Anwendungsorientiertes Archivieren in Datenbanksystemen – vertieft am Beispiel von EXPRESS und SDAI

Axel Herbst*

IBM Wissenschaftliches Zentrum
Vangerowstr. 18
69115 Heidelberg
E-mail: aherbst @ vnet.ibm.com

Zusammenfassung. Zunehmend müssen Daten aus wirtschaftlichen und juristischen Gründen langfristig, kostengünstig und wiederverwendbar aufbewahrt werden. Während die elektronische Archivierung von Dokumenten eine etablierte Technik ist, wurde ein adäquater Datenbankservice bislang kaum untersucht. Auch in Datenbanksystemen belasten die Daten, mit denen längere Zeit nicht gearbeitet wird und die von "aktiven" Daten unterschieden werden sollen, unnötig das Sekundärspeichersystem. Tertiärspeicher werden nur unzureichend unterstützt. Dieser Beitrag stellt einen auf Datenbanksysteme bezogenen Archivierungsansatz vor, bei dem die zu archivierenden Daten auf der konzeptuellen Ebene der Datenbankanwendung (hier exemplarisch STEP/EXPRESS) bestimmt werden. Im Kontext der ISO-Norm EXPRESS bedeutet anwendungsorientiertes Archivieren eine Erweiterung der Zugriffsschnittstelle SDAI (Standard Data Access Interface). Aus unserer Spezifikation von SDAI–integriertem Archivieren leiten wir Anforderungen an Datenbanksysteme ab und zeigen verschiedene Implementierungsansätze auf.

1 Archivierung als Datenbankservice

Der Ursprung des Wortes *Archiv* (griechisch "archeion" = Rathaus) beeinflußt noch immer das Verständnis vom Archivieren als eine geordnete Aufnahme von Dokumenten und deren sichere Aufbewahrung auf Medien wie z.B. Papier, Microfilm oder optischen Platten. Neben dem traditionellen reinen Dokumentationscharakter eines Archivs tritt heute die *Wiederverwendbarkeit* der *kostengünstig* archivierten Daten immer mehr in den Vordergrund. Dies hat betriebswirtschaftliche Gründe [VS91] und kommt Forderungen des Gesetzgebers nach [Sam93]. Aktueller Forschungsgegenstand sind u.a. multimediale Archive, wobei hier die Verschiedenartigkeit der elektronisch zu archivierenden Dokumente (Video, Ton oder Bild) gemeint ist [RDMP94]. Beispiele für Dokumente im Konstruktionsumfeld sind technische Zeichnungen und Stücklisten. Aber auch sog. Geometriemodelle in Form von CAD–systemspezifischen Dateien oder in neutralen, standardisierten Austauschformaten zählen inzwischen dazu [HM94].

* auch: Universität Kaiserslautern, FB Informatik, AG Datenverwaltungssysteme

Mit dem Übergang zur Produktmodelltechnologie [GAP93] verschwimmt in diesem Anwendungsbereich der Dokument–Begriff: Produktdaten sind nicht länger nur inhaltstransparente Dateien, die ausschließlich über zusätzliche Beschreibungsattribute aufgefunden werden können. Produktdaten werden modelliert und feingranular *in Datenbanksystemen* gespeichert. Geblieben ist die Forderung nach ihrer Archivierung. Das Auslösen des Archivierungsvorgangs ist dabei nicht vom Alter der Daten abhängig, sondern wird durch andere "Reife"–Kriterien (z.B. Zeitpunkt der Freigabe) bestimmt. Eine derartige Archivierungsfunktionalität wurde erstmals in [EL90] von einer *Engineering Database* gefordert. Durch die wachsende Datenflut gewinnt die Archivierungsproblematik über technische Datenbanken hinaus an Brisanz. Neuartige Speichermedien motivieren zusätzlich, **anwendungsorientiertes Archivieren** als allgemeinen Datenbankservice mit folgender Semantik zu untersuchen:

– *Benutzerveranlassung*
  Das Datenbanksystem bietet dem Benutzer (ein Anwendungsprogrammierer oder auch Datenbankadministrator) Funktionen an, mit denen er bewußt die Archivierung auslöst. Nur aus den Anwendungen heraus ist bekannt, *welche Daten* langfristig aufbewahrt werden sollen. Vom Zeitpunkt der Archivierung an unterliegen diese Daten in aller Regel einem wesentlich anderen Zugriffsprofil (seltene, zeitunkritische, überwiegend lesende Zugriffe).
– *Abstraktionsebene Datenmodell*
  Datenbankanwendungen identifizieren die zu archivierenden Daten in der Terminologie und dem Granulat des jeweiligen Datenmodells. Im relationalen Fall sind dies etwa Tupel, Tabellen oder Sichten. Anwendungsorientiertes Archivieren kann aber auch auf Instanzen eines semantischen Datenmodells, das beispielsweise ein Konzept "Entity" definiert, übertragen werden.
– *Datenauslagerung*
  Die "weniger interessanten" Daten werden auf billigere, aber dafür langsamere Massenspeicher ausgelagert, um Platz für wichtigere, aktive Daten zu schaffen und die Anfrageauswertung nicht unnötig zu belasten. Angesichts Größenordnungen von mehreren 100 Terabyte werden verschiedene neue Tertiärspeicher – Speicher "unterhalb" von Magnetplatten in der Speicherhierarchie (z.B. optische Plattenroboter) – unumgänglich [KPCD+92].

Abstrahiert man von dem originären Ansatz in Verbindung mit dem Datenbankprototypen "PRODAT", stimmen wir in diesen Grundzügen mit [EL90] überein. Anders als in [EL90] nutzen und erweitern wir systemunabhängige Schnittstellen, die derzeit weltweit standardisiert werden und analysieren, inwieweit heutige Datenbanksysteme anwendungsorientiertes Archivieren unterstützen. Auf weitere Unterschiede verweisen wir im Verlauf des Beitrags. Unser Konzept grenzt sich aber schon jetzt deutlich von dem ab, was Datenbanksysteme üblicherweise zur Archivierung anbieten: ein Zusatzwerkzeug, das physische Datenbankgranulate (Seiten, Segmente, Log–Dateien) auf ein nicht weiter verwaltetes Speichermedium kopiert, um bei Ausfall des Sekundärspeichers eine Wiederherstellung der Datenbank (Recovery) zu ermöglichen. Auch in [MN93] gehen die Autoren von dieser *Backup–Semantik* aus.

Der Beitrag ist im weiteren wie folgt aufgebaut: Abschnitt 2 definiert unsere Abstraktionsebene für anwendungsorientiertes Archivieren, spezifiziert Archivierungsfunktionen und leitet Anforderungen an Datenbanksysteme ab. Abschnitt 3 klassifiziert Implementierungsansätze und untersucht, inwieweit relationale und objektorientierte Systeme unsere Spezifikation unterstützen. Wir illustrieren einen objektorientierten Ansatz an Hand unseres Prototyps in Abschnitt 4. Abschnitt 5 zieht Schlußfolgerungen und gibt einen Ausblick.

2 Anwendungsorientiertes Archivieren am Beispiel einer EXPRESS/SDAI–Datenbank

Wir wählen Teile von STEP (Standard for the Exchange of Product Model Data [Owe93]) aus drei Gründen als Kontext für anwendungsorientiertes Archivieren: Erstens sehen wir einen Bedarf für derartige Archivierungskonzepte in den aufkommenden STEP–Datenbanken [LRW93]. Zweitens sind die Modellierungssprache EXPRESS und die Zugriffsschnittstelle SDAI (Standard Data Access Interface) unabhängig vom primären Anwendungsbereich Produktdatenverwaltung [DHSV94, Her94, Wil94]. Schließlich versprechen die Standardisierung von EXPRESS und SDAI durch die ISO und die zunehmende weltweite Akzeptanz von STEP in der Industrie eine langfristigere Stabilität dieser beiden Teilnormen.

