Versioning and Historiography in Automated Generic Electronic Flight Log Book Transfer

Arne Koschel, **Carsten Kleiner**
Univ. of Applied Sciences and Arts,
Faculty IV (Dept. of Computer Science)
Hannover, Germany
akoschel@acm.org, ckleiner@acm.org

Björn Koschel
edatasystems GmbH

Gelsenkirchen, Germany
bjoern.koschel@edatasystems.de

***Abstract*—The automated transfer of flight logbook information from aircrafts into aircraft maintenance systems leads to reduced ground and maintenance time and is thus desirable from an economical point of view. Until recently, flight logbooks have not been managed electronically in aircrafts or at least data transfer from aircraft to ground maintenance system has been executed manually, since only latest aircraft types (e.g., Airbus A380, Boeing 777) support electronic logbooks. This paper introduces a top level distributed system architecture of a generic system for automated flight logbook data transfer. The system includes a generic mapping component that facilitates flexible mappings between aircraft logbook systems as input and aircraft maintenance systems in the backend. As its main contribution this paper details versioning and historiography concepts for our system. The former makes it possible to deal with different versions of input and output systems in a single mapping component. The latter explains the practically important aspect of historizing data in order to comply with legal regulations. Due to its flexible design the mapping component could also be used for other domains with similar requirements.**

Keywords-System integration; versioning; historiography; aerospace domain; generic interface; flexible data mapping

I. INTRODUCTION

Ground and maintenance time is very costly for airline operators. They try to minimize them for economical reasons. Today's mostly manual transfer of flight logbook data from an aircraft into the operator's maintenance systems should be automated to get closer to this goal. This should reduce information transfer time and is likely to be less error prone as well. Thus, in total it should result in reduced ground maintenance time and cost.

A generic automated flight log data transfer system needs to support differently structured flight log information from different aircraft types and manufacturers on one side. On the other side different aircraft maintenance systems used by different operators have to be supported. Technically, all these systems are likely distributed, even though typically in a common network within a single organization. Moreover, fault tolerance and (transactional) persistence to prevent data loss are required. Not very critical are performance demands due to a limited amount of log data per flight in practise.

To support those requirements, a *generic* transfer system for flight logbooks into maintenance systems needs to be designed and implemented. In a joint industry and research cooperation (*Verbundprojekt*) the eLog system has been designed and prototypically implemented by the University of Applied Sciences Hannover in cooperation with Lufthansa Technik AG and edatasystems GmbH. Technically this system supports different – currently Extensible Markup Language (XML)-based, but with heterogeneous XML schemata – versions of different aircraft flight log systems on the *input* side. On the *output* side different airline systems are supported, which may have different data models. As an example a Relational DataBase Management System (RDBMS/DBMS) with tables is used, that map (almost) 1:1 to the data structures of the *output* system, which is Lufthansa's core aircraft maintenance system. Note that the number of potential input and output systems in a real world scenario will be rather small (e.g. a one-digit number), thus limiting the number of potential combinations of input and output systems and the mappings required.

The resulting eLog system offers a generic distributed system architecture for integration and mapping of the different XML-based flight log input data formats to different output aircraft maintenance systems. The mapping is configurable and flexible, including mapping of arbitrary entities.

Mapping of input objects to output objects is specified by an XML mapping document (conforming to a specific mapping schema) which is dynamically loaded into the mapping component (cf. sec. IV). Information in the mapping document is handled dynamically making the component extremely agile once in operation. In total all these features contribute towards our goal of a generic flight logbook tool. In [13], a high level overview on eLog has been given and [14] discusses the mapping component in detail. This paper's main contribution are details on the versioning and historiography concept for eLog. The versioning concept is important in order to deal with different versions of input and output systems with a minimal operational overhead. Thus, a single (or at least small number) of mapping system instances should be able to take care of several versions of input and output systems with each combination requiring individual mapping specifications. Historiography is important both from an operational point of view (e.g., to limit the amount of data to be held in the operational mapping systems) as well as from a legal perspective as certain accountability requirements for all Information Technology (IT) systems related to aircraft are in place.

