Maßnahmen durchgeführt werden, so sollte es aber möglich sein, dass der Benutzer ein alternatives Ziel der betreffenden Rückwertsreferenzen angeben kann. Als Zielvorschlag zeigt das System die Revision oder Datenreferenz, von der, die durch die Maßnahme betroffenen Daten, ein Branch durchgeführt wurden. Man kann dies so betrachten, als wird die Referenz durch gereicht.
Des weiteren kann das Unterscheiden zwischen gelöscht, verschoben oder umbenannt aber Rechner erreichbar und Rechner offline bzw. andere IP oder Name, mittels Exception Handling ermöglicht werden. Hierdurch könnte man entsprechende Fehlermeldungen ausgeben. Diese genauere Fehlermeldung könnte dem Daten-Administrator bezüglich der Fehlerermittlung hilfreich sein.
Eine mögliche Lösung bezüglich falsch vergebener Rechte oder dem mergen auf einen Ast, der nicht mehr weiter verfolgt wird, ist eine Pull-Aktion. Diese wird im späteren Verlauf kurz genauer beschrieben.

6.4 Fehlerermittlung im SESIS-Datenmanagement

Die Fehlerermittlung wird, ebenso wie die Ermittlung von alten oder nicht mehr benötigten Daten, mittels der zuvor beschriebenen lokalen und globalen Ermittlung gestartet. Die jeweilige Stelle innerhalb der zwei beschriebenen Ermittlungskonzepte, an der die Fehlerermittlung gestartet wird, wurde zuvor in den Abbildungen 5.1.2-1 bis 3 und 5.2-1 als grünes Kästchen gekennzeichnet. Es existiert lediglich ein kleiner Unterschied zur lokalen Ermittlung. Der Unterschied zur lokalen Ermittlung von Abhängigkeiten im speziellen von alten oder nicht mehr benötigten Daten betrifft die Startauslöser. Zusätzlich zu den schon definierten Startauslösern kommt nun noch ein weiterer dazu. Dieser zusätzliche Startauslöser wird durch eine Löschung (Löschaktion) ausgelöst. Des weiteren werden die ermittelten Daten, ebenso wie bei der Ermittlung von alten oder nicht mehr benötigten Daten, in einer XML-Datei oder “Apache Derby“-Datenbank gespeichert. Eine Änderung (Änderungsaktion) wird hier wie auch in der lokalen Ermittlung nicht als ein eigener Startauslöser angesehen, da diese der Speicheraktion entspricht. Es darf auf keinen Fall eine automatisierte Korrektur der gefundenen Fehler durchgeführt werden. Deshalb hat der Daten-Administrator jederzeit die Möglichkeit, eine Bereinigung der gefundenen Fehler durchzuführen. Ebenso existiert auch hier ein Aktualitätsproblem der Datei. Dies besteht darin, dass die gefundenen Fehler zum Zeitpunkt der Bereinigung durch den Daten-Administrator nicht mehr existieren, da sie schon korrigiert wurden. Aus diesem Grund muss auch hier die Möglichkeit existieren, dass die Ermittlung der fehlerhaften Daten manuell durch den Daten-Administrator gestartet werden kann. Da aber durch das manuelle Starten der Ermittlung weitere Probleme wie zum Beispiel Performance-Verlust oder sogar Blockade der Benutzerarbeit entstehen können, müssen wichtige Vorbereitung getroffen werden. Eine dieser Vorbereitungen ist die rechtzeitige Benachrichtigung über die Maßnahme. Die besagte Benachrichtigung muss, wie zuvor beschrieben, auf zwei unabhängigen Wegen geschehen.

Ein Beispiel hierfür wäre einmal über das RCE-System selbst, aber parallel dazu über E-Mail. Der Daten-Administrator startet hierbei lediglich einmal die Benachrichtigung für eine Datenreferenz, bei der etwas behoben werden soll. Das System schaut nun, wie die Abhängigkeiten gestaltet sind und benachrichtigt nun jeden direkt oder indirekt betroffenen Benutzer.
Diese Benachrichtigung sollte, wie anfänglich beschrieben, auf zwei Wegen geschehen, da hierdurch eine Sicherheit bezüglich der Kommunikation gewährleistet werden kann. Hieran folgend wird nun das Flussdiagramm der Fehlerermittlung gezeigt, welches anschließend beschrieben wird.

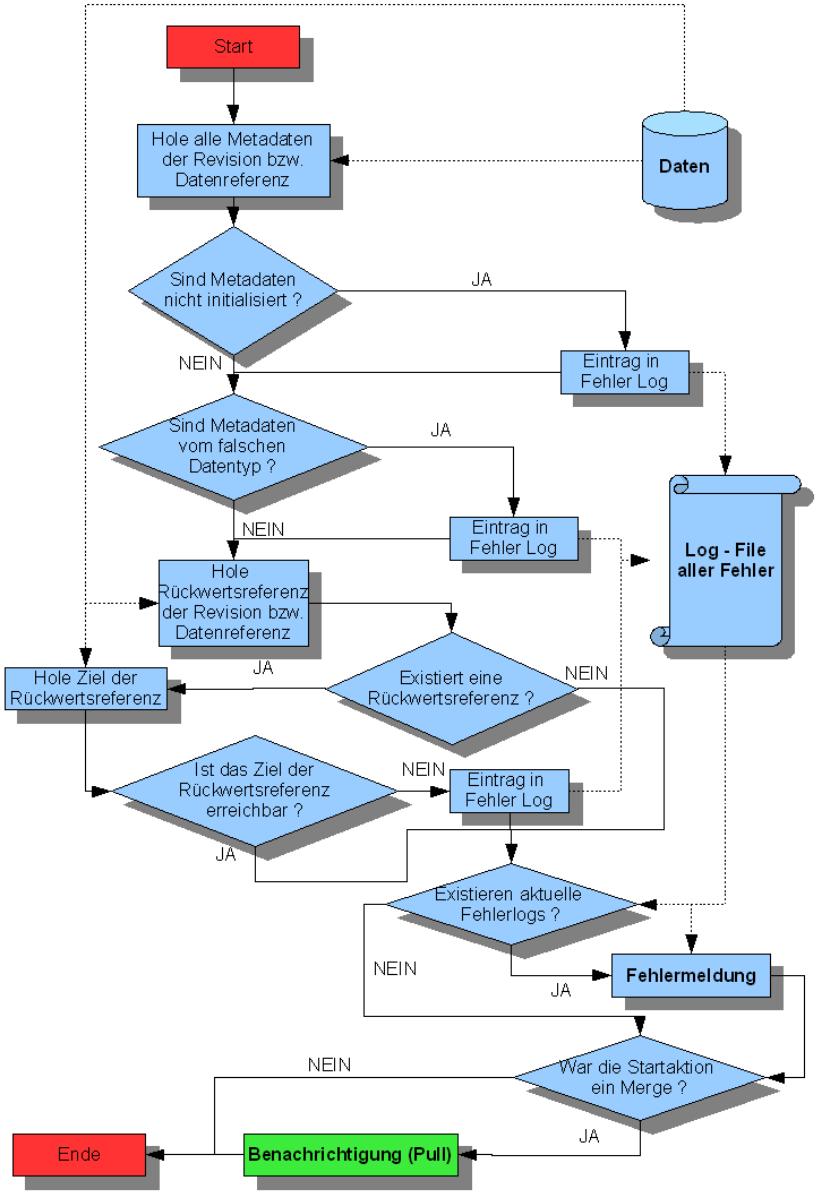

Abbildung 6.4-1: Flussdiagramm zur Fehlererkennung und Fehlermeldung

(Eigener Entwurf)

Wie zuvor schon angekündigt, wird nun an dieser Stelle das Flussdiagramm der Fehlerermittlung genauer beschrieben. Als erstes muss erwähnt werden, dass die Fehlerermittlung bei jedem Startauslöser der lokalen Ermittlung von Abhängigkeiten mit gestartet wird. Des weiteren wird die Fehlerermittlung ebenfalls auch bei der Ausführung der globalen Ermittlung von Abhängigkeiten mit gestartet. Die jeweils entsprechenden Stellen in den Flussdiagrammen zuvor wurden mit grünen Kästchen gekennzeichnet. Als erstes werden alle Metadaten der betreffenden Revision bzw. Datenreferenz aus der “Apache Derby“-Datenbank geholt. Als nächstes wird nun zuerst überprüft, ob die Metadaten mit Werten initialisiert wurden, also nicht leer sind. Diese Überprüfung ist sehr wichtig, da diese Werte bezüglich der Nachvollziehbarkeit und Ermittlung von alten und nicht mehr benötigten Daten einen wichtigen Faktor darstellen. Falls nun eine Metadatenvariable mit keinem Wert initialisiert wurde, so wird für diese ein entsprechender Fehlereintrag in der Log-File, welche alle Fehler merkt, gespeichert. Diese Log-File kann eine XML-Datei oder eine “Apache Derby“-Datenbank sein. In diesem Beispiel Flussdiagramm handelt es sich um eine XML-Datei. Ein entsprechendes Beispiel für diese XML-Datei wird hieran folgend gezeigt. Dieser Eintrag enthält die Revision bzw. Datenreferenz, den Zeitpunkt der Fehlerermittlung, wodurch die Fehlerermittlung gestartet wurde und was der Fehler ist. Bezüglich unseres Falls wäre das “Was“ die besagte Nicht-Initialisierung einer bestimmten Metadatenvariable. Nach der Eintragung wird nun mit der nächsten Überprüfung fortgefahren. Aber auch wenn keine Nicht-Initialisierung festgestellt werden sollte, wird mit der nächsten Überprüfung, ohne einen Fehlereintrag, weiter gemacht. Die nächste Überprüfung bezieht sich auf die Datentypen der initialisierten Metadatenvariablen. Hierbei wird geschaut, ob die Metadaten mit den richtigen Datentypen initialisiert wurden. Die Ermittlung der möglichen Fehler wird mittels Exception Handling durchgeführt. Falls nun ein Fehler gefunden wurde, wird wie auch zuvor schon beschrieben ein entsprechender Fehlereintrag in die Log-File durchgeführt. Auch hier wird nach der Eintragung mit der nächsten Überprüfung vorgefahren. Dies geschieht aber auch, wenn kein Fehler gefunden wurde.

Nachdem die ersten zwei Überprüfungen durchgeführt wurden, wird nun die mögliche Rückwertsreferenz der betreffenden Revision bzw. Datenreferenz aus der “Apache Derby“-Datenbank geholt. Die nächste Überprüfung schaut nun ob eine solche Rückwertsreferenz existiert. Falls dies der Fall sein sollte, wird nun versucht, das Ziel der Rückwertsreferenz zu holen bzw. ausfindig zu machen. Wenn nun dies nicht möglich ist, wird auch in diesem Fall ein entsprechender Eintrag in die Log-File durchgeführt. An dieser Stelle wäre es sogar möglich, um eine genauere Fehlermeldung zu erhalten, mittels Exception Handling festzustellen, wie der Fehler entstanden ist. Die genauere Fehlermeldung äußert sich in diesem Fall so, dass unterschieden werden kann, ob der Rechner im Netzwerk erreichbar ist, aber die gesuchte Revision bzw. Datenreferenz nicht existiert, oder ob sich der Rechner einfach nur nicht mehr im Netzwerk befindet bzw. offline ist. Durch diese Unterscheidung bzw. genauerem Log-File Eintrag, wird es dem Daten-Administrator ermöglicht, die Fehlerquelle genauer zu ermitteln. Also wie beschrieben wird beim Nichtauffinden der Quelle, auf die die Rückwertsreferenz zeigt, ein entsprechender Eintrag in die Log-File durchgeführt. Nach diesem Eintrag wird nun geschaut, ob aktuelle, also durch diese Fehlerermittlung entstehende, Fehler in der Log-File eingetragen wurden. Diese Abfrage wird auch dann durchgeführt, wenn das Ziel der Rückwertsreferenz erreichbar ist oder erst keine Rückwertsreferenz existiert. Wenn nun aktuelle Fehler vorhanden sein sollten, so werden diese nun mittels einer Fehlermeldung gezeigt. Diese Meldung fordert nun den Benutzer auf, diese Fehler zu beheben. Eine automatische Fehlerbehebung darf unter keinen Umständen durchgeführt werden, da die Richtigkeit der automatisch korrigierten Werte nicht gewährleistet werden und hierdurch zu unerwarteten Problemen führen kann. Nachdem nun die Fehlermitteilung gezeigt wurde, wird nun als letztes überprüft, ob die Fehlerermittlung durch den Startauslöser Merge gestartet wurde. Diese Überprüfung wird auch durchgeführt, wenn keine aktuellen Einträge in der Log-File existieren sollten. Wenn nun tatsächlich die Ermittlung durch eine Merge-Aktion gestartet wurde, wird nun eine mögliche Benachrichtigung bezüglich einer notwendigen Pull-Aktion durchgeführt. Auf diese Benachrichtigung bzw. deren Bedingungen wird im späteren Verlauf eingegangen. Nach dieser möglichen Benachrichtigung wird die Fehlerermittlung beendet und mit der jeweiligen Stelle der lokalen oder globalen Ermittlung fortgefahren.

Falls keine Merge-Aktion zum Start dieser Fehlerermittlung geführt haben sollte, so wird diese Ermittlung ebenfalls beendet. Wie zuvor schon beschrieben wurde, ist die Speicherung der Fehler in diesem Konzept mittels XML realisiert, kann aber ohne weiteres auch als “Apache Derby“-Lösung realisiert werden. An dieser Stelle wird nun das entsprechende Beispiel bezüglich der Log-File gezeigt.

```
<Failure>
        <Revision Key=“ddd1“ Failuredate=”15.05.2008”
        Action=”Merge”>
                <Author>Otto</Author>
                <Date>09.05.2008</Date>
                <Extension>rrr</Extension>
                <Size>123123</Size>
                <Name>Tatjana</Name>
                <Failurereason>lala</Failurereason>
        </Revision>
        <Datareference Key=“ddd2“ Failuredate=”20.12.2008”
        Action=”Branch”>
                <Author>Hans</Author>
                <Date>12.12.2008</Date>
                <Extension>fff</Extension>
                <Size>123123</Size>
                <Name>Flo</Name>
                <Failurereason>lolo</Failurereason>
        </Datareference>
</Failure>
```

Nachdem nun das Flussdiagramm der Fehlerermittlung und das XML-Beispiel beschrieben bzw. gezeigt wurden, wird nun im weiteren Verlauf auf einen weiteren Startauslöser eingegangen, die aber nur in Hinsicht auf die Fehlerermittlung eine Rolle spielt. Bevor dies geschieht, muss auch hier darauf aufmerksam gemacht werden, dass niemand zum Zeitpunkt der Fehlerermittlung mit den Daten arbeiten sollte. Um 100% sicher zu gehen, dass die Ermittlung bzw. der Zeitpunkt der Ermittlung (zum Beispiel 01:00) niemanden blockiert, könnte diesbezüglich eine Read-only-Kopie der Datenstruktur erstellt werden. Hierdurch wird es möglich, auf dieser gegebenenfalls weiter zu arbeiten, während die Ermittlung auf den Originaldaten durchgeführt wird. Ein weiteres Problem ist die Aktualität der ermittelten Daten, da nach einer Ermittlung die gefundenen Fehler korrigiert sein könnten. Aus diesem Grund wurde die Fehlerermittlung in die lokale und globale Ermittlung von Abhängigkeiten eingebunden. Die globale Ermittlung führt eine komplette Fehlerermittlung durch und trägt dadurch alle zum Zeitpunkt der Ermittlung bekannten Fehler in die Log-File ein. Die lokale Ermittlung ergänzt diese Log-File nun durch die Startauslöser gebundene Fehlerermittlung. Wenn nun ein Fehler auftritt, der noch nicht in der Log-File existiert, wird dieser eingetragen. Der Daten-Administrator behebt nun die Fehler und löscht den entsprechenden Eintrag aus der Log-File raus. Sowohl Log-File der gefundenen Fehler, als auch die Log-File der alten und nicht mehr benötigten Daten haben bezüglich der Statusermittlung eine wichtige Funktion (Hierzu später mehr). Als nächstes wird nun an dieser Stelle, wie zuvor schon angesprochen, ein weiterer Startauslöser vorgestellt. Dieser Startauslöser betrifft die Löschung einer Revision oder Datenreferenz. Genauer gesagt ist die Aufgabe dieses Startauslösers die Vermeidung potentieller Fehler, die durch eine solche Löschung entstehen könnten.
Ein Beispiel hierfür wäre zum Beispiel die Löschung einer Revision, auf die eine Rückwertsreferenz einer Datenreferenz zeigt. Dieses Problem wurde zuvor schon beschrieben. Ein weiteres Beispiel betrifft die Löschung der Historie. Wenn dieses Beispiel eintreten sollte, entsteht der Fehler dadurch, dass sämtliche Daten nicht mehr zu zuordnen sind und dadurch verworfen werden können. Falls dies der Fall sein sollte, wird eine Meldung gezeigt, die vor dieser Löschung warnen soll. Hieran folgend wird nun das entsprechende Flussdiagramm dieses Startauslösers gezeigt und danach beschrieben.

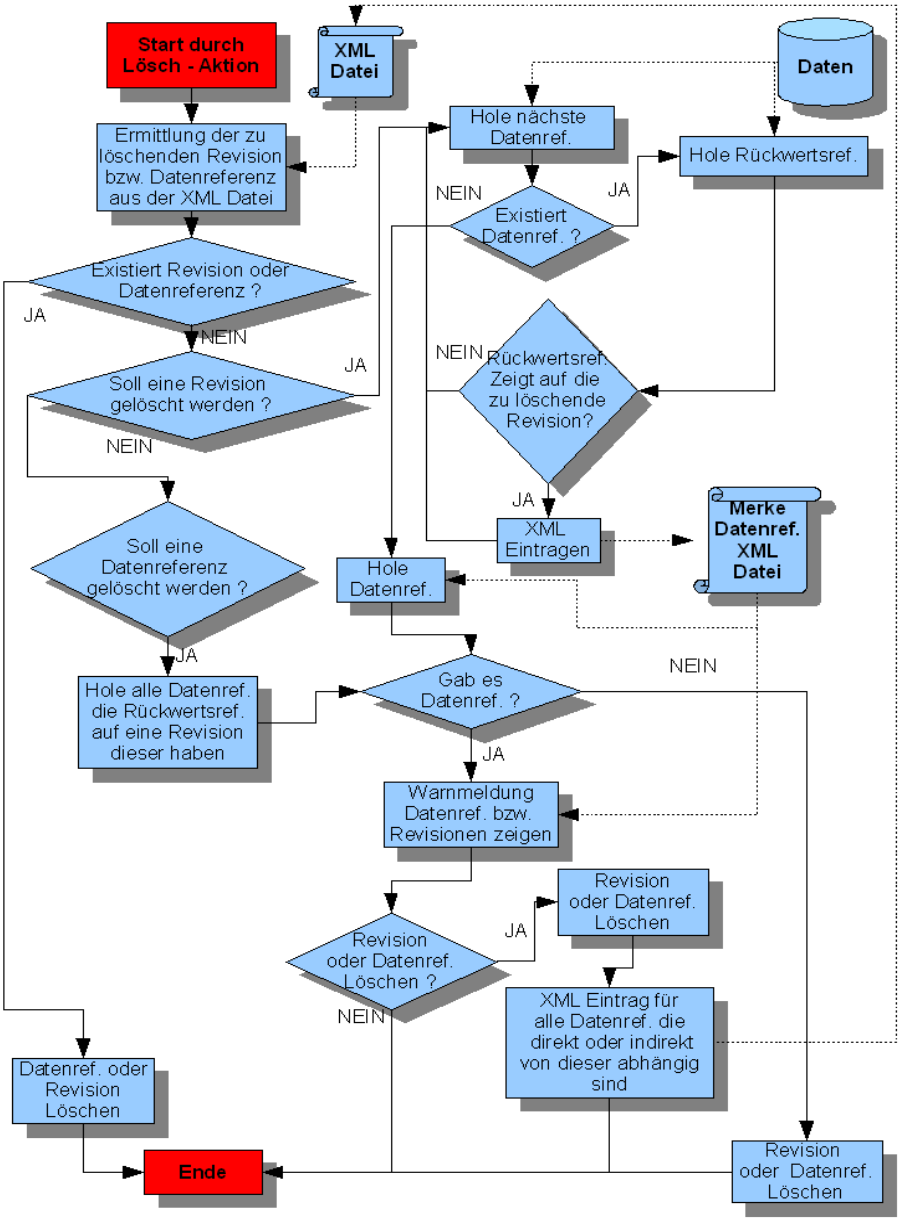

Abbildung 6.4-2: Flussdiagramm des Startauslöser Löschen

(Eigener Entwurf)

Es wird nun das gezeigte Flussdiagramm beschrieben: Die erste Aktion nach dem Starten durch den Startauslöser Löschen ist das Ermitteln der zu löschenden Revision oder Datenreferenz aus der XML-Datei. Diese XML-Datei enthält alle Revisionen und Datenreferenzen, welche durch die Ermittlung von alten und nicht mehr benötigten Daten gefunden wurden. Dies enthält sowohl die lokale, als auch die globale Ermittlung. Diese Revisionen und Datenreferenzen haben aufgrund der Bedingungen der Ermittlung keine Rückwertsreferenzen, welche auf diese zeigen. Aus diesen Grund kann nach dem Auffinden der zu löschenden Revision oder Datenreferenz in der XML-Datei direkt die tatsächliche Löschung durchgeführt werden. Nach der Löschung wird keine weitere Ermittlung durchgeführt und die Aktion beendet. Falls aber nun die zu löschende Revision oder Datenreferenz nicht in der XML-Datei gefunden wird, so muss als erstes ermittelt werden, ob es sich um eine Revision handelt. Falls es sich um eine Revision handelt, so wird als nächstes eine Schleife aufgerufen, welche durchlaufen wird, so lange es Datenreferenzen gibt. Beim ersten Durchlauf wird die erste Datenreferenz geholt, später immer die nächste. Wenn nun eine Datenreferenz gefunden wurde, wird das Ziel ihrer Rückwertsreferenz ermittelt. Wenn nun das Ziel der Rückwertsreferenz die zu löschende Revision ist, so wird ein entsprechender Eintrag in eine Merk-XML-Datei durchgeführt. Ein Beispiel für diese XML-Datei wird hier im Anschluss gezeigt. Nach dieser Eintragung wird anschließend die nächste Datenreferenz bzw. deren Rückwertsreferenz geholt. Aber auch wenn die Rückwertsreferenz nicht auf die zu löschende Revision zeigt, wird die nächste Datenreferenz ermittelt. Diese Schleife, wie zuvor schon beschrieben, läuft so lange, wie es Datenreferenzen gibt. Wenn nun alle Datenreferenzen überprüft wurden, wird anschließend geschaut, ob eine oder mehrere Datenreferenzen in der Merk-XML-Datei gemerkt wurden. Falls keine Datenreferenzen gemerkt wurden, kann sofort die Löschung der Revision durchgeführt werden. Die ist deshalb möglich, da keine Datenreferenz auf diese Revision referenziert. Nach der Löschung wird die Aktion beendet. Wenn nun aber Datenreferenzen in dieser Merk-XML-Datei gemerkt wurden, so wird eine entsprechende Warnmeldung ausgegeben. Diese Warnmeldung zeigt darüber hinaus die betreffenden Datenreferenzen und Revisionen, die von der zu löschenden Revision abhängig sind. Wenn der Benutzer nun die Löschung ablehnt, so wird die Aktion ohne weitere Maßnahmen beendet.