2.1 EXPRESS und SDAI aus Datenbanksicht

In [MSRD91] wurde EXPRESS unter dem Gesichtspunkt des Datenbankentwurfs analysiert[1]. Auch wenn die endgültige Veröffentlichung der Sprache als internationale Norm [ISO95] in einigen Punkten von der Darstellung in [MSRD91] abweicht, halten wir diese Analyse im wesentlichen für aktuell: Ein EXPRESS–Schema als Datenbankschema aufzufassen (und EXPRESS als DDL) bringt Probleme mit sich, die u.a. im komplexen Typsystem der Sprache liegen. Üblicherweise wählt man eine Abbildung von EXPRESS auf die DDL des zugrundeliegenden Datenbanksystems. Je nach Datenmodell ist diese Abbildung mehr oder weniger aufwendig (z.B. bei dem *SELECT*–Typ in EXPRESS, der ein Vereinigungstyp über Entities ist oder bei den vielfältigen Vererbungsbeziehungen, die zwischen Entities in Form von Super- und Subtypen spezifizierbar sind).

In [MSRD91] werden die globalen Regeln in EXPRESS noch als Datenbank–Integritätsbedingungen interpretiert. Inzwischen beantwortet die Schnittstellenspezifikation SDAI [ISO94] die Frage, wann diese Regeln zu überprüfen sind: bei expliziten Aufrufen aus Programmen heraus, die über SDAI auf Daten zugreifen, die durch das EXPRESS–Schema beschrieben werden.

[1] Wir verzichten daher auf eine detaillierte Vorstellung der Sprache. Eine umfassende Einführung findet man in [SW94]. Die Beispiele in Abschnitt 2.2 sollten auch dem Nicht-EXPRESS-Kenner einen ausreichenden Eindruck vermitteln. Vorerst genügt es festzuhalten, daß instanziierbare Entities das zentrale Modellierungskonzept sind. Entities haben Attribute (einfache und Entity–wertige Datentypen), erlauben Vererbung und werden syntaktisch zu EXPRESS-Schemata zusammengefaßt.

Neben den Operationen zur Regelauswertung enthält die SDAI–Spezifikation z.B. die folgenden, weitgehend selbsterklärenden Operationen für den Zugriff auf Instanzen von EXPRESS–Entities (bzgl. *model* siehe Abschnitt 2.2.):

```
CREATE_INSTANCE       <entity>, <model>
DELETE_INSTANCE       <instance>
PUT_ATTRIBUTE         <instance>, <attribute>, <value>
GET_ATTRIBUTE         <instance>, <attribute>
UNSET_ATTRIBUTE_VALUE <instance>, <attribute>
GET_INSTANCE_TYPE     <instance>
```

Transaktionsklammern und einfache Anfragen (Selektion von Instanzen eines Entitys über einfache Prädikate) wurden erst vor kurzem in die Spezifikation aufgenommen. Dies liegt daran, daß SDAI nicht ursprünglich als Datenbankschnittstelle, sondern für Hauptspeicherzugriffe ohne Berücksichtigung eines Mehrbenutzerbetriebs konzipiert wurde.

Die verschiedenen sog. Language–Bindings der Spezifikation und die aktuell definierten Konformitätsklassen für SDAI–Implementierungen widersprechen nicht dem Prinzip, SDAI–Anwendungsprogramme weitgehend von dem zugrundeliegenden Speichersystem (Hauptspeicher, Dateisystem, Datenbanksystem) zu isolieren. SDAI ist damit zwar keine DML im strengen Sinne, aber doch *die* in der Standardisierung befindliche Schnittstelle, über die EXPRESS–Daten gelesen, modifiziert und unserer Meinung nach auch **archiviert** werden sollen.

2.2 Integration von Archivierungsfunktionalität in SDAI

Nachdem wir gerade die "Stelle SDAI" lokalisiert haben, an der Archivierungsdienste angeboten werden sollen, wollen wir diese zusätzliche Funktionalität genauer spezifizieren. Dabei gehen wir von der (konzeptionellen) Sicht des SDAI–Benutzers aus. Konkret ist dies die Person, die SDAI–Anwendungen entwirft und programmiert. Wir berücksichtigen gleichzeitig erste Implementierungsaspekte von *SDAI–integriertem Archivieren.*

SDAI–Metadatendefinitionen als Archivierungsgranulate. Das Abstraktionsniveau des SDAI-Benutzers, EXPRESS, spiegelt sich in Metadatendefinitionen wider, die ebenfalls Bestandteil von [ISO94] sind. Dazu zählen insbesondere das *SDAI Dictionary Schema* und das *Session Schema*, die wiederum in EXPRESS geschrieben sind. Beide Schemata sind ein geeigneter Anknüpfungspunkt für die Bestimmung der zu archivierenden Daten – interessanterweise auch aus Sicht des SDAI–Entwicklers: Um normkonform zu sein, muß die Funktionalität beider Schemata von einer SDAI–Implementierung bereitgestellt werden. Folglich entsprechen diese EXPRESS–Definitionen mehr oder weniger direkt Datenstrukturen der SDAI–Implementierungssprache.

Aus dem **Dictionary Schema** kann man z.B. ablesen, daß *schema*, *entity* und *attribute* anwendungsorientierte Dateneinheiten sind. Praktische Bedeutung als Granulat zum Archivieren besitzen jedoch nur die ersten beiden, da niemand langfristig auf eine Menge einzelner Attribut–Werte zugreift. Dagegen ist es vorstellbar, daß nur gewisse "archivierungswürdige" Entities aus einem Anwendungsschema ausgewählt werden sollen. Diese Auswahl ist eine Projektion auf der Ebene des Anwendungsschemas und entspricht der Selektion von Instanzen von Meta–Entities. Hierzu betrachte man den folgenden Ausschnitt aus dem Dictionary Schema, wo das (Meta–) Entity *entity_definition* vom (Meta–) Entity *schema_definition* referenziert wird:

```
ENTITY schema_definition;
 name: STRING;
 entities: SET OF entity_definition;
 global_rules: SET OF global_rule;
 ...
END_ENTITY;
```

```
ENTITY entity_definition;
 attributes: LIST OF attribute;
 ...
 INVERSE parent_schema:
  schema_definition FOR entities;
END_ENTITY;
```

Orthogonal zu der Auswahlmöglichkeit auf Typ– bzw. Schema–Ebene kann man die zu archivierenden Instanzenmengen gemäß den Definitionen aus dem **Session Schema** festlegen. In diesem Schema ist auch dokumentiert, wie SDAI die Anwendungen von der zugrundeliegenden Speichertechnologie isoliert: Ein sog. *SDAI Repository* ist der abstrakte Speicherort für sämtliche Daten, die von einer SDAI–Implementierung verwaltet werden.

```
ENTITY sdai_repository;
 name: STRING;
 contents: sdai_repository_contents;
 schemas: SET OF schema_definition;
 ...
END_ENTITY;
```

```
ENTITY sdai_repository_contents;
 models: SET OF sdai_model;
 INVERSE repos:
  sdai_repository FOR contents;
END_ENTITY;
```

```
ENTITY sdai_model;
 underlying_schema: schema_definition;
 name: STRING;
 contents: sdai_model_contents;
 repos: sdai_repository;
 ...
END_ENTITY;
```

```
ENTITY schema_instance;
 name: STRING;
 contents: SET OF sdai_model;
 base_schema: schema_definition;
 ...
END_ENTITY;
```

Hierbei steht *sdai_model_contents* für die Menge aller Instanzen eines *SDAI model*s. Dies ist eine willkürliche Zusammenfassung von Instanzen, die zu beliebigen Entities aus genau einem EXPRESS–Schema gehören. Jede Instanz ist in genau einem *model* enthalten. In einem Repository werden in der Regel mehrere Schemata und die aktuell existierenden *model*s verwaltet.

