Document Exchange Considered Useful – Extending the Reach of DBMS-Functionality to Document Driven Processes

Alexander Hilliger von Thile
DaimlerChrysler Research and Technology
P.O. Box 2360, 89013 Ulm
Germany
+49 (0) 731 505 4840
alexander.hilliger_von_thile@daimlerchrysler.com

ABSTRACT

In today's enterprises documents are preferred by most users for data exchange in spontaneously started and frequently changing processes because documents (e.g. spreadsheets) are easy to create and to edit. This causes well known problems such as poor data quality due to missing constraint checks, no up-to-date data in backend systems (e.g. for analysis), and missing multi-user support. These problems could be avoided by using special applications that store their data in a DBMS. But for end users this is significantly more complex and expensive than just creating a new spreadsheet.

Instead of refraining from document exchange this paper describes how the reach of well-known database concepts (e.g. multi-user support, data quality checks, and trigger) can be extended to support document-driven data exchange.

To this end, this paper presents a novel approach to handle data inside common documents as objects that are under control by a DBMS. This is realized by extending the concept of common views to materialized external views. Afterwards, this paper describes how documents can be turned into a DBMS themselves by combining data, meta-data, and execution logic within a single file. These 'smart' files can be exchanged like other documents but they are able to check their data for integrity and propagate changes to backend systems automatically.

1. INTRODUCTION

Many processes of an enterprise span between changing networks of external partners, are started spontaneously, and are subject to frequent changes. As a consequence, they need to be highly flexible. Due to development time and cost, for most of these processes no enterprise application and no database for data analysis exist. Furthermore, such processes are executed by non-IT-experts without deeper knowledge of DBMS. Thus, in these processes documents (flat files) are used for data exchange because they are easy to create, edit (even offline), and exchange.

From a technical perspective, highlighting advantages of documents might seem absurd because disadvantages are predominating. However, for non IT experts in many cases other criteria such as simplicity, flexibility, and cost-effectiveness of just creating a desktop document without the need to code are key criteria. It is important to understand that the main reasons for document exchange are non-technical. This explains why documents are still preferred even though plenty of alternatives – such as WfMS or groupware systems – are available.

Today, using documents causes serious problems because in contrast to DBMS, they are not stored centrally, they contain minimal (schema) information defining consistency, and they offer no control mechanism enforcing these constraints by preventing inconsistent changes. As a consequence, data from these processes is not accessible for up-to-date data analysis (real-time data warehousing) and data quality is unpredictable because it is unchecked during process execution.

In this paper, we tackle these issues by asking the research question whether and how proven functionality of DBMS such as multi-user support, data quality checks (constraints), trigger and alerter (for real-time and active data warehousing) can be made available in the domain of personal data management and ad hoc processes that are executed using documents from desktop applications (e.g. cad, office) as described above.

2. OVERVIEW OF THE PROPOSED APPROACH

To answer the question of how to overcome the disadvantages of document exchange this paper first describes how the reach of DBMS functionality can be extended to documents. Therefore, in section 3, an approach is presented that handles data from common documents (such as spreadsheets) as objects that are under control by a DBMS. This approach extends the concept of common views to materialized external views (MEV). These views store their physical content outside the database by mapping the view's data to file formats that are commonly used for document exchange.

To support offline editing and exchange of documents, in section 4, this approach is extended to cover the entire document-driven process by turning documents into stand-alone DBMS. This is achieved by combining data, metadata, and execution logic within a single file we refer to as smart file. Smart files can be exchanged like common documents but they are able to check their data for integrity and propagate changes to backend systems automatically.

Proceedings of the VLDB2006 Ph.D. Workshop
Seoul, Rep of Korea, 2006

This concept combines the advantages of cost-effective and easy document exchange with the powerful features of today's DBMS.

3. HANDLING DOCUMENT DATA AS DATABASE OBJECTS

Most process relevant data that is exchanged using documents consists of excerpts of data which is maintained by backend systems that access a DBMS. Within a DBMS, (updateable) views can be used to get a customized perspective on data that is relevant within a process. However, these views are materialized in the tablespace of the DBMS. To map data between a DBMS and documents (and vice versa), our work extends the concept of views as they are used in today's DBMS by using a concept we refer to as materialized external views (MEV). An MEV uses a relational expression (like common views do) to create a custom view on data. However, the physical materialization of the view is not held within the database but is stored externally (e.g. in a file share). Instead of a proprietary file format, the view's data is mapped to file formats that are commonly used for document exchange using a wrapper that maps rows to records (binary files) or nodes (XML) and vice versa.