Stimmt der Benutzer trotz der Warnmeldung der Löschung zu, so wird die Revision gelöscht. Hierdurch entsteht nun folgendes Problem, welches zuvor schon im Kapitel 5.3 beschrieben wurde: Die Rückwertsreferenzen der betroffenen Datenreferenzen zeigen ins Leere, da ihre Quelle gelöscht wurde. Nicht nur die direkte, sondern auch die indirekte Abhängigkeiten sind hiervon betroffen, zum Beispiel eine Datenreferenz, die auf eine Revision referenziert, deren Datenreferenz wiederum auf die gelöschte Revision zeigt. In diesem Fall sind schon zwei, eine direkt und eine indirekt, betroffen. Dieses Spiel kann beliebig weitergeführt werden. Was ist nun in einem solchen Fall zu tun ? Es existieren zwei Möglichkeiten.

1. Die Datenreferenz, die auf die gelöschte Revision gezeigt hat, wird weiter beibehalten. Diese stellt aber nun eine “Basis Datenreferenz“ dar. Dies bedeutet, dass eine Revision dieser Datenreferenz keine Merge-Aktion mehr durchführen kann. Man kann aber ohne Weiteres von dieser branchen und in diese mergen.

2. Alle Datenreferenzen, die direkt von dieser Revision abhängig waren, werden wie unter Punkt 1 vorerst weiter geführt. Zusätzlich aber landen diese Datenreferenzen, also die direkten und indirekten, in der XML-Datei der alten und nicht mehr benötigten Daten, die infolgedessen zu einem späterem Zeitpunkt gelöscht, archiviert oder zu einem anderen Ast hinzugefügt werden können.

Die Entscheidung fiel auf den zweiten Punkt. Der Grund hierfür war der, dass durch die Speicherung der entsprechenden Werte, also die direkt und indirekt betroffenen Datenreferenzen, mehr Optionen zur Verfügung stehen. Ein weiterer Grund für diese Wahl war, dass bei einer späteren, beliebigen Maßnahme ein sehr performanter Zugriff auf die betreffenden Daten möglich wird, das heißt nach der Löschung der Revision werden alle direkt und indirekt abhängigen Datenreferenz in der XML-Datei gespeichert. Nach der Speicherung der Einträge wird die Aktion beendet. Nachdem nun das Vorgehen bei der Löschung einer Revision beschrieben wurde, wird im Folgenden der Fall angeschaut, bei dem bei der anfänglichen Abfrage, ob eine Revision gelöscht werden soll, ein nein heraus kommt.

Nachdem es keine Revision ist, die gelöscht werden soll, wird nun gefragt, ob es eine Datenreferenz ist. Da in diesem Fall nur dies sein kann, werden nun anschließend alle Datenreferenzen ermittelt, die auf eine Revision der zu löschenden Datenreferenz eine Rückwertsreferenz besitzen. Wenn nun alle Datenreferenzen überprüft wurden, wird anschließend geschaut, ob eine oder mehrere Datenreferenzen in der Merk-XML-Datei gemerkt wurden. Falls keine Datenreferenzen gemerkt wurden, kann sofort die Löschung der Datenreferenz durchgeführt werden. Diese ist deshalb möglich, da keine Datenreferenz auf die Revisionen der zu löschenden Datenreferenz referenzieren. Nach der Löschung wird die Aktion beendet. Wenn nun aber Datenreferenzen in dieser Merk-XML-Datei gemerkt wurden, so wird eine entsprechende Warnmeldung ausgegeben. Diese Warnmeldung zeigt zudem die betreffenden Datenreferenzen und Revisionen die von der zu löschenden Datenreferenz abhängig sind. Wenn der Benutzer nun die Löschung ablehnt, so wird die Aktion ohne weitere Maßnahmen beendet. Stimmt der Benutzer, trotz der Warnmeldung, der Löschung zu, so wird die Datenreferenz gelöscht. Hierdurch entsteht nun dasselbe Problem, welches zuvor schon beschrieben wurde. Aber auch dasselbe Vorgehen nach Punkt Zwei wird hier verwendet und besitzt dadurch auch dieselben Vorteile. Nach der Löschung der Datenreferenz werden also alle direkt und indirekt abhängigen Datenreferenz in der XML-Datei gespeichert. Nach der Speicherung der Einträge wird die Aktion beendet. Wie zuvor schon beschrieben wurde, ist die Speicherung bzw. das Merken in diesem Konzept mittels XML realisiert, kann aber ohne weiteres auch als
“Apache Derby“-Lösung realisiert werden. Das hieran folgende Beispiel zeigt die Merk-XML-Datei.

```
<Notice>
        <Revision Key="xyz1">
                <Author>Otto</Author>
                <Date>11.01.2008</Date>
                <Extension>abc</Extension>
                <Size>123123</Size>
                <Name>Tatjana</Name>
        </Revision>
        <Datareference Key="xyz2">
                <Author>Hans</Author>
                <Date>11.10.2008</Date>
                <Extension>cba</Extension>
                <Size>321321</Size>
                <Name>Flo</Name>
        </Datareference>
</Notice>
```

Nachdem nun das Flussdiagramm des Startauslösers Löschen und das XML-Beispiel beschrieben und gezeigt wurde, wird nun auf die Bedingungen der Benachrichtigung bei der Fehlerermittlung eingegangen.

6.5 Benachrichtigung über Pull-Aktion bei der Fehlerermittlung

Die Möglichkeit einer Pull-Aktion und die damit verbundene Benachrichtigung über neue Revisionen, soll es gewährleisten, dass Entwicklungen nicht verloren gehen. Die Gründe für das Verlorengehen von Entwicklungen, könnten zum Beispiel aufgrund fehlender Rechte bestehen. Dies ist der Fall, wenn zum Beispiel in einen Ast ein Merge durchgeführt wurde, dessen Besitzer nur Leserechte auf den Ast hat, von dem er selbst ein Branch durchgeführt hat. In diesem Fall würde diese Entwicklung in diesem Ast verbleiben und nicht mehr in den ursprünglichen Ast zurück geführt werden. Ein weiterer Grund, weshalb eine Entwicklung verloren gehen könnte, entsteht zum Beispiel, wenn in einem Ast ein Merge durchgeführt wurde, der zuvor schon selbst gemerged wurde und nicht weiter verfolgt wird.

In diesem Fall wird zwar eine Revision erzeugt, aber nicht mehr in den ursprünglichen Ast gemerged. Hierdurch geht auch diese Entwicklung verloren. Eine Möglichkeit, dieses Problem zu verhindern, ist der Einsatz einer Pull-Strategie. Mittels dieser Strategie wäre es möglich, eine Merge-Aktion nicht nur von einer Seite auszuführen, sondern auch von der Seite, in die die Merge-Aktion durchgeführt werden soll. Diese Aktion darf aber nicht willkürlich durchführbar sein, sondern muss an bedingte Bedingungen gebunden sein. Der Grund hierfür ist der, dass der Status der Revision, auf die ein Merge durchgeführt werden soll, nicht bekannt ist. Mit Status ist gemeint, dass nicht bekannt ist, ob eine Revision fertig, also die letzte ihres Astes ist. Falls dies der Fall sein sollte, so ist sie zum mergen bereit, ansonsten nicht. Wenn nun jedes mal, wenn eine betreffende Revision mit den entsprechenden Bedingung auftaucht, automatisch ein Merge durchgeführt werden sollte, ginge dies sehr zu Lasten der Performance.
Aus diesem Grund sollte die Pull-Aktion auf einem Request / Response-Prinzip aufbauen. Bevor aber mittels dem Request / Response-Prinzip ein Pull durchgeführt werden kann, muss eine automatische Benachrichtigung an den Benutzer erfolgen, dem der Ast gehört, in welchen die Merge-Aktion durchgeführt werden soll. Diese automatische Benachrichtigung über eine neue Revision erfolgt aufgrund von einer der folgenden zwei Bedingungen. Diese Bedingungen sind :

1. **Aufgrund fehlender Rechte :**
   Genauer gesagt bedeutet dies, dass eine neue Revision aufgrund einer Merge-Aktion entstanden ist und der Ast bzw. Benutzer dieses Astes nur Leserechte auf den Ast hat, von dem dieser gebranched wurde. Mit dieser Revisionserzeugung wird gleichzeitig eine Meldung über diese Erzeugung an den Benutzer des Astes geschickt, von dem ein Branch durchgeführt wurde. Anschließend hat dieser nun die Möglichkeit, ein entsprechendes Request für die Revision zu stellen. Dieser Request wird durch ein Response beantwortet. Dieser Response stellt die Merge-Aktion der betreffenden Revision dar. Durch dieses Vorgehen wird verhindert, dass diese Revision verloren geht.

2. **Aufgrund einer Merge-Aktion auf einen nicht weiter verfolgten Ast :**
   Im genauen bedeutet es, dass ein Ast, mittels einer Merge-Aktion zurück geführt wurde, von dem aber ein Branch durchgeführt wurde. Des weiteren wird dieser Ast, der gemerged wurde, nicht weiter verfolgt. Wenn nun der Ast, welcher durch den Branch entstanden ist, in diesen Ast gemerged wird, so besteht die Gefahr, dass diese neu erzeugte Revision verloren geht. Dies entsteht dadurch, da der Ast in den gemerged wurde, nicht mehr weiter verfolgt wird. Wenn nun ein Merge gemacht wird bzw. eine Revision durch einen Merge entsteht und die vorige Revision aber ebenfalls auch gemerged wurde, so kann man davon ausgehen, das der Ast nicht weiter verfolgt wird. Diese Problematik kann an der vorigen Abbildung (“Abbildung 6.2.2-2: Problematik bezüglich dem verloren gehen einer Merge-Aktion“) entnommen werden. Falls nun die Bedingung, dass in einen Ast gemerged wurde bei dem die vorige Revision ebenfalls gemerged wurde, eintreten sollte, so wird in diesem Fall eine entsprechende Mitteilung an den Benutzer des ursprünglichen Astes geschickt. Dieser kann nun diesen Sachverhalt prüfen und bei einer beabsichtigten Merge-Aktion einen entsprechenden Request bezüglich der betreffenden Revision, die durch die Merge-Aktion entstanden ist, schicken. Wenn nun dies durchgeführt wurde, wird mittels eines Response die beantragte Merge-Aktion durchgeführt.

Hierdurch ist es nun möglich, dass Verlorengehen von Entwicklungen bzw. Revisionen zu verhindern. Nachdem nun die Bedingungen gezeigt wurden, wird nun im Folgenden das entsprechende Flussdiagramm gezeigt. Dieses Flussdiagramm zeigt das Vorgehen bezüglich einer Benachrichtigung falls einer der vorigen Bedingungen eintreffen sollte. Des weiteren stellt dieses Flussdiagramm eine genauere Beschreibung der Benachrichtigungsaktion dar, welche im Flussdiagramm der Fehlerermittlung erwähnt wird. Die Benachrichtigungsaktion wird, da sie Bestandteil der Fehlerermittlung ist, ebenfalls mit der lokalen Ermittlung von Abhängigkeiten gestartet. Dies bedeutet, dass aufgrund einer Merge-Aktion der lokalen Ermittlung ebenfalls die Benachrichtigungsaktion aufgerufen wird und entsprechend der Bedingungen eine Benachrichtigung durchgeführt wird. In dem nun folgenden Flussdiagramm wird das Vorgehen beschrieben.

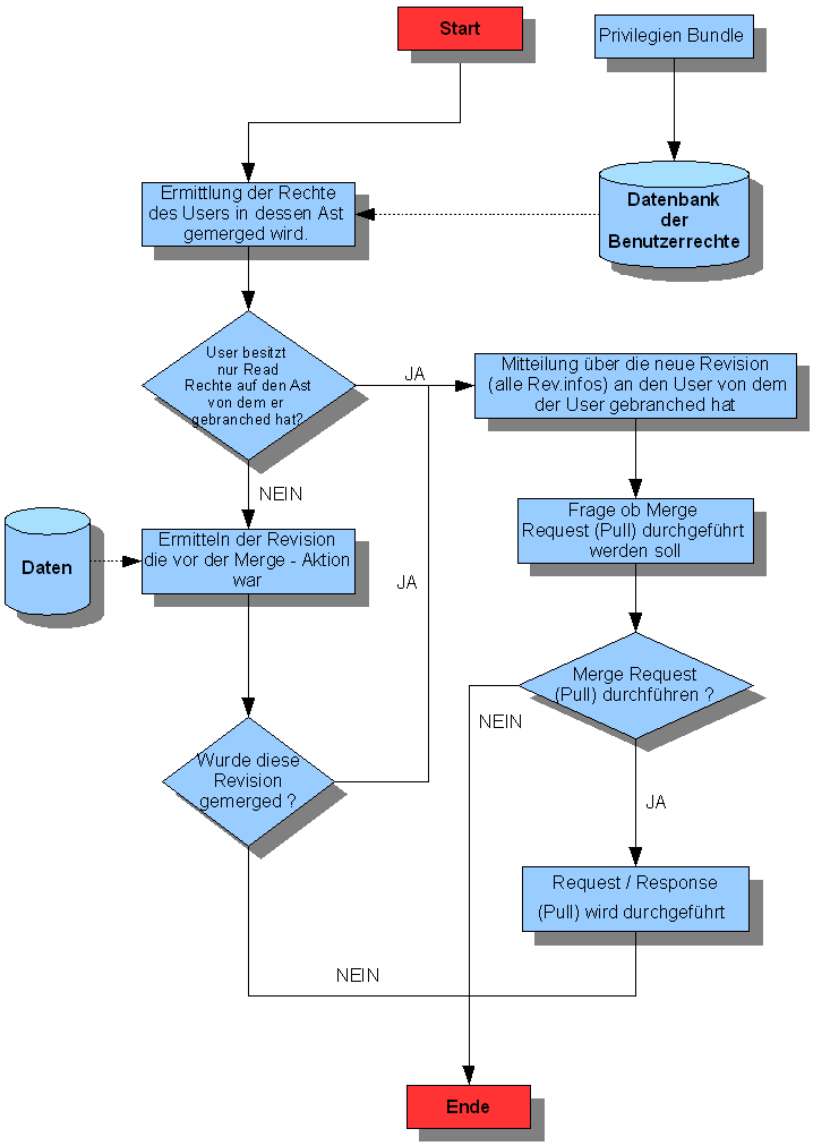

Abbildung 6.5-1: Beschreibung der Pull Aktion

(Eigener Entwurf)

Nachdem nun das Flussdiagramm gezeigt wurde, wird nun eine entsprechende Beschreibung geliefert. Nach dem Starten dieser Aktion werden als erstes die Rechte des Benutzers ermittelt, in dessen Ast ein Merge durchgeführt wird. Die Ermittlung kann durch das im RCE-Basissystem enthaltene Privilegien-Bundle geliefert werden. Des weiteren wird eine Benachrichtigung bzw. die Anfrage auf eine Pull-Aktion nur im Falle einer Merge-Aktion durchgeführt. Nachdem nun die Rechte des Benutzers ermittelt wurden, wird nun geprüft, ob dieser Benutzer nur
Read-Rechte auf den Ast hat, von dem er selbst ein Branch durchgeführt hatte. Wenn nun dies der Fall sein sollte, so erhält der Besitzer des Astes von dem der Read-only-Benutzer ein Branch durchgeführt hat, eine entsprechende Meldung über die neue Revision, die durch den Merge erzeugt wurde.
In dieser besagten Meldung wird nun dem Benutzer mitgeteilt, dass eine neue Revision in einem Ast erzeugt wurde, die aufgrund fehlender Rechte nicht zu seinem selbständig zurück geführt werden kann. Nun kann dieser Benutzer einen “Merge Request“ starten. Wenn er diesen nun gestartet hat, wird anschließend ein Response ausgeführt, der den gewünschten Merge darstellt. Diese gesamte Aktion wird als Pull-Aktion bezeichnet. Wenn der Benutzer diesen Request aber verneint, so wird keine weitere Maßnahme durchgeführt und die Aktion beendet.
Besitzt der Benutzer, in dessen Ast ein Merge durchgeführt wurde,
Lese- und Schreibrechte auf den Ast von dem er selber gebranched hat, so wird überprüft ob die vorige Revision gemerged wurde. Mit der vorigen Revision ist die Revision gemeint, die vor dem Merge die letzte war. Wenn dies nicht sein sollte, so wird ebenfalls die Aktion ohne weitere Maßnahmen beendet.

Wurde aber die vorige Revision schon gemerged, so kann davon ausgegangen werden, dass dieser Ast nicht weiter verfolgt wird bzw. eine weitere Merge-Aktion nicht beabsichtigt ist. Hierdurch könnte die Revision verloren gehen. Wenn nun dies der Fall sein sollte, so erhält nun der Besitzer des Astes von dem der Benutzer, der Lese- und Schreibrechte besitzt, gebranched hat, eine entsprechende Meldung über die neue Revision, die durch den Merge erzeugt wurde. In dieser besagten Meldung wird nun wiederum dem Benutzer mitgeteilt, dass eine neue Revision in einem Ast erzeugt wurde, von dem unmittelbar zuvor schon auf seinen Ast gemerged wurde. Nun kann dieser Benutzer einen “Merge Request“ starten.
Wenn er diesen nun gestartet hat, wird anschließend ein Response ausgeführt, der den gewünschten Merge darstellt. Diese gesamte Aktion wird ebenfalls als
Pull-Aktion bezeichnet. Wenn der Benutzer diesen Request aber verneint, so wird, ebenfalls wie zuvor, keine weitere Maßnahme durchgeführt und die Aktion beendet.

7 Konzept zur Lokalisierung der Daten und Optimierung von Speicherorten

In diesem Kapitel wird nun darauf eingegangen, wie es möglich ist, einen Überblick darüber zu erhalten, wo Daten im verteilten Datenmanagement liegen.
Des weiteren wird in diesem Kapitel ein Konzept gezeigt, welches es ermöglicht, eine Optimierung der Zugriffsgeschwindigkeit durch Verlagerung des Speicherortes zu erreichen.
Dieses Kapitel stellt den dritten Hauptteil dieser Diplomarbeit dar. Als erstes wird nun darauf eingegangen wie es möglich ist, einen Überblick über alle Daten zu erhalten.

7.1 Konzept zur Lokalisierung der Daten

In diesem Unterkapitel wird nun darauf eingegangen, wie es möglich ist, einen Überblick darüber zu erhalten, wo die Daten des verteilten Datenmanagements liegen. Dieser Überblick soll in allen Basissystemen des SESIS Projektes verfügbar sein. Der Überblick ist aber nur für die Basissysteme möglich, die direkt oder indirekt eine Rückwertsreferenz aufeinander haben.
Eine Möglichkeit, diese Abhängigkeiten zu ermitteln, könnte mittels eines systeminternen Broadcast geschehen. Wenn nun ein einzelnes Basissystem einen Überblick erhalten möchte, wo Daten liegen, würde dieser nun einen Broadcast an alle Basissysteme schicken. Diese würden dann jeweils mit ihrem Datenkatalog antworten. Dieses Vorgehen setzt aber voraus, dass ein Basissystem immer alle Basissysteme kennt. Diese Voraussetzung ist aber zur Zeit nicht in dem System implementiert und stellt dadurch ein Problem bezüglich der Ermittlung der anderen indirekten Basissystemen dar. Aus diesem Grund muss nun eine andere Strategie her. Eine mögliche Strategie wäre zum Beispiel, dass man einen systeminternes Broadcast verwendet, aber nicht in dem maßen wie zuvor beschrieben. Damit ist gemeint, dass der Broadcast nicht an alle, sondern nur an die Basissysteme geschickt wird, die ihm bekannt sind bzw. zu denen eine direkte Beziehung mittels Rückwertsreferenz existiert. Die Broadcast-Versendung wird durch das “Kommunikation Bundle“ ermöglicht, welches Bestandteil eines jeden Basissystems ist. Mit diesem Broadcast wird das eigene Basissystem als Antwortziel mitgegeben.

Wenn nun die ihm bekannten Basissysteme diesen Broadcast erhalten, senden diese zuerst ihren eigenen Datenkatalog und ihre eigene Beschreibung an das mitgelieferte Antwortziel der Broadcast. Nach dem Zusenden des jeweiligen Datenkatalogs, senden diese Basissysteme einen weiteren Broadcast mit dem selben Antwortziel, an das sie ihren Datenkatalog geschickt haben, an alle Basissysteme die ihnen bekannt sind. Diese indirekten Basissysteme senden nun, wie auch zuvor, ihre Datenkataloge und ihre eigene Beschreibung an das gleiche Ziel. Nach diesem Versenden, senden diese wieder einen Broadcast an allen ihnen bekannten und so weiter.
Das Resultat, nach der Beendigung der Broadcast-Versendung, ist, dass nun alle benötigten Datenkataloge an einer Stelle vorhanden sind. Diese eine Stelle ist nun das Basissystem, das den ersten Broadcast verschickt hat. Des weiteren besitzt dieses Basissystem nun die Möglichkeit, einen kompletten Überblick der direkt und indirekt abhängigen Daten zu erzeugen. Das Ergebnis dieser Erzeugung bzw. die hierarchische Struktur wird in einer XML-Datei oder “Apache Derby“-Datenbank gespeichert. In der nun folgenden Abbildung wird das entsprechende Versenden der Broadcast und Empfangen der Datenkataloge gezeigt.

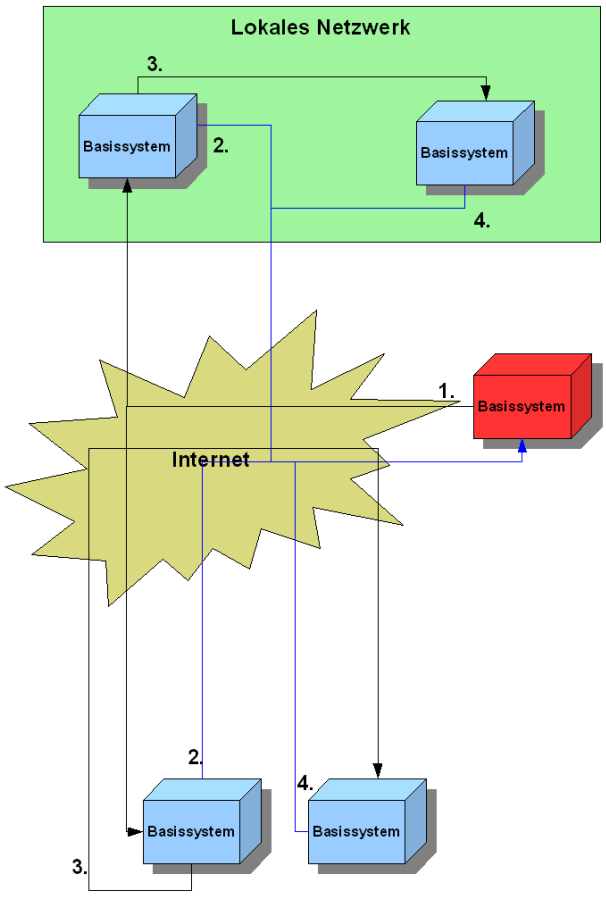

Abbildung 7.1-1: Beispiel für das Versenden der Broadcast und Empfangen der Datenkataloge

(Eigener Entwurf)

1. Versenden der Broadcast an alle bekannten Basissysteme
2. Zusenden der entsprechenden Datenkataloge
3. Versenden der Broadcast an alle bekannten Basissysteme
4. Zusenden der entsprechenden Datenkataloge .....

7.1.1 Broadcast-Problem

Nachdem nun das Versenden der Broadcast und der Datenkataloge bzw. deren Empfangen gezeigt wurde, wird nun auf ein großes Problem der Broadcast eingegangen. Dieses Problem, welches allgemein bei Broadcast existiert, sind Broadcaststürme bzw. mögliche Endlosschleifen. Dies bedeutet, dass zum Beispiel ein Broadcast zwischen drei Basissystemen hin und her geschickt wird. Ein Beispiel für diese Problematik wird nun in der folgenden Abbildung gezeigt.