Das Entity *schema_instance*[2] hat einen sehr irreführenden Namen. Es steht für eine Menge von *model*s, die den Gültigkeitsbereich für globale EXPRESS–Regeln und Referenzen zwischen (Entity–) Instanzen definiert. Letztere dürfen einander nur referenzieren, wenn sie in *model*s enthalten sind, die in der gleichen *schema_instance* liegen.

[2] Dieses neue Konstrukt wurde in SDAI aufgenommen als man feststellte, daß der Gültigkeitsbereich *repository* "zu groß", aber *model* "zu klein" für Datenbankanwendungen ist.

Wir gehen davon aus, daß SDAI–Benutzer den Empfehlungen in [ISO94] folgen und semantisch zusammenhängende Daten in *model* und *schema_instance* gruppieren. Folglich bilden so zusammengefaßte Instanzen auch sinnvolle Mengen von Daten, die **als Ganzes** archiviert werden. Wenn Instanzen bestimmter Entities stets ausgeblendet werden sollen (etwa weil sie nicht archiviert werden brauchen), kann dies zusätzlich spezifiziert werden.

Neue SDAI Operationen. Mindestents zwei Operationen müssen in SDAI eingeführt werden, um anwendungsorientiertes Archivieren zu ermöglichen. Den Zugriff auf archivierte Daten klären wir anschließend.

1. **SELECT**
   Diese Operation bestimmt die zu archivierenden Daten. Im einfachsten Fall wird der Aufruf mit einem Verweis auf ein Granulat aus dem Session Schema parametrisiert. Zusätzliche Einschränkungen durch Vorgaben aus dem Dictionary Schema erhöhen zwar die Auswahlmächtigkeit, verlangen aber weitere Klarstellungen: Wie sollen z.B. offene Referenzen zwischen zu archivierenden und aktiven Daten behandelt werden, die entstehen können, wenn bestimmte Entities von der Archivierung ausgenommen werden? Das Auswahlvermögen der SELECT–Operation wäre noch höher, wenn Prädikate als Parameter zugelassen werden, wie sie auch in SDAI–Queries Verwendung finden. Auch eine deskriptive, EXPRESS–basierte Sichtbeschreibungssprache ist auf den ersten Blick eine wünschenswerte Erweiterung. Wir sehen jedoch keinen Bedarf für diese Funktionalität, da – wie bereits angesprochen – Daten langfristig nur grobgranular, also konkret in kompletten SDAI *schema_instance*s oder *model*s wiederverwendbar sind.
   Bereits das Archivieren ausgewählter *model*s ist nicht trivial: Neben den potentiellen offenen Referenzen kann im allgemeinen nicht davon ausgegangen werden, daß zuvor geltende EXPRESS–Integritätsbedingungen in Form globaler Regeln nach dem Archivieren immer noch erfüllt sind, da ihr Gültigkeitsbereich eine *schema_instance* ist. In diesem Fall kommt der SELECT–Operation die Aufgabe zu, die Korrektheit der Auswahl zu überprüfen. Dies führt zu einem iterativen Auswahlprozeß in Form einer Folge von SELECT–Aufrufen: Die Archivierungsanwendung wertet die Rückmeldungen des Archivierungssubsystems aus (Zurückweisungen von ausgewählten Granulaten oder Vorgaben zur Archivierung weiterer Daten, um einen referentiellen Abschluß zu erreichen) und setzt ggf. ein erneutes SELECT ab. Zusammenfassend halten wir fest, daß eine Folge von SELECT–Aufrufen die endgültige Menge der gewünschten und tatsächlich archivierbaren Daten festlegt. Als Granulate wählen wir vorzugsweise *schema_instance* und *model*.
2. **ARCHIVE**
   Entsprechend unserer Motivation löst ein SDAI–Anwendungsprogramm mit dieser Operation das Archivieren der zuletzt selektierten Daten aus. Ein Parameter identifiziert das zu verwendende Archiv. Dieser Vorgang wird aus Sicht der Anwendung synchron ausgeführt, so daß die betroffenen Daten im

Anschluß an den ARCHIVE–Aufruf im aktiven Repository nicht mehr sichtbar sind. Ihre tatsächliche Übertragung in das physische Archiv, d.h. das Auslagern auf Tertiärspeicher, sollte bei großen Datenmengen asynchron geschehen[3]. Auch bei weniger Daten ist es vorteilhaft, wenn man die Datenauslagerung auf Zeiten mit geringerer Systemlast verschieben kann.
Als Konsequenz aus der Trennung von *logischem Archivieren* und *asynchroner Datenmigration* kann sich eine "Zugriffslücke" derart ergeben, daß archivierte Daten zwar nicht mehr aktiv zugreifbar aber auch noch nicht in das Archiv migriert sind. Wenn eine SDAI–Implementierung vorliegt, die sämtliche Zugriffe auf Instanzen über logische Zugriffspfade abwickelt, kann dieses Problem in Analogie zu dem Vorschlag für logisches Archivieren von Dokumenten in [ZPD90] behandelt werden: Die ARCHIVE–Operation modifiziert die Zugriffspfade (Verweise, Indexe) so, daß die mittels SELECT ausgewählten Daten über die üblichen SDAI–Zugriffsoperationen nicht mehr erreichbar sind. Gleichzeitig wird ein Pfad eingerichtet, der nur vom Archivierungssubsystem zum Auffinden von Daten benutzt wird. Erst bei erfolgreicher *physischer Archivierung* werden diese Daten – ebenfalls von einem asynchronen Prozeß – in ihrem bisherigen Speicher freigegeben. Damit können archivierte, noch nicht migrierte Daten ebenso schnell über einen Archivzugriffspfad erreicht werden wie aktive Daten. Im allgemeinen halten wir die "Zugriffslücke" aber für nicht sehr kritisch: Beachtet man typische Arbeitsabläufe, so wird auf archivierte Daten erst nach geraumer Zeit zugegriffen.
Ein weiterer Aspekt bei dem Archivierungsvorgang ist die unterschiedliche Behandlung von Daten (Instanzen des Anwendungsschemas) und Metadaten (Instanzen des Dictionary und Session Schemas). Während den Daten eine *move*–Semantik unterliegt, wenden wir auf die Metadaten eine *copy*–Semantik an. Metadaten sicher *mit* den Anwendungsdaten zusammen zu archivieren (und nicht nur zu referenzieren!) ist unabdingbar für die Interpretation der Daten über Jahre oder sogar Jahrzehnte hinweg. Desweiteren erscheint es sinnvoll, Metadaten, die z.B. ein spezielles EXPRESS-Schema beschreiben, im aktiven Datenbestand zu halten, um das Schema sofort neu instanziieren zu können. Außerdem ist der Umfang der Metadaten im allgemeinen viel geringer als der der Anwendungsdaten, so daß die Auslagerung der Metadaten das Platzproblem nur unwesentlich entschärfen würde.

