Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2010 Proceedings

Americas Conference on Information Systems (AMCIS)

8-2010

Supporting the Production of High-Quality Data in Concurrent Plant Engineering Using a MetaDataRepository

Juliane Blechinger
Friedrich-Alexander University of Erlangen and Nuremberg Department of Computer Science, juliane.blechinger@informatik.uni-erlangen.de

Frank Lauterwald
Friedrich-Alexander University of Erlangen and Nuremberg Department of Computer Science, frank.lauterwald@informatik.uni-erlangen.de

Richard Lenz
Friedrich-Alexander University of Erlangen and Nuremberg Department of Computer Science, richard.lenz@informatik.uni-erlangen.de

Supporting the Production of High-Quality Data in Concurrent Plant Engineering Using a MetaDataRepository

Juliane Blechinger, Frank Lauterwald, Richard Lenz
Friedrich-Alexander University of Erlangen and Nuremberg
Department of Computer Science
Chair for Computer Science 6 (Data Management)
Martensstrasse 3
91058 Erlangen, Germany
{juliane.blechinger, frank.lauterwald, richard.lenz}@informatik.uni-erlangen.de

ABSTRACT (REQUIRED)

In recent years, several process models for data quality management have been proposed. As data quality problems are highly application-specific, these models have to remain abstract. This leaves the question of what to do exactly in a given situation unanswered.

The task of implementing a data quality process is usually delegated to data quality experts. To do so, they rely heavily on input from domain experts, especially regarding data quality rules. However, in large engineering projects, the number of rules is very large and different domain experts might have different data quality needs. This considerably complicates the task of the data quality experts. Nevertheless, the domain experts need quality measures to support their decision-making process what data quality problems to solve most urgently.

In this paper, we propose a MetaDataRepository architecture which allows domain experts to model their quality expectations without the help from technical experts. It balances three conflicting goals: non-intrusiveness, simple and easy usage for domain experts and sufficient expressive power to handle most common data quality problems in a large concurrent engineering environment.

Keywords (Required)

Data quality, concurrent engineering, metadata, rules.

INTRODUCTION

`Fitness for use` (Wang and Strong, 1996; Strong, Lee and Wang, 1997; Scannapieco and Batini, 2005; Wang, Ziad and Lee, 2000), `meeting user expectations` (English, 1999) and `conformity to specifications` (Kahn, Strong and Wang, 2002) are some of the best-known definitions for data quality. Although there is no standardized data quality definition all the most-cited definitions have two things in common: Data is only of high quality if the users can use it satisfyingly for their task at hand and if the specifications and business rules are met. While these definitions are quite sensible, they are too abstract to provide guidance to concrete data quality projects. This leaves a gap between the abstract definitions and quality models and their implementation in real projects.

In this paper we introduce our approach that narrows this gap by defining a system architecture for a specific application area, namely plant engineering. We, however, expect this architecture to be applicable in other areas as well.

The basic idea relies on characterizing the existing data quality problems in an application area and subsequently ensuring their prevention with means of data quality rule types. The novelty of this approach lies in storing these rules in a separated MetaDataRepository (MDR) and allowing domain experts to express and alter rules without the help of technical or data quality (DQ) experts. This is important for two reasons: First, the number of rules in real-world projects can become very large, making their definition by dedicated experts very expensive. Secondly, to cite the above definition by (English, 1999), user expectations may be highly individual which makes it unfeasible to impose rules defined by a heavyweight management process. We believe that users should be empowered to work on their DQ problems themselves, deciding by means of their own DQ rules which problems to solve most urgently. This only makes sense if appropriate tools are available, which we strive to provide.

In the following sections we will introduce our approach by firstly characterizing the DQ problems and specific system requirements in our application domain of plant engineering. Secondly, after sketching the features of two state-of-the-art DQ

products we show how these products fail to solve the problems in large engineering projects. Then we propose our idea, the MDR, for ensuring a high data quality in plant engineering before we conclude.