As a general concept, we define a materialized external view as follows:

1. its content is defined by a relational expression (select-query) on a base relation or another view
2. results of the view's relational expression are materialized (physically stored)
3. the materialized data is maintained outside the tablespace of the DBMS in a file format that is used by desktop applications

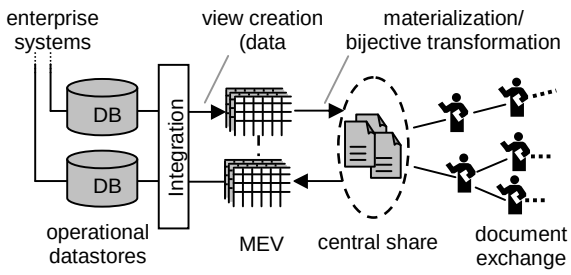

Figure 1. Mapping views to MEV-documents

The figure above depicts necessary steps to make data – which is managed by a DBMS – available as documents. In contrast to simple import/export processors it should be denoted that these documents are virtual files (views) that are managed by the DBMS itself. To realize this idea, creation of an MEV, mapping its data, and working with an MEV (editing the document and integrating changes back to the database) is described next.

3.1 View creation

To create an MEV, a select-statement like in common updateable views is used (see line 6-7). We extended the create-materialized-view SQL statement to support an externalize-in clause (line 5) that defines where the physically materialized content has to be stored.

```
1 CREATE MATERIALIZED VIEW name [<COLUMN-LIST>]
2 [EXTENDED BY (<TYPED-COLUMN-LIST>)]
3 USING (<TRANSFORM-WRAPPER-ALIAS
4    [WRAPPER-OPTIONS]> | <TRANSFORM-STATEMENT>)
5 EXTERNALIZE <OPTIONS> IN <DIRECTORY-ALIAS>
6 AS <FULL-SELECT-STATEMENT>
7 [WITH <CHECK-OPTIONS>]
```

How the transformation is performed can be specified by using a transformation statement (e.g. as done with Oracle's external tables) or by using a wrapper. Today, descriptive approaches are used for text or XML files primarily. Binary files are usually handled by wrappers.

If the DBMS does not support updateable views, this statement is also used to generate instead-of triggers to handle updates. The view can optionally be extended by new columns (line 2). Data of these columns is stored in a new table (tuples within this table reference tuples of the view). In this case, instead-of triggers are used to merge and update the content originating from the view's select statement and the new columns.

The materialized MEV-files are made accessible in a directory denoted by the *externalize-in* clause (e.g. as a shared directory).

3.2 Data mapping

The primary intention of an MEV is to make the view's data accessible by users who are using desktop applications. Therefore, the rows inside the base relation(s) must be mapped to a file format that can be handled by such applications. To integrate changes performed on these documents, the transformation has to be performed in the reverse direction as well. Therefore, a bijective mapping description is required.

We specified and implemented two transformation descriptions. The first is a general mapping that can be used with XML-documents. Recently, XML has become more important for desktop applications due to initiatives such as the open document alliance as well as Microsoft's open XML which is used in Office 2007.

```
<company>
  <?MEV %SELECT *
        FROM Employees ?>
    <employee id="%id">
      <firstname>%fname</firstname>
       …
      <?MEV %SELECT *
          FROM Projects
          WHERE leader = $parent.id ?>
        <leaderOf id="%projects.id" />
      <?MEV /?>
    </employee>
  <?MEV /?>
</company>
```

The example above shows a simple transformation statement to bijectively map rows to XML-nodes. To transform rows from the view to the file, this transformation statement is firstly translated to an SQL/XML statement consisting of a set of XMLELEMENT and XMLGEN operators. This temporary result set is then transformed to the file format used by desktop applications by using XQuery.

The statements for the inverse transformation are generated as well from the mapping shown above. It uses XQuery and

SQL/XML's XMLTABLE operator to reconstruct the view's rowset from the document. Using SQL/XML has the advantage that the entire mapping process can be executed by the DBMS itself.

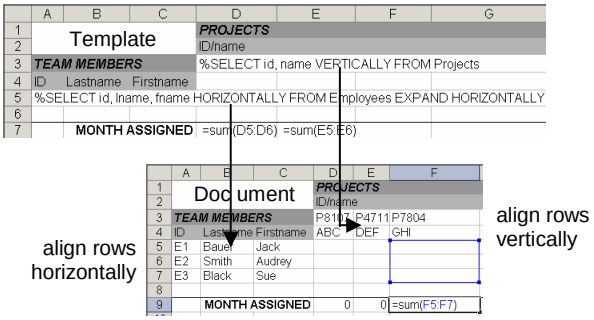

Figure 2. Template based mapping definition

The second way of specifying the MEV-mapping is based on a template document. In our experiments, we used binary Excel files as an example because they are commonly used for personal data management and ad-hoc business processes. We specified statements to map 1:1, 1:N, and N:M relations.