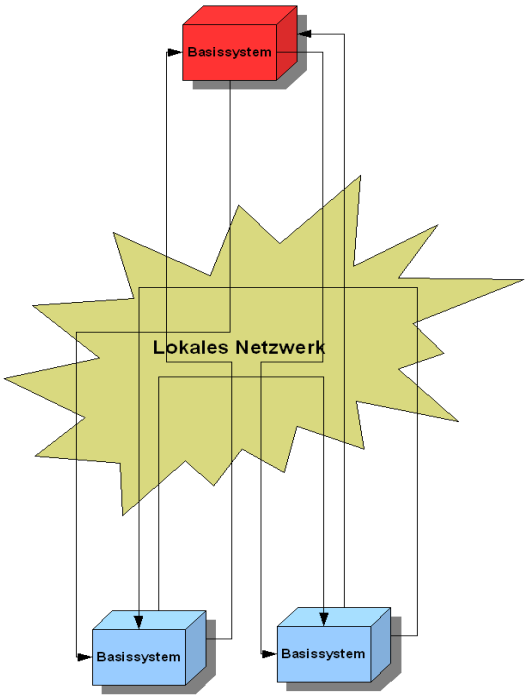

Abbildung 7.1.1-1: Beispiel für Broadcaststürme

(Eigener Entwurf)

In diesem Beispiel würden sich die einzelnen Basissystem ständig gegenseitig Broadcasts schicken. Dieses Problem könnte das gesamte Netzwerk lahm legen.

7.1.2 Lösung des Broadcast-Problems

Die Lösung des Problems, ist die Verwendung einer Broadcast-ID und einer Erinnerungsvariable. Der Nutzen der Broadcast-ID ist der, dass das jeweilige Basissystem sich merken kann, dass es diesen bestimmten Broadcast schon erhalten hat und dadurch nicht weiter an die ihm bekannten Basissysteme ein Broadcast schickt bzw. nicht nochmal seinen Datenkatalog an das Zielsystem (Basissystem) schicken muss. Das Vorgehen ist folgendes: Das Basissystem schaut in seinem Datenkatalog und ermittelt die Systeme, auf die es eine Referenz besitzt. Nachdem das Basissystem nun die Ziele bzw. Pfade hat, schickt es nun einen Broadcast an die Systeme, die noch nicht in der Erinnerungsvariable gemerkt wurden. Wenn jetzt aber ein System zwar nicht in der Variablen steht, aber den Broadcast schon bekommen hat, ist die ID der Broadcast gemerkt worden. Hierdurch ist es nun möglich, lediglich eine Meldung an den Sender der Broadcast zu schicken, dass er die Broadcast schon einmal bekommen hatte und deshalb nicht weiter verfolgt.
Der Nutzen der Erinnerungsvariable ist der, dass hierdurch gemerkt wird, welchen Weg ein Broadcast gegangen ist. Durch diese Variable wird es möglich, neben dem Merken, dass ein bestimmter Broadcast schon einmal eingegangen ist, überhaupt das senden einer Broadcast an ein Basissystem zu verhindern. Die Verhinderung des Sendens einer Broadcast wird dann durchgeführt, wenn eine Rückwertsreferenz des Datenkatalogs auf ein Ziel zeigt, welches ebenfalls in der Erinnerungsvariable der Broadcast steht. Diese Maßnahmen lösen das Problem der Broadcaststürme. Es besteht aber ein weiteres Problem. Dieses Problem ist die Broadcast-ID, da diese in dem gesamten verteilten Datenmanagement eindeutig sein muss und es dementsprechend keine zentrale Stelle existiert, die diese verwalten könnte.

Die Lösung des Problems sind die benutzerspezifischen Zertifikate, die in dem Basissystem mittels des Privilegien-Bundle implementiert sind. Jeder Benutzer, der sich an diesem System anmeldet, erhält ein solches Zertifikat, um ihn eindeutig zu identifizieren. Die folgende Abbildung zeigt nun die entsprechende Lösung des Broadcast-Problems.

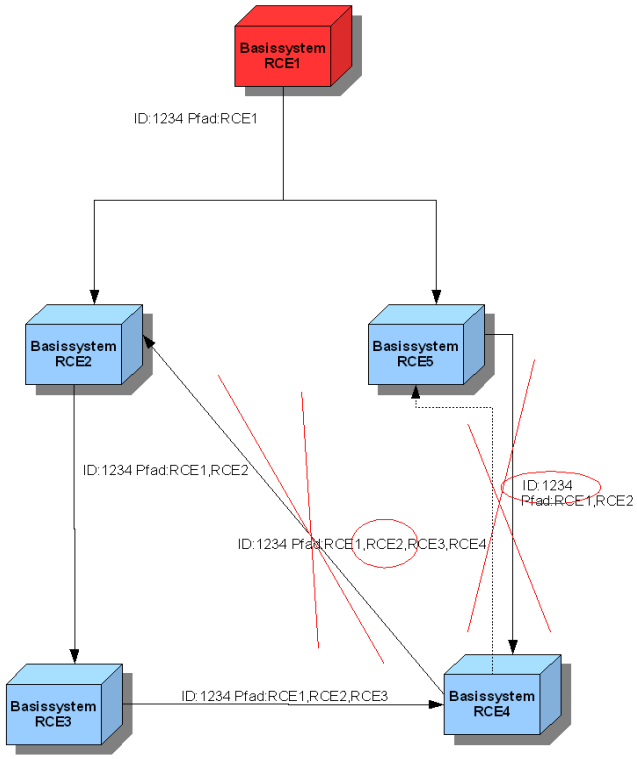

Abbildung 7.1.2-1: Lösung für das Broadcast Problem

(Eigener Entwurf)

7.1.3 Versenden und Empfangen der Broadcast

Nachdem nun das Versenden und Empfangen von Broadcasts und der Datenkataloge bzw. deren Probleme und Lösungen gezeigt wurden, wird auf die entsprechenden Flussdiagramme eingegangen. Als erstes wird das Flussdiagramm bezüglich des Sendens und Empfangens von Broadcasts bzw. dem Senden der Datenkataloge gezeigt.
Im letzten Teil dieses Unterkapitels wird dann auf die Erzeugung der XML-Datei
(Es kann auch eine “Apache Derby“-Datenbank verwendet werden) bzw. das Empfangen des Datenkataloges eingegangen. Wichtig zu erwähnen ist, dass das Versenden der ersten Broadcast nicht automatisch gestartet wird, sondern nur durch den Benutzer selbst. Dies wird durch eine GUI realisiert. Nun aber wird als erstes auf das Flussdiagramm eingegangen, welches das Versenden und Empfangen der Broadcasts bzw. das Senden des Datenkatalogs darstellt.

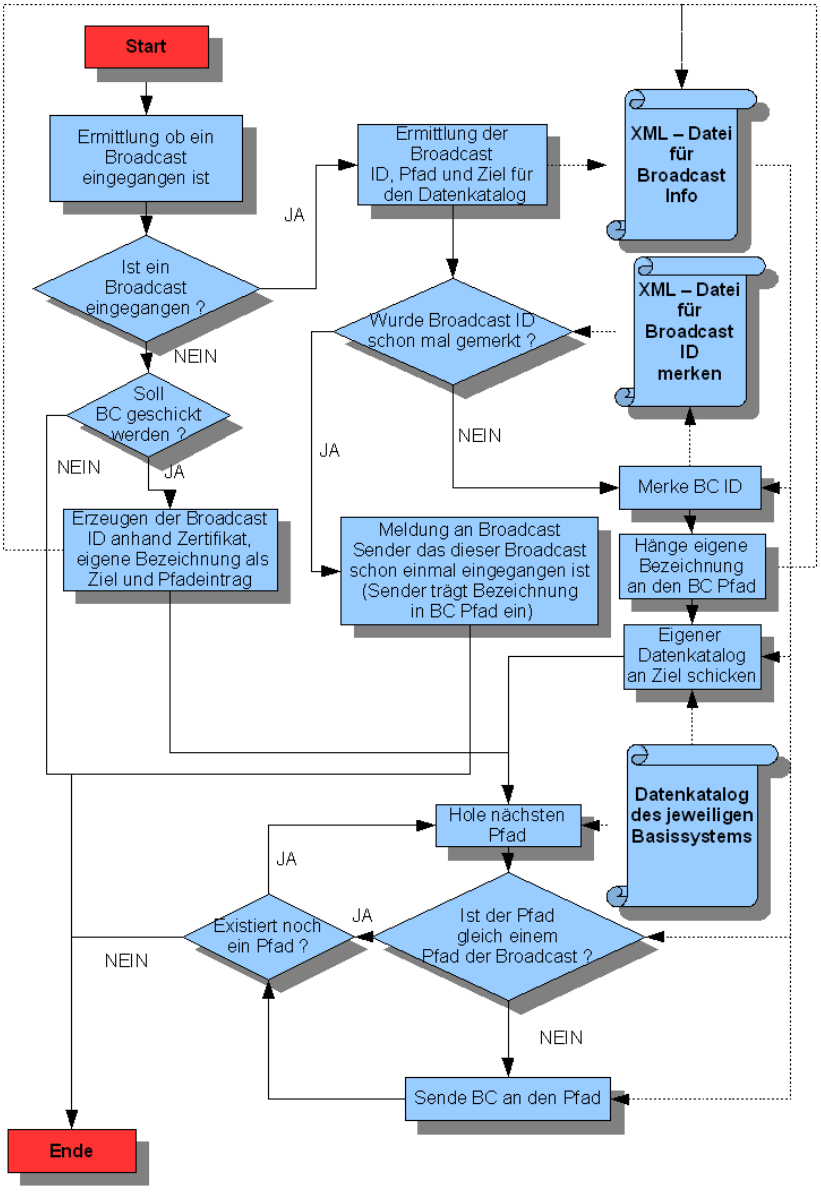

Abbildung 7.1.3-1: Versenden und Empfangen der Broadcast bzw. Senden des Datenkatalogs

(Eigener Entwurf)

Nachdem nun das Flussdiagramm gezeigt wurde, wird dieses kurz beschrieben. Die erste Maßnahme ist die Ermittlung, ob eine Broadcast eingegangen ist. Wenn dies der Fall sein sollte, werden die Broadcast-Informationen aus der erhaltenen Broadcast ermittelt und in eine XML-Datei gespeichert. Ein Beispiel dieser XML-Datei wird hieran folgend gezeigt. Nach der Ermittlung und Speicherung der Broadcast-ID, des Ziels des Datenkatalogs und des Pfads, den die Broadcast zurück gelegt hat, wird nun überprüft, ob dieser Broadcast schon einmal eingegangen ist. Diese Überprüfung wird mittels einer Merk-XML-Datei vorgenommen. Ein Beispiel dieser XML-Datei wird hieran folgend gezeigt. Wenn nun die ermittelte Broadcast-ID in der besagten XML-Datei stehen sollte, so wird lediglich eine Mitteilung an den Sender geschickt und die Aktion beendet.
Wenn nun aber die Broadcast-ID nicht in der XML-Datei sein sollte, so wird diese dort eingetragen. Zusätzlich zum Eintrag der Broadcast-ID wird die eigene Bezeichnung an den Pfad der Broadcast-Info angehängt. Nachdem nun diese Informationen gesichert wurden, wird der eigene Datenkatalog an das Ziel, welches in der Broadcast steht, geschickt. Hiermit ist eine Hauptaufgabe für dieses System erledigt worden. Eine weitere Aufgabe ist jetzt an alle einen Broadcast zu schicken, die dieser kennt. Um dies zu erreichen wird nun beim ersten mal der erste, ansonsten immer der nächste Pfad aus dem Datenkatalog ermittelt und überprüft, ob dieser Pfad in dem Broadcast-Pfad vorhanden ist. Falls dies der Fall sein sollte, wird einfach der nächste Pfad geholt. Wenn nun aber kein Pfad vorhanden sein sollte, wird die Aktion beendet. Wenn der Pfad bzw. Bezeichnung aber nicht im Broadcast Pfad sein sollte, wird an diesen Pfad eine Broadcast geschickt. Diese Schleife läuft so lange es Pfade gibt.
Wie zuvor schon beschrieben, werden durch dieses Vorgehen einerseits die Informationen ermittelt, welche später einen Überblick über die Daten und ihren Ort ermöglichen, und andererseits werden hierdurch Broadcaststürme verhindert. Wie zuvor schon beschrieben wurde, ist die Speicherung der Broadcastinformationen in diesem Konzept mittels XML realisiert, kann aber ohne weiteres auch als “Apache Derby“-Lösung realisiert werden. Hieran folgend wird nun zunächst ein XML-Beispiel bezüglich der Speicherung der Broadcast-Informationen gezeigt. Danach wird ein XML-Beispiel bezüglich der Broadcast-ID Speicherung dargestellt.

```
<Broadcastinfo>
        <Broadcast>
                <ID>234235</ID>
                <Destination>\\lala.dcs.lolo.de\xyz</Destination>
                <Path>
                        <Step>\\lala1.dcs.lolo2.de\xyz3</Step>
                        <Step>\\lala3.dcs.lolo5.de\xyz7</Step>
                </Path>

        </Broadcast>
</Broadcastinfo>
```

```
<Broadcastids>
        <Broadcast>
                <ID>234234</ID>
        </Broadcast>
        <Broadcast>
                <ID>678788</ID>
        </Broadcast>
</Broadcastids>
```

7.1.4 Analyse der erhaltenen Datenkataloge zur Lokalisierung der Daten

In diesem Unterkapitel werden nun der Empfang und die Verwendung der Datenkataloge beschrieben. Diesbezüglich wird, wie auch zuvor, zuerst das entsprechende Flussdiagramm gezeigt und anschließend beschrieben.
Ein wichtiger Punkt bei der Erzeugung der Übersicht XML-Datei ist, dass nach der kompletten Erzeugung alle Merk-Dateien geleert werden. Zu diesen Merk-Dateien gehören zum Beispiel die Dateien, in denen die Broadcast-ID gemerkt wird bzw. die Datei, in der der Eingang eines Datenkatalogs gemerkt wird. Bezüglich der Datei, in der die Broadcast-ID gemerkt wird, besteht aber ein kleines Problem, nämlich, dass diese Datei bei jedem Basissystem geleert werden muss. Da aber jedes Basissystem, das den Broadcast erhalten hat, dementsprechend einen Datenkatalog geschickt hat, kann nun zuerst, vor der Leerung der Datei, in der die Datenkataloge gemerkt wurden, diese Datei verwendet werden. Der Nutzen dieser Datei ist der, dass nun direkt der Absender angesprochen werden kann, um die entsprechende Leerung zu beantragen.
Nachdem nun dieser Sachverhalt geklärt wurde, wird nun das entsprechende Flussdiagramm gezeigt, mit welchem nun der Eingang der Datenkataloge gesteuert wird. Des weiteren beinhaltet dieses Flussdiagramm die Erzeugung der XML-Datei, welche am Ende die Daten-Hierarchie darstellt.

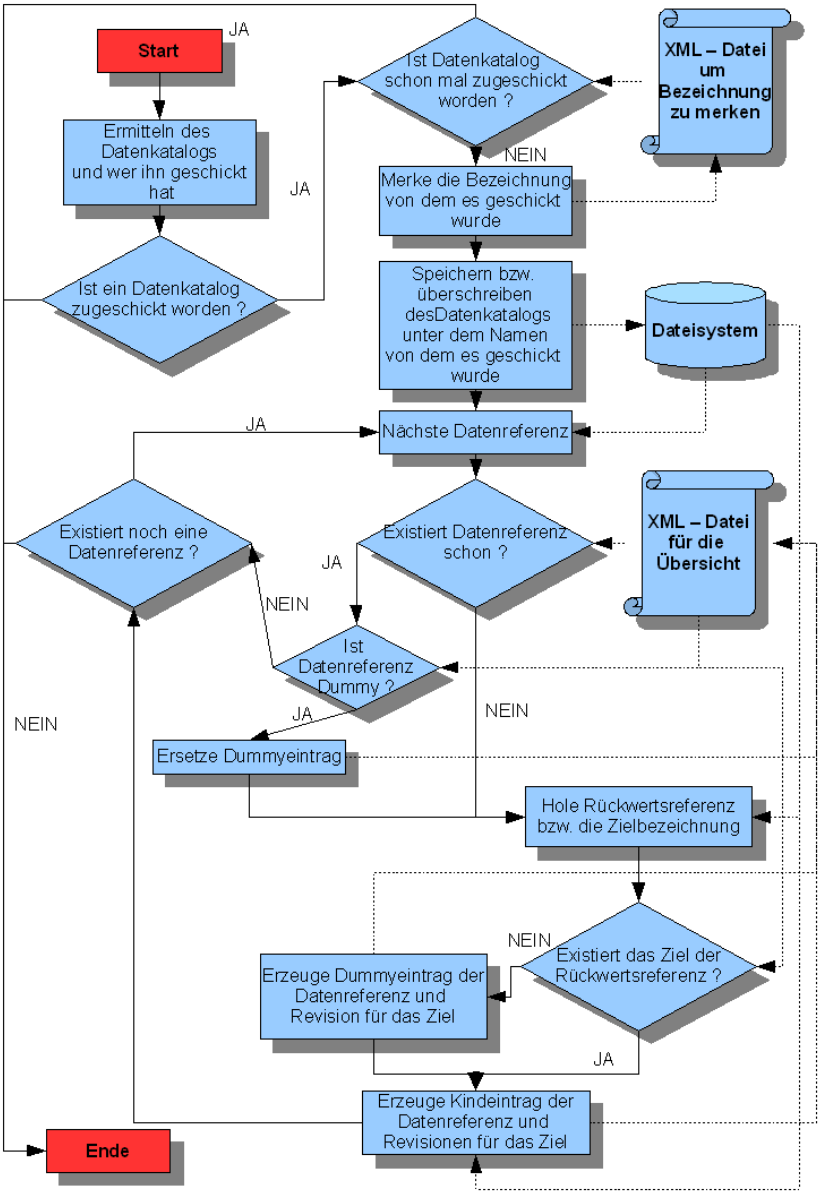

Abbildung 7.1.4-1: Empfang und die Verwendung der Datenkataloge

(Eigener Entwurf)

Nachdem nun das Flussdiagramm gezeigt wurde, wird es beschrieben. Das Flussdiagramm zeigt den Empfang und die Verarbeitung der eingehenden Datenkataloge, welche aufgrund einer Broadcast zugeschickt werden. Diese Verarbeitung ist notwendig, da durch diese am Ende eine XML-Datei resultiert, welche einen Überblick über alle Daten und ihrer Lokalität ermöglicht. Als erstes wird der Datenkatalog und wer diesen geschickt hat ermittelt. Wenn nun ein Datenkatalog eingegangen sein sollte, wird als nächstes anhand des Benutzers, der ihn geschickt hat geschaut, ob dieser Datenkatalog schon einmal eingegangen ist. Falls dies der Fall sein sollte, wird die Aktion direkt beendet. Wenn aber dieser Datenkatalog noch nicht eingegangen sein sollte, wird als erstes dieser Eingang anhand einer XML-Datei gemerkt. Ein entsprechendes XML-Beispiel wird hieran folgend gezeigt. Nach dem Eintrag in die XML-Datei wird der erhaltene Datenkatalog im Dateisystem gespeichert. Nach der Speicherung des Datenkatalogs wird nun beim ersten mal die erste, ansonsten immer die nächste Datenreferenz aus dem gespeicherten Datenkatalog geholt. Als nächstes wird nun geschaut, ob diese Datenreferenz schon in der XML-Datei, in welcher die Übersicht erzeugt wird, gespeichert wurde. Auch bezüglich dieser XML-Datei wird im Anschluss ein entsprechendes Beispiel gezeigt. Wenn nun diese Datenreferenz noch nicht in der Datei existieren sollte, wird versucht, das Ziel der Rückwertsreferenz in dieser Datei zu ermitteln. Falls das Ziel existiert, wird in diesen Eintrag (Revision) ein Kindeintrag der Datenreferenz erzeugt. Nachdem nun die Datenreferenz mit seinen Revisionen in XML gespeichert wurde, wird die nächste Datenreferenz geholt und überprüft. Falls aber keine mehr existieren sollte, so wird die Aktion beendet. Wenn aber das Ziel der Rückwertsreferenz noch nicht existieren sollte, wird ein Dummyeintrag für diese erzeugt. Dieser Dummyeintrag ist in der XML-Datei mittels eines Attributs (Dummy = “true“) auch als solche gekennzeichnet. Weitere Bestandteile des Dummyeintrags sind die Dummydatenreferenz und eine Dummyrevision. Die Dummyrevision, auf die die Rückwertsreferenz zeigt, trägt die gleich Bezeichnung bzw. Key wie das Original.
Innerhalb dieses Dummyrevision-Tags wird nun der abhängige Datenreferenzeintrag erzeugt. Nach diesem Eintrag wird nun, ebenso wie zuvor, die nächste Datenreferenz geholt. Falls keine existieren sollte, wird auch hier die Aktion abgebrochen bzw. beendet.

Nachdem dieser Weg beschrieben wurde, muss nun kurz auf die Situation eingegangen werden, die eintritt wenn die Datenreferenz in der XML-Datei existieren sollte. Hierbei können nun zwei Gründe die Ursache sein. Der erste Grund ist der, dass die tatsächliche Datenreferenz schon eingetragen ist. Wenn dies der Fall ist, wird versucht, die nächste Datenreferenz zu holen und eine weitere Überprüfung durchzuführen. Falls es aber keine weiteren Datenreferenzen mehr existieren sollten, so wird die Aktion beendet. Wenn aber nun die gefundene Datenreferenz ein Dummy sein sollte, wird diese durch das gefundene Original ersetzt. Dies stellt den zweiten Grund dar. Nach dem Ersetzen des Dummyeintrags wird wie schon beschrieben weiter vorgegangen. Durch dieses Vorgehen wird nun nach und nach die Hierarchie der erreichbaren Datenreferenz mittels der XML-Datei abgebildet und kann in einer GUI dargestellt werden. Wie zuvor schon beschrieben wurde, ist die Speicherung der Übersicht und der Merk-Datei in diesem Konzept mittels XML umgesetzt worden, kann aber ohne weiteres auch als “Apache Derby“-Lösung realisiert werden. Das folgende XML-Beispiel stellt die Struktur der Übersicht dar.

```
<AllData>
 <Datareference Dummy = "false" Key="ddd2" Author="Hans"
 Date="12.12.2008" Extension="fff" Size="123123" Name="Flo">
  <Revision Key="rrr2" Author="Otto" Date="20.01.2009" Extension="rrr"
  Size="678678" Name="Tatjana">
  </Revision>
  <Revision Key="rrr3" Author="Tatjana" Date="30.02.2009"
  Extension="rrr2" Size="456324" Name="Flo">
   <Datareference Dummy = "false" Key="ddd2" Author="Hans"
   Date="12.12.2008" Extension="fff" Size="123123" Name="Flo">
    <Revision Key="rrr2" Author="Otto" Date="20.01.2009"
    Extension="rrr" Size="678678" Name="Tatjana">
    </Revision>
   </Datareference>
  </Revision>
 </Datareference>
 <Datareference Dummy = "true" Key="343444" Author="" Date=""
 Extension="" Size="" Name="">
  <Revision Key="234234" Author="" Date="" Extension="" Size=""
  Name="">
   <Datareference Dummy = "false" Key="ddd2" Author="Hans"
   Date="12.12.2008" Extension="fff" Size="123123" Name="Flo">
    <Revision Key="rrr2" Author="Otto" Date="20.01.2009"
    Extension="rrr" Size="678678" Name="Tatjana" >
    </Revision>
   </Datareference>
  </Revision>
 </Datareference>
</AllData>
```

Das zweite XML-Beispiel zeigt die Speicherung der der Broadcast-ID.

```
<ListOfDataObtained>
        <BroadcastIdentification >
                <ID>234234</ID>
        </BroadcastIdentification>
        <BroadcastIdentification>
                <ID>678788</ID>
        </BroadcastIdentification>
</ListOfDataObtained>
```

Nachdem nun die Strategie gezeigt wurde, wie ein Überblick über die Daten und ihrer Lokalität möglich ist, wird anschließend auf eine mögliche Optimierung der Zugriffsgeschwindigkeit eingegangen.