PLANT ENGINEERING – BASICS AND DATA QUALITY ISSUES

To generalize, the application domain of plant engineering can be attributed to the engineering of huge physical objects that are only built on explicit order and consist of many composite items. According to their functionality and construction, these items can be classified into itemtypes. Other domains with the same characteristics are for example ship building, astronautical engineering or the design and construction of any kind of industrial factories, e.g. for the chemical industry.

In the following, we describe the general information flow in this application area, the arising DQ problems and resulting requirements.

Information Flow

Every huge, composite object that is built on explicit order has three major stakeholders: The engineers who design the object based on initial inputs, the management who receives orders and controls its own engineering department, and the ordering party. Additionally, there are the craftsmen who finally build the object based on the item specifications of the engineering department. The craftsmen can be under the control of the ordering party or of the management, depending on the contract. If the craftsmen on site determine wrong specifications, they communicate these problems – depending on the contract - either to the ordering party or directly to the management. Besides, the management is informed personally in a non-standardized oral or written form about the engineering progress via human reports. Based on these reports and the problems on site, they give the engineering department advice. To fulfill their tasks, the engineers use various interdependent information management systems (IMSs); problems that are detected in the item specifications are reported backwards to iterate.

These information flows are depicted in Figure 1.

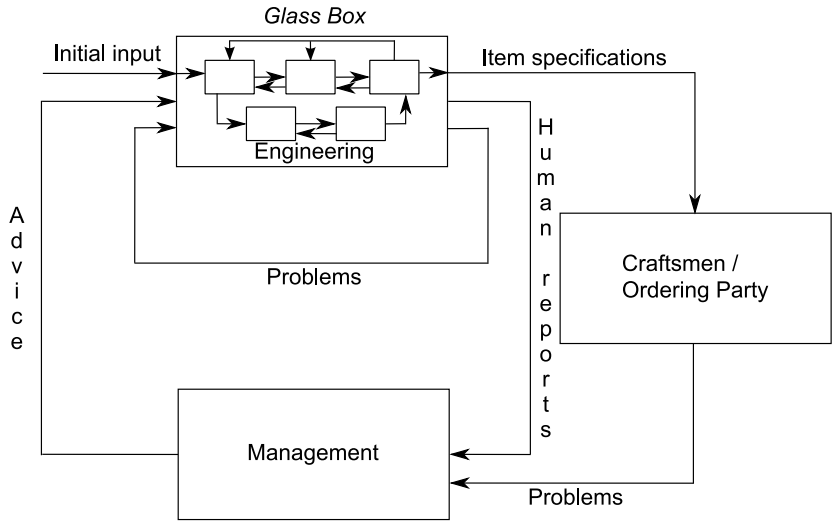

Figure 1. Information flows between engineering, management and ordering party

The IMS architecture can be described as a so-called glass box. In contrast to a black box that can only be viewed in terms of its input and output, a glass box is a system where we can additionally take a look into its internal activities but are not allowed to change anything. The reason for that is threefold:

1) Projects with the criteria mentioned above are huge and last several years. Of course it is not recommended to change the whole IMS architecture during a running project.
2) The IMS architecture in such huge and long-lasting projects consists of many subsystems that are interdependent, providing many interfaces to adjacent tools with upstream and downstream information flows. That is why it is hard to change the whole architecture. Changing only a subsystem is not recommended and would not lead solve the overall DQ problem. This is due to the fact that almost all of the DQ problems originate from the whole architecture and not from only one subsystem.

3) Finally, changing the IMS architecture is a management decision and can hardly be influenced by the engineering department. So, the required DQ solution must handle future IMS decisions.

Inside the glass box, the engineering follows the principle of concurrency. Concurrent engineering implies that there are several process steps that can proceed in parallel. However, many of them are interdependent, requiring designers to proceed with partial information, with incomplete knowledge, and subjective interpolations (Prasad, Morenc and Rangan. 1993; Prasad 1999). To make parallel design of interdependent process steps possible, early preliminary upstream information has to be shared with downstream design stages. Each iteration in the design results in changes that must propagate through all dependent design stages. Consequently, (iterating) rework, upstream and downstream, can be needed to reach a consistent design (Yassine and Braha. 2003).