Using an Excel wrapper, the template can easily be mapped to the format described above. For end users the template based approach is more intuitive.

3.3 Working with MEV-documents

In contrast to commonly used import/export processors the mapped content of the view is not stored as a file. Instead, the MEV-document is a virtual file that is being managed by the DBMS itself and that is accessible to desktop applications via a virtual (shared) drive. This mechanism is required to allow 'one' file to be edited by multiple users concurrently. To make DBMS features such as those provided by the ACID-paradigm available we compared two approaches in our experiments.

To allow access for legacy applications, a conflict avoiding approach that uses locks is used (comparable to transactions compliant to the ACID-paradigm). This approach is not applicable for end users due to long edit-periods (locks) and loss of changes due to constraint violations (rollback of work). For end users, we evaluated a conflict-resolving approach that does not require locks but saves changes to an isolated shadow copy. To avoid anomalies due to conflicting changes by multiple users (detected using the wrapper that reconstructs rowsets from the MEV-file and compares them to change-timestamps or change-logs) these have to be handled manually. Compared to other CSCW [9] approaches to detect and handle conflicts in this case a significant difference must be denoted: desktop applications usually access entire files and therefore read all tuples of an MEV. Therefore, dependencies between tuples (e.g. tuples t_1 and t_2 have been read before t_3 has been updated) cannot be discovered. This prevents automatic detection of read-write conflicts. Such dependencies could only be checked by additional constraints. For automatic conflict detection, only write-write conflicts can be considered.

After the detected conflicts have been resolved, constraints are checked. In this approach (and related approaches such as used for snapshot isolation [10]) it is insufficient to just check changes to guarantee serializability. Since integration of changes happens rarely (started by the user after editing of one or all documents is complete), constraints are checked on the entire MEV as well as dependent MEVs in our prototype.

Using this approach, end users can work concurrently on a single file (even the base relation can be changed) and conflicts and errors can be detected without rolling back the transaction (explicit rollback only).

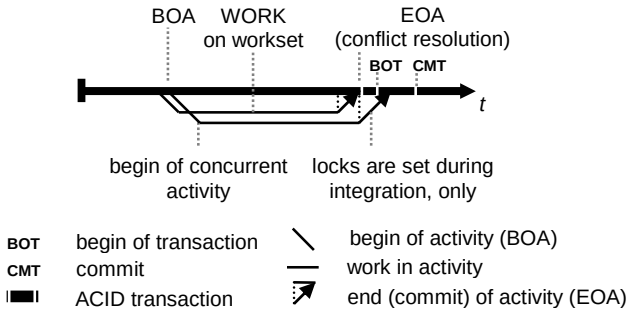

Figure 3. Conflict resolving approach

The figure above depicts how users work with documents using the conflict resolving approach. At the beginning of an activity (BOA) by a user, an isolated shadow copy (SC) is created that is invisible to other users. The SC stores all changed documents (after-images) during this activity by this user as well as the original documents (before-images). These images among with the mapping definition are used to reconstruct performed operations. At the end of an activity (EOA) – which is comparable to a commit in the transactional case – firstly, the change operations are reconstructed. Secondly, these changes are compared for conflicts with changes made by other users. Finally, the changes are checked for integrity (constraint-checks). As mentioned above, constraint violations do not rollback changes. Instead, the user can correct the violations by editing the document again. However, in order to successfully integrate an activity, all conflicts and all constraints must be handled by the user.

4. TURNING DOCUMENTS INTO A STAND-ALONE DBMS

The MEV approach requires users to be connected to the file share provided by the DBMS. Downloading and exchanging MEV-documents by email still bears the problem that only data is being exchanged in documents. Thus, constraints cannot be checked and changes cannot be propagated to backend systems as soon as a user downloads a document (macros are potentially dangerous and are filtered by most enterprise firewalls).

Since all benefits of a DBMS are lost if the document is being exchanged, we turned the document itself into a stand-alone DBMS by combining the document's data with metadata and the execution logic of a DBMS.