Eine Folge von SELECT– und einem ARCHIVE–Aufruf kann durch die vorhandenen SDAI–Transaktionsprimitive mit Datenbank–Transaktionssemantik (START_TRANSACTION..., COMMIT, ABORT) geklammert werden. Dieser Vorschlag berücksichtigt, daß SDAI SELECT und ARCHIVE in Schreibzugriffe des darunterliegenden Datenbanksystems und SDAI–Transaktionen in Datenbanktransaktionen umgesetzt werden. Potentielle Konflikte zwischen SDAI SELECT, ARCHIVE und anderen Datenbankzugriffen im Mehrbenutzerbetrieb werden so ohne zusätzlichen Aufwand durch die ohnehin vorhandene Synchronisationskomponente gelöst.

[3] Wir weisen an dieser Stelle auf den zusätzlichen Einsatz von Komprimierungsmethoden hin [RV93].

Abschließend stellt sich die Frage nach dem **Zugriff auf archivierte Daten**. Im Gegensatz zu [EL90] lassen wir uns von dem Vorgehen beim Archivieren und Zurückholen konventionell archivierter Dokumente leiten: Der Anwender sucht an einem anderen Ort (z.B. dem Zeichnungsarchiv), wendet dort aber die gleichen Techniken zum Wiederauffinden an. Analog dazu wollen wir die *logische Trennung* zwischen archivierten und aktiven Daten beibehalten und auch keine neuen SDAI–Zugriffsprimitive einführen.

Auch eine explizite **RESTORE**–Operation zum Zurückladen von Daten vor dem Zugriff sehen wir nicht vor. Zwar ist dadurch nicht das Installieren von ggf. *off-line* verwalteten Datenträgern gelöst, aber dies ist auch nicht Aufgabe eines SDAI-Anwendungsprogramms. Das gezielte Kopieren archivierter Daten in eine Speicherumgebung mit kürzeren Zugriffszeiten (**ELEVATE**) lohnt sich dann, wenn ein intensives Arbeiten mit diesen Daten geplant ist. Ein Umlagern archivierter Daten ist erforderlich, wenn Schreibzugriffe auf solche Daten beabsichtigt sind, die ausdrücklich als nicht modifizierbar klassifiziert und unter Umständen auf *read-only* Speichermedien ausgelagert wurden.[4]

Einführung von Archiv–Repositories. Die logische Trennung zwischen dem Archiv und dem aktiven Datenbestand sowie die Perspektive, auf archivierte Daten mit vertrauten SDAI–Operationen zugreifen zu können, sprechen dafür, in SDAI **Archiv–Repositories** einzuführen[5]. Wir erweitern deshalb das SDAI Session Schema wie folgt:

```
ENTITY sdai_session;
 known_servers:   SET OF sdai_repository;     -- zugreifbare Repositories
 archive_servers: SET OF archive_repository;  -- neu! (siehe unten)
 active_servers:  SET OF sdai_repository;     -- geoeffnete Repositories
 ...
END_ENTITY;
```

In einer SDAI–Session stehen dem Anwendungsprogramm jetzt nicht nur "normale" Repositories, sondern auch Archiv–Repositories zur Auswahl. Wenn ein beliebiges Repository geöffnet wird, ist es (konzeptionell) auch in der Menge der *active_servers* enthalten. Anschließend sind grundsätzlich GET_ATTRIBUTE, SDAI QUERY usw. anwendbar.

Dieser Ansatz erlaubt es, spezielle Anforderungen an die Archivierung aus Anwendungssicht durch verschiedene Arten von Repositories auszudrücken:

- *read-only* für Archive, die ausschließlich lesende Zugriffe unterstützen,
- *long-term* für Archive mit extrem langen Aufbewahrungszeiten oder
- *vaulted* für Archive, die in besonderem Maße gegen Umwelteinflüsse wie Feuer und Wasser geschützt werden sollen.

[4] Nicht alle Anwendungen sprechen gegen Updates in Archiven: Technische Zeichnungen werden oft lokal ("in place") korrigiert und mit einem Änderungsvermerk versehen.

[5] Wie Repositories *erzeugt* werden, ist nicht Gegenstand der Spezifikation [ISO94]. Deshalb haben wir auch keine Operation CREATE ARCHIVE eingeführt.

Wir spezifizieren diese Erweiterung auf der konzeptuellen Ebene des SDAI–Benutzers und gehen davon aus, daß die Implementierung eines SDAI–Archivierungsdienstes geeignete Tertiärspeicher als physische Grundlage der logischen Archive vorsehen muß:

```
TYPE
 archive_type = ENUMERATION OF (read-only,long-term,vaulted,...);
END_TYPE;

ENTITY archive_repository
 SUBTYPE OF (sdai_repository);
 characteristics = SET OF archive_type;
END_ENTITY;
```

Eine Alternative zum Archiv-Repository-Ansatz ist die Kennzeichnung archivierter Daten (z.B. durch ein Status–Flag) unter Beibehaltung ihres bisherigen logischen Speicherortes, d.h. des aktuellen SDAI–Repositorys. Gegen eine derart uniforme Betrachtung spricht die wesentlich andere Qualität archivierter Daten aus Sicht vieler Anwendungen. Beispielsweise werden Produktdaten einer abgeschlossenen Baureihe oder Bilanzen aus zurückliegenden Jahren nicht im operationalen Datenbestand erwartet. Das andere Zugriffsprofil (siehe Abschnitt 1) und die erwartete Stabilität eines Archivs über lange Zeit rechtfertigen ebenfalls dessen besondere, eigenständige Rolle – sowohl logisch als auch physisch.

2.3 Resultierende Anforderungen an Datenbanksysteme

Da wir SDAI in diesem Beitrag als eine Softwareschicht ansehen, die auf einem Datenbanksystem implementiert wird, leiten sich aus der spezifizierten Archivierungsfunktionalität Anforderungen an die zugrundeliegenden Systeme ab. Bezüglich der Externspeicherebene haben wir bereits angedeutet:

- *kostengünstigere Speichermedien als Magnetplatten*
  Nicht nur rein magnetische Speicher unterliegen dem anhaltenden Preisverfall. Neue, leistungsfähigere Laufwerke für magneto–optische (MO) Platten sind nur ein Beispiel für die attraktiver werdende MO–Technologie [NHVR93]. In [GSSZ93] werden hohe Erwartungen in optische Bänder gesetzt.
- *Langzeitspeicherung*
  Sofern die Haltbarkeit von Datenträgern ohne Informationsverlust über Jahrzehnte nicht garantiert werden kann, müssen geeignete Refresh–Techniken oder Kopierverfahren dies kompensieren [Wal94].
- *ggf. WORM–Medien* [Zab90]
  Höchstens einmal beschreibbare Medien erhöhen die handels– und steuerrechtliche Beweiskraft der Daten [Gei93]. Ein authentischer Nachweis liegt auch im Interesse von Herstellern, die für evtl. Produktfehler haften [Sam93].
- *ggf. "Electronic Vaulting"* [GR93]
  Die räumliche Unterbringung der Speichermedien erfolgt in entfernten, besonders abgesicherten Spezialräumen.