Data Quality Problems

After comprehensively observing the architecture, analyzing the real data, and interviewing engineers of many design steps in our application domain, we have recognized the following DQ problems that we split into problems of unavailable information for the engineers, problems of wrong data values, and necessary management information.

Information Deficits for the Engineers

This category consists of information that is currently not provided to the engineers but would help them regarding decisions in their daily work and prohibit mistakes resulting in wrong data values. For example, if an engineer easily and immediately gets the information about the master source of an attribute that seems incorrect to him/her and whom to contact to talk about this concern, s/he will be prevented from illegally changing the attribute resulting in a huge rework cycle up- and downstream.

The reason why the following information is currently lacking is due to their absent incorporation into state-of-the-art plant engineering tools. Of course, the information is available in the enterprise, but their detection is very tedious because of the necessity of consulting many different information sources and/or persons. That is why the engineers are tempted to trust their own judgment, skipping process-oriented feedback loops. Finally, this leads to very late error detection, often not before placing the purchase order or even starting the construction on site, resulting in huge iteration loops.

In detail we have identified the following information shortages:

a) Responsibility and trustworthiness: For the attribute values the engineer receives, s/he needs to know the name of the person
   - who has filled the attributes (executor)
   - who is responsible for the filling (person in charge)
   - who has reviewed the attributes before their release
   - who owns the attributes

   and by using which tool or method (e.g. direct SQL, third party tool, primary tool) the attribute was filled.

b) Incompleteness: The engineer needs to know to which itemtype the item s/he designs belongs. This makes it possible for the engineer to see what attributes have to be filled and through what path the item can be derived. We call the corresponding help structure the tree structure of itemtypes (see Figure 2) that we have derived by analyzing the actual data, documentations and the information products: the data sheets containing the final item specifications. Additionally, the attributes get priorities to mark the attributes that need to be filled first.
   As an example, Figure 2 shows the classification path to the existing itemtype *ControlValvePlugEMSR*. This way, the engineer gets the information that items of this type have the control function, two separate inputs and a plug moving device (see (Dubbel, 2004) for valve-specific explanations). Every sub-class has its relevant – optional and mandatory - attributes (omitted in Figure 2) so that the joined result forms a complete item.
   Besides, it shall be mentioned that our hierarchical classification is compatible to the classes of the ISO15926 (a standard for the integration of life-cycle data for process plants), especially to the initial reference data in part 4 where also an exemplified valve classification can be found (International Organization for Standardization, 2007). In addition to the ISO15926, our model includes also an attribute classification and the mapping of existing itemtypes in plant engineering.

c) Problems because of concurrency: The parallel and iterating nature of concurrent engineering poses some unique problems that form this category.

- Validity: Mistakes in form of released but invalid items arise due to the fact that the engineers are not oblivious of release dependencies. For example, attributes in the master process module must be released before the more specific attributes in slave modules are allowed to be released, too.

- Notification: If already released items change in their master IMS, engineers working with copies in downstream IMSs must be notified about the change. Additional information to what point in time the changed values have to be incorporated into the existent design should also be provided. Likewise detected errors in slave IMSs have to be notified to the responsible engineer in the master IMS. Providing this information would help preventing downstream engineers from changing the attributes themselves.

- Progress: In concurrent engineering, huge problems arise because of waiting for attributes or whole items that are necessary for subsequent design stages. That's why the engineers need information about what has to be filled in which design step until when and by whom.

- Transfer: For all the points mentioned above the engineer needs an overview about source and destinations of attribute and item transfers with the corresponding master IMS and the timestamp of the last transfer.