 ISBN: 978-1-61208-270-7

Initial tests of the prototypical implementation already validated the practical usefulness of the concept. Very few logbook data transfer systems exist. Those that do are flight operator internal and/or aircraft type and maintenance system specific.

Although concepts and approaches for application/data integration in general of course do exist (important ones are briefly discussed in section II) applying them to flight logbook data is a novelty. To the best of our knowledge a *generic* flight logbook data transfer system has not been implemented yet and is thus the key contribution of our overall work.

The remainder of this article discusses some related work in section II before giving a high level system overview of the eLog system in section III that has been the result of an extensive comparison of options (cf. [13]). Section IV briefly introduces the generic mapping specification before section V discusses as the main contribution the eLog versioning and historiography approach. The article ends with a conclusion and some outlook to future work in section VI.

II. Related Work

Related work originates from different areas. *Conceptually* different enterprise application integration approaches [2], [4], [12] provide a potential foundation for the technical system architecture. The most important ones include: messaging based enterprise integration patterns [7], transactional DBMS based approaches [5], [3], using an Enterprise Service Bus (ESB)/Service-Oriented Architecture (SOA) as foundation [11], and finally an Extract Transform Load (ETL)-like data warehouse concept [8].

All of them potentially deliver feasible architectures for a system like eLog and have been evaluated (cf. [13]). Eventually a transactional DBMS based architecture was chosen similar to an ETL approach. We omit a detailed discussion about the architecture selection at this point. The selection has been undertaken and is described in project internal documentation. It might be published by us in a future article. In this paper we will rather focus on the achieved results.

From an *application* point of view many systems exist, which transfer and map data from multiple input sources to different output sinks. For example, Deutsche Post uses an ESB for XML-based data transfer within a SOA [6].

A generic XML mapping architecture is discussed in [9]. Similarly [19] presents an algorithm for mapping of XML Schemas which we did not use to potentially missing XML schemats on output side. Graph based mapping to transform structured documents is explored in [16]. A tool for semantic-driven creation of complex XML mappings is presented in [17] whereas [20] presents a similar mapping approach to ours but employs XSLT for the mapping step. Moreover (semi-)automated mapping of XML schemas has been discussed in many research papers, a pretty comprehensive survey is given in [15]. However, for the limited complexity of the XML schemas used in flight logbooks the project has decided that the overhead of employing a complex automated mapper, let alone choosing the most suitable one, is too big. Nevertheless the output of an automated mapper could be used as a base to define the mapping documents (cf. section IV) if our approach is applied to more complex domain schemas. Also [18] describes the design of a histoy database but is restricted to a completely different domain.

Looking at recent aircraft models in particular shows that only the latest models support electronic log books. In addition standardization of the data format is still in its early stages. Recently initial standardization has been designed in the ATA specification [1]. This specification is quite helpful for the flight log input data within our work – although it is still significantly in flux. One e-logbook tool for the aerospace industry is presented in [10]. But no *generic* flight logbook data transfer system has been documented yet.

III. eLog: System Overview

The designed generic flight log data transfer system is based on ideas frequently found in ETL processes in data warehouse systems. Note though that the implementation itself is not based on data warehouses but rather uses a transactional DBMS-based approach as stated in the previous section. In order to provide a brief eLog system overview we explain the data flow within eLog in the sequel.

A. Data flow within eLog

Data flow within eLog follows a sequence of steps to map the XML input data to arbitrary aircraft maintenance systems. Figure 1 shows the high level input data flow from the flight logbook system until procured to the maintenance system.