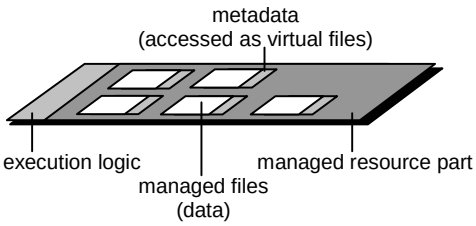

Figure 4. Components of a smart file

The basic principle is based on a mobile DBMS consisting of a single file (fig. 4) that can easily be exchanged by e-mail. This file is executable and contains a managed resource part where common documents (files) can be stored. Metadata can be attached to these files to define integrity constraints or access rights as done in a data dictionary of a DBMS. Since this concept enhances traditional file usage paradigms it is called 'smart' file (SF) [12]. A SF is a file container, comparable to self-extracting zip-archives. It mounts its resource part as a virtual drive. However, any read/write access is managed by the SF itself, it controls who can do what, when, and where. This concept enables a SF to be autonomous and therefore responsible for its contents which opens great possibilities for consistency and security. Because a SF allows storage of documents and still can be e-mailed as easily as any other file, all advantages of document exchange are preserved. For end users, accessing documents managed by a smart file is as simple as accessing them on a file share.

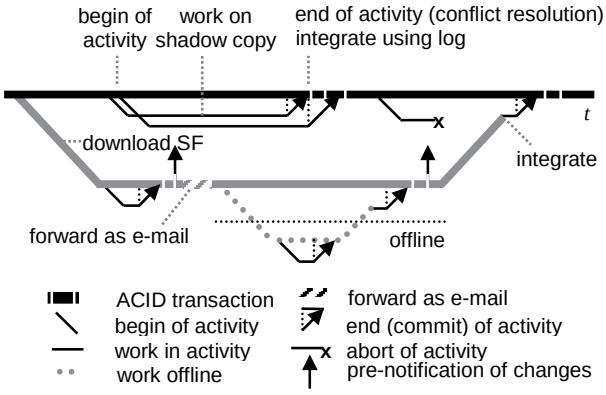

Figure 5. Nested activity based approach

To support reintegration of pending changes and handling of concurrent changes we extended the conflict resolving approach to a so called nested activity based approach depicted in fig. 5.

Downloading a smart file can be compared to starting a new activity. All changes performed on a smart file are stored in its container (similar to the shadow copy approach described above). This approach is referred to as nested because every copy of a smart file (with all changes) is reintegrated into its originating source, only. At the time of reintegration all changes made in activities on that smart file are handled as one activity that has been executed on the originating smart file. Reintegration of this activity can be compared to ending an activity (see section 3.3).

5. EXPERIENCES

To proof that the benefits of working with documents can be combined with proven functionality from DBMS we implemented a prototype that realizes the presented concepts of materialized external views and smart files.

We used this prototype in ad-hoc processes that were executed by non IT experts using spreadsheets. These processes could seamlessly be executed using MEVs and smart files. Compared to other approaches (see table on next page) only mapping definitions had to be defined, the process did not need to be changed and the benefits of working with documents were preserved.

Using the prototype data changed in documents can instantly (online only) be propagated to a central DBMS which e.g. allows real-time data warehousing and monitoring of process execution (BPM). By adding constraints the unpredictable data quality was improved drastically.

6. RELATED WORK

Related work can be categorized into three parts. First, approaches to solve the problems of document exchange. Second, approaches to materialize views and third, approaches that combine data with metadata and execution logic.

In contrast to other approaches, we do not substitute document exchange, even though plenty of alternatives are available today (e.g. WfMS or groupware systems). As described in the introduction, documents are preferred by end users for non-technical reasons that are not preserved by these alternatives (cost, development time).

Instead, we are using well known concepts from DBMS to bridge the gap between DBMS and documents by extending the concept of views. In the literature, plenty of work exists in this domain (e.g. Gray/Reuter, [1]). Especially in the context of data warehouses, problems of view materialization [2] and maintenance [3], [4] are of great importance. Since MEVs are materialized externally, our work is also related to the domain of external data management.

However, since support for document driven processes is not in the scope of today's mainstream DBMS, such processes are usually handled with a mix-up of concepts from federated databases (see [5]), data integration (e.g. with foreign tables in SQL/MED [6]) and data transformation. In contrast to data links, our approach handles external data not as a blob but on a fine grained (attribute based) granularity.