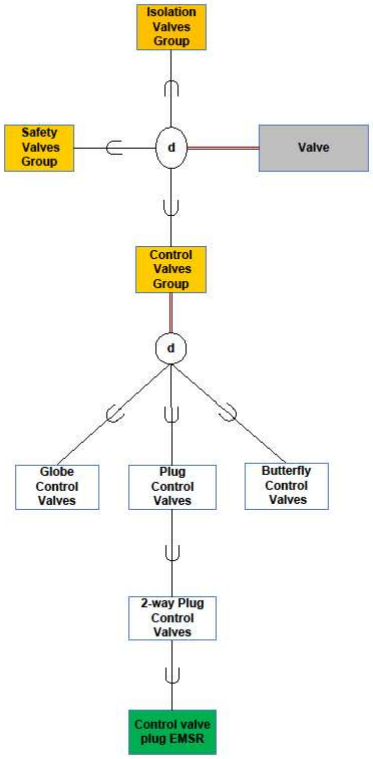

Figure 2. Tree structure of itemtypes (extract of valve classification – attributes omitted)

Wrong Data Values

This category consists of data error types that frequently exist in plant engineering and could be detected by applying appropriate completeness, correctness, and consistency rules.

These error types are already extensively classified in the literature; for example in the taxonomies of (Kim, Choi, Hong, Kim and Lee, 2003) and (Oliveira, Rodrigues, and Henriques, 2005). Besides, definitions for the DQ dimensions completeness, accuracy/correctness and consistency with detailed explanations are given in (Batini and Scannapieco, 2006).

a) Incompleteness: In our tree structure of itemtypes (see Figure 2) it must be specified in the right classification step what attributes are necessary to completely design an item of the corresponding type. Optional and mandatory attributes are possible.

b) Incorrectness: To prevent wrong data values regarding range or type violations, rules must be specified that define necessary attribute ranges, lists of values, formats or impossibilities (e.g. delivery dates in the past). Of course, not all erroneous data can be detected by such rules (e.g. safety class 1 is specified, but the real class is number 2). Validated replicated reference data can be used to correct mistakes, if available. Attribute dependencies might also help to detect mistakes. Otherwise, only manual inspection can solve the problem (Kim et al., 2003).

c) Inconsistency: Concurrent engineering causes the following types of inconsistencies:

- Different values of replicated items in different IMSs.
- Different numbers of replicated items in different IMSs (e.g. the count of valves has to be the same in several valve design sub-systems).
- Mismatches between adjacent items (e.g. a valve that restricts the fluid of a corresponding pipe has to be compatible with that pipe so that the materials are appropriate for the fluid in both items, and of course the pipe diameter has to satisfy the valve's one).
- Mismatches between attributes of one item that have conditional functional dependencies (Fan, Geerts and Jia, 2009) (e.g. the type of a valve constrains the allowed nominal diameter range of it or the safety class constrains allowed materials).
- Mismatches between items and corresponding catalogs (e.g. to order a valve from a specific supplier it has to satisfy the modeling ranges of the according catalog).
- Mismatches between items and the corresponding system environment (e.g. a cooling system should have cooling item types).

Information Deficits for the Management

In addition to the engineers, also the management needs information about the whole design progress of the running project to make overall decisions regarding resource allocations or completion time estimations, for example. Using current state-of-the-art plant engineering tools it is difficult to get an overview of the amounts of finished items, open items, items in progress, and design stages where bottleneck problems arise. As a consequence the management needs as decision support aggregated information in the following areas:

a) Progress: Show the management aggregated information about the amounts of finished and uncompleted items with corresponding design stages and intended completion dates. This way the management can identify overloaded design stages and possibly shift employees.

b) Responsibility and trustworthiness: Provide information about responsible persons for each design stage and review.

Additional Requirements

Additionally, the static IMS architecture requires an adaptable and IMS neutral DQ concept. Due to the glass box characteristic of the IMSs, and the project-related quality conditions and attribute assignments in this application domain, the

DQ solution must be separated from the IMSs. This way we are compatible with future IMSs and necessary modifications of rules in new projects.

Finally, because of the fact that only the engineers have comprehensive knowledge about valid completeness, correctness, and consistency rules, they have to fill in the rules into the intended DQ solution. So, the solution needs to be easy to use with understandable attribute names, rule type definitions and an appropriate rule language.