An input reader component – where different implementations for different source data formats may exist – uses polling (step 1) to check whether new XML files have been delivered by aircrafts. Polling is implemented by frequently checking a predefined directory on a dedicated server for newly added files. The data is validated against the XML schema (step 2) of the particular aircraft's electronic logbook format, e.g., against those defined in [1]. If the data is formally valid, it is transferred into a buffer database (step 3), where it is stored in its original format in an *XML-type* attribute. Else an error handling takes place. As long as the covered aircrafts

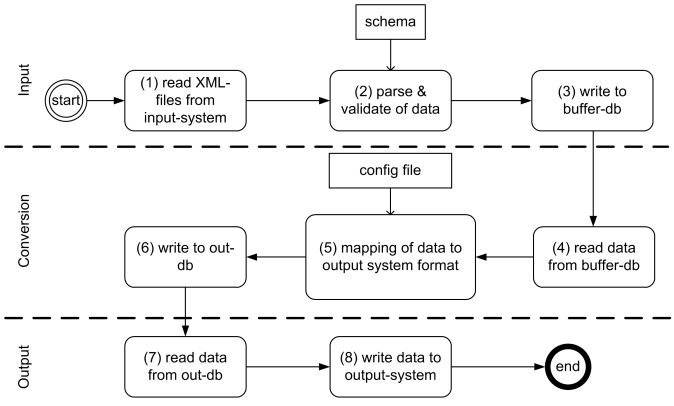

Fig. 1. Generic eLog data flow

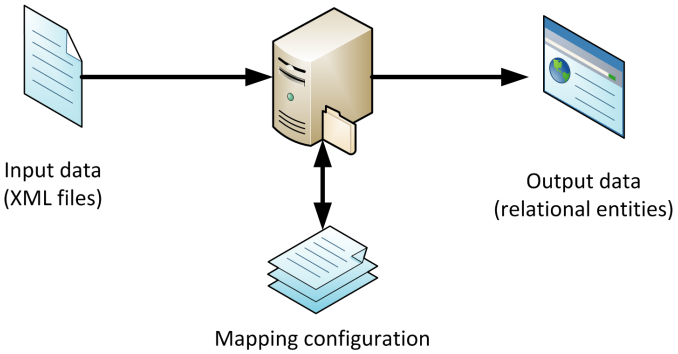

Fig. 2. XML based mapping process

provide source data in XML format a single database schema is sufficient for the buffer database.

Again using polling a mapping component checks for newly arrived source data (step 4) in the buffer database. It utilizes a flexibly configurable sequence of conversion functions to map the input data to a specific output target system (step 5); this step is explained in detail in section IV. The output data is stored in a database again (step 6). It closely resembles the data structure of the airline's maintenance system.

Consequently for each different maintenance system there will be an individual schema in the output database. Options in the mapping configuration include checks for dependent source information, flexible mapping and features to update existing entities in the output system; cf. details section IV.

Eventually another upload component transfers data from the output database (step 7) into the airline maintenance system (step 8) whenever an entity of the maintenance system has been completely assembled. As one option the airline maintenance system used provided a Web services interface for programmatic access.

IV. Mapping Rules

The mapping of domain specific information between input and output systems is expressed by XML mapping files conforming to a specific mapping schema. There is an individual mapping file for each entity of any input system that is triggering the creation of an entity in the output system (cf. fig. 2). Mapping files can take care of any type of mapping between input and output on entity, attribute and attribute value level. Mappings on entity level use an individual mapping specification for each of the input entity types. Within these entities nested elements specify the attribute mappings by individual converters whereas transformations on attribute values are defined on the innermost level within the target attribute specification.

For eLog it is important to note that there is an individual mapping configuration for each combination of input and output system where a mapping has to take place. This includes different versions of a system on each side, i.e., mapping from a different version of the same input system leads to a new mapping configuration file to be used. This has the advantage from an operational point of view that each combination and version of input and output system can be configured and operated independently of other potential mappings.

In the case study an implementation of the whole mapping process in Java has been performed. Technically the system is designed according to the ETL paradigm and it is implemented with different Java processes which together provide the tasks from figure 1. They are combined with a relational DBMS, that also allows for XML data storage (Oracle). Throughout the conversion steps DB transactions are utilized to ensure data consistency. In combination with operating system based fault tolerance (automated process restarts), a high degree of fault tolerance of the overall system is thus achieved.