7.2 Konzept zur Optimierung von Speicherorten

In diesem Unterkapitel wird nun auf eine mögliche Optimierung der Zugriffsgeschwindigkeit eingegangen. Die Strategie basiert auf dem Einsatz von so genannten Proxy-Servern [Proxy08]. Da die Geschwindigkeit im Internet nicht immer und vor allem nicht für alle Seiten so schnell ist, wie man sich es wünschen würde, kann man mit relativ einfachen Mitteln eine Verbesserung erzielen. Eine solche Verbesserung der Zugriffsgeschwindigkeit kann zum Beispiel durch den Einsatz eines Proxy-Servers, in Verbindung mit der Caching-Funktion, ermöglicht werden [Proxy08]. Bevor die Strategie genauer gezeigt wird, wird zunächst kurz der Begriff Proxy erläutert. Ein Proxy (-Server) stellt eine zwischengeschaltete Komponente bei der Kommunikation zwischen zum Beispiel einem Browser oder in diesem Fall einem Basissystem und dem Rechner bzw. dem anderen Basissystem, von dem man eine Internetseite oder Datenreferenz abrufen will, dar. Der Proxy-Server ist ein Stellvertreter sowohl für den “Client“ als auch für den “Server“, denn die Anfrage nach der Internetseite oder Datenreferenz richtet der “Client“ an den Proxy, und von diesem erhält er dann auch die angeforderten Daten.

Auf der anderen Seite kommuniziert der “Server“ nur mit dem Proxy [Proxy08]. Wenn jetzt jedoch, wie beschrieben, nur eine Komponente zwischengeschaltet würde, so kann sich die Geschwindigkeit nicht steigern, sondern im Gegenteil nur geringer werden. Ein Proxy ist des weiteren auch nur für lokale Netze sinnvoll bzw. optimiert nur die Zugriffsgeschwindigkeit für die Systeme, die sich in diesem Netzwerk befinden. Der Proxy stellt dadurch das Bindeglied zum Internet dar, durch welches aber auch eine bessere Kontrollmöglichkeit gegeben ist. Die tatsächliche Optimierung der Zugriffsgeschwindigkeit liegt letztendlich darin, dass man üblicherweise einen Proxy in Verbindung mit einer Caching-Funktion betreibt. Diese Caching-Funktion speichert nach dem ersten Aufruf die übermittelten Daten [Proxy08]. Hierdurch wird es nun zum Beispiel möglich, weitere Aufrufe nicht wie gewohnt über das Internet zu schicken, sondern man greift auf die Daten des Proxys zu und nutzt diese. Da der Proxy sich im lokalen Netzwerk befindet, können dadurch die Vorteile (wie zum Beispiel Geschwindigkeit und Überprüfbarkeit) des LANs genutzt werden. Des weiteren können die Daten, die im Proxy gespeichert werden, als eine Art Backup betrachtet werden. In speziellen Fall des
SESIS-Systems würde man nun zwar einmal einen Branch von dem tatsächlichen System durchführen, aber der Merge würde dann auf den Daten des Proxy-Servers vollzogen werden. Um nun einen Merge auf das ursprüngliche System zu ermöglichen, muss nun lediglich eine entsprechende Merge-Aktion im Proxy-Server bzw. RCE-Proxy-Server durchgeführt werden. Dies kann sowohl manuell als auch automatisch geschehen. Die automatische Synchronisation sollte einerseits immer in bestimmten Intervallen durchgeführt werden, aber andererseits auch zu einen Zeitpunkt, zu dem keiner mit den Daten arbeitet. Der Grund hierfür ist eine mögliche Blockade des Arbeitsprozesses, was bereits bei der Darstellung der lokalen und globalen Ermittlung gezeigt wurde. Bei einer manuellen Synchronisierung sollte, wie ebenfalls zuvor beschrieben, eine redundante Mitteilung über die bevorstehende Maßnahme an die betroffenen Benutzer gehen. Im Endresultat wird bei einem Branch einer Revision die gesamte Datenreferenz auf den Proxy gespiegelt und auf diese referenziert. Wie zuvor schon beschrieben, wird ein Merge einer Revision nun mit der des RCE-Proxys durchgeführt.
Eine Optimierungsmöglichkeit bezüglich der Speichernutzung wäre zum Beispiel der Einsatz von Durchschnittswerten.

Man könnte zum Beispiel die Durchschnittswerte der Branch- und Merge-Aktionen aller Datenreferenzen ermitteln. Wenn nun der Proxy diese Aktionen zählen würde, könnte man bei einer Überschreitung der Durchschnittswerte die Speicherung der Datenreferenz auf dem RCE-Proxy veranlassen. Hierdurch würden nun nicht alle Datenreferenzen auf dem Proxy gespeichert, sondern nur die, bei denen ein ungewöhnlich hoher Zugriff stattfindet. Hierdurch würde man die Zugriffsgeschwindigkeit teilweise optimieren und Kosten bezüglich teurer Speichermedien einsparen. Für den Benutzer sieht aber das branchen und mergen wie gewohnt aus, da er nur die Datenreferenzen sieht, aber nicht, wo sie tatsächlich liegen. Bezüglich der Ermittlung der Durchschnittswerte werden folgende Informationen benötigt.

- Benötigte Informationen
  - Von wo und wie oft wurde auf eine Datenreferenz zugegriffen ?
  - Ist die Anzahl der Zugriffe größer als der Durchschnitt ?
  - Wie lange ist der letzte Zugriff her ?
  - Ist der Zeitraum des letzten Zugriffs größer oder kleiner als das durchschnittliche Alter ?
  - Logisch von einer Datenreferenz abhängige Daten können ebenfalls direkt mit kopiert werden obwohl noch nicht auf diese Daten zugegriffen wurde ... aber davon auszugehen ist, dass dies noch geschieht
  - Es muss festgelegt werden ob ein “move“ oder ein “copy“ der Datenreferenz auf den Zielrechner oder Server geschieht .... ein “copy“ kann ebenfalls funktionieren, da die Verwaltung durch das Revisioncontrollsystem sowieso geschieht.
  - Beim Verlagern des Speicherortes muss geprüft werden ob Lese- oder Schreibberechtigung zugrunde liegen, da Schreibberechtigung die höchste Priorität besitzen und nur in diesem Fall eine Verlagerung Sinn macht. Falls nur von den Daten gelesen wird, ist der Aufwand einer Verlagerung bzw. mergen nicht gerechtfertigt.

In der hieran folgenden Abbildung wird nun der Aufbau eines solchen Netzwerkes dargestellt.

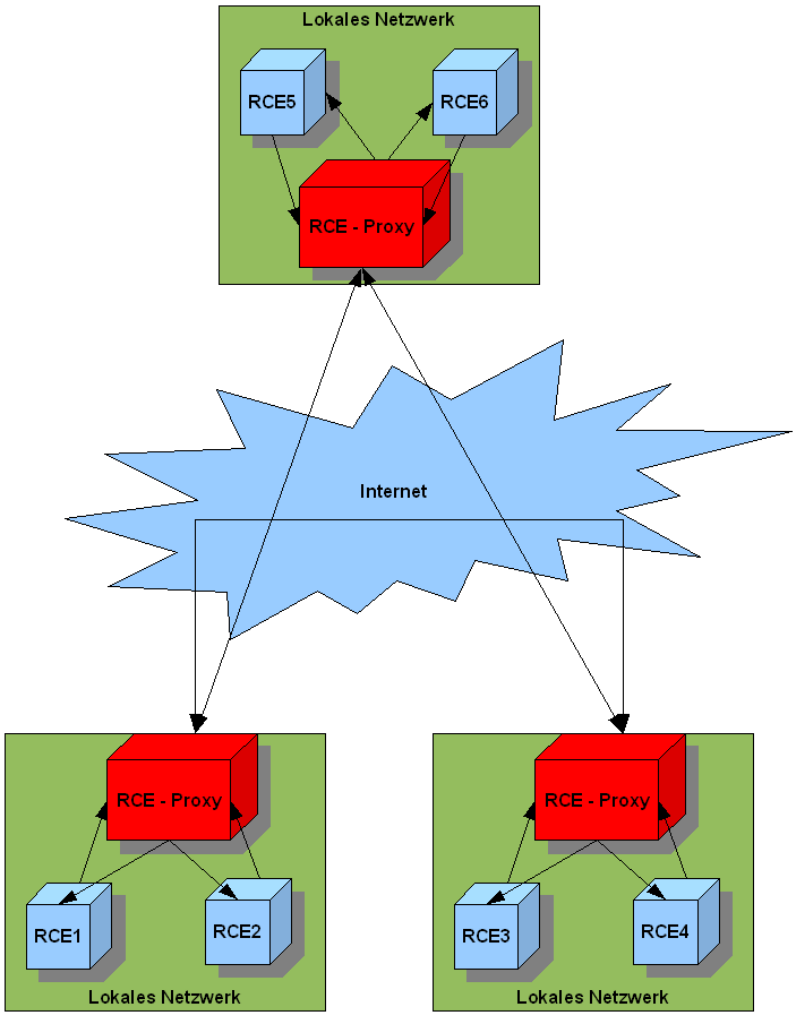

Abbildung 7.2-1: Aufbau eines Netzwerkes mit RCE-Proxy Server

(Eigener Entwurf)

8 Konzept zur Statusermittlung des Systems

In diesem Teil dieser Diplomarbeit wird nun eine mögliche Statusermittlung gezeigt. Diese Ermittlung soll dem Daten-Administrator eine Hilfestellung ermöglichen, um das System besser beurteilen zu können.

8.1 Benötigte Informationen zur Statusermittlung

An dieser Stelle wird kurz darauf eingegangen, welche Informationen dafür nötig sind, um dem Daten-Administrator eine Hilfestellung zu ermöglichen. Im späterem Verlauf wird dann auf die Ermittlung derselben eingegangen.
In der hieran folgenden Aufzählung werden nun die Informationen gezeigt, welche es dem Daten-Administrator erleichtern sollen, den Status des Systems zu ermitteln.

Benötigte Informationen, um das System bzw. dessen Status zu erkennen :

- Darstellung wo die Daten liegen bzw. der Lokalität der Daten
- Für den ausgewählten Datenserver
  - Darstellung und Nachvollziehbarkeit (ISO 9000) der aktuellen Struktur bzw. Zusammenhänge der Revisionen bzw. Datenreferenzen
  - Anzahl der Datenverzweigungen bzw. Abhängigkeiten pro Revision
  - Gesamtanzahl der Revisionen bzw. Datenreferenzen
  - Welche Revisionen (Namen) sind vorhanden ?
  - Wann wurde welche Revision gebranched ?
  - Wann wurde welche Revision gemerged ?
  - Anzahl der Zugriffe pro Revision
  - Wann war der letzte Zugriff auf eine Revision ?
  - Von wem wurde die Revision angelegt ?
  - Wann wurde die Revision angelegt ?
  - Wer hat als letztes auf eine Revision zugegriffen (siehe Datenschutzbestimmungen) ?
  - Farbliche Kennzeichnung bei selten (rot) bzw. bei häufig (grün) genutzten Revisionen (Anzahl der Zugriffe bezüglich Rot-Grün-Blindheit)

- Kennzeichnung von fehlerhaften Revisionen bzw. Datenreferenzen durch ein Symbol, zum Beispiel :

- Gesamte Anzahl aller fehlerhaften Revisionen bzw. Datenreferenzen
- Gesamte Anzahl aller “nicht mehr benötigten“ Revisionen bzw. Datenreferenzen
- Gesamte Statusdarstellung anhand von Abstufung von rot (schlechter Gesamtstatus) bis grün (guter Gesamtstatus) zusätzlich eine Prozent-Anzeige von 0% bis 100%, wobei 100% keine fehlerhaften Revisionen und keine nicht mehr benötigten Daten entspricht.
  - Faktoren zur Ermittlung der Prozentzahl :
    - Gesamte Anzahl aller Fehlerhaften Revisionen bzw. Datenreferenzen
    - Gesamte Anzahl aller “nicht mehr benötigten“ Revisionen bzw. Datenreferenzen
    - Gesamte Anzahl aller Revisionen bzw. Datenreferenzen, die durch Verlagerung des Speicherortes eine Geschwindigkeitsoptimierung ermöglichen

Neben diesen gezeigten Punkten, könnte man aber noch zusätzlich Informationen über die Komponenten liefern, welche zusätzlich zu dem gerade betrachteten Basissystem eingebunden wurden. Diese Informationen könnten dem Daten-Administrator helfen, um zum Beispiel fest zu stellen, ob Komponenten fehlen und um diese einzubinden oder aber auch um fest zu stellen, ob Komponenten veraltete sind und um auch diese zu erneuern. In der folgenden Aufzählung werden nun diesbezüglich Informationen gezeigt, die dies ermöglichen können.

- Darstellen welche Komponenten zusätzlich zum RCE-Basissystem eingebunden sind
  - Wann wurde die Komponente eingebunden ?
  - Welche Version hat die Komponente ?
  - Von wem wurde die Komponente erzeugt (Firma etc.) ?
  - Wer hat die Komponente eingebunden ?
  - Auf welchen Plattformen kann die Komponente betrieben werden ?
  - Name der Komponente
  - Komponentenbemerkung

8.2 Überblick- und Statusdarstellung anhand einer GUI

Nachdem die Informationen aufgezählt wurden, welche die Erkennung des Systemstatus ermöglichen, wird nun eine mögliche Darstellung dieser Informationen gezeigt. Anhand der folgenden Abbildung wird nun eine mögliche GUI dargestellt, welche dem Daten-Administrator die benötigten und zuvor beschriebenen Informationen liefert. Diese Darstellung wird als “Status Explorer“ bezeichnet.

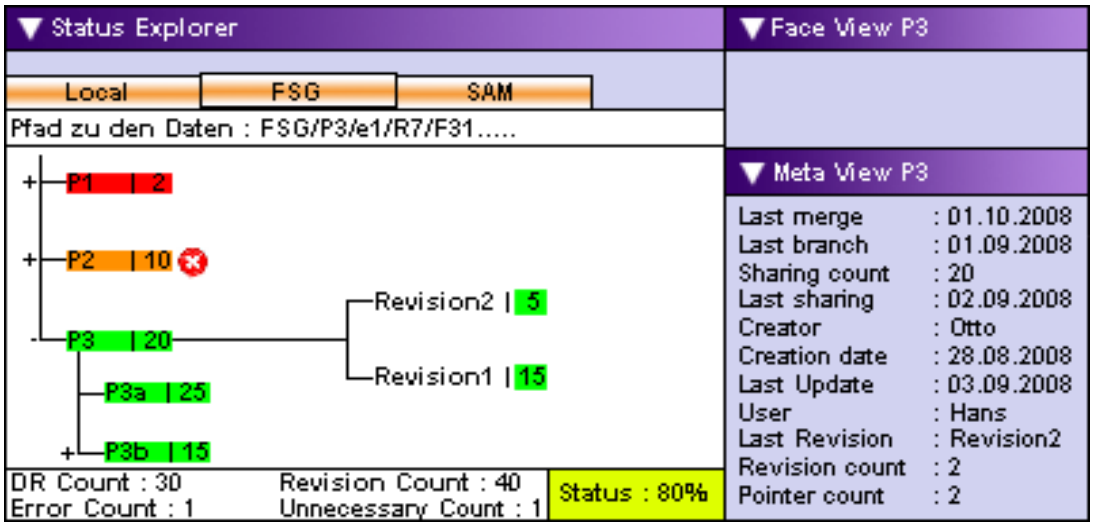

Abbildung 8.2-1: Beispiel für Überblick- und Statusdarstellung mittels "Status Explorer"

(Eigener Entwurf)

Diese gezeigte GUI hat eine ähnliche Gestaltung wie der schon existierende "Daten Explorer". Der Grund hierfür ist das einheitliche Erscheinungsbild und die dadurch entstehende, verbesserte Useability.

Diese GUI ist folgendermaßen aufgebaut. Im linken zentralen Teil wird die Hierarchie der Datenreferenzen bzw. Revisionen gezeigt. Dies wird mittels einer Baumstruktur realisiert. Hierbei bestehen aber zwei Möglichkeiten der Darstellungsart. Die erste Darstellungsart wäre, die komplette Abhängigkeit der Datenreferenzen und Revision in eine Darstellung zu packen. Hierbei besteht aber eine große Gefahr, dass ab einer bestimmten Anzahl an Datenreferenzen und Revisionen diese ganze Geschichte unübersichtlich wird.

Aus diesem Grund fiel die Entscheidung auf die zweite Darstellungsart. Diese Darstellungsart sieht es vor, dass für jedes Basissystem bzw. jeden Datenkatalog ein eigener Reiter existiert. Dieser Reiter wäre einmal das lokale System, welches immer vorhanden ist, und die anderen Basissysteme.

Zusätzlich wird unter dem jeweiligen Reiter noch der absolute Pfad des Systems gezeigt.
Zusätzlich zu der Baumstruktur, welche die Abhängigkeiten zeigt, wird bei jeder Datenreferenz und Revision die Anzahl der Zugriffe gezeigt. Hierbei wird diese Anzahl sowohl farblich als auch als Zahl gezeigt. Der Grund hierbei ist der, dass Benutzer mit einer Rot-Grün-Blindheit ebenfalls die Anzahl der Zugriffe erkennen müssen. Des weiteren wird bei jeder Datenreferenz und Revision kenntlich gemacht, ob diese fehlerhaft ist. Dies geschieht mittels eines Symbols.
Im Bereich unten links wird in dieser Abbildung der allgemeine Status des Systems gezeigt. Hierzu gehört zum Beispiel die Anzahl aller Datenreferenzen und Revisionen, aber auch die Anzahl aller Fehler und alten bzw. nicht mehr benötigter Daten. Diese vier Informationen werden des weiteren für die letzte Informationsquelle in diesem Bereich benötigt. Dies ist der allgemeine Status des Systems. Dieser wird ebenfalls mittels einer Prozentzahl und einer entsprechenden Farbe kenntlich gemacht. Der Grund hierfür, wie auch zuvor, ist eine mögliche Rot-Grün-Blindheit. Wie schon gesagt, wird dieser Prozentwert durch die genannten Faktoren bestimmt. In der folgenden Aufzählung werden diese nochmal gezeigt.

- Faktoren zur Ermittlung der Prozentzahl :
  - Gesamtanzahl aller Datenreferenzen und Revisionen
  - Gesamte Anzahl aller fehlerhaften Revisionen und Datenreferenzen
  - Gesamte Anzahl aller “nicht mehr benötigten“ Revisionen und Datenreferenzen
  - Gesamte Anzahl aller Revisionen und Datenreferenzen die durch Verlagerung des Speicherortes eine Geschwindigkeitsoptimierung ermöglichen

Der letzte Bereich dieser Statusdarstellung ist der Bereich rechts in der zuvor gezeigten GUI. In diesem Bereich werden nun genauere Informationen für die ausgewählte Datenreferenz oder Revision gezeigt.

Zu diesen Informationen gehört zum Beispiel das Datum des letzten Merge und Branch, die Anzahl, wie viele Datenreferenzen diese Datenreferenz oder Revision nutzen, das Datum der letzten Benutzung, wer es erzeugt hat, wann es erzeugt wurde, wann das letzte Update war, welchem Benutzer es gehört, was die letzte Revision war, die Anzahl der Revisionen und Rückwertsreferenzen.
Durch diese Informationen kann sich der Daten-Administrator nun ein genaueres Bild über den Status des Systems machen.

8.3 Ermittlung der Statuswerte

Nachdem nun die Darstellungsart und Auswahl an gezeigten Informationen beschrieben wurden, wird auf die Ermittlung dieser Werte eingegangen.
Die Ermittlung basiert auf Werten, welche mittels der zuvor beschriebenen Ermittlungen erzeugt und gespeichert wurden. Zu diesen Ermittlungen zählt zum Beispiel die Ermittlung von alten und nicht mehr benötigten Daten, von Fehlern im System und die Identifizierung bzw. Optimierung der Daten.
Diesbezüglich wird zuerst ein entsprechendes Flussdiagramm gezeigt, welches dann anschließend beschrieben wird.

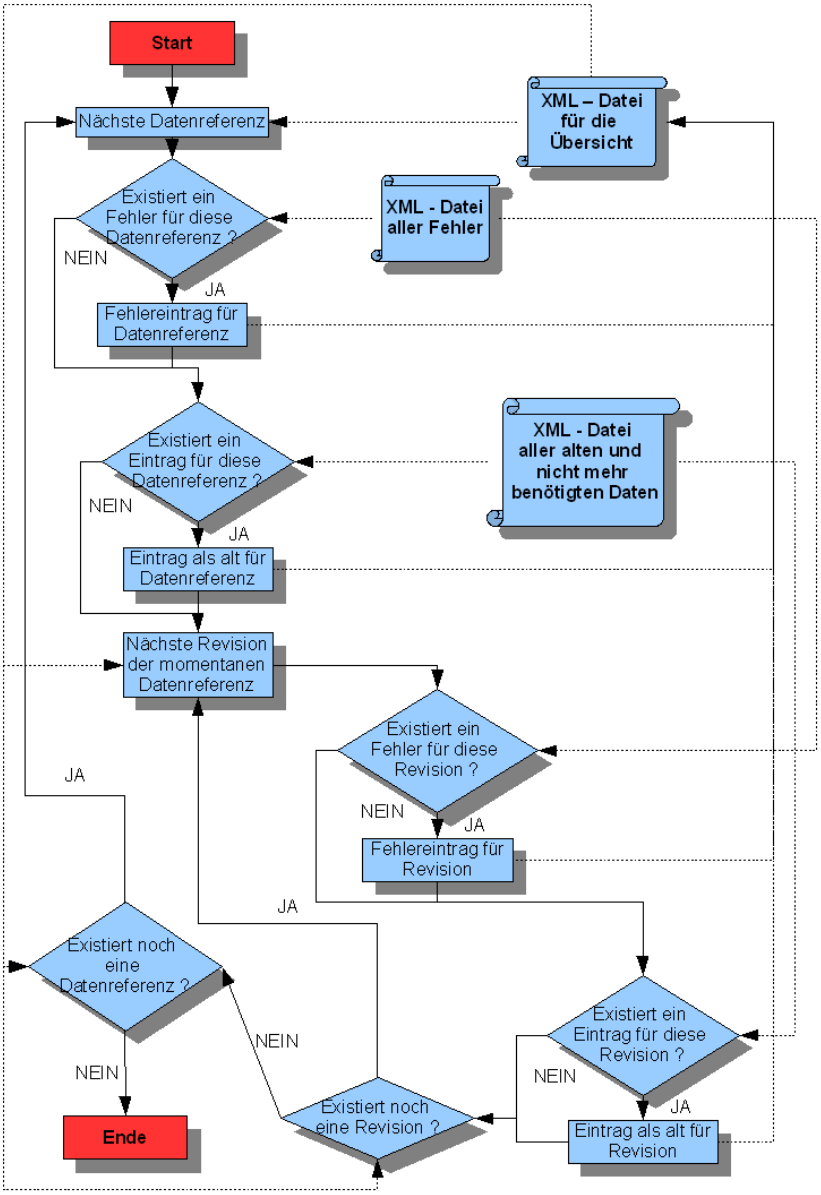

Abbildung 8.3-1: Ermittlung der Informationen für die Statusdarstellung

(Eigener Entwurf)

Nachdem das Flussdiagramm gezeigt wurde, wird es nun beschrieben:
Beim ersten mal wird die erste, ansonsten immer die nächste Datenreferenz geholt. Die jeweiligen Datenreferenzen werden aus der zuvor erzeugten XML-Datei, welche im Rahmen der Übersichtserzeugung angelegt wurde, geholt. In dieser Datei sind die kompletten Abhängigkeiten abgebildet.
Nachdem nun die Datenreferenz geholt wurde, wird geschaut, ob diese momentane Datenreferenz in der XML-Datei, in der alle Fehler gespeichert wurden, vorhanden ist. Falls dies der Fall sein sollte, wird nun ein entsprechender Fehlereintrag für jeden Fehler dieser Datenreferenz durchgeführt. Ein entsprechender Eintrag könnte zum Beispiel wie folgt aussehen.

```
<Error ErrorID="ID" ID="ID" Date="Tag:Monat:Jahr" Action="Wodurch
wurde die Ermittlung gestartet">
            "Beschreibung"
</Error>
```

Dieser Eintrag besteht aus Attributen wie der ErrorID, der ID der Datenreferenz oder Revision, dem Datum an dem der Fehler festgestellt wurde und der Aktion durch die die Fehlererkennung gestartet wurde. Der Wert des Tags ist hingegen die Beschreibung des Fehlers.
Wenn aber nun für die momentane Datenreferenz kein Fehler existieren sollte, so wird mit der nächsten Überprüfung vorgefahren.
Bei dieser Überprüfung wird nun geprüft, ob diese Datenreferenz unter den alten und nicht mehr benötigten Daten ist. Falls dies nun der Fall sein sollte, wird ebenso wie zuvor, ein entsprechender Eintrag in die XML-Datei der Übersicht durchgeführt.
Hierbei sieht der entsprechende Eintrag wie folgt aus.

```
<Notused NotusedID="ID" ID="ID" Date="Tag:Monat:Jahr"
Aktion="Wodurch wurde die Ermittlung gestartet">
            "Beschreibung"
</Notused>
```

Zudem ähneln sich auch die Attribute bzw. die Beschreibung dem der Fehlereinträge.
Wenn aber bei dieser Überprüfung die Datenreferenz nicht in der XML-Datei der alten und nicht mehr benötigten Daten gefunden wurde, wird nun auf die Revisionen dieser Datenreferenz eingegangen.
Aus diesem Grund wird nun die erste Revision der Datenreferenz geholt. Nachdem die Revision geholt wurde, wird nun diese auf einen möglichen Eintrag in der XML-Datei, welche alle Fehler beinhaltet, geprüft. Wenn ein Fehler gefunden wurde, wird ebenso wie zuvor für jeden Fehler ein XML-Eintrag durchgeführt.
Wenn für die Revision kein Fehler in der Datei existieren sollte, so wird die nächste Überprüfung durchgeführt. Bei dieser wird ebenfalls auf alte und nicht mehr benötigte Revisionen geprüft. Falls ein Eintrag gefunden wurde, wird ein entsprechender XML-Eintrag durchgeführt. Ansonsten wird, soweit vorhanden, die nächste Revision geholt. Falls keine Revisionen mehr vorhanden sein sollten, wird nun auf eine weitere Datenreferenz geprüft. Wenn eine existiert, geht das ganze Spiel von vorne los. Ansonsten wird die Aktion beendet.
Durch dieses Vorgehen wurden nun alle relevanten Informationen in eine XML-Datei zusammen gefasst. Die einzelnen Teildateien werden durch die jeweiligen Überprüfungen geliefert.
Anhand dieser neuen XML-Datei ist es nun möglich, die Darstellung im zuvor beschriebenen Hauptfenster zu realisieren.
Das hieran folgende Flussdiagramm zeigt nun die Ermittlung der allgemeinen Statuswerte, die im unteren Bereich des “Status Explorer“ gezeigt werden.