COMPARISON OF EXISTING TOOLS WITH OUR REQUIREMENTS

According to the existent environment described above, a tool to improve data quality must be located outside of the engineering glass box. Furthermore, to help both engineers and management, the tool will take the input from the engineering glass box and provide information to both. The management will be provided with information based on engineering data, and the engineers get feedback in form of information and design mistakes derived from rule definitions. Figure 3 integrates this DQ concept into the information flows based on Figure 1.

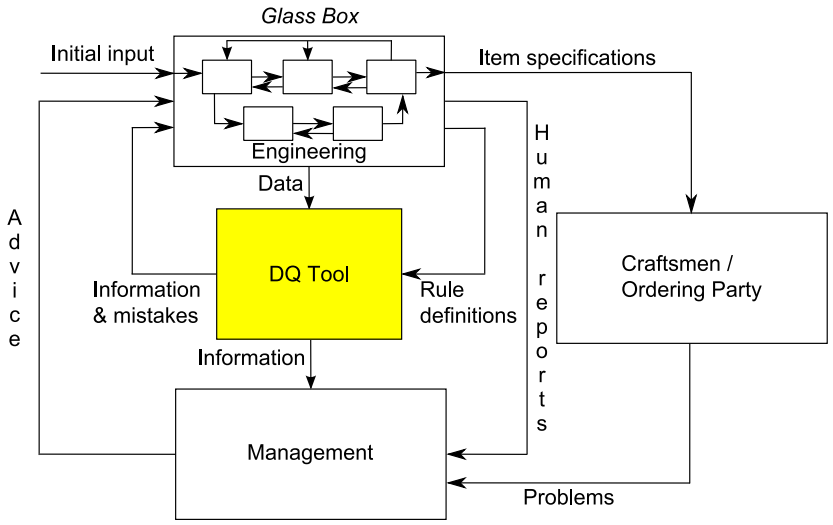

Figure 3. DQ-Tool-extended version of the information flows based on Figure 1

After the appropriate placement of the DQ initiative, now the question arises, if a commercial tool could fit into the environment and solve the DQ problems described above.

A survey of both commercial and research DQ tools can be found in (Barateiro and Galhardas, 2005). After categorizing DQ problems on schema- and instance-level they describe the features of 37 DQ tools they have analyzed, and categorize the tools according to the coverage of cleaning or profiling activities.

According to our analysis of existing DQ tools, besides the tool list in (Barateiro and Galhardas, 2005) two new tools have to be considered: The Oracle Data Profiling and Oracle Data Quality for Data Integrator Product (Oracle, 2007) and the Uniserv Data Quality Solution (UniservGmbH, 2009) consisting of the four sub-solutions Explorer, Batch Suite, Real Time Services and Monitor.

The Oracle Data Profiling and Oracle Data Quality for Data Integrator Product extends the features of the Oracle Data Integrator regarding data quality needs. The Oracle Data Profiling project type offers key, dependency and join analysis for finding possible primary keys, attribute dependencies within a single entity and possible join candidates between two entities, respectively. The Oracle Data Quality project type provides checks at the attribute level - like data type, uniqueness, and null value checks -, duplicate detection and enrichment and cleaning of data by means of a master reference file. To perform these features, Oracle's Metabase must be populated, i.e. all the data that shall be analyzed or cleaned must be copied to it.

The Uniserv sub-solutions Explorer and Batch Suite offer similar features as Oracle Data Profiling and Oracle Data Quality. Additionally, the Monitor provides reports based on the results of the Explorer and Batch Suite activities by defining rules via SQL-statements.

Summing up the tool analysis, we can state that many of the DQ tools concentrate on cleaning and enrichment activities, mostly for address related data. Even if business related data is concerned, two problems arise:

- Plant engineering data are very sensitive regarding safety-related requirements. Thus, it is not rational to aim for automatic cleaning or enrichment of these data. A manual correction is unavoidable.
- The rules for correct data in plant engineering are often very context-sensitive so that only experienced plant engineers know how to state them.