The Java mapping application dynamically reads the mapping specification documents and checks for syntactical correctness. Thus, changes in the mapping specification simply require a restart of the mapping process without any changes to the source code. After starting, the transformation process polls the input folder (or database) for newly arrived XML input. Whenever the first input entity of a given type arrives, the mapping specification document is used to instantiate the required converters and functions as Java objects. The names of the converter XML elements are used to instantiate a corresponding Java class using reflection. This enables dynamic provisioning of converter classes and facilitates complex converters as the full power of the Java language can be used for implementation. Arguments of the converters and functions are provided to the Java classes based on the specific objects to be processed. Thus, additional converters and functions can be easily implemented and added to mapping processing by observing given interfaces without any changes to the mapping process source code. After successful mapping of an input entity the result is procured to the output system (a relational database in our case) with standard JDBC operations. Note that while the first step and the instantiation of mapping objects in the Java application are executed only once, the other steps are executed for each input entity with the number of executions depending on the particular mapping specification.

V. Versioning and Historiography

While the previous sections provided a brief general overview about eLog summarized from [13], [14], this section newly contributes approaches for two additional requirements, namely versioning and historiography within eLog.

Versioning within the eLog context in particular means to have independent version changes for the attached input systems such as ATA conformant systems as well as for the attached output airline maintenance systems. In principal arbitrary combinations of input and output system versions should be possible for eLog.

Historiography in the eLog context means what happens to the different input and output data within eLog's databases. Since this is mostly future work for eLog we only briefly sketch pragmatic ideas, which nevertheless might well form the basis for a pragmatic solution in this space.

This section will first examine the different resources within the eLog system architecture, which are affected by versioning.

 ISBN: 978-1-61208-270-7

It is followed by a look at some additional assumptions regarding foreseeable eLog usage. A discussion of several options to enable versioning for eLog based on the resources and the additional assumptions is performed. The section concludes with a look at eLog's historiography concept.

A. Versioning within eLog

1) Common resources in eLog affected by versioning: Taking a closer look at the eLog system architecture as sketched in figure 1 shows several resources, which are commonly used within eLog. Table I describes them in more detail.

2) Additional assumptions for versioning in eLog: Discussion with potential future users of eLog directed us to a few additional assumptions:

- There will be different versions of input data systems.
- There will be different versions of output data systems.
- Due to typical time and release schedules, the actual number of different versions on both sides – input and output – will be less then 10 per system.
- Individual administration is of importance on a 'per version' basis. Especially important is a very low impact of one system version (such as one particular Airman version on input side) to other system versions.
- A 'pragmatic' solution for version control with base technologies such as RDBMS and comparatively simple data structures is prefered by the proeject partners compared to an 'over-engineered' approach. For example, ontologies or version control systems are not required here.

3) Discussing input/output system versions for eLog: While the versioning discussion in principal is required for all of the mentioned system environments and common resources from table I, we will – due to space limitations – focus here on the versioning of the eLog database(s). Please note, that this nevertheless forms a baseline for a similar discussion of all those resources. For example, one could use a particular version of the eLog program 'tied to' a certain version of the eLog database or its users and schemas.

The discussion of versioning for the eLog database(s) is of particular importance, since it affects pretty much all other resources. We also do take into account for the discussion the 'additional assumptions' from above however. Based on an example we examine different approaches for versioning of the eLog database(s) below.

TABLE I
eLog Resources

System Environment	*Key Common Resources*
Physical or virtual machines to run the operating system	DB schema (user data, flight log entities, etc.)
Databases management systems such as Oracle 11	eLog program versions (and their respective run time process incarnations)
A certain runtime environment. For eLog Java virtual machines.	eLog data directories within the file system
	eLog configuration information

Typically a (relational) database installation will have a few database instances, which will have some schemas that will consist of some tables. For the following approaches let there be 2 input systems E,F with 3 version 0,1,2 as well as 2 output systems R,S also with 3 versions 0,1,2. From this example – which adheres to the 'additional assumptions' from above and would also be reasonably typical in practise for our project partner – the following versioning concepts arise:

- D1 – 1 DB instance per input/output system
  One full database instance per input/output system combination would allow for the best decoupling between different input/output system combinations. However, the price for many full DB instances is of course a relatively high overhead.
  Our example would result in 12 DB instances, one for each of the three versions of each of the four systems.
- D2 – 1 schema per input/output system and version
  In this concept only 1 database instance would be used. The separation between input/output system and resp. versions would be performed based on different DB users with their individual DB schemas.
  For the example this concept results in $\{E_0, E_1, E_2, F_0, F_1, F_2, R_0, R_1, R_2, S_0, S_1, S_2\}$ thus 12 DB schemas within a single DB instance, one schema per input and output system and version.
- D3 – 1 DB schema per input/output system combination for all versions
  The number of different schemas could be reduced by using the same schema for all versions of a single input or output system. In this option all tables for a single system would be multiple for each version.
  For the example this gives $\{E_{0,1,2}, F_{0,1,2}, R_{0,1,2}, S_{0,1,2}\}$ thus 4 DB schemas with three times the tables each compared to D2. While the number of schemas is reduced significantly, the decoupling of input/output system combinations is much lower compared to D1 or D2 leading to increased dependencies during operation.
- D4 – 1 DB schema for all input systems and 1 DB schema for all output systems
  The lowest meaningful number of schemas would be just one for the data from all input systems and another one for all output systems.
  Here $\{(E, F), (R, S)\}$ would be the result for our example. This gives only two schemas, thus relatively low overhead but only minimal decoupling as the price.

A solution for a good relation between system / DB maintenance effort versus decoupling of systems may be found by looking at table II.

Option D2, which consists of 1 DB instance and 1 DB schema per input/output system and version provides a pragmatic solution with a suitable flexibility combined with a reasonable DB administration effort. It does require a substantial number of DB users – the sum of number of input systems and their versions plus number of output systems and their versions. However, the systems are decoupled. They can be

 ISBN: 978-1-61208-270-7

TABLE II
DB VERSIONING CONCEPT: EVALUATION

System	*Evaluation*	*Comment*
D1	–	Max. decoupling of versions, quite high overhead
D2	+	Maintenance effort vs. decoupling in a reasonable relation
D3	+-	Decoupling lesser, maintenance effort higher
D4	–	only 2 DB users required, but no decoupling for maintenance tasks given

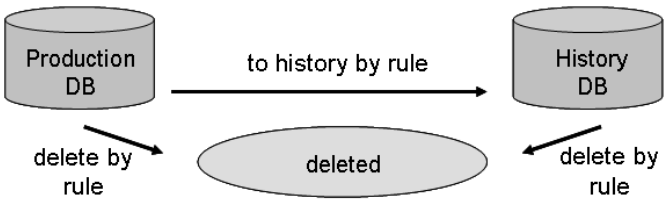

Fig. 3. Garbage Collection

maintained individually on a per version base, e.g., allowing for individual system starts and stops, backups or updates.

In total option D2 is thus the authors recommendation for eLog. However, individual airline IT environments, maintenance procedures or accounting strategies may have to lead to other options, e.g., to option D3. This would provide fewer flexibility, but it would limit the number of required DB users and thus reduce the DB administration effort to some degree.

Beside maintenance aspects the single point of failure aspect also has to be considered, at least, whenever system resources are shared. For this reason it is recommended for system resources, to have them separated whenever possible. For example, a shared DBMS as in option D2 might be used, but within a highly available environment. For separate eLog instances individual virtual or even physical machines shall be used. The number of parallel installations seems reasonable, while the advantages of independent administration, maintenance, and fault tolerance should be significant.

B. Historiography and 'Garbage Collection' for eLog

Persistent data within the eLog context resides in the buffer and in the output data base (cf. section III). Over time, the data stored there becomes outdated and thus – at least if in the future many aircrafts deliver input data for the system – the stored data wastes database memory in the eLog production databases. Still however, for control and accounting purposes outdated data should remain accessible in some form. For this reason some kind of data historiography and 'garbage collection' is a required feature for eLog. The base requirements for data historiography and garbage collection in eLog are thus:

- R1: Archive database
  Historiography and garbage collection in the eLog context means to archive data from the production databases into some other long term archive database according to certain rules (see below). When those rules apply the associated data must be moved from eLog's production database to an eLog archive database. Figure 3 sketches this process.
- R2: Rules for data historiography
  It shall be possible to declare certain (simple) time bound rules, which specify when and which data is moved to the archive database from R1 (and deleted afterwards).
- R3: Rules for data deletion
  Certain data items within eLog only serve for internal purposes, for example internal key and reference information for data mappings in eLog's output database. Once this information is outdated, it might safely be deleted ('garbage collected') with no need for further archiving. As in R2 it shall be possible to declare certain (simple) time bound rules, which specify when and which data might safely be deleted from eLog's production database.