Methoden zur vollen Einbeziehung von Tertiärspeicher in Datenbanksysteme, d.h. als gleichrangige Externspeicheralternativen, die sich nur durch Zugriffszeit und Kapazität unterschieden, sind aktueller Forschungsgegenstand [CHL93, SS94]. Zu den ersten Produkten, die optische Platten(roboter) integrieren, zählen DB2, Illustra und Transbase/CD. Der Grad der Integration (z.B. nur Emulation einer Magnetplatten–Schnittstelle duch den Jukebox–Controller) und die Erweiterbarkeit um neue Speichermedien sind von System zu System verschieden. Beschränkungen findet man vor allem bezüglich folgender Anforderungen:

- Auswahl einschließlich Korrektheitskontrolle der Daten, die auf Tertiärspeicher archiviert werden sollen (Abbildung der selektierten SDAI–Granulate)
- Benutzerkontrolle der Archivierung (Abbilden der logischen ARCHIVE–Operation, Überprüfen der Zugriffsberechtigung)
- auslagerndes Archivieren (ggf. Nullwerte erforderlich, asynchrone Migration bzgl. "move data"/"copy metadata", Komprimierungsoption)
- effiziente Zugriffspfade für tertiärspeicherresidente Daten (Transformation von Indexen)
- Navigation durch / Anfragen an Daten, die nicht explizit auf Sekundärspeicher zurückgeladen werden (optimales Retrieval, Schreibzugriffe)
- Logging des Archivierungsvorgangs

3 Ausnutzung von Datenbanksystemen für SDAI–integriertes Archivieren

3.1 Ansätze im Überblick

Es überrascht nicht, daß es keine persistenten SDAI–Implementierungen auf der Basis von Netzwerk– oder hierarchischen Datenbanksystemen gibt. Verschiedene Systeme bilden EXPRESS auf das relationale Modell ab [Wil94], belassen es aber bei der systemeigenen DML (meist SQL) ohne eine zusätzliche SDAI–Schicht. In [RM94] wird argumentiert, daß das C++-Binding von SDAI die am erfolgversprechendste Variante ist, weil hier am ehesten EXPRESS-Semantik (insbesondere Vererbung) "gerettet" werden kann. Außerdem können viele Typüberprüfungen vom Compiler übernommen werden. Desweiteren wird C++ von den meisten objektorientierten Datenbanksystemen unterstützt. Wir gehen deshalb nur auf relationale und objektorientierte Systeme ein.

Die zweite Ebene unserer Klassifikation von Implementierungen anwendungsorientierter Archivierungsfunktionalität ist der Grad der Integration eines Archivierungssubsystems in ein Datenbanksystem. Wir geben im Rest des Abschnitts die markantesten bzw. einzig bekannten Beispiele für folgende Ansätze an:

a) Ausnutzung "üblicher" Schnittstellen, d.h. allgemein verfügbare oder standardisierte DDL/DML ohne systemspezifische Erweiterungen
b) Ausnutzung spezieller (Archivierungs–) Funktionen, die aber noch zur Anwendungsschnittstelle des Datenbanksystems gehören
c) Enge Integration von Archivierungskomponenten in das Datenbanksystem unter Ausnutzung interner Schnittstellen

3.2 (Erweitert) relationale Systeme

a) Systeme mit SQL–Schnittstelle. Die spezifizierte SDAI SELECT– und ARCHIVE–Operation kann in Markierungen durch tupelweises SQL UPDATE umgesetzt werden. Dazu muß das Datenbankschema so erweitert werden, daß die Basisrelationen zusätzliche Attribute erhalten oder ein vorhandenes Datenbank–Dictionary ausgenutzt wird. Auch die SDAI–Metadaten können um Statusinformationen angereichert werden. Diese sind in Anfragen auszuwerten. Die asynchrone Datenauslagerung in Archiv–Tabellen, die zuvor auf Tertiärspeicher angelegt worden sind, muß in separaten Transaktionen durch Folgen von SQL INSERT und SQL DELETE erfolgen.

b) Codd's Vorschlag für RM/V2. Codd schlägt in [Cod90] ein Kommando ARCHIVE vor, das unserer Spezifikation auf den ersten Blick ähnlich, aber im Detail weniger mächtig ist: Der Datenbankadministrator kann damit Relationen (komplette Basistabellen oder Sichten) synchron auslagern. Allerdings ist ein erneuter Zugriff darauf erst nach dem REACTIVATE-Kommando möglich. Bei dem Zurückladen werden evtl. existierende Relationen gleichen Namens überschrieben. Was im Fall einer Sicht passieren soll (Materialisierung von Basistabellen?) wird nicht näher beschrieben. Jegliche Implementierungshinweise fehlen. Da verschiedene Arten von Archiven nicht näher spezifizierbar und Anfragen an archivierte Daten nicht vorgesehen sind, wirkt dieser Ansatz eher wie ein tabellenbezogenes Backup/Restore. Datenbanksysteme, die diesen Vorschlag in die Praxis umsetzen, sind uns nicht bekannt.

c) Eingriffsmöglichkeiten in Postgres. Das erweitert relationale Datenbanksystem Postgres sah frühzeitig das Auslagern veralteter Sätze auf WORM-Datenträger vor [Sto87]. Der ursprüngliche *Vacuum Cleaner* überträgt die sich qualifizierenden Tupel einer Basis– in eine Archiv–Relation sofern der Modus *light*– oder *heavy–archive* angegeben ist.

Mit dem Einziehen einer Schnittstelle (*Storage Manager Switch*) zwischen dem Zugriffssystem (*Data Manager*) und den geräteabhängigen Speichersubsystemen (*Storage Device Manager*) wurde Postgres prinzipiell um beliebige Speicher erweiterbar [Ols92]. Neue Device Manager müssen sich an Postgres–Konventionen halten: Sie müssen z.B. Postgres–Relationen auf dem Speichermedium anlegen und in Blöcken von 8 KByte lesen und schreiben. Nach der wiederholten Übersetzung des gesamten Postgres(system)–Quelltextes werden die neuen Device Manager von den selben Zugriffsmethoden angesprochen.

Eine derart enge Integration bringt selbstverständlich Abhängigkeiten vom Datenbanksystem und damit eine geringere Autonomie des Archivierungssubsystems mit sich. Auch wenn man von Postgres-Spezifika wie z.B. der "no-overwrite"[6]–Speicherverwaltung absieht, kann das Vacuuming nur bedingt für SDAI–integriertes Archivieren ausgenutzt werden. So ist etwa das Kriterium,

[6] Tatsächlich erfolgen durchaus Updates in Blöcken, z.B. beim Setzen des Transaktionszustandes eines Tupels.

das zum Auslagern von Sätzen führt, anders zu fassen: Die SDAI SELECT–Operation und nicht das Alter eines Tupels veranlassen dessen Archivierung. Unabhängig vom Vacuuming ist das Granulat Relation für die (feste!) Zuordnung eines Speichermediums zu einer Relation sehr grob. Grundsätzlich ist aber erst durch die Offenlegung des Storage Manager Switch die gleichrangige Einbeziehung von Tertiärspeicher möglich geworden.

Zu den inhärenten Problemen und Lösungsansätzen bei der Abbildung von EXPRESS (ohne SDAI) auf das Datenmodell von Postgres verweisen wir auf [Gud94]. In dieser Implementierung wurden viele Erweiterungen von Postgres gegenüber (rein) relationalen Systemen, wie z.B. benutzerdefinierte abstrakte Datentypen, bereits ausgenutzt. Die Modellierungsmächtigkeit (Datentypvielfalt, Vererbungsbeziehungen) objektorientierter Systeme ist jedoch weitaus höher.