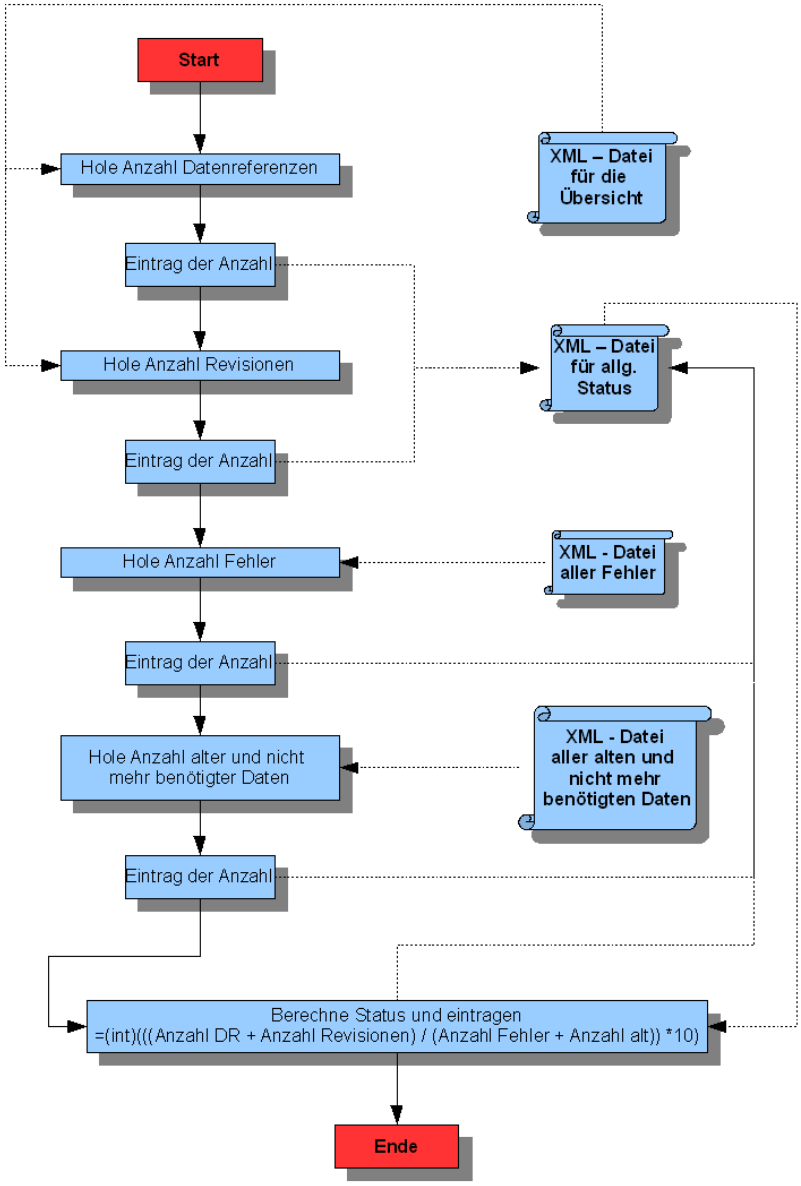

Abbildung 8.3-2: Allgemeine Statusermittlung

(Eigener Entwurf)

In diesem Flussdiagramm wird die allgemeine Statusermittlung bzw. Berechnung gezeigt. Bevor aber die Berechnung durchgeführt werden kann, müssen noch einige Werte ermittelt werden. Als erstes werden nun alle Datenreferenz gezählt. Hierbei wird jede nur einmal gezählt. Nach der Ermittlung des Wertes wird dieser in einer XML-Datei gesichert. Ein Beispiel für den Eintrag wird nun kurz gezeigt.

<Datenreferenzen>“Anzahl aller Datenreferenzen“</Datenreferenzen>

Nachdem nun dieser Wert ermittelt wurde, werden nun alle Revisionen gezählt. Auch hier wird jede Revision nur ein mal gezählt. Das entsprechende Tag, welches gespeichert wird, sieht wie folgt aus.

<Revisionen>“Anzahl aller Revisionen“</Revisionen>

Nach der Ermittlung des Wertes wird dieser nun in die selbe XML-Datei gesichert wie der Eintrag zuvor. Anschließend werden nun alle Fehler gezählt. Auch hier wird jeder Fehler nur ein mal gezählt. Das entsprechende Tag sieht wie folgt aus.

<Error>“Anzahl aller Fehler“</Error>

Nach der Ermittlung des Wertes wird dieser nun wieder in die selbe XML-Datei gesichert wie der Eintrag zuvor. Anschließend werden nun alle alten und nicht mehr benötigten Daten gezählt. Auch hier wird jeder alte und nicht mehr benötigte Wert nur ein mal gezählt. Das entsprechende Tag sieht wie folgt aus.

<Notused>“Anzahl aller alten und nicht mehr benötigten Daten“</Notused>

Wenn nun alle Werte in der XML-Datei gespeichert wurden, wird die Berechnung des Status durchgeführt und ebenfalls in die Datei gespeichert.
Die Berechnung des Wertes sieht wie folgt aus: Zuerst werden alle Datenreferenzen und Revisionen addiert. Anschließend wird dieser Wert durch die Summe aus allen fehlenden und nicht mehr benötigten Daten dividiert. Wenn dies geschehen ist, wird das Ergebnis mal zehn genommen und am Ende zu einem Integer gecasted.
Hierdurch erhält man nun einen Wert, der den allgemeinen Status darstellt. Die Werte in der erzeugten Datei ermöglichen nun die Darstellung.

9 Fazit / Ausblick

Im Rahmen dieser Diplomarbeit wurde eine Analyse und Monitoring eines verteilten Datenmanagements konzeptioniert. Im speziellen betrifft es das verteilte SESIS-Datenmanagement. In diesem Zusammenhang wurden die entsprechenden bzw. bestehenden Methoden bezüglich elementaren Datenmanagement Operationen, Strategien und Fehlererkennungsalgorithmen analysiert. Zudem wurde auf bestehende Datenschutzbestimmungen, Normen (ISO,DIN) und existierende Systeme eingegangen. Im Detail wurde das Diplomarbeitsthema in vier Unterthemen, bzw. deren Konzepte, unterteilt und gezeigt. Mit dem Konzept des ersten Unterthemas (“Konzept zur Ermittlung von alten und nicht mehr gebrauchten Daten“) wurde eine Ermittlung anhand einer Strategie mit einem lokalen und globalen Konzept und den damit verbundenen Algorithmen gezeigt.
Des weiteren wurde mit dem Konzept des zweiten Unterthemas (“Konzept zur Fehleraufdeckung im verteilten SESIS-Datenmanagement“) eine Ermittlung der Fehler im verteilten SESIS-Datenmanagement beschrieben. Dieses Konzept beinhaltet einen Fehlererkennungsalgorithmus welches in die lokale und globale Strategie eingebunden wurden und hierdurch eine permanente und Intervall gesteuerte Fehlerermittlung ermöglicht. Mit dem Konzept des dritten Unterthemas (“Konzept zur Lokalisierung der Daten und Optimierung von Speicherorten“) wurde ein Algorithmus zur Lokalisierung der Daten und Optimierung deren Speicherorten entwickelt und gezeigt. Anhand des vierten und letzten Konzeptes (“Konzept zur Statusermittlung des System“) wurde abschließend eine Statusermittlung und Statusdarstellung des Systems anhand der zuvor ermittelten Parameter entwickelt und beschrieben. Abschließend können diese entwickelten Konzepte der einzelnen Unterthemen als Zahnräder angesehen werden. Diese greifen ineinander bzw. arbeiten miteinander und stellen dadurch eine Abhängigkeit untereinander dar.

Das Resultat dieser Abhängigkeit ermöglicht dadurch eine Analyse und Monitoring des verteilten SESIS-Datenmanagements. Mögliche Ausblicke in diesem Zusammenhang wäre eine Anpassung und Optimierung der einzelnen Konzepte an mögliche neue Gegebenheiten bzw. Änderungen im Basissystem. Des weiteren könnte zum Beispiel die Lokalisierung der Daten durch das “Spanning Tree“-Prinzip optimiert werden [Spant08].
Eine weitere Optimierung könnte auch im Bereich der Zugriffsgeschwindigkeit auf die Daten liegen. Hierbei könnte man eine Individuelle, Basissystem gebundene, Optimierung entwickeln. Bezüglich der Anpassungen, könnte man zum Beispiel die GUI der Statusdarstellung an die entsprechenden Gegebenheiten anpassen bzw. weiter detaillieren.

Ehrenwörtliche Erklärung

Ich versichere,

1. dass ich die Kapitel der Diplomarbeit, für die ich als Verfasser genannt werde, selbständig verfasst habe,
2. dass ich keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt habe,
3. dass ich diese Arbeit bei keinem anderen Prüfungsverfahren vorgelegt habe.

Heidelberg, im September 2008

_____

Clemens Alexander Decker

Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde eine Analyse und Monitoring eines verteilten Datenmanagements konzeptioniert. Die Diplomarbeit wurde in einem Zeitraum von vier Monaten im Institut für Simulations- und Softwaretechnik (SISTEC) des DLR (Deutsches Zentrum für Luft- und Raumfahrt) erarbeitet. Das Konzept unterteilte sich in vier Unterthemen. Diese Unterthemen waren erstens das Ermitteln von alten und nicht mehr benötigten Daten und zweitens das Ermitteln von Fehlern im SESIS-Datenmanagement drittens ging es darum, einen Überblick darüber zu erhalten, wo Daten liegen und ob durch Speicherort-Verlagerung die Zugriffsgeschwindigkeit optimiert werden kann. Letzte Aufgabe war es, den Status des Systems zu ermitteln und darzustellen. In dieser Reihenfolge wurden diese Unterthemen erarbeitet. Die entwickelten Konzepte bzw. deren Basen sind an sich allgemeiner Natur, aber in diesem speziellen Fall an die Gegebenheiten des SESIS-Projekt (Schiffsentwurfs- und Simulationssystem) angepasst. Bevor diese Konzepte gezeigt werden, wurde zuerst auf das Datenmanagement, das bestehende SESIS-Projekt bzw. deren relevanten Bestandteile und die Anforderungen an das SESIS-Datenmanagement eingegangen. Des weiteren wurde in diesem Rahmen das bestehende verteilte Datenmanagement Konzept beschrieben. Neben den behandelten Unterthemen wurde zudem auf die Definition bzw. Beschreibung von Branch- und Merge-Aktionen eingegangen. Darüber hinaus wurden diesbezüglich unter Kapitel A, die existierenden Branch- und Merge-Strategien beschrieben. Weitere zusätzliche Punkte, die im Anhang beschreiben werden, sind einmal die Datenschutzbestimmungen (Kapitel C), da mit Personen bezogenen Daten gearbeitet wird, und bestehende internationale und deutsche Normen (Kapitel B), die für die ordnungsgemäße Umsetzung elementar wichtig sind. Bezüglich der Fehlererkennung in einem verteilten Datenmanagement wurde des weiteren auf bestehende und allgemein gültige Fehlererkennungsalgorithmen eingegangen. Diese zusätzlich genannten Punkte sind für die Rahmenbedingung bzw. Verständnis dieser Diplomarbeit sehr wichtig. Das Ergebnis dieser wissenschaftlich ausgearbeiteten Diplomarbeit zeigt ein Konzept, welches in das bestehende SESIS-Projekt implementiert werden kann und dadurch eine Analyse und Monitoring des verteilten Datenmanagements ermöglicht.

Abstract

My thesis describes a concept for a possible analysis and monitoring of a distributed data system. This thesis was developed during a period of four months at the Institute for Simulation- and Software Technology (SISTEC) of the DLR
(Deutsches Zentrum für Luft- und Raumfahrt). This concept can be subdivided into four sub-topics. These sub-topics were the determining of old and no longer needed data, the identification of errors in the system, an overview as to where data is available and whether the location shift access speed can be optimized and finally the status of the entire system which will be shown. Appearing In this order, these sub-topics are developed in this thesis. The developed concepts and their bases are to be general, but in this particular case they are related to the realities of the SESIS-project. Before these concepts are shown, there was first the need of explaining the SESIS-project or its relevant components, such as interfaces, etc.. Furthermore, the existing distributed data management concept was explained. In addition to the sub-topics there was further a definition and description of branch- and merge-actions. But there are more extra points to describe, for example the data protection provisions, since people with related data are in charge, and existing International and German standards for the proper implementation of fundamental importance. Regarding the error detection in a distributed data management existing and universal error detection algorithms were mentioned. These points are also needed for the framework condition or understanding of this thesis. The result of this thesis shows a possible concept which can be implemented in the existing SESIS-project and thus an analysis and monitoring of distributed data management.

Literaturverzeichnis

[Kasp08]
Kasper, Juliane : Entwicklung eines Systems zur kollaborativen und räumlichen Organisation von Informationsquellen.
http://www.ifis.uni-luebeck.de/lehre/StuDiplBaArbeiten/erledigte/da_informationsorganisation.html, Abruf am 20.05.2008

[CVS08]
CVS (“Concurrent Versions System“) :
http://cvsbook.red-bean.com/translations/german/, Abruf am 25.05.2008

[SESIS108]
Thomas Brandes, Katja Christiansen, Eric Esins, Jürgen Klein, Ottmar Krämer-Fuhrmann, Thijs Metsch, Dirk Rossow, Andreas Schreiber, Sandra Schrödter : System Design für ein Schiffbauliches Entwurfs- und Simulationssystem, Version 1.2

[SESIS208]
CMT, DLR, FhG, FSG, Lindenau,SAM, TU HH : Benutzeranforderungen für ein schiffbauliches Entwurfs- und Simulationssystem, Version 1.3

[GNU08]
GNU-Projekts :
http://www.gnu.org/software/rcs/rcs.html, Abruf am 05.06.2008

[Spant08]
Spanning Tree :
http://en.wikipedia.org/wiki/Spanning_tree_(mathematics), Abruf am 05.07.2008

[DVR08]
Jürgen de Haas, Sixta Zerlauth : DV-Revision, Vieweg+Teubner, 1995, ISBN : 3-528-05446-8

[OSGi08]
OSGi ("Open Services Gateway initiative") Alliance :
http://www.osgi.org/About/Technology, Abruf am 12.07.2008

[Equi08]
Equinox Projekts :
http://www.eclipse.org/equinox, Abruf am 20.07.2008

[Oden94]
Odental, R.: Voraussetzungen für den erfolgreichen Einsatz einer Prüfsoftware im Revisionsbereich, Wpg 1994

[Minz87]
Minz, R. : Computergestützte Jahresabschlussprüfung, IDW-Verlag Düsseldorf, 1987

[Proxy08]
Proxy-Server :
http://www.onlinefan.de/OTIP/Proxyinfo/Proxy-Info.htm, Abruf am 10.08.2008

[Def08]
Branch- und Merge-Definition :
www.codinghorror.com/blog/archives/000968.html, Abruf am 01.06.2008

[Bran08]
Branching :
www.cmcrossroads.com/bradapp/acme/branching, Abruf am 06.06.2008

[BDG08]
Bundesdatenschutzgesetz :
http://www.dfpug.com/loseblattsammlung/loseblatt/auflage/lose2/09_soft/bundesdatenschutzgesetz.htm, Abruf am 25.05.2008

[HSM08]
Hierarchisches Speichermanagement :
http://dnserver.nt.fh-koeln.de/grebe/Diplom/DA/mirko_wiedemann_2005.pdf, Abruf am 17.08.2008

[JFS08]
Journaling Filesystem :
http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html, Abruf am 01.07.2008

[ILM108]
Informationslebenszyklusmanagement :
http://www.project-consult.net/Files/ILM_W2005.pdf, Abruf am 16.08.2008

[ILM208]
SNIA Data Management Forum / SNIA Information Lifecycle Management Initiative, ILM Vision. März 2004

[Baz08]
Bazaar :
http://bazaar-vcs.org/, Abruf am 28.08.2008

[GIT08]
GIT :
http://stud-in.fh-swf.de/Matthias.Faulstich/tools/bazaar/#sec_1_2, Abruf am 28.08.2008

[ISO108]
ISO-Normen :
http://www.vsa-aas.org/uploads/media/Normenkatalog_Version1-1_20070901.pdf, Abruf 10.079.08

[ISO208]
ISO-Normen :
http://www.quality.de/lexikon/din_iso_9000.htm, Abruf am 15.07.08

Abbildungsverzeichnis

Abbildung 2.3-1: Risiken aus Sicht der DV-Revision....9
Abbildung 3.1-1: Schichtenmodell....14
Abbildung 3.2-1: Basissystem....16
Abbildung 3.3-1: Verwendung des Analyse- und Monitoring-Konzeptes im Basissystem 19
Abbildung 3.4-1: Datenmanagement und Daten-Katalog....20
Abbildung 3.4.2-1: Beispiel für ein SESIS-Datenmanagement-Szenario....22
Abbildung 5.1.1.1-1: Beispiel für ersten Startauslöser: Speichern einer Änderung und dadurch Erzeugung einer neuen Revision....35
Abbildung 5.1.1.2-1: Beispiel für zweiten Startauslöser: Branch-Aktion....37
Abbildung 5.1.1.3-1: Beispiel für dritten Startauslöser: Merge-Aktion....38
Abbildung 5.1.2-1: Flussdiagramm des ersten Startauslösers bei der lokalen Ermittlung von Abhängigkeiten....39
Abbildung 5.1.2-2: Flussdiagramm des zweiten Startauslösers bei der lokalen Ermittlung von Abhängigkeiten....41
Abbildung 5.1.2-3: Flussdiagramm des dritten Startauslösers bei der lokalen Ermittlung von Abhängigkeiten....44
Abbildung 5.2-1: Flussdiagramm der globalen Ermittlung von Abhängigkeiten (1/2)....49
Abbildung 5.2-2: Flussdiagramm der globalen Ermittlung von Abhängigkeiten (2/2)....50
Abbildung 5.3-1: Beispiel für Zuordnungsproblematik 55
Abbildung 5.3-2: Beispiel für mehrere, parallele Mitteilungswege für eine Warnmeldung 57
Abbildung 5.4-1: Ergebnis der Lösungsansätze 59
Abbildung 6.1-1: Zusammenwirken der Prüfungsverfahren....63
Abbildung 6.1-2: Ablauf Einzelfallprüfung....66
Abbildung 6.2.2-1: Problematik bezüglich Lese/Schreibrechten und Merge-Aktion....72
Abbildung 6.2.2-2: Problematik bezüglich dem verloren gehen einer Merge-Aktion....74
Abbildung 6.4-1: Flussdiagramm zur Fehlererkennung und Fehlermeldung....82
Abbildung 6.4-2: Flussdiagramm des Startauslöser Löschen....87
Abbildung 6.5-1: Beschreibung der Pull Aktion....94
Abbildung 7.1-1: Beispiel für das Versenden der Broadcast und Empfangen der Datenkataloge....99
Abbildung 7.1.1-1: Beispiel für Broadcaststürme....100
Abbildung 7.1.2-1: Lösung für das Broadcast Problem....102
Abbildung 7.1.3-1: Versenden und Empfangen der Broadcast bzw. Senden des Datenkatalogs....104
Abbildung 7.1.4-1: Empfang und die Verwendung der Datenkataloge....108
Abbildung 7.2-1: Aufbau eines Netzwerkes mit RCE-Proxy Server....115
Abbildung 8.2-1: Beispiel für Überblick- und Statusdarstellung mittels "Status Explorer" 119

Abbildung 8.3-1: Ermittlung der Informationen für die Statusdarstellung.....122
Abbildung 8.3-2: Allgemeine Statusermittlung.....125
Abbildung A.1-1: Branch Beispiel.....141
Abbildung A.1-2: Beispiel eines Entwicklungsbaums von Daten oder Projekten.....143
Abbildung A.3.2-1: Branch per Release.....146
Abbildung A.3.3-1: Branch per Promotion.....147
Abbildung A.3.4-1: Branch per Task.....148
Abbildung A.3.5-1: Branch per Component.....149
Abbildung A.3.6-1: Branch per Technology.....150

Tabellenverzeichnis

Tabelle 4.1-1: Überblick Zielgruppen.....26
Tabelle A.3.7-1: Merge und Branch Vorgehensmuster.....152

Anhang

A Überblick über elementare Operationen

In dem hieran folgenden Kapiteln wird nun zuerst eine genauere Beschreibung der Begriffe und Notation, zum Beispiel bezüglich Branch und Merge, gegeben. Des weiteren wird daran folgend, ein Überblick über bestehende Verzweigungsmuster bzw. deren Strategien gezeigt.

A.1 Begriffe und Notation

In dieser Arbeit wurden verschiedene Begriffe und Notation verwendet. Wo es möglich war, wurde versucht Namen und Konzepte zu benutzen, die häufig in der Praxis wiederkehren. Die hieran folgenden Informationen wurden der Quelle [Bran08] entnommen.