Looking at those core requirements and the requirement of a pragmatic, established approach, the following solution seems sufficient: Conceptually each attribute gets two constraints (or markers). One specifies when it must be moved from the production database to the history database at the earliest (and deleted afterwards). The second specifies when the data item may be deleted from either production or history (whereever it resides) at the earliest. Please note, that from a regulatory persepctive may be a real deletion in the history table could even be 'never' or only in a very long time frame. At the moment, out project partners do not see this demand.

In table III fictive examples (although practically reasonable acc. to our project partners) are given for archiving and deleting certain flight log and maintenance data after a certain time period.

Technically again a pragmatic, proven solution is eLog's approach of choice. Periodic database jobs, which use triggers and stored procedures are a sufficient implemenation for eLog. External database programs, which run periodically, would be an alternative.

VI. CONCLUSION AND OUTLOOK

A. Conclusion

So far the concept and prototypical implementation of eLog have proven to be very promising. The implementation already covers a single exemplary input and also output system. It includes entities that require almost all possible mapping options between the systems. The mapping configuration as described above is defined in an XML file based on a proprietary schema.

TABLE III
TYPICAL ENTRIES: HISTORIOGRAPHY AND GARBAGE COLLECTION

Database	*Entity*	*toHistory after*	*toDelete after*
Buffer_DB	MaintLog	1 week	3 weeks
	MaintAction	1 week	3 weeks
	MaintRelease	no	3 weeks
Out_DB	mapping of MaintLog	no	3 weeks
	mapping of MaintAction	no	3 weeks
	mapping of MaintRelease	no	6 weeks

 ISBN: 978-1-61208-270-7

Different versions of input and output systems are an integral part of eLog's architectural concept (cf. section V). The mentioned XML schema may easily be adjusted to different input and output formats (for different versions of the same or other logbooks and maintenance systems). The mapping specification is read dynamically by the mapping component which makes easy and fast adjustments to different versions or products possible without any software development, just by configuration. The current output DB resembles the simple relational structure of most airline maintenance systems but may also be adjusted for a different maintenance system. Both mapping configuration as well as input and output data formats are sufficiently generic in order for the system to be easily adjusted to specific data formats. They might well be usable in other domains as well.

The overall modular design of the system by decoupling input and output system leads to a highly scalable overall architecture. Dealing with different versions and output systems will be a major issue as, e.g., logbook data will be procured to different output systems dealing with specific aspects of aircraft management. Since the actual number of different versions in an installation in a typical time period will in practise likely be small enough (in total likely less than 20), the scalability of the overall architecture will hold true.

Consequently, eLog's solution for versioning – as discussed in section V – is of highly practical importance for a working interface component. Moreover, the general discussion of versioning aspects could well be of practical importance for other systems as well.

Elog's pragmatic concept for data historiography based on history tables and (simple) rules could also be a solution for other systems with similar (relatively small) requirements.

B. Outlook

After a brief successful evaluation of the first eLog prototype we have finished the conceptual and detailed specification phase of eLog's 2^{nd} project stage and are about to finish this phase's implementation.

There the genericity of the mapping specification already proved to be quite useful as a different logbook provider was used than in the first stage. Also, the schema of the output system has changed from a relational schema towards a more XML-oriented schema using domain-specific objects from the aircraft maintenance domain. Moreover as of late two slightly different output systems will have to be procured positively testing the feasibility of our generic mapping approach as well as the versioning.

The next steps include evaluation of eLog in a production environment. All this will provide additional proof for the scalability and reliability aspects. Within the mapping specification we might also consider as future work some examination of ontology based approaches. Also, we will have to generalize the format of the mapping specification to an an XML schema in order to do development/compile time correctness checking of specifications in the future.