3.3 Objektorientierte Systeme

a) ODMG-93. Anders als im relationalen Fall unterscheiden sich die Konzepte und Schnittstellen von objektorientierten Datenbanksystemen erheblich [Heu92]. Der Standardisierungsversuch in [Cat94] strebt eine Vereinheitlichung an, die im Falle ihres Erfolgs eine geeignete Basis für die persistente Implementierung des C++-Bindings von SDAI einschließlich Archivierung wäre. Für die gegenwärtige Diskussion des Ansatzes "oberhalb ODBMS" ziehen wir ein konkretes System vor: Wir besprechen unseren *ObjectStore*–basierten Prototyp in Abschnitt 4.

b) Archivierungs–Methoden in Versant. Versant ODBMS Release 3 bietet dem Anwendungsprogrammierer eine Methode *archive()* an: Man kann Objekte (Instanzen) spezifizieren, die in eine Archiv-Datenbank ausgelagert werden sollen. Dabei gibt es eine Reihe von Einschränkungen. Referenzen auf Objekte, die nicht archiviert werden können (z.B. Schema–, Klassen–, System– oder versionierte Objekte), liefert die Methode zurück. Versant geht davon aus, daß die Klasse, dem das Objekt entstammt (Schema–Objekt), mit der ggf. bereits archivierten Klasse gleichen Names kompatibel ist. In der aktiven Datenbank verbleiben Statusinformationen, sog. Stellvertreter oder *proxy objects*. Diese Funktionalität läßt sich prinzipiell für die Archivierung von Anwendungsobjekten ausnutzen. Die SDAI Dictionary Instanzen müssen kopiert werden.

Der Aufruf der archive()–Methode muß in normale Transaktionsklammern eingeschlossen werden, wodurch sich der Einfluß des Benutzers auf den Zeitpunkt des tatsächlichen Datentransfers auf das Absetzen des Commits beschränkt. Dies wirkt sich bei einer beabsichtigten separaten Implementierung von SDAI SELECT, SDAI ARCHIVE und asynchroner Datenauslagerung nachteilig aus.

Versant erzwingt vor dem Zugriff auf archivierte Objekte den Aufruf einer *restore()*–Methode. Dabei werden keine Kopien der Objekte zurückgeladen, sondern die Objekte selbst, so daß sie anschließend nicht mehr in der Archiv–Datenbank enthalten sind. Versant erhält somit streng die Identität eines Objekts – unabhängig von dessen Status bzw. Speicherort. Dieser Service ist für SDAI–integriertes Archivieren dann von Vorteil, wenn die Auslagerung von Granulaten zugelassen werden soll, die zu offenen Objektreferenzen führen.

c) Storage–Manager–Klassen in ONTOS/DB. Wir wählen ONTOS/DB als Beispiel für die Verwendung "tiefer" bzw. interner Schnittstellen, weil dieses ODBMS bezüglich der Externspeicherverwaltung erweiterbar ist. Die Speicherverwaltung ist selbst als Menge von instanziierten Klassen implementiert und offengelegt. Ein Anwendungsprogramm kann **pro Objekt** eine *Storage Manager*–Instanz festlegen, über die das Anlegen, die Identifikation, das Auffinden und die Zugriffe auf das Objekt abgewickelt werden. Im Zusammenhang mit einem Archivierungssubsystem besteht die Idee darin, eigene Storage Manager unter Verwendung der Klasse "OC_ExternalSM" zu entwickeln, die Tertiärspeicher ansprechen (vgl. Device Manager in Postgres). Natürlich erfordert dies ein tiefes Systemverständnis, da viele virtuelle Methoden zur Sperrverwaltung, zur ONTOS–spezifischen Typregistrierung oder "Aktivierung" von Objekten neu zu implementieren sind.

Allerdings gibt es neben den allgemeinen Nachteilen bei der engen Integration eines Archivierungssubsystems (siehe 3.2.c) eine weitere konkrete Beschränkung: Die Zuordnung eines Storage Managers zu einem Objekt besteht für die gesamte Lebenszeit des Objekts. Die Auslagerung von Daten vom Sekundär- auf Tertiärspeicher im Zuge von SDAI ARCHIVE ist demnach nur über Kopieren, Zuordnen eines neuen Storage Managers und Löschen zu erreichen.

4 Implementierungsvariante "oberhalb ODBMS"

4.1 Prototyp eines EXPRESS/SDAI-Datenbanksystems

Unsere ersten praktischen Erfahrungen mit einem persistenten SDAI resultieren aus einem Prototyp, der sich an der Spezifikation des C++-Bindings von [ISO94] orientiert und die dort vorgegebenen Definitionen unter Ausnutzung des ODBMS ObjectStore implementiert. Abbildung 1 zeigt die vier Komponenten, die beim Anlegen und während des Zugriffs auf eine EXPRESS/SDAI-Datenbank zusammenspielen [Pri93, Her94].

1. *EXPRESS Parser*
   Wir wollten nach Möglichkeit auf existierende Softwarebausteine zurückgreifen und haben deshalb den frei verfügbaren Schema–Parser ausgewählt, der von NIST ("National Institute of Standards and Technology") in den USA entwickelt wurde. Dieser Parser ist mit einem Quelltex–Generator gekoppelt, der u.a. aus EXPRESS–Entities C++-Klassen mit attributbezogenen Zugriffsmethoden erzeugt.
2. *Präprozessor*
   Effiziente *awk*–Programme modifizieren den C++-Code so, daß SDAI–Konventionen strenger eingehalten werden. Der Präprozessor stellt auch sicher, daß persistente Objekte tatsächlich in der Datenbank und nicht durch herkömmliches *new* im Hauptspeicher angelegt werden.
3. *SDAI–Klassenbibliothek*
   Diese C++-Klassen sind unabhängig vom EXPRESS–Anwendungsschema. Sie realisieren das SDAI Dictionary- und Session Schema.