- **Branches, Change-Tasks, and Codelines**
  In der Regel, beschreibt ein Branch bzw. Zweig einen Austausch mit einer Entwicklungslinie in Form von mehreren logischen Veränderungen, verweist man auf auf die Zweige als codeline, obwohl sie nicht
  Source-Code-Artefakte brauchen. Oft wird ein Branch für eine einzige logische Änderung (auch Change-Task) benötigt. Wird ein Branch nur für eine einzige Änderungsaufgabe benötigt, und anschließend zurück zu seinen Eltern zusammengelegt, nennt man es ein „activity-branch“, oder einfach nur „Branch“ oder „Sub-Branch“. In der Theorie wird der Passus "Branch" und "codeline" als Synonyme verwendet. Bei der Beschreibung von Branching Muster, sollte versucht werden eine kohärente Verwendung des Begriffs "codeline", bezieht sich auf eine längere Dauer des Workstreams, und "Branch", eine einzige activity-branch oder einen Sub-Branch einer codeline, zu benutzen.

Gründe für den Einsatz einer Branch-Aktion :

- Release Versionen einer Software zur Fehlerbehebung
- Fehlerbehebungsmanagement, experimentelle Versionen eines Projektes
- In der Regel versucht man Verzweigung so schnell wie möglich wieder mit dem Hauptzweig zu vereinigen.

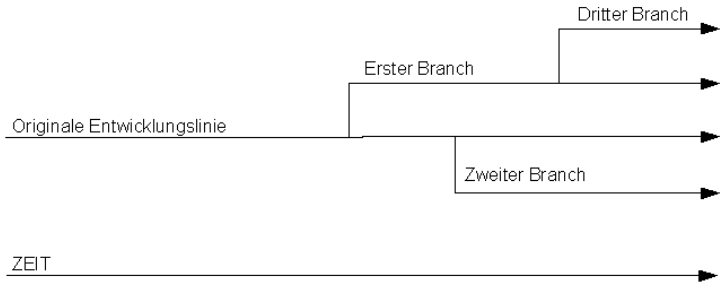

Abbildung A.1-1: Branch Beispiel

(Eigener Entwurf)

- **Versions, Change-Packages, and Baselevels**
  Eine Version kann auf eine Revision einer einzelnen Datei bzw. Daten oder auf eine Reihe von Revisionen Datei bzw. Daten zeigen, die das gesamte Projekt (oder eine ihrer Komponenten und Subsysteme) darstellt. Ein change-package ist die Gruppe von Revisionen, die als Teil eines change-task geändert oder erstellt wurden. Ein baselevel ist eine benannte und konsistente Konfiguration des Projekts, die als eine stabile Basis für die spätere, weitere Entwicklung gilt. Eine Baseline ist ein baselevel, die sich für eine förmliche, interne oder externe Version eignet.

- **Merging, Propagating, and Syncing**
  Merging ist der Prozess welcher die Integration der Revisionen mittels eines change-package mit dem Inhalt einer codeline zusammenfasst. Manchmal muss aber auch ein Änderung in einer codeline in einem anderen codeline angepasst werden. Zum Beispiel, ein Bug-Fix in einer Wartungs codeline kann es erforderlich machen, dass die abhängigen Entwicklungs codelines ebenfalls für den nächsten großen Release geändert werden müssen. Dadurch verweist man auf diese als change propagation, oder simply propagation. Wenn der gesamte Inhalt einer codeline mit einer anderen codeline oder in einen workspace verschmolzen wird, wird diese besondere Art der Verschmelzung, die Synchronisierung (Syncing) mit der codeline genannt.

- **Version Tree Diagrams**
  Da Revision Namen wie "1.4.1.2", die von VC-Tools wie RCS (und viele andere) genutzt werden sind nicht besonders aussagekräftig, normal verwendet man mehr symbolische Branch-Namen, bestehend aus Buchstaben und Zahlen (und einige andere Zeichen). Außerdem nutzt man das' / '-Zeichen, um den Beginn eines Branch-Namen anzuzeigen, so können Versionen eindeutig mit einem Bezeichner wie "/ main/rel1-maint/fix232/4" bestimmt werden. Daher ist ein vollständig spezifizierter Versionsname ähnlich einem Verzeichnis-Pfad in Unix oder DOS. Ein paar VC-Tools (vor allem ClearCase und Perforce) haben gleichen oder ähnliche Vereinbarungen für die Nutzung der Versionsnamensgebung.

Bei der Erstellung von codelines, branches, change-tasks und ihre Beziehungen, verwendet man eine Baumstruktur mit Branch-Namen und Versionsnamen innerhalb der Kreise ("Box" oder "Kreis" ohne Namen sind "anonym"). Branches und codelines sind mit durchgezogenen Linien und Merges und propagations sind mit gestrichelten Linien verbunden.

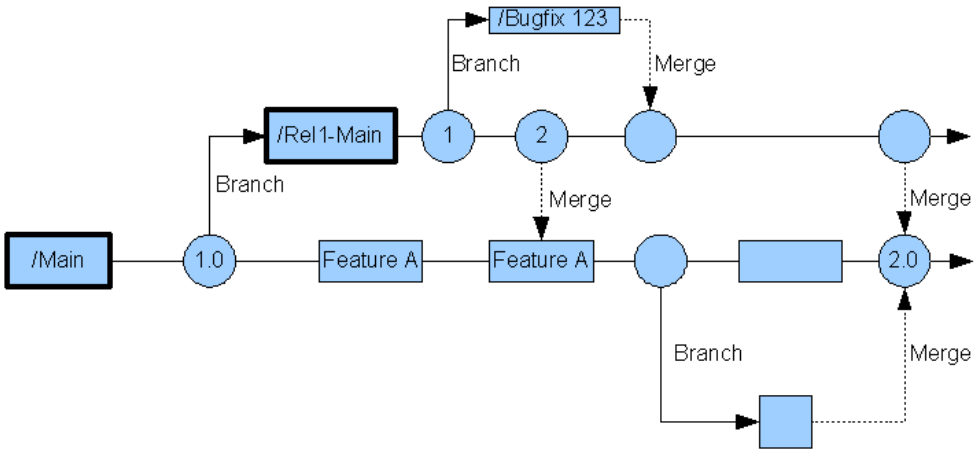

Abbildung A.1-2: Beispiel eines Entwicklungsbaums von Daten oder Projekten [Bran08]

Dieses Versions-Baum-Diagramm erinnert an die Interaktion Sequenzdiagramme in der UML, aber in diesem Beispiel geht die Timeline von links nach rechts, anstatt von oben nach unten. Branch Namen erscheinen immer am Anfang der Timeline und sind mit einem ' / ' gekennzeichnet. Eine "Box" die mittig in einer Timeline liegt entspricht einem Branch bei dem ein "on-line" (direkt am codeline, anstatt auf eigenem Branch) change-task durchgeführt wurde. Des weiterem gibt es keinen führenden Schrägstrich vor dem Name. Die Länge eines change-task "box" kann verwendet werden, um seine Laufzeit, im Vergleich zu anderen change-task, anzuzeigen.

A.2 Dimensionen des Branching

Es gibt im Wesentlichen fünf verschiedene Formen des Branching, von denen jede in, Datei-basierten Branching, VC-Tools eingesetzt wird [Bran08]:

1. **Physikalische :**
   Branching der physischen Konfiguration des Systems-Branches werden für Daten, Dateien, Komponenten und Teilsysteme erzeugt.

2. **Funktionelle :**
   Branching der funktionellen Konfiguration des Systems-Branches werden für Funktionen, logische Änderungen (Bug-Fixes und Erweiterungen) und andere wichtigen Einheiten lieferbarer Funktionalitäten wie zum Beispiel Patches, Releases und Produkte erzeugt.

3. **Umfeldorientierte :**
   Branching des betrieblichen Umfeldes des Systems-Branches werden für verschiedene Aspekte der Build- und Run-Time-Plattformen wie zum Beispiel Compiler, Windows-Systeme, Bibliotheken, Hardware, Betriebssysteme und / oder für die gesamte Plattform erzeugt.

4. **Organisatorische :**
   Branching der Arbeitsbemühungen des Teams-Branches werden für Tätigkeiten / Aufgaben, Teilprojekte, Rollen und Gruppen erzeugt.

5. **Procedurale :**
   Branching des Arbeit Verhalten des Teams-Branches werden erstellt um verschiedene Strategien, Prozesse und Zustände zu erzeugen.

A.3 Verzweigungmuster

Unter diesem Kapitel wird nun auf die allgemein gültigen Verzweigungsmuster eingegangen. Hierbei werden diese besagten Verzweigungsmuster gezeigt und kurz beschrieben.

A.3.1 Einleitung

Verzweigungen und damit die Möglichkeit des parallelen Arbeiten mit Daten ist ein sehr komplexes Unterfangen. Es gibt Dutzende Möglichkeiten bzw. Muster der Verzweigung, und dadurch kann niemand wirklich sagen, ob eine richtig oder falsch ist. Hier werden nun ein paar gemeinsame Verzweigungsmuster gezeigt.
Unter den Verzweigungsmustern sind zum Beispiel „Branch per Release“, „Branch per Promotion“, „Branch per Task“, „Branch per Component“ und „Branch per Technology“. Diese stellen damit die wichtigsten Muster dar. Detaillierte Informationen bezüglich dieser Muster wurden der Quelle [Def08] entnommen.

A.3.2 Branch per Release

Jedes Release ist ein neuer Bereich, häufige Änderungen werden zwischen den Releases zusammengelegt. Gebranchte Daten werden gelöscht, wenn die Versionen nicht mehr benötigt werden.

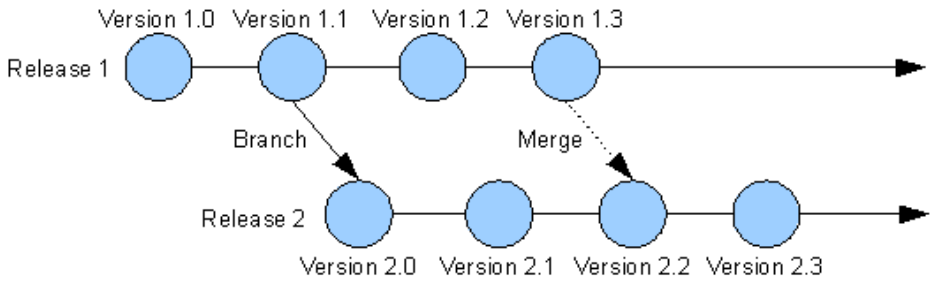

Abbildung A.3.2-1: Branch per Release

A.3.3 Branch per Promotion

Jeder Branch ist ein beständiger Bereich. Wenn Änderungen fertig und getestet sind, werden diese in den Ast gemerged von dem der Branch durchgeführt wurde.

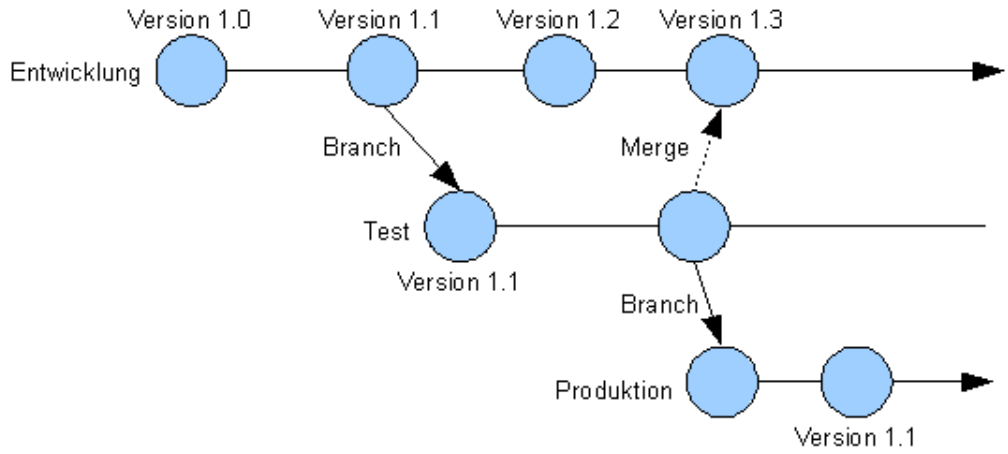

Abbildung A.3.3-1: Branch per Promotion

A.3.4 Branch per Task

Jeder Entwicklungsprozess ist ein neuer, eigenständiger Zweig. Der Entwicklungsprozess wird mit dem ständigen Hauptzweig zusammengeführt, wenn dieser abgeschlossen ist.

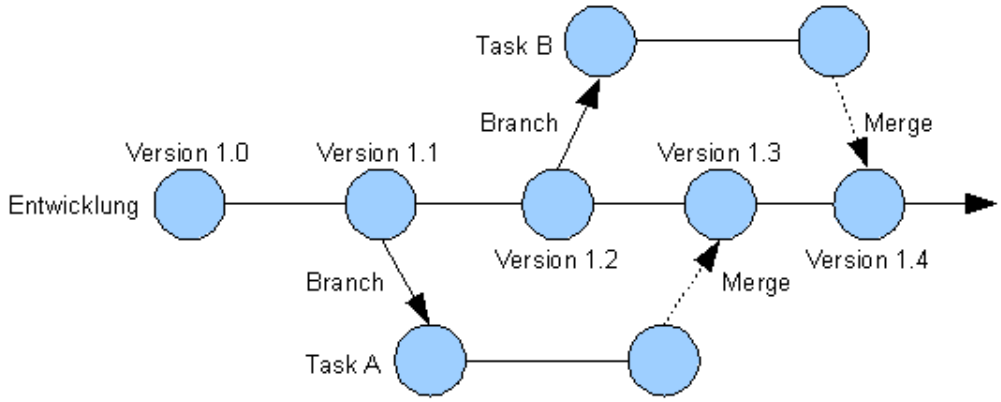

Abbildung A.3.4-1: Branch per Task

A.3.5 Branch per Component

Jede architektonische Komponente des Systems ist ein neuer, unabhängiger Bereich. Abgeschlossene Komponenten werden in den Hauptzweig zusammengeführt.

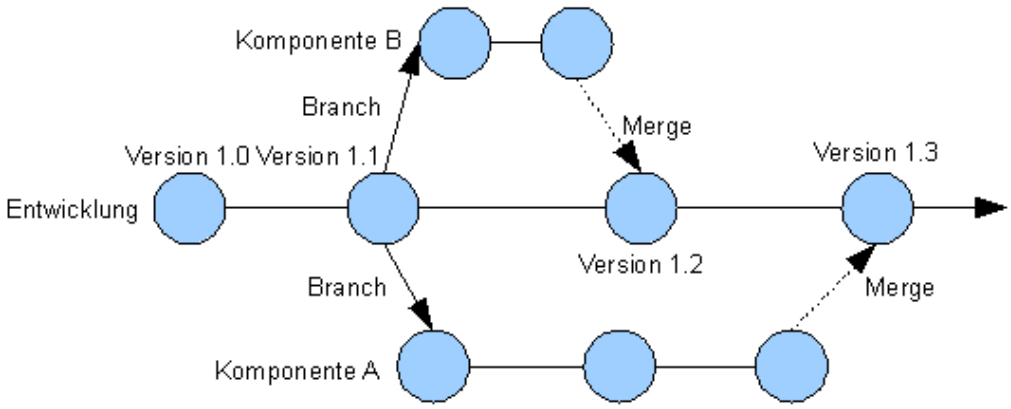

Abbildung A.3.5-1: Branch per Component

A.3.6 Branch per Technology

Jede Technologie-Plattform, zum Beispiel Unix oder Windows, ist ein beständiger Bereich. Gemeinsame Teile der Codebasis werden zwischen den einzelnen Plattformen zusammengeführt.

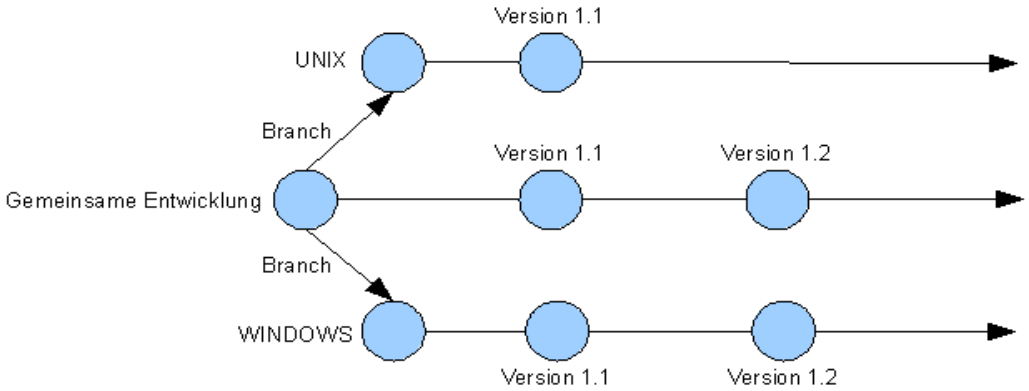

Abbildung A.3.6-1: Branch per Technology

A.3.7 Zusammenfassung

Es ist sehr schnell erkennbar, dass sich einige Muster ähneln:

- Alle Zweige haben einen klar definierten Lebenszyklus. Sie leben entweder ewig, oder sie sind schließlich gelöscht.
- Alle Zweige werden mit der Absicht gebranched, schließlich gemerged zu werden. Ein Zweig ohne eine Zusammenführung ist sinnlos.
- Wenn man Zweige hinzufügt, wird dieses Entwicklungs-Modell bzw. Datenmodell komplizierter.

Aber diese Komplikation ist oft gerechtfertigt. Je mehr Entwickler an einem Projekt arbeiten, desto höher sind die Chancen, dass einer dieser Entwickler fehlerhafte Daten in die Stammdaten einchecked und so die Arbeit aller anderen zerstört. Es ist ganz einfach, Menschen machen Fehler.

Je mehr Entwickler an einem Projekt arbeiten, desto mehr Fehler in den Daten hat man. Und je mehr Entwickler an einem Projekt arbeiten, desto größer sind die Folgen, wenn eine Arbeit, durch einen schlechten Check-In, alle anderen Arbeiten gleichzeitig stoppt. Also, was sind die zur Verfügung stehenden Möglichkeiten ?

1. **Maximale Produktivität :**
   Jeder arbeitet in einem gleichen, gemeinsamen Raum. Es gibt keine Branches, nur eine lange, ununterbrochene gerade Linie der Entwicklung.
   Es gibt nichts zu verstehen, so sind Absprachen einfach jeder kann durch Check-Ins das gesamte Projekt unterbrechen und dadurch alle Fortschritte zum stoppen bringen.

2. **Minimales Risiko :**
   Jede einzelne Person an dem Projekt arbeitet in einem eigenen, unabhängigen Zweig. Dies minimiert die Gefahr, jeder arbeitet unabhängig, und niemand stört die Arbeit eines anderen. Aber dies erzeugt auch unglaublich viel Prozess-Overhead. Die Zusammenarbeit wird hierdurch sehr erschwert - jeder Merge wird hierdurch zu einer sehr mühsamen Aufgabe, sogar die kleinsten Teil des kompletten Systems. Die Antwort liegt in der Regel irgendwo zwischen diesen beiden Extremen. Des weiteren können Verzweigungen missbraucht werden. „Chris Birmele“ stellt fest, dass die Verzweigungen über eine eigene Reihe von Anti-Muster verfügen.

Merge Paranoia	Merging um jeden Preis vermeiden, und zwar wegen einer Furcht vor den Konsequenzen.
Merge Mania	Das Team verbringt übermäßig viel Zeit bei zusammenführen von Softwareteilen, anstatt sie zu entwickeln.
Big Bang Merge	Das Zusammenlegen wird an den Schluss der Entwicklung gelegt. Am Schluss der Entwicklung wird ein Versuch gemacht, alle Zweige gleichzeitig zusammenzufassen.
Never Ending Merge	Merge Aktivität scheint nie zu Enden, es gibt immer mehr was zusammen gefügt werden muss.
Wrong Way Merge	Ein Software Asset wurde mit einer früheren Version zusammen geführt.
Branch Mania	Grundloses branchen wird durchgeführt.
Cascading Branches	Gebranchede Zweige werden nie wieder zum Hauptast zurückgeführt.
Mysterious Branches	Niemand kann Ihnen sagen, was die Zweige sind oder von wo sie herkommen.
Temporary Branches	Der Zweck eines Astes hat sich verändert, dient nun als ständigen, "vorübergehenden" Arbeitsbereich.
Volatile Branches	Wenn von einem Zweig gebranched (shared) wurde oder in eine anderen Ast gemerged wurde.
Development Freeze	Alle Aktivitäten sind gestoppt während Branching, Merging und dem Bau neuer Basislinien.
Berlin Wall	Branchen wird verwendet, um die Entwicklung auf Teammitglieder zu verteilen, anstatt der Spaltung der Arbeit, die sie durchführen.

Tabelle A.3.7-1: Merge und Branch Vorgehensmuster [Def08]

A.4 Risikofaktoren beim Branching und parallelem Entwickeln

Die parallele Entwicklung wirft einige wichtige Fragen und Anliegen für den Erfolg des Entwicklungsprojektes auf. Hieran folgend werden nun kurz die Risiko-Faktoren identifiziert bzw. beschrieben. Informationen wurden der Quelle [Bran08] entnommen.

- **Teamwork**
  Kommunikation, Anstrengung und Interaktion müssen wirksam organisiert und ausgeführt werden, damit eine parallele Entwicklung gelingt. Wichtige Themen sind hierbei :

  - **Kommunikations- und Sichtweite**
    Team-Mitglieder müssen immer auf dem laufenden bleiben, dies beinhaltet was die anderen Teammitglieder tun und warum. Wichtige Änderungen und Baselines müssen sichtbar und häufig an das gesamte Team, sowie an anderen wichtigen Akteuren, mitgeteilt werden. Gleichzeitige bzw. parallele Entwicklungen müssen isoliert voneinander, aber auch zusammen koordiniert werden. Hierdurch ist es möglich dass Änderungen isoliert durchgeführt und anschließend erfolgreich integriert werden können. Man muß aber sehr vorsichtig sein, dass man nicht die Teammitglieder voneinander oder von ihren eigenen Entwicklung isoliert. Es müssen die häufigen Veränderungen bzw. der Status durch Kommunikation gesichert werden, als auch das sorgfältig kontrollieren und isolieren von parallelen Entwicklungen und change-tasks.

- **Rollen und Verantwortung**
  Wenn jeder für eine Sache verantwortlich ist, dann ist es oft niemand. Funktionen, Änderungen, Komponenten und Meilensteine müssen verantwortliche Eigentümer besitzen, die deren Zweck verstehen und zur Rechenschaft gezogen werden können. Die Besitzer müssen aber auch für das ganze Team greifbar sein.

- **Workflow-Organisation**
  Groß, komplexe Aufgaben und Veränderunge müssen in überschaubare Einheiten untergliedert werden. Des weiteren müssen diese Einheiten geeignete Besitzer mit entsprechenden Meilensteinen zugeordnet werden. Es müssen aber auch die Ergebnisse ständig neu in Einklang gebracht werden um letztendlich ein funktionierendes Ganzes zu erhalten. Abhängigkeiten zwischen den Aufgaben, Veränderungen, Meilensteine und ihre Besitzer sollten minimiert werden, um die Risiken im Zusammenhang mit der Sicherheit, Lebensdauer und Wiederverwendbarkeit zu umgehen. Und natürlich brauchen wichtige Aufgaben und Meilensteine auch ihren, für das Team sichtbaren.