As a consequence, the engineers themselves have to state the rules and, according to these rules, possible mistakes must be reported to them; but finally, by combining the reports with their plant-specific knowledge they decide if and how to correct them.
Besides, there are also DQ tools that offer profiling, analysis or even monitoring capabilities. Under the assumption that the plant engineers are able to state the rules by themselves, those features can solve parts of the incorrectness problems described above like range or format constraints and to find null values of mandatory attributes. It should be mentioned that these correctness rules can also be simply stated in SQL via NOT NULL and CHECK constraints; drawbacks are the limitation on Boolean results and the necessary change of source IMSs, violating the glass box principle. Additionally, conditional functional dependencies of attributes of one entity can be defined via business rules for example in the Oracle solution. Drawbacks are again the limitation on Boolean results.

What cannot be solved by the analyzed DQ tools, are all the DQ problems arising due to the lack of information and concurrent engineering:

- There is no way to store the itemtype tree structure as well as the responsibility information the engineers need for their task at hand.
- Additionally, process related information in forms of validity, notification, progress and transfer issues cannot be stated.
- The inconsistency problems due to concurrent engineering are not solvable by the tools, because - besides the conditional functional dependencies of attributes of one item - all other inconsistency problems arise between different items, partly in even completely separated IMSs.

Additionally, management information in form of aggregated progress reports with responsibilities are not provided by the DQ tools.
Furthermore regarding correctness and consistency rules, we are not limited to Boolean results. In plant engineering, of course there are lists of values, ranges and rules for adjacent items and so on, but often there is no pure wrong and right, so we want to provide normalized, interpretable and interval-scaled results as also recommended in (Heinrich, Klier and Kaiser, 2009). That way, we want to support the engineers in their decisions what to correct or review most urgently.

Finally, the additional requirements mentioned above are not sufficiently met by the analyzed tools:

- Usability and understandability: For example, the user interface of the Oracle tool is quite overloaded for an engineer whose daily work does not focus on IT. Furthermore, to use the quality processes, many parameters have to be configured before the processes can start. Of course, this allows for various configurations suitable for many situations, but for engineers this is too complex and would not be used in their daily work where data quality comprises only one small part of their tasks.
- Adaptability and IMS neutrality: Due to concurrent engineering there are many different IMSs with separated databases and complicated data flows between these systems on same and related items. So, firstly, a physical integration of the entire data set to a DQ tool – as for example to Oracle's metabase - is not useful. Secondly, the changing nature of project-related definitions, itemtypes and even IMSs should have minimal influence on the DQ tool. This leads to an architecture that is decoupled from the IMSs and whose rule definitions, process and item model are easily modifiable.

Referring to this discussion, we propose our approach of a MetaDataRepository (MDR) that will be able to address all our requirements described above.

METADATAREPOSITORY APPROACH

This section describes the architecture, the rule logic and possibilities for a usable user interface of our solution, called the MetaDataRepository (MDR). In the MDR we will store all metadata that are necessary to

- support the engineers in guaranteeing a high data quality in terms of fulfilling business rules and specifications and providing intra- and inter-tool consistency and
- show itemtype-related information like tree structures and mandatory and optional attributes as well as process-related information like progress data, transfer overview and validity information and
- be a basis as responsibility reference and for notification possibilities.

The MDR will get an intuitive user interface so that the users are able and motivated to fill in rules besides their daily tasks.

Other metadata-based DQ solutions are those in (Helfert and Herrmann, 2002) and (Becker, McMullen, and Hetherington-Young, 2007). In contrast to our approach, the solution in (Helfert and Herrmann, 2002) focuses on a conceptual description without concrete rule types, metrics and interval-scaled measures. The approach in (Becker et al) is more concrete but the rules are defined via SQL-statements and are thus also limited to Boolean results. Additionally, none of the mentioned approaches in the literature deals with DQ problems arising due to concurrent engineering.