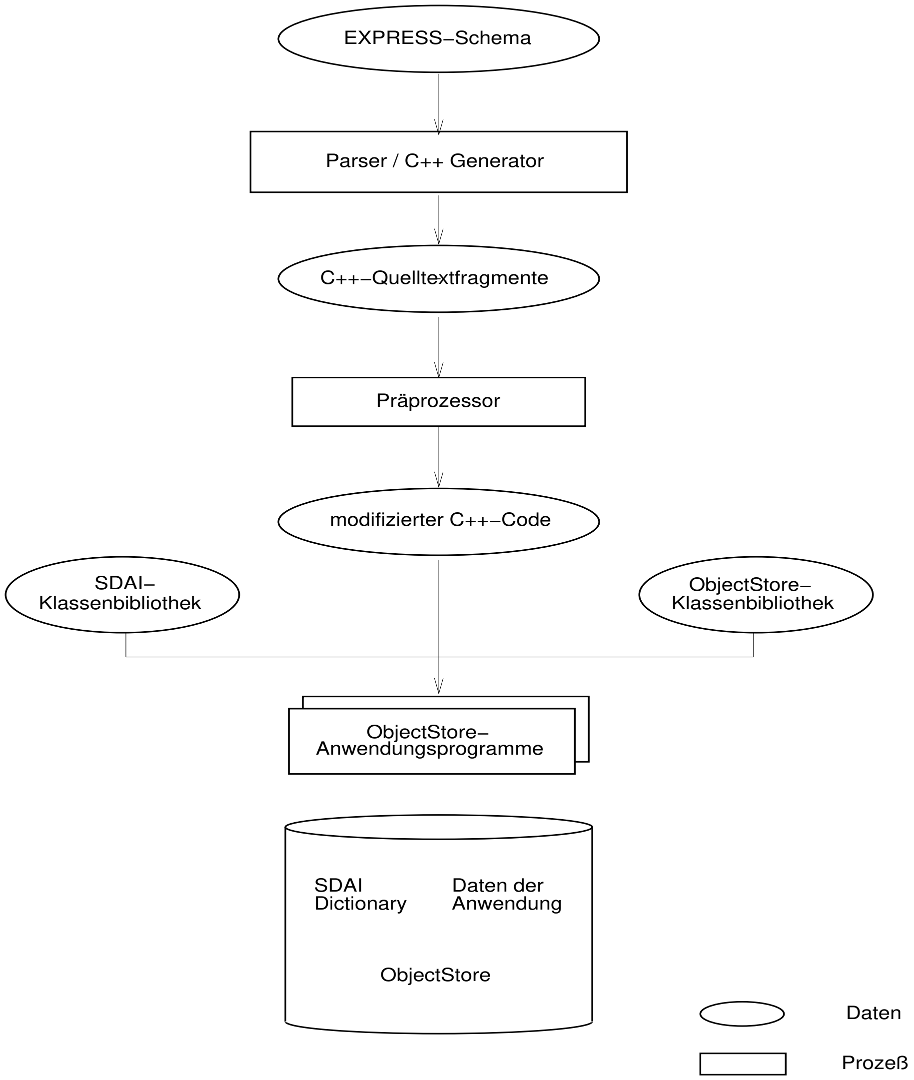

Abbildung1. Komponenten unseres EXPRESS/SDAI–Datenbanksystems

4. *ObjectStore*
   Die folgenden Eigenschaften dieses ODBMS lassen sich gut für die Implementierung des C++-Bindings ausnutzen:
   – C++-Einbettung (“seamless integration”)
   – Persistenz orthogonal zum Typsystem
   – Unterstützung für inverse EXPRESS Attribute (siehe Entities in Abschnitt 2.2)
   – Kollektionsgebundene Anfragen

Auch wenn das Anlegen eines SDAI–Repositorys in [ISO94] nicht normiert ist (vgl. Fußnote 5), geht jedem SDAI–Anwendungsprogramm (*run time*) das Überführen eines EXPRESS–Schemas in persistente Daten des SDAI Dictionarys voraus (*build time*). Auch Informationen aus dem SDAI Session Schema (z.B. die Zuordnung von Instanzen zu SDAI *models*) werden im Datenbanksystem verwaltet. Dies verlangsamt zwar wesentlich die Schreibzugriffe, ermöglicht aber ein sog. *SDAI late binding* – also eine Anwendungsentwicklung, die weitgehend unabhängig von einem konkreten Anwendungsschema ist. Die langfristige Interpretierbarkeit der EXPRESS–Daten durch die SDAI–Metadaten ist gleichzeitig eine notwendige Bedingung für die Archivierung.

4.2 Erweiterungen des Prototyps in Richtung Archivierung

Unsere ersten Untersuchungen von ObjectStore hinsichtlich der Unterstützung von SDAI–integriertem Archivieren beziehen sich auf den Aspekt der gemeinsamen Archivierung von SDAI–Metadaten und Anwendungsdaten. ObjectStore kennt sog. *Konfigurationen.* Dies sind Gruppierungen von Objekten, die sich in ihrer Gesamtheit weiterentwickeln und dabei versioniert werden. Eine neue Version einer Konfiguration führt beim *check-in* zum "Einfrieren" ihrer Vorgängerversion. Auf alte Versionen kann dann nur noch lesend zugegriffen werden, oder es ist zuvor ein *check-out* erforderlich. Vereinfachend assoziieren wir *check-in* mit SDAI ARCHIVE.

Im SDAI C++-Binding sind Klassen für Anwendungsdaten und Metadaten von der Klasse *SdaiEntityInstance* abgeleitet. Um Konfigurationen von beliebigen SDAI–Granulaten bilden zu können, führen wir eine neue Wurzelklasse (ObjectStore's *os_configuration*) für die Implementierungshierarchie ein. Jetzt ist es möglich, *check-in* und *check-out* auf eine Konfiguration anzuwenden, die sowohl (Kopien der) EXPRESS-Schemainformationen als auch Anwendungsdaten umfaßt. Letztere gehören zum Extent der Klasse *SdaiAppInstance*:

```
class SdaiEntityInstance:
      public os_configuration { ... };

class SdaiAppInstance:
      virtual public SdaiEntityInstance { ... };
```

Allerdings deckt dieser Ansatz nur mit Einschränkungen die im Abschnitt 2.2 spezifizierte Funktionalität ab: Ein dynamisches SDAI SELECT, das erst zur Laufzeit des Anwendungsprogramms die Daten zur Archivierung auswählt, kann nicht unmittelbar realisiert werden, da Konfigurationen in folgendem Sinn statisch sind: Ein Objekt muß bereits bei seiner Erzeugung einer Konfiguration zugeordnet werden. Dies erzwingt wiederum das Markieren der zu archivierenden Daten bei SDAI SELECT und das Anlegen einer entsprechenden Konfiguration (z.B. durch Copy-Konstruktoren) während SDAI ARCHIVE. Die beabsichtigte Verringerung des aktiven Datenbestandes muß durch anschließendes explizites Löschen der Objekte erfolgen.

Aus einem anderen Grund kann unser konkretes Systemszenario den vollen Umfang von SDAI–integriertem Archivieren derzeit nur simulieren: Um die Idee eines tertiärspeicherresidenten SDAI–Archiv–Repositorys umzusetzen, bietet ObjectStore nur die Möglichkeit, **ganze** *ObjectStore–Datenbanken* auf einem Externspeicher zu plazieren. Dabei werden entweder *raw partitions* von Festplatten ausgewählter Hardwareplattformen unterstützt oder das UNIX–Dateisystem als geräteunabhängige Schnittstelle benutzt. Mit der Version 3.1 ist erstmalig ein *NFS mount* von nicht–lokalen (*remote server*) Dateisystemen erlaubt. Datengranulate, wie sie der ObjectStore–Server intern verwaltet (Seiten, Segmente) oder anwendungsbezogene Einheiten, die ein ObjectStore–Client kennt (Cluster, Versionen von Konfigurationen), können jedoch nicht selektiv auf Tertiärspeicher ausgelagert werden. Andererseits gewährleist eine "on–top"–Lösung im Vergleich zur Verwendung tieferer Schnittstellen eine größere Datenunabhängigkeit, die angesichts einer langfristigen Archivierung einen hohen Stellenwert besitzt.

5 Resümee

Wir sehen einen wachsenden Bedarf für anwendungsorientiertes Archivieren, wobei Fortschritte in der Massenspeichertechnologie neuartige Systemrealisierungen erst ermöglichen. Derzeit befindet sich die Erweiterbarkeit von Datenbanksystemen um eine integrierte Archivierungskomponente noch in den Anfängen. Aber auch eine lose Kopplung eines Datenbanksystems mit einem dedizierten Archivierungssystem (wie sie vergleichsweise im Zusammenhang mit einem Speichersystem für Multimediaobjekte in [KMMW93] konzipiert wurde) hat Vorteile, zu denen insbesondere Autonomie und Stabilität zählen [Her93].

Aus Sicht der spezifizierten Archivierungsfunktionalität kann durchaus auf die absolute Gleichberechtigung von Sekundär– und Tertiärspeicher ("1st class citizen" [CHL93]) zugunsten einer geräteunabhängigen Schnittstelle zu einem Archivierungssubsystem in einem Datenbanksystem verzichtet werden. Denn archivierte und aktive Daten unterscheiden sich grundlegend im Zugriffsprofil und dem tolerierbaren Zugriffszeitverhalten. Dies heißt jedoch nicht, daß der weitverbreitete einschlägige Datenbankservice – ein Backup/Restore – das benutzerveranlaßte, datenmodellbasierte und auslagernde Archivieren hinreichend unterstützt. Die angegebenen Vorschläge für höhere Archivierungskonzepte in relationalen und erste Realisierungen in objektorientierten Systemen decken sich ebenfalls nicht mit der von uns spezifizierten Semantik.