- **Wiederverwendbarkeit**
  Wiederverwendbarkeit ist die Eigenschaft, dass baselevels und Änderungen sich problemlos verwenden lassen und in bestehende Versionen einwandfrei eingebunden werden können. Es soll hiermit erreicht werden, dass der Entwicklungsumfang minimiert wird. Dies kann zum Beispel durch Verwendung schon erstellter Entwicklungen eines anderen in diesem eigenen Arbeitsbereich und in jede neue Basislinie geschehen. Es sollte bei jeder Entwicklung das Ziel sein, dass jemand anderes ohne große Änderungen diese Entwicklung ebenfalls nutzen kann.
  Wiederverwendbarkeitsbedenken für die parallele Entwicklung sind :

- **Reproduzierbarkeit**
  Wenn man den Inhalt einer Veränderung oder einer baselevel nicht reproduzieren kann, dann kann man nicht einfach die Wiederverwendung im Entwickler-Workspace oder in einer abgeleiteter Konfigurationen in Erwägung ziehen.

- **Rückverfolgbarkeit**
  Wenn man nicht die Inhalte über eine Änderung oder ein baselevel, ohne weiteres finden oder identifizieren kann, dann kann man diese nicht so leicht wieder verwenden.

- **Trennbarkeit**
  Wenn man nicht ohne weiteres gewünschte Änderunge von unerwünschten Änderungen oder baselevels trennen kann, dann kann man nicht einfach die Wiederverwendung dieser Änderungen in Erwägung ziehen.

- **Sicherheit**
  Mit Sicherheit ist die Eigenschaft gemeint, dass eine Entwicklung keine negative Auswirkungen auf das gesamte Projekt hat. Sicherheitsbedenken für die parallele Entwicklung sind in der Regel im Zusammenhang mit Qualität zu nennen :

  - **Konsistenz der Codeline**
    Man versucht die Codeline in einem konsistenten Zustand zu behalten. Das heißt, es wurden alle beabsichtigten Veränderungen erfolgreich durchgeführt (keine partielle Änderungen), es wurde erfolgreich integriert und es steht nichts im Widerspruch zueinander.

- **Zuverlässigkeit der Codeline**
  Andere Entwickler verlassen sich auf dem aktuellen Stand der Codeline und das diese korrekt funktioniert. Ist dies nicht der Fall, kann die Arbeit bzw. die Anstrengungen verworfen werden.

- **Integrität und Stabilität der Codeline**
  An einem bestimmten Zeitpunkt möchte man, dass die Codeline konsistent als auch zuverlässig ist. Aber auch über diesen Zeitraum hinweg sollte die Codeline zuverlässig, konsistent und konsequent sein. Ansonsten könnte eine Teilentwicklung, die an sich richtig funktioniert, im Zusammenspiel mit den anderen zu Fehlern und dadurch Destabilisierend auf das gesamte Projekt wirken.

- **Verlorene Änderungen**
  Eine Parallel-Entwicklung ohne kontrollierte Wechselwirkung zwischen gleichzeitigen Veränderungen können dazu führen, dass Änderungen verloren gehen oder beschädigt werden und dadurch einen großen Aufwand bei der Nacharbeitung hervorrufen.

- **Wiederkommende Fehler**
  Eine Strategie die sicherstellt, dass die in der vorherigen Version Defekten Bestandteile nicht wieder in eine neuere Version des Produktes eingehen (besonders, wenn mehrere Versionen gleichzeitig entwickelt werden).

- **Lebendigkeit**
  Lebendigkeit ist die Eigenschaft, dass in der Entwicklung immer Fortschritte stattfinden. Lebendigkeit steht in engem Zusammenhang mit der Teamarbeit im Allgemeinen und im Besonderen die Entwicklung der Produktivität. Lebendigkeitssorgen sind auch häufig im Widerspruch zu den Bedenken hinsichtlich der Sicherheit (suggeriert eine Art Gleichgewicht). Lebendigkeitsbedenken für die parallele Entwicklung sind :

- **Erhöhte Effizienz der Arbeit / Produktivität**
  Erhöhung der Gleichzeitigkeit in einem Entwicklungsprojekt kann auf einmal mehr Arbeit ermöglichen, was zu einer verringerten Time-to-Market führt.

- **Verstärkte Koordinierungsbemühungen**
  Erhöhung der Gleichzeitigkeit erhöht die Notwendigkeit und Höhe der, Synchronisation Bemühungen (übertragende und Integration). Zusammenführen und Integration sind oft nicht triviale Tätigkeiten, die sich sowohl zeitaufwändig und fehleranfällig erweisen.
  Die zu weit oder nicht intelligent geplanten Synchronisationsoverheads von Branching und Merging, können aufgrund Parallelisierter Entwicklung als Produktivitätssteigerung ins Gewicht fallen.

- **Abstand zu Streitigkeiten und Arbeitsniederlegungen nehmen**
  Übermäßiges Sperren und "work queuing" kann dazu führen, dass sehr lange "busy waits", in dem die Mitarbeiter gezwungen sind, einige ihrer Aktivitäten zu suspendieren, bis die Daten zur Verfügung stehen. Dies könnte zu Schlafzuständen oder sogar Deadlocks von parallelen Aktivitäten führen.

- **Erzeugungszeit**
  Der Zeitraum die das System zum integrieren und erzeugen braucht, aber auch die Erzeugungsdauer der einzelnen Systeme kann dramatische Auswirkungen auf die Gesamtproduktivität und Koordination haben. Wenn die Systemerzeugung nur wenige Minuten in Anspruch nimmt, können viele Veränderungen, innerhalb eines einzigen Tages, getestet und integriert werden. Wenn hingegen die Systemerzeugung mehrere Stunden oder sogar einen ganzen Tag dauert, können deutlich weniger Veränderungen, innerhalb eines einzigen Tages, getestet und integriert werden.

Dies betrifft nicht nur die Kommunikation und Produktivität, sondern auch die Sichtbarkeit und die Häufigkeit der neuen Basislinien. Je weniger Zeit für das Erzeugen gebraucht wird, desto seltener muss jedes Mal eine Neuerzeugung durchgeführt werden, und desto produktiver kann man sein, aber auch mehr muss kommuniziert werden, was die Änderungen der neuesten, stabilen Grundlinie des Systems betrifft.

- **Merge Komplexität**
  Wenn merging und Veränderungen gleichzeitig geschehen, kann die Art und Komplexität dieser Veränderungen einen großen Unterschied in der Höhe bzw. Vereinbarkeit von gemeinsamen Zusammenhängen erforderlich machen. Jede Änderung betrifft eine Reihe von Zeilen oder Algorithmen, die sich von den anderen Änderungen ergeben. Erfolgreiches merging der Änderungen können in der Regel erreicht werden, indem ein einziger Entwickler eine
  Daten-Vergleich-Werkzeug nutzt. Aber wenn gleichzeitige Änderungen die gleichen Zeilen oder Algorithmen betreffen, ist eine erfolgreiche Lösung der Konflikte mit zwei oder mehr Entwickler möglich. Im extremen Fall können widersprüchliche Änderungen, die sich auf grundlegende Teile eines Teilsystems oder eines Moduls beziehen, es verlangen, dass die Zusammenarbeit von Entwicklern (vielleicht sogar ein oder mehreren Geschäftsführern) eine Neugestaltung und (Wieder-) Aufbau eines gemeinsamen Verständnisses des Teilsystems oder eines Bauteils erfordert. Das letzte Szenario sollte dringend durch eine effektive Verwaltung von Teamarbeit und Workflow vermeiden werden.

- **SCM-Tool-Unterstützung**
  Förderungsmechanismen (oder fehlenden) in Ihrem SCM-Tools für Change Control / Tracking- und Versionskontrolle haben einen gewaltigen Einfluss auf den Erfolg oder Misserfolg der parallelen Entwicklungsbemühungen. Einige dieser Mechanismen sind :

  - **Branching Support**
    Nicht alle VC-Tools ermöglichen ein konzeptuell, einfache Weise für die Verwaltung von Branches und ihre Verwendung für Gruppefunktionalen Änderungen. Viele VC-Tools (wie zum Beispiel RCS und SCCS) zeigen, Zweige mit einem hierarchischen Version Nummerierungsschema wie 1.2.3.4, bei der jede nachfolgende Komponente eine Zweigstelle seiner Vorgänger identifiziert.
    Andere VC-Tools (zum Beispiel ClearCase und Perforce) benutzen hierarchische, symbolische Namen, wie / main/rel_1.1/fix_file_menu, diese sind besser für das Menschliche Verständnis geeignet. CVS erlaubt mnemonische "Tags" als Branch-Namen, aber solche Branch-Namen sind nicht hierarchisch zusammenhängende Branch-Pfade.

  - **Merging Support**
    Merging ist wahrscheinlich am meisten gefürchtet, da sie den höchste Risikoaspekt der parallelen Entwicklung bildet. Der Grad der visuellen und automatischen merging Hilfe durch ein VC-Tool kann den großen Unterschied ausmachen. Die meisten Tools bieten ein Daten-Vergleichprogramm an, welches in der Lage ist, Änderungen zwischen 2-3 verschiedene Versionen derselben Daten zu zeigen. Immer mehr VC-Tools bieten jetzt auch GUI-Unterstützung für Daten Mergeing und Vergleiche an, dass ist einer der Hauptgründe, warum diese immer komfortabler mit Verzweigung für die parallele Entwicklung umgehen.

Die besseren VC-Tools sind nicht nur wegen dem GUI unterstützten Merge hilfreich, sie sind auch in der Lage „tracking Merge-history“ Informationen und Abhängigkeiten zu nutzen um das automatisierte Mergeing leichter zu erfüllen. Die meisten Werkzeuge unterstützen jedoch nur das Mergen von Text-Dateien. Man sollte eher Werkzeuge wählen die in der Lage sind Nicht-Text-Dateien zu mergen, wie zum Beispiel Wort-Prozessor-Dokumente und Tabellen. Mergeing von Nicht-Text-Dateien ist äußerst schwierig und werden oft nicht unterstützt.

- **Logische Gruppierung ändern**
  Einige SCM-Tools unterstützen das Konzept der logischen Veränderungen (manchmal auch als “change-tasks“ oder “change-packages“ beschrieben) direkt. Diese sind in der Lage, die Änderungen an mehreren Daten, die den gleichen Zweck haben, in eine einzige, logische Änderung zusammenzufassen.
  Das kann sehr nützlich für das Management der funktionalen Konfiguration eines Softwareprodukts, für die Ermittlung, Verfolgung und zur Einbeziehung oder beheben von Änderungen für eine bestimmte Funktion sein.

- **Abgeleitete Objekte Wiederverwenden**
  Die meisten Software-Build-Tools (zum Beispiel "make") sind in der Lage, bestehenden Zielobjekt Daten wiederzuverwenden, um so unnötige Neukompilierungen und / oder relinking zu vermeiden. Einige Software-Build-Tools (wie ODIN und ClearCase "clearmake") sind in der Lage, Zielobjekt Daten wiederzuverwenden, die zuvor in einem anderen Arbeitsbereich waren, und so noch mehr unnötige Neukompilierung und / oder relinking zu vermeiden. Diese Fähigkeiten können für den Entwickler erheblich die Build-Zeit und das Testen ihrer Änderungen reduzieren, bevor diese für eine neue baselevel oder Baseline genutzt werden.

- **Individualisierbarkeit**
  Einige SCM-Tools sind mehr als andere konfigurierbar. So schön wie ein Instrument sein könnte, kann es nicht, mit dem Aufruf eines einzigen Befehl (zum Beispiel Makro- oder Maus-Klick), alles was Sie wollen. Einige Tools ermöglichen die Anpassung und Erweiterung von Programmier- und / oder GUI-Schnittstellen.

- **Erweiterbare Event / Aktion Haken**
  Einige Tools bieten zum Beispiel "Trigger" die Benutzerdefinierbare "Haken" oder "Plug-Ins" zu einem bestimmten Zeitpunkt ausführt. Trigger können helfen, eine nahtlose Integration zwischen Ihrem Werkzeugen und Prozessen zu ermöglichen. Wenn Ihr Werkzeug keine Trigger oder benutzerdefinierte Erweiterungen unterstützt, muss man unter Umständen auf die Umsetzung von "Wrapper" für eine gewünschte Operation zurückgreifen. Selbst erzeugte Skripts oder Befehle, die als Front-End-Tool auf bestimmte Funktionen oder das erzeugen dieser Funktionen zusammen in eine sinnvolle Art und Weise erledigen.

B Die deutschen und internationalen Normen

Nun wird an dieser Stelle auf die bestehenden deutschen (DIN) und internationalen (ISO) Normen im Bereich der DV kurz eingegangen.
Da aber keine direkten Normen bezüglich eines Datenmanagements auffindbar waren, wurden vergleichbare bzw. allgemein gültige Normen ermittelt. Diese Normen bilden eine sehr wichtige Grundlage für die Umsetzung eines Datenmanagementsystems. Im Bereich der deutschen Normen regeln diese [DVR08] :

- DIN 66230
  Diese Norm wurde 1981 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit der Programmdokumentation.
- DIN 66231
  Diese Norm wurde 1982 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit der Projektentwicklungs- und Dokumentationsumfang.
- DIN 66232
  Diese Norm wurde 1985 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit der Datei-, Datensatz- und Datenfelddokumentation.
- DIN 66285
  Diese Norm wurde 1994 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit den Software Pakete, Qualitätsforderungen und Prüfbestimmungen (übereinstimmend mit ISO / IEC 12119 von 1993).

Die DIN 66230 unterscheidet ähnlich wie bereits andere aufgeführte Quellen im wesentlichen zwischen dem

- Anwendungshandbuch sowie dem
- Datenverarbeitungstechnischen Handbuch

Im Bereich der internationalen Normen regeln diese [ISO108,ISO208]:

- ISO 14721
  Diese Norm wurde Januar 2002 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit den “Open Archival Information System“. OAIS ist ein ISO-Standard in Form eines Referenzmodells für ein dynamisches, erweiterungsfähiges Archivinformationssystem. Es stellt den Rahmen zur Beschreibung von Modulen, Schnittstellen und Prozessen in einem digitalen Langzeitarchiv dar. Ein offenes Archivinformationssystem besteht aus sechs Funktionsbereichen:

  1. Übernahme (Ingest)
  2. Archivspeicher (Archival Storage)
  3. Datenverwaltung (Data Management)
  4. Verwaltung (Administration)
  5. Nutzung (Access)
  6. Archivierungsplanung (Preservation Planning)

  Das Referenzmodell schreibt kein spezielles technisches Vorgehen bei der Implementation vor.

- ISO 23081
  Diese Norm wurde Januar 2006 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit den “Information and documentation- Records management processes-Metadata for records“. Diese Norm befasst sich bzw. regelt die Erstellung und Anwendung von Metadaten, die für die Verwaltung von geschäftsrelevanten Aufzeichnungen (Records) nach ISO 15489 benötigt werden. Die Norm definiert, welche Metadaten notwendig sind, um die in ISO 15489 postulierten Zwecke (unter anderem Authentizität, Verlässlichkeit, Integrität, Benutzbarkeit) sicherzustellen.

Es werden fünf Kategorien von Metadaten beschrieben:

1. metadata about the record itself ;
2. metadata about the business rules or policies and mandates;
3. metadata about agents;
4. metadata about business activities or processes;
5. metadata about records management processes.

- ISO 9000 (und folgende)
  Diese Norm wurde 2005 festgelegt und befasst sich, im Bereich der Informationsverarbeitung, mit den Qualitätsmanagement. Mit der Normenreihe EN ISO 9000 ff. sind Normen entwickelt worden, die die Grundsätze für Maßnahmen zum Qualitätsmanagement dokumentieren. Jedes Produkt unterliegt anderen spezifischen Anforderungen und ist dadurch nur unter individuellen Qualitätssicherungsmaßnahmen zu erzeugen.

C Datenschutzgesetze bezüglich personenbezogener Datenspeicherung

Da im Rahmen der Metadatenspeicherung auch personenbezogene Daten gespeichert werden, wird nun hier kurz auf die geltenden Bestimmungen bzw. Gesetze eingegangen um eine Rechtliche Grundlage zu gewährleisten.
Bei der Metadatenspeicherung wird unter anderem gespeichert wer und wann eine Datenrevision erzeugt hat. Hierdurch wäre es rein theoretisch möglich, produktive Mitarbeiter von weniger produktiven Mitarbeiten zu selektieren. Des weiteren ist diese Art der Problematik auch bei gängigen Revision-Control-System wie zum Beispiel CVS oder Supversion vorhanden. Da diese Systeme ebenfalls benutzerspezifische Metadaten speichern. Unter anderem aus diesem Grund gibt es seit dem 1.1.1978 das "Bundesdatenschutzgesetz", kurz "BDSG" [BDG08]. Es definiert die notwendigen Datenschutzmaßnahmen im Zusammenhang mit der elektronischen Verarbeitung von personenbezogenen Daten. Darüber hinaus legt es die Rechte und Pflichten des "Beauftragten für Datenschutz" für Betriebe ab 5 (mit EDV) bzw. ab 20 (ohne EDV) Angestellten fest. Der Originaltitel des Gesetzes lautet "Gesetz zum Schutz vor Missbrauch personenbezogener Daten bei der Datenverarbeitung". Das Gesetz enthält sehr detaillierte und praxisnahe Regelungen zum Bereich Datenschutz, die nicht nur für den betrieblichen Datenschutzbeauftragten von Bedeutung sondern eigentlich für jeden Programmierer wichtig sind. Ein großer Teil der Datenschutzmaßnahmen sowie ein gewisser Teil der Tätigkeiten des Datenschutzbeauftragten sind nämlich ohne Vorbereitung und/oder Vorarbeit durch den Programmierer überhaupt nicht realisierbar. Es ist wesentlich besser, dieses Gesetz und seine Vorschriften zu kennen und gleich bei der Programmerstellung zu berücksichtigen, als dann beim Kunden nachbessern zu müssen. Die Einhaltung des Datenschutzgesetzes ist für ein ordentlich erstelltes Programm eigentlich selbstverständlich und kann wohl kaum im Pflichtenheft oder bei der Auftragserteilung abgelehnt werden. Im Folgenden wird deshalb besonderer Wert gelegt auf die notwendigen und möglichen Arbeiten des Programmierers bei der Umsetzung des innerbetrieblichen Datenschutzes sowie im Zusammenhang mit einem Datenschutzbeauftragten [BDG08].

C.1 Vorbemerkung

Das Bundesdatenschutzgesetz besteht aus insgesamt 47 Paragraphen mit meist mehreren Abschnitten. Für den eigenen Betrieb bzw. für Endanwenderprogramme sind die allgemeinen Vorschriften (1-6) sowie die Paragraphen 22 bis 40 wichtig. Die anderen Bereiche regeln den Datenschutz für öffentliche Betriebe/Verwaltung (7-21) sowie für Rechenzentren bzw. Auftragsverarbeiter von Fremddateien (31-40) und enthalten ansonsten die Straf- und Bußgeldvorschriften (41-47). Für den EDV-Alltag sind zwei wesentliche Bereiche herauszustellen : Zum einen die Vorschriften über Datenschutzmaßnahmen und zum anderen die Regelungen zum betrieblichen Datenschutzbeauftragten. Die Datenschutzmaßnahmen treffen jeden Betrieb mit EDV-geführten Dateien welche personenbezogene Daten enthalten. Zusätzlich sei noch darauf hingewiesen, dass sich manche der vom Bundesdatenschutzgesetz geforderten Maßnahmen mit Notwendigkeiten aus anderen Vorschriften oder Regeln überschneiden. Beispielsweise seien hier die Qualitätsrichtlinien aus der ISO 9001, die "Grundsätze ordnungsgemäßer Speicherbuchführung", kurz "GOS", oder die Vorschriften des Bankenvereins über die Durchführung von EDV-Prüfungen in Banken genannt [BDG08].

C.2 Datenschutzmaßnahmen

Im Paragraph 6.1. des BDSG wird die Einhaltung von umfangreichen Datenschutzmaßnahmen festgelegt. Diese Datenschutzmaßnahmen umfassen im Einzelnen die Kontrolle des physischen Zugangs, des Datenträgerabgangs, der Speicherung, der Benutzer, des Zugriffs, der Übermittlung, der Eingabe, der Auftragsverarbeitung, des Transports sowie der allgemeinen Organisation.
Glücklicherweise wird in Paragraph 6.2. festgelegt, daß nicht sämtliche Maßnahmen von allen Betrieben realisiert werden müssen. Es muß jeweils der Aufwand in einem angemessenen Verhältnis zum angestrebten Schutzzweck stehen. Man kommt aber nicht umhin, sämtliche Maßnahmen einzeln zu betrachten und den jeweils notwendigen Aufwand für die Realisierung abzuschätzen. Bei dem Entwurf eines Programmes sollte man von vornherein festlegen, bis zu welcher Stufe man Schutzmaßnahmen realisieren möchte [BDG08].

Bei der Programmierung eines später einmal frei zu verkaufenden Standardprogrammes wird man sich zwangsläufig nach dem potentiell größten Kunden und dessen Anforderungen richten müssen - oder man muss entsprechende Zusatz- und Aufrüstmodule von vornherein einplanen. In letzterem Fall muss man bei netzwerkfähigen Programmen auf jeden Fall die Anwendung sämtlicher Vorschriften zu Grunde legen. Die Vorschriften zu den Datenschutzmaßnahmen kann man grob in drei Teilbereiche aufteilen. Besonderer Wert wird im Gesetz auf die Überwachung aller Tätigkeiten der Benutzer im Zusammenhang mit personenbezogenen Daten gelegt (1-5). Dazu sind auch externe Sicherheitseinrichtungen notwendig, auf die der Auftraggeber hingewiesen werden sollte. Ein weiteres wichtiges Teilgebiet ist die Übermittlung und der Transport der Daten (6-8). Darüber hinaus gibt es noch zwei diverse Punkte (9-10), deren Klassifizierung nicht eindeutig ist [BDG08].

1. Zugangskontrolle:
Unbefugten ist der physische Zugang zu Datenverarbeitungsanlagen mit Zugriff auf personenbezogene Daten zu verwehren.

2. Benutzerkontrolle:
Die unbefugte Benutzung von Datenverarbeitungsanlagen mit Zugriff auf personenbezogene Daten ist zu verhindern.

3. Zugriffskontrolle:
Für die verschiedenen Bereiche der Verarbeitung von personenbezogenen Daten sind Zugriffsberechtigungen zu vergeben.

4. Eingabekontrolle:
Das Erfassungsprotokoll, welches nachträglich feststellen läßt, welche personenbezogenen Daten zu welcher Zeit von wem erfasst oder geändert wurden.

5. Speicherkontrolle:
Des weiteren ist zu kontrollieren, welche personenbezogenen Daten notwendigerweise gespeichert werden müssen und wo bzw. in welcher Form dies erfolgt.

6. Abgangskontrolle:
Die Abgangskontrolle schreibt lapidar vor, dass das unbefugte Entfernen von Datenträgern zu verhindern ist.

7. Übermittlungskontrolle:
Die Übermittlungskontrolle bezieht sich hauptsächlich auf automatische Einrichtungen, die ohne Mitarbeitereingriff funktionieren und über die Unbefugte an die personenbezogenen Daten gelangen könnten (zum Beispiel Modems zur Fernwartung).

8. Transportkontrolle:
Während eines Transportvorganges oder einer Datenübermittlung dürfen die Daten weder unbefugt gelesen, geändert oder gelöscht werden.

9. Organisationskontrolle:
Die Organisation ist so zu gestalten, dass sie den besonderen Anforderungen des Datenschutzes gerecht wird.

10. Auftragskontrolle:
Sofern ein Betrieb Daten im Auftrag Dritter verarbeitet, dürfen diese ausschließlich nach den Weisungen des Auftraggebers behandelt werden.

Zusammenfassend lässt sich sagen, dass der Bereich der Datenschutzmaßnahmen in sehr unterschiedlichem Maß realisiert werden kann. Der Benutzer- und Zugriffskontrolle sollte aber bei der Programmerstellung ein wesentlicher Aspekt zukommen. Es sollte ein standardisiertes System für die Verwaltung der Benutzer und der jeweiligen Zugriffsrechte existieren.

Des weiteren sollte ein internes Protokoll über den Aufruf von wesentlichen Funktionen sowie der Durchführung von Ausdrucken und Datensicherungen existieren.