Since all benefits of a DBMS are lost if MEV-documents are being exchanged, we turned such document into a stand-alone DBMS by combining the document's data with metadata and the execution logic of a DBMS. Related work can be found in the domain of electronic forms. These store data, metadata (for layout and simple data types) and scripts to check integrity constraints. Recently, electronic forms gain popularity due to efforts by Adobe (Intelligent Document Platform) and Microsoft (InfoPath). However, this approach is primarily intended for forms. Common documents such as CAD-files or spreadsheets can only be handled as attachments. In Active XML [11], concepts to include data from backend-system into XML documents are described.

criteria	application based	document based	MEV	smart file
spontaneous creation	-	+	+	+
qualification: easy to create	-	+	+	+
training time	+	+	+	+
multi-user support	+	-	+	+
offline data availability	-	+	+	+
offline data integrity checks	-	-	-	+
up to date data availability	+	-	+	+
data quality	+	-	+	+
execution monitoring	+	-	(-)	+
ad hoc changes	-	+	+	+
cost	--	+	+	+

Table 1. Comparison of the approaches

A more general approach is based on active documents (AD) [7]. ADs contain data and control over that data. Therefore, they are commonly referred to be autonomous. Most work with ADs is related to the concept of mobile agents utilizing artificial intelligence. Support for common document driven processes is out of scope. Living documents [8] (LD) are a special kind of ADs (without artificial intelligence) that are used as lightweight document management systems. However, for document driven processes, issues of concurrent editing and concepts of today's DBMS such as transactions to rollback inconsistent changes are important for our approach which are out of scope of LDs.

7. SUMMARY AND CONCLUSION

Documents – such as spreadsheets – are widely (and wildly) used in ad-hoc processes that are executed by non IT-experts. In contrast to enterprise applications and WfMS they can be created and used instantly, and they can be exchanged by e-mail. However, using documents causes problems such as poor data quality and missing up-to-date data in backend systems: good reasons to argue against them.

But document exchange does not necessarily determine loss of all benefits from DBMS. In this paper we described an approach to combine the flexibility of document exchange with proven DBMS functionality.

Firstly, our approach handles data processed in documents as objects that are managed under control by a DBMS. We extended the well-known concepts of views to materialize their contents externally in file formats that are commonly used by desktop applications. Wrappers along with end user maintainable template documents are used to externalize data managed by a DBMS outside the tablespace, i.e. as documents that are used in document driven processes.

Secondly, we focused on the problem that when documents are downloaded, edited offline, and exchanged between users, only data is available. We extended the previous approach by turning documents into a DBMS themselves. Such a smart file combines data, metadata and executable data within a single file that can be exchanged in ad-hoc processes just like a common document. Changes performed on MEV-documents managed by a smart file are checked for integrity and conflicts automatically.

8. REFERENCES

[1] Karenos, K., Samaras, A, et. al.: Mobile agent-based services for view materialization. *ACM SIGMOBILE, Special Issue,* Volume 8, 2004.

[2] Blakely, J.A., Larson, P., Tompa, F.: Efficiently updating materialized views. *Proceedings of the ACM SIGMOD* 1986.

[3] Hammer, J., Garcia-Molina et al: The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):41-48, June 1995.

[4] Zhuge, Y., Garcia-Molina, H., Hammer, J., and Widom, J.: View Maintenance in a Warehousing Environment. *Proceedings of the ACM SIGMOD*, pages 316-327, San Jose, USA, May 1995.

[5] Sheth, A. P., Larson, J. A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys* 22(3), Sep. 1990.

[6] SQL/MED: see ISO/ANSI SQL-99 part 9

[7] Werle, P.: Active Documents to Support Work Processes in a Ubiquitous Computing Environment, *Proceedings of the first workshop on Resource Sensitive Mobile Human-Computer Interaction*, Bristol, September 2000.

[8] Schimkat, R.-D., Küchlin, W.: Living Documents – Micro Servers for Documents, XML-Based Data Management and Multimedia Engineering. *EDBT 2002 Workshops*, LNCS 2490, Prague, Czech Republic, March 2002.

[9] Lee, Y.W.; Leung, K.S.; Satyanarayanan, M.: Operation-based Update Propagation in a Mobile File System. *Proceedings of the USENIX Annual Technical Conference*, Monterey, USA. June 1999.

[10] Berenson, H., Bernstein, P., Gray, J, Melton, J, O'Neil, E., O'Neil, P.: A Critique of ANSI SQL Isolation Levels. *Proceedings of the ACM SIGMOD,* San Jose, USA. 1995

[11] Abiteboul, S., Bonifati, A., Cobena, G., Manolescu, I., Milo, T.: Dynamic XML Documents with Distribution and Replication. *SIGMOD*, 2003

[12] Hilliger von Thile, A.; Melzer, I.: Smart Files: Combining the advantages of DBMS and WfMS with the simplicity and flexibility of spreadsheets. *BTW*. Karlsruhe, Germany. March, 2005.