Wir haben in diesem Beitrag einen Ansatz gewählt, der auf Standards basiert. Wir betrachten EXPRESS und die sich in der Entwicklung befindliche SDAI–Schnittstelle über STEP hinaus als geeigneten Kontext für anwendungsorientiertes Archivieren. Notwendige Erweiterungen von SDAI sind die Operationen SELECT und ARCHIVE sowie die Einführung von Archiv–Repositories.

Unsere weiteren Arbeiten werden die Spezifikation und Implementierungsaspekte von SDAI–integriertem Archivieren detaillieren. Wir werden z.B. Tertiärspeicherzugriffe und Archivzugriffspfade simulieren müssen, solange wir auf Grenzen der internen Erweiterbarkeit von eingesetzten Datenbanksystemen stoßen.

Danksagung

Mein Dank gilt Herrn Prof. Dr. K. Küspert sowie den anonymen Gutachtern für die sorgfältige und hilfreiche Durchsicht des Manuskripts.

Literatur

[Cat94] R. G. G. Cattell (Hrsg.). *The Object Database Standard: ODMG–93, Release 1.1.* Morgan Kaufmann, 1994.

[CHL93] M. J. Carey, L. M. Haas, M. Livny. Tapes Hold Data, Too: Challenges of Tuples on Tertiary Store. In *ACM SIGMOD*, S. 413–417, Washington, 1993.

[Cod90] E. F. Codd. *The Relational Model for Database Management: Version 2.* Addison–Wesley, Massachusetts, 1990.

[DHSV94] M. Dach, N. Hoimyr, J. Saarela, J. Vuoskoski. Using EXPRESS in a High Energy Physics Research Environment. In *4th Int. EXPRESS Users Group Conf.*, Greenville, Oktober 1994.

[EL90] J. Encarnacao, P. C. Lockemann. *Engineering Databases.* Springer–Verlag, Berlin Heidelberg New York, 1990.

[GAP93] H. Grabowski, R. Anderl, A. Polly. *Integriertes Produktmodell.* Beuth–Verlag, Berlin, Wien, Zürich, 1993.

[Gei93] I. Geis. Rechtliche Aspekte der elektronischen Dokumentenerarbeitung und -verwaltung. In *NormDOC'93*, Berlin, November 1993. Beuth–Verlag.

[GR93] J. Gray, A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, 1993.

[GSSZ93] J. Gulbins, M. Seyfried, H. Strack-Zimmermann. *Elektronische Archivierungssysteme.* Springer–Verlag, Berlin, Heidelberg, New York, 1993.

[Gud94] W. Guddat. Realisierung einer STEP–Produktdatenbank auf POSTGRES. Diplomarbeit, Friedrich-Alexander-Universität Erlangen–Nürnberg, Februar 1994.

[Her93] A. Herbst. STEP–basierte Ansätze für Archivierungssysteme. Technischer Bericht TN 93.01, IBM WZH, Heidelberg, August 1993.

[Her94] A. Herbst. Long-Term Database Support for EXPRESS Data. In *7th Int. Working Conference on Scientific and Statistical Database Management*, Charlottesville, September 1994.

[Heu92] A. Heuer. *Objektorientierte Datenbanken: Konzepte, Modelle, Systeme.* Addison–Wesley, Bonn u.a., 1992.

[HM94] A. Herbst, B. Malle. Perspektiven für die Archivierung von CAD–Daten in einer STEP–Umgebung. In *CAD'94*, Paderborn, März 1994. Hanser–Verlag.

[ISO94] *ISO 10303-22: Product Data Representation and Exchange - Part 22: Standard Data Access Interface (CD).* 1994.

[ISO95] *10303-11: Product Data Representation and Exchange - Part 11: EXPRESS Language Reference Manual (IS).* 1995.

[KMMW93] R. Käckenhoff, D. Merten, K. Meyer-Wegener. Eine vergleichende Untersuchung der Speicherungsformen für multimediale Datenobjekte. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, Braunschweig, März 1993. Springer–Verlag.

[KPCD+92] R. H. Katz, D. A. Patterson, A. Chervenak-Drapeau, J. Fine, E. Miller. An Approach to Cost-Effective Terabyte Memory Systems. In *COMPCON Spring'92*, San Francisco, Februar 1992.

[LRW93] H. Lührsen, T. Ruf, H. Wedekind. STEP–Datenbanken. *CIM Management*, 9(5):9–13, 1993.

[MN93] C. Mohan, I. Narang. An Efficient and Flexible Method for Archiving a Data Base. In *ACM SIGMOD*, Washington, 1993.

[MSRD91] U. Mehlhaus, S. Schneider, U. Rembold, R. Dillmann. Die Schemabeschreibungssprache EXPRESS des STEP–Standards und technische Datenbanksysteme — Eine Analyse. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, Kaiserslautern, März 1991. Springer–Verlag.

[NHVR93] T. Nakagomi, M. Holzbach, R. VanMeter, S. Ranade. Re-Defining the Storage Hierarchy: An Ultra–Fast Magneto–Optical Disk Drive. In *12th IEEE Symposium on Mass Storage Systems*, Monterey, April 1993.

[Ols92] M. A. Olson. Extending the Postgres Database System to Manage Tertiary Storage. Master's thesis, Univ. of California, Berkeley, 1992.

[Owe93] J. Owen. *STEP - An Introduction.* Information Geometers, Winchester, 1993.

[Pri93] A. Primbs. STEP/EXPRESS–Datenverwaltung mit einem objektorientierten Datenbanksystem. Diplomarbeit, IBM WZH / Universität Mannheim, Dezember 1993.

[RDMP94] T. C. Rakow, P. Dettling, F. Moser, B. Paul. Development of a Multimedia Archiving Teleservice using the DFR Standard. In *Workshop on Advanced Teleservices and High Speed Communication Architectures*, Heidelberg, September 1994.

[RM94] T. Rando, L. McCabe. Issues in Implementing the C++ Binding to SDAI. *Computer Standards and Interfaces*, 16(4):331–340, 1994.

[RV93] M. A. Roth, S. J. VanHorn. Database Compression. *SIGMOD Record*, 22(3):31–39, September 1993.

[Sam93] U. E. Samel. Produkthaftungsgesetz und CAD–Archivierung. *CAD–CAM Report*, 9(5):138–144, 1993.

[SS94] S. Sarawagi, M. Stonebraker. Single Query Optimization for Tertiary Memory. Technischer Bericht Sequoia 2000, 94/45, University of California, Berkeley, 1994.

[Sto87] M. Stonebraker. The Design of the Postgres Storage System. In *13th VLDB*, S. 289–300, Brighton, 1987.

[SW94] D. A. Schenck, P. R. Wilson. *Information Modeling: The EXPRESS Way.* Oxford University Press, 1994.

[VS91] S. Vajna, W. Stenke. Wirtschaftliche Nutzung des digitalen Archivs. *CAD–CAM Report*, 7(5):143–149, 1991.

[Wal94] S. Wallace. Managing Mass Storage. *Byte*, 19(3):78–89, 1994.

[Wil94] P. R. Wilson. EXPRESS Tools and Services. Rensselaer Polytechnic Institute, Troy, August 1994.

[Zab90] P. Zabback. Optische und magneto–optische Platten in File– und Datenbanksystemen. *Informatik Spektrum*, 13:260–275, 1990.

[ZPD90] P. Zabback, J. B. Paul, U. Deppisch. Office Documents on a Database Kernel – Filing, Retrieval, and Archiving. In *5th Conf. on Office Information Systems*, S. 261–270, Cambridge, April 1990.