C.3 Datenschutzbeauftragter

Im Paragraph 28 des BDSG wird die Notwendigkeit sowie die Rechte und Pflichten des "Beauftragten für Datenschutz" festgelegt. Sofern eine Firma personenbezogene Daten verarbeitet ist ein Datenschutzbeauftragter ab einer bestimmten Anzahl von insgesamt vorhandenen Arbeitnehmern vorgeschrieben (5 mit EDV oder 20 ohne EDV). Gemäß Paragraph 28.2. muss die ernannte Person die notwendige Fachkunde und Zuverlässigkeit mitbringen. Gemäß Paragraph 28.3 ist der Beauftragte im Bereich seiner Datenschutz-Tätigkeit weisungsfrei und berichtet direkt an die Geschäftsleitung. Die Aufgaben des Datenschutzbeauftragten lassen sich in die Bereiche Mitarbeiter (1-3), Zulässigkeit (4-5), Kontrolle (6-8) und Listenführung (9-12) gliedern. Insgesamt handelt es sich um 12 verschiedene Tätigkeitsbereiche, die von stark unterschiedlicher Bedeutung für einen Programmierer sind. Der erste Bereich über Mitarbeiter ist irrelevant, der letzte Bereich mit der Listenführung hingegen möglicherweise völlig automatisierbar bzw. kann zumindest mit Hilfsprogrammen stark vereinfacht werden [BDG08].

1. Datengeheimnisverpflichtung:
Gemäß Paragraph 5 ist der Datenschutzbeauftragte verpflichtet, alle Mitarbeitern im Bereich der personenbezogenen Datenverarbeitung auf das Datengeheimnis zu verpflichten.

2. Mitarbeiterbelehrung:
Gemäß Paragraph 29 Absatz 3 ist es notwendig, die Mitarbeiter über die Vorschriften des Datenschutzgesetzes sowie die hausinterne Richtlinien zu unterrichten.

3. Mitarbeiterauswahl:
Gemäß Paragraph 29 Absatz 4 hat der Beauftragte eine "beratende Mitwirkung" bei der Neueinstellung bzw. beim Einsatz von Personen im Bereich der personenbezogenen Datenverarbeitung.

4. Zulässigkeitskontrolle:
Gemäß Paragraph 23. bzw. 25. prüft der Datenschutzbeauftragte die Verarbeitung personenbezogener Daten auf ihre gesetzliche Zulässigkeit.

5. Anwendungsüberwachung:
Gemäß Paragraph 29. Absatz 2 überwacht der Datenschutzbeauftragte die ordnungsgemäße Anwendung der Datenverarbeitungsprogramme.

6. Benachrichtigung:
Gemäß Paragraph 26. Absatz 1 ist der Datenschutzbeauftragte auch für die Benachrichtigung der Personen zuständig, deren personenbezogene Daten gespeichert werden.

7. Datenverifikation:
Der Datenschutzbeauftragte prüft die Korrektheit der personenbezogenen Daten.

8. Berichtigung:
Gemäß Paragraph 27. können Personen nicht nur eine Auskunft über ihre gespeicherten personenbezogenen Daten erhalten, sondern diese im Bedarfsfall auch berichtigen.

Listenerstellung:

Der vierte und letzte Aufgabenbereich des Datenschutzbeauftragten ist im Paragraph 29 Absatz 1 definiert und umfasst die Führung von verschiedenen Listen und Aufstellungen, die nun vorgestellt werden sollen. Dieser Bereich ist für einen Programmierer sehr interessant, da ein Großteil der hier erhobenen Daten in anderer Form oft bereits als Datei vorliegt.

Die entsprechenden Dateien müssen nur einige wenige zusätzliche Felder erhalten, um als Grundlage für die Listen des Beauftragten zu dienen. Im Gegensatz zu den Datenschutzmaßnahmen gibt es hier keine Aufwandsabwägung, sondern die Listen müssen bei Vorhandensein eines Beauftragten auf jeden Fall geführt werden. Von daher sollte die Integration in die Anwendung von Anfang an vorgesehen werden [BDG08].

9. Dateiinventar:
Die erste zu erstellende Liste ist ein Dateiinventar. Das Dateiinventar enthält eine Übersicht über die Art der gespeicherten personenbezogenen Daten. Die Metadaten eines Data-Dictionary können um ein Flag-Feld für den Datenschutz (0=irrelevant, 1=personenbezogene Daten) ergänzt werden.

10. Geschäftszweckeverzeichnis:
Das Geschäftszweckeverzeichnis enthält eine Übersicht über alle Geschäftszwecke und Ziele, zu deren Erfüllung die Kenntnis personenbezogener Daten erforderlich ist. Interessant wäre hier die Verwaltung in einer kleinen Datei mit einem Verweis auf die entsprechende Datenbank.

11. Empfängerverzeichnis:
Des weiteren führt der Datenschutzbeauftragte eine Übersicht über regelmäßige Empfänger personenbezogener Daten. Da alle in einem Netzwerk angemeldeten Benutzer ohnehin mit Ihren jeweiligen Zugriffsrechten im Rahmen der Datenschutzmaßnahmen 1-5 verwaltet werden sollten, könnte man die Benutzerdatei ebenfalls mit einem Schalter versehen. Sofern es dem Beauftragten möglich ist, Datensätze einzufügen, die keine Benutzer sondern einfach nur Datenempfänger sind, wäre eine Liste sofort druckbar. Sofern man eine Protokolldatei führt, in der man den Aufruf von Programmen und Berichten speichert und darüber hinaus die Benutzung von personenbezogenen Dateien vermerkt, könnte man zusätzlich eine Liste von Programm- und Berichts aufrufen mit personenbezogenen Daten zu den jeweiligen Empfänger ausdrucken.

12. Anlagenverzeichnis:
Die vierte und letzte zu führende Übersicht befasst sich mit der Art der eingesetzten Datenverarbeitungsanlagen. Der Datenschutzbeauftragte muss hier zusammenstellen, wo Daten gespeichert oder abgerufen werden können. Das Anlagenverzeichnis lässt sich von der Programmierung her von verschiedenen Seiten angehen. Sofern man die Arbeitsstationen zwecks Konfiguration von lokalen Druckern, Datenpfaden usw. in einer Datenbank verwaltet, kann diese als Grundlage einer Liste dienen. Andernfalls kann man eine Anlagenbuchhaltung mit den entsprechenden Markier- und Memofeldern ergänzen und von dort aus eine Aufstellung drucken lassen.

C.4 Zusammenfassung

Zusammenfassend lässt sich sagen, dass dem Datenschutzbeauftragten sehr viele Verantwortlichkeiten aufgeladen werden, ohne dass damit wirkliche Rechte wie zum Beispiel bei einem Betriebsrat einhergehen würden. Einzig die Weisungsfreiheit und seine beratende Funktion wurden im Gesetz festgehalten. Seitens der Programmierung könnte man unterstützend eingreifen und zumindest den Bereich der Listenführung stark vereinfachen. Bei der Beschreibung der Datenschutzmaßnahmen und den Aufgaben des Datenschutzbeauftragten wurde jeweils kurz auf die Möglichkeiten des Programmierers eingegangen, die Vorgaben des Bundesdatenschutzgesetzes schon bei der Programmkonzeption zu berücksichtigen. Nachfolgend nochmals eine kurze Aufstellung über die wesentlichen Punkte [BDG08]:

- Benutzer- und Zugriffskontrolle
- Eingabekontrolle
- Abgangs-, Transport-, Übermittlungskontrolle
- Verpflichtungserklärung
- Anwendungskontrolle
- Listenerstellung

D Mögliche Alternative zur Löschung von nicht mehr gebrauchten Daten

In diesem Unterkapitel werden nun zwei mögliche Alternativen zur Löschung von alten und nicht mehr benötigten Daten gezeigt. Hierbei wird zuerst auf das Hierarchisches Speichermanagement und danach auf das Informationslebenszyklusmanagement eingegangen.

D.1 Hierarchisches Speichermanagement

Unter einem Hierarchischen Speichermanagement (kurz HSM) versteht man eine Systemkomponente, die Dateien bzw. Daten, auf welche über längere Zeit nicht zugegriffen wurde, auf ein Speichermedium auslagert, welches einer niedrigeren Speicherhierarchiestufe zuzuordnen ist, also preiswerter ist, aber den Nachteil einer größeren Zugriffszeit besitzt. Ein solches Speichermedium kann ein Magnetband oder ein optisches Speichermedium sein. Versucht ein Benutzer, auf eine solche Datei zu zugreifen, so werden die Daten von dem langsamen Speichermedium wieder auf das schnellere Speichermedium kopiert, dieser Prozess wird als Paging bezeichnet. Vorteil dieses Verfahrens ist die Kostenersparnis bei der Bereitstellung von Massenspeichern. Hierarchisches Speichermanagement wird häufig in Kombination mit elektronischen Archiv-, Dokumentenmanagement-,
Enterprise - Content-Management- und Datensicherungs-Systemen eingesetzt.
Eine Weiterentwicklung des hierarchischen Speichermanagements ist Information Lifecycle Management (ILM), in dem die Information entsprechend ihrem Wert nach einem Regelwerk auf das jeweils günstigste Speichermedium verschoben wird [HSM08].

D.1.1 HSM unter z/OS

Dieses Verfahren wird insbesondere im Großrechner-Bereich eingesetzt. Zum Beispiel existiert unter z/OS eine Systemkomponente, die als Hierarchical Storage Manager, kurz HSM, bezeichnet wird, und für ein hierarchisches Speichermanagement zuständig ist. HSM ermöglicht die Definition von bis zu zwei Migrationsstufen für die Auslagerung der Dateien bzw. Daten.

In der ersten Migrationsstufe werden die Daten komprimiert auf einer preiswerten Festplatte gespeichert. In der zweiten Stufe werden die Daten auf ein Magnetband oder einen Virtual Tape Server ausgelagert. Außerdem kann die Zeitspanne, die bis zur Auslagerung einer Datei verstreichen muss, vom Systemprogrammierer über diverse Parameter (Storage class, Name, Größe etc) festgelegt werden [HSM08].

D.1.2 HSM unter Unix

Für das Unix-Derivat AIX von IBM ist HSM als Tivoli Storage Manager for Space Management für das JFS2- und das GPFS-Dateisystem verfügbar. Dieses Produkt gibt es auch für Linux, Solaris und HP-UX und ist eng mit dem
Backup- / Archive- Produkt der selben TSM-Produktfamilie integriert.
Für Solaris von Sun Microsystems ist die HSM-Implementierung SAM-FS verfügbar, diese stellt ein HSM-System auf Dateisystem-Basis bereit, welche es erlaubt, bis zu vier Medien-Kopien anzulegen, um die Gefahr eines Verlustes von Daten durch defekte Medien zu minimieren. Für Linux gibt es von SGI das HSM DMF, das kommerziell vertrieben wird [HSM08].

D.1.3 Vorteile von HSM

Ein wesentlicher Vorteil von HSM-Systemen im Vergleich zu konventionellen
Backup - Systemen liegt darin, dass die Zeit, die für eine Rücksicherung benötigt wird, extrem verkürzt werden kann. Es wird lediglich die Verwaltungsinformation des Dateisystems zurück gespielt, entweder in Form einer Inode-Sicherung oder einer Datenbank. Das Dateisystem kann wieder für Anwender-Zugriffe freigegeben werden. Die benötigten Nutzdaten werden dann bei Zugriff auf die Datei auf den Primärspeicher zurück kopiert.
Ein weiterer Vorteil von HSM besteht in der Kostenersparnis durch die Nutzung preiswerterer Massenspeicher zur Aufbewahrung von Datenbeständen, die über einen längeren Zeitraum nicht benötigt werden. Der Anwender muss hierbei fast keinen Komfortverlust hinnehmen, da die Datenbestände automatisch rückgesichert werden, wenn auf diese zugegriffen wird [HSM08].

Implementierungen:

- IBM Tivoli Storage Manager for Space Management (HSM)
- OpenStore for File Servers by Intercope (resold as IBM TSM for HSM for Windows)
- GRAU ArchiveManager (GAM) by Grau Data Storage for Windows/Linux
- EMC DiskXtender, formerly Legato DiskXtender, formerly OTG DiskXtender
- Caminosoft Corporation, Caminosoft Managed Server Netware/Windows/Linux
- Moonwalk (Columbia and Eagle), for NetWare/Windows/Linux
- SAM-QFS
- CommVault QiNetix DataMigrator
- SGI DMF (Data Migration Facility)
- Symantec VERITAS Enterprise Vault (KVS acquisition and Veritas acquisition)
- QStar
- Quantum StorNext Storage Manager
- Compellent Data Progression Automated Tiered Storage

D.2 Informationslebenszyklusmanagement

Informationslebenszyklusmanagement (ILM, englisch information lifecycle management) sind Strategien, Methoden und Anwendungen um Information automatisiert entsprechend ihrem Wert und ihrer Nutzung optimal auf dem jeweils kostengünstigsten Speichermedium bereitzustellen, zu erschließen und langfristig sicher aufzubewahren. Der zusammengesetzte Begriff stammt aus den USA und wurde von verschiedenen Speichersystemanbietern 2003 als Marketing-Slogan auserkoren. Verschiedentlich und in anderer Bedeutung wurde der Begriff bereits in den 90er Jahren benutzt. Der Begriff setzt aus den Komponenten „Information“, „Lifecycle“ (engl. für Lebenszyklus; zusammengeführt aus life cycle oder life-cycle, um ein dreibuchstabiges Akronym bilden zu können) und „Management“ im Sinne der Verwaltung, Handhabung und Kontrolle von Informationen in einem Informationssystem [ILM108,ILM208].

D.2.1 Definition

ILM beschreibt eine Speicherstrategie. Ziel dieser Strategie ist die Speicherung von Informationen entsprechend ihrem Wert auf dem jeweils günstigsten Speichermedium einschließlich der Regeln und Prozesse, wie Information auf die geeigneten Speichermedien verschoben wird. Die Steuerungsmechanismen der Verwaltung und der Speicherung orientieren sich an Wichtigkeit, Wertigkeit und Kosten der elektronischen Information. Hierfür wird eine Klassifizierung der Daten, Quellen und Speichersysteme vorgenommen, die innerhalb einer Speicherhierarchie die automatisierte Bereitstellung erlauben.
Der Herstellerverband für Speichersystemlösungen SNIA definiert ILM wie folgt :

ILM Vision :

“A new set of management practices based on aligning the business value of information to the most appropriate and cost effective infrastructure“ [ILM108,ILM208].

ILM Definition :

“Information Lifecycle Management is compromised of the policies, processes, practices, and tools used to align the business value of information with the most appropriate and cost effective IT infrastructure from the time information is conceived through its final disposition. Information is aligned with business processes through management policies and service levels associated with applications, metadata, information, and data“[ILM108,ILM208].

D.2.2 Funktionalität von ILM - Lösungen

Um diese Anwenderanforderungen erfüllen zu können, benötigen ILM-Lösungen eine Vielzahl von Funktionen. Diese Funktionen sind in einzelnen Komponenten zusammengefasst. Die Komponenten wiederum bilden ein geschlossenes Rahmenwerk, um alle Anforderungen an ILM integrativ abdecken zu können.

Zu den wichtigsten Komponenten von ILM gehören [ILM108,ILM208] :

- Erfassung : Subsysteme und Software zur Erfassung, Aufbereitung, Verarbeitung, Indexierung und Ordnung unterschiedlichster Formen von Informationen
- Verwaltung von Dokumenten, Content und Media Assets : Software zur kontrollierten Erstellung, Verwaltung, Publikation und Verteilung von Information
- Speicherung : Subsysteme zur optimierten Speicherung beliebiger Typen von Information mit Unterstützung unterschiedlichster Hardware, softwaregestützter Verdrängungsstrategien, verteilter Umgebungen und Nutzbarkeit durch alle Anwendungen in einem System
- Zugriff und Verwaltung : datenbankgestützte Registratur-, Dokumenten-, Metadaten- und Indexverwaltung für den geordneten, schnellen Zugriff auf die gespeicherte Information
- Prozessunterstützung : Workflow- und Business-Process-Management-Software zur Bereitstellung, Zusammenführung und Kontrolle von Information und zur Steuerung der Speicherprozesse
- Langzeitarchivierung : Subsysteme zur unveränderbaren, langzeitigen elektronischen Archivierung entsprechend rechtlichen und regulativen Anforderungen

Durch das Zusammenspiel der verschiedenen Komponenten wird der gesamte Lebenszyklus der Information von seiner Entstehung bis zur Aussonderung unterstützt.

E Existierende Systeme

In diesem Unterkapitel werden nun kurz zwei existierende Systeme im Bericht der verteilten Revisionsverwaltung gezeigt. Die hierbei gezeigten Systeme sind einerseits das Bazaar-System und andererseits das Git-System.

E.1 Bazaar

Bazaar (vormals Bazaar-NG) ist ein verteiltes Versionskontrollsystem, dessen Entwicklung vor allem durch Canonical Ltd. finanziert und vorangetrieben wird. Hauptziel ist, die Entwicklung von Open-Source-Projekten zu erleichtern. Die bekanntesten Projekte, welche Bazaar benutzen, sind zur Zeit Ubuntu (Stand Januar 2008) und MySQL (Stand Juni 2008). Das Entwicklerteam von Bazaar legt das Hauptaugenmerk auf einfache Bedienung, Zuverlässigkeit und Flexibilität. Die Verwaltung von Entwicklungszweigen (Branching und Merging) gestaltet sich sehr einfach und kann mit einem sehr kleinen Satz von Kommandos bewältigt werden. Bazaar kann von einem einzelnen Entwickler mit mehreren Entwicklerzweigen auf einem lokalen System genauso benutzt werden wie von Teams, die über ein Rechnernetz an einem Projekt zusammenarbeiten. Bazaar ist in Python geschrieben und als fertige Pakete für alle gängigen Linux-Distributionen, Mac OS X und Windows verfügbar. Es gehört zum GNU-Projekt [Baz08].

E.1.1 Eigenschaften von Bazaar

Bazaar ist auf einfache Benutzbarkeit ausgelegt. Die Kommandos ähneln denen von CVS und Subversion, und es ist sehr leicht, ein neues Projekt ohne Server zu starten und zu betreuen. Im Gegensatz zu rein verteilten Versionskontrollprogrammen, unterstützt Bazaar sowohl den Ansatz mit als auch den ohne zentralen Server. Es ist darüber hinaus möglich, beide Ansätze gleichzeitig bei einem Projekt anzuwenden. Die Website Launchpad bietet einen kostenlosen Hosting-Dienst für Bazaar-Projekte im Bereich Open Source an.

Bazaar kann mit einigen anderen Versionskontrollprogrammen zusammenarbeiten. Dies ermöglicht Benutzern, Entwicklerzweige aus diesen anderen Systemen transparent als Bazaar-Zweig zu nutzen. Bazaar unterstützt auf diese Weise rudimentär Subversion. Unterstützung für Mercurial und Git befinden sich in den Anfängen. Bazaar unterstützt Dateinamen, die Zeichen aus dem kompletten Unicode enthalten. Unicode-Zeichen sind ebenso erlaubt in zum Beispiel Commit-Beschreibungen und Entwickler-Namen. Vor allem weil es in einer interpretierten Sprache geschrieben ist, ist Bazaar merklich langsamer als Git. Allerdings ist die Performance vergleichbar zu anderen Versionskontrollprogrammen [Baz08].

E.1.2 Geschichte von Bazaar

Am 1. Februar 2005 erklärte Martin Pool, dass er von Cannonical Ltd. damit beauftragt worden war, ein neues Versionskontrollprogramm zu erstellen, das Open-Source-Entwickler gerne benutzen werden. Martin Pool hatte bis dato mehrere Versionskontrollprogramme in Vorträgen und in seinem Weblog beschrieben und kritisch untersucht. Im März 2005 wurde eine öffentliche Website und Mailingliste eingerichtet. Das Projekt war eine völlige Neuimplementierung, die versuchte, auf den Erfahrungen mit anderen in der Entwicklung befindlichen Versionskontrollprogrammen aufzubauen. Seit Februar 2008 ist Bazaar ein Teil des GNU-Projektes [Baz08].

E.1.3 Vorgängerprojekte

Obwohl es keine gemeinsame Codebasis gibt, wurzelt Bazaar im GNU-arch-Protokoll und -Projekt. Am 29. Oktober 2004 kündigte Robert Collins einen Fork von Arch namens Baz an (der ursprünglich selbst Bazaar genannt wurde). Der Name “Baz“ war angelehnt an den Kommandozeilenaufruf “baz“. Cannonical Ltd. unterstützte die Entwicklung von Baz bis Mitte 2005, als auf der Baz-Webseite ein Parallel-Projekt namens Bazaar-NG angekündigt wurde, welches neu in Python erstellt würde und welches Baz ablösen würde. Dieses wurde dann später in Bazaar umbenannt. Baz wird nicht mehr weiterentwickelt. Im Oktober 2005 gab es die letzte veröffentlichte Version von Baz (1.4.3).

Gleichzeitig gab Robert Collins bekannt, dass er nicht die Ressourcen habe, Baz 1.5 fertigzustellen, und dass Teile von Baz in Arch zurück verschmolzen worden waren. Canonical betrachtet Baz als veraltet [Baz08].

E.2 GIT

GIT im Gegensatz zu herkömmlichen Versionsverwaltungen ein dezentrales System. Das heißt nicht, dass es kein gemeinsames, zentrales Repository geben kann. Jeder Benutzer hat sein eigenes Repository, dass er vorher von jemand anderem geklont hat (entspricht ungefähr einem CVS checkout). In diesem Repository sind sämtliche Branches und die komplette History des Projektes. Somit braucht man nach einem git-clone-Aufruf keinen Netzwerkzugriff, um zum Beispiel zu einer älteren Revision zurückzukehren. Daher die Bezeichnung dezentrales System. Sollte man ein gemeinsames Repository benutzen wollen, so ist das meistens einfach nur das eigene Repository eines Benutzers. Im Falle des Linux-Kernels, kann man sich zum Beispiel von Linux Torvalds Repository bedienen. Man wählt im Projekt einfach eine Person aus, der man vertraut und überlässt es ihr, ein Repository zu pflegen, welches für die Öffentlichkeit zugänglich ist [GIT08].

E.2.1 Vorteile von GIT

An dieser Stelle werden nun die Vorteile von GIT gegenüber zentralen Versionskontrollen gezeigt [GIT08].

Vorteile :

- Kein/Kaum Netzwerkverkehr, daher sehr schnell bei Standard-Kommandos wie diff.
- Jeder kann eigene Branches erstellen (vom Hauptteil abzweigen) und im Vergleich zu CVS / SVN sehr einfach mergen
- Kaum Administrationsaufwand von öffentlichen Repositories. Ein SSH-Zugang reicht meist schon aus.

- Man authentifiziert sich bei GIT und kann einen gemeinsamen SSH / SFTP-Account verwenden.
- Keine lästigen CVS-Verzeichnisse in jedem Unterverzeichnis (es gibt nur noch ein .git-Verzeichnis im Root-Verzeichnis des Projektes).

E.2.2 Nachteile von GIT

An dieser Stelle werden nun die Nachteile von GIT gegenüber zentralen Versionskontrollen gezeigt [GIT08].

Nachteile :

- Man muss ein neues Programm lernen
- Nur „msysgit“ als Grafische Oberfläche

E.2.3 Unterschied zwischen GIT und Bazaar

In diesem letzten Unterkapitel werden nun kurz die Unterschiede zwischen GIT und Bazaar beschrieben.
Bazaar ist so etwas ähnliches wie GIT. Allerdings liegen bei Bazaar die Branches nicht alle in einem Verzeichnis, sondern in gesonderten Verzeichnissen ähnlich wie bei SVN. Außerdem ist Bazaar durch die Programmiersprache Python sehr plattformunabhängig und integriert sich auch in Windows sehr gut [GIT08].