In summary, automated integration of flight logbooks shows the potential for reduced transfer times of maintenance information coupled with increased correctness when compared to the current manual process. This will eventually lead to reduced maintenance times for aircrafts and thus increase profitability of the airline. The presented real world eLog project is a very promising step into this direction.

ACKNOWLEDGMENT

The authors thank our cooperation partner Lufthansa Technik AG and our master students A. Hödicke and S. Nitz. The overall project is part of the Aviation Cluster Hamburg Metropolitan Region's Leading-Edge Cluster Strategy and is sponsored by the German Ministry of Education and Research.

REFERENCES

[1] Air Transport Association of America, Inc. (ATA), "Industry Standard XML Schema for the Exchange of Electronic Logbook Data," 1.2 edition, May 2008.

[2] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch, "Enterprise Application Integration," Spektrum, Germany, 2005.

[3] C.J. Date. "An Introduction to Database Systems, 7^{th} edt.," Addison-Wesley, U.S.A., 2003.

[4] J. Dunkel, A. Eberhart, S. Fischer, C. Kleiner, and A. Koschel, "Systemarchitekturen f. Verteilte Anwendungen," Hanser, Germany, 2008.

[5] R. Elmasri and S. Navathe, "Fundamentals of Database Systems, 5^{th} Edt.," Add.-Wes., U.S.A, 2006.

[6] M. Herr, U. Bath, and A. Koschel, "Implementation of a Service Oriented Architecture at Deutsche Post MAIL," In ECOWS, LNCS vol. 3250 Springer, 2004, pp. 227–238.

[7] G. Hohpe and B. Woolf, "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions," Addison-Wes., USA, 2003.

[8] W. H. Inmon, "Building the Data Warehouse," Wiley, U.S.A, 2005.

[9] W. Jiyi, "An Extensible XML Mapping Architecture," Chinese Control Conference (CCC2007), July 2007, pp. 291–293.

[10] A. T. Kavelaars, E. Bloom, R. Claus, et al, "An Extensible XML Mapping Architecture," In IEEE Transactions On Aerospace And Electronic Systems, 45(1), U.S.A, 2009.

[11] D. Krafzig, K. Banke, and D. Slama, "Enterprise SOA: Service Oriented Architecture Best Practices," Prentice Hall, U.S.A, 2005.

[12] D. S. Linthicum, "Enterprise Application Integration," Addison-Wesley, U.S.A, 1999.

[13] O. Hunte, C. Kleiner, U. Koch, A. Koschel, B. Koschel, and S. Nitz, "Automated generic integration of flight logbook data into aircraft maintenance systems," 17^{th} GI/ITG KiVS, OASIcs, Dagstuhl, Germany, 2011, pp. 201–204.

[14] C. Kleiner and A. Koschel, "Towards Automated Generic Electronic Flight Log Book Transfer," 15^{th} IC on Business Information Systems (BIS 2012), Springer, Germany, LNBIP 117, 2012, pp. 177–188.

[15] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," The VLDB Journal, Springer, U.S.A, vol. 10(4), Dec. 2001, pp. 334–350.

[16] A. Boukottaya and C. Vanoirbeek, "Schema matching for transforming structured documents," Proc. 2005 ACM symposium on Document engineering. ACM, U.S.A, pp. 101–110.

[17] A. Morishima, T. Okawara, J. Tanaka, and K. Ishikawa, "SMART: A tool for semantic-driven creation of complex XML mappings," Proc. SIGMOD 2005. ACM, New York, NY, U.S.A., pp. 909–911.

[18] Z. Youzhi, P. Peng, and Z. Geng, "Design of History Database for Networked Control Systems," CCC2007, 2007, pp. 292–296.

[19] L. Checiu and D. Ionescu, "A new algorithm for mapping XML Schema to XML Schema," International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI), May 2010, pp. 625–630.

[20] M. Roth, M. Hernandez, P. Coulthard, et. al., "XML mapping technology: Making connections in an XML-centric world," IBM Systems Journal, vol.45, no.2, 2006, pp. 389–409.

 ISBN: 978-1-61208-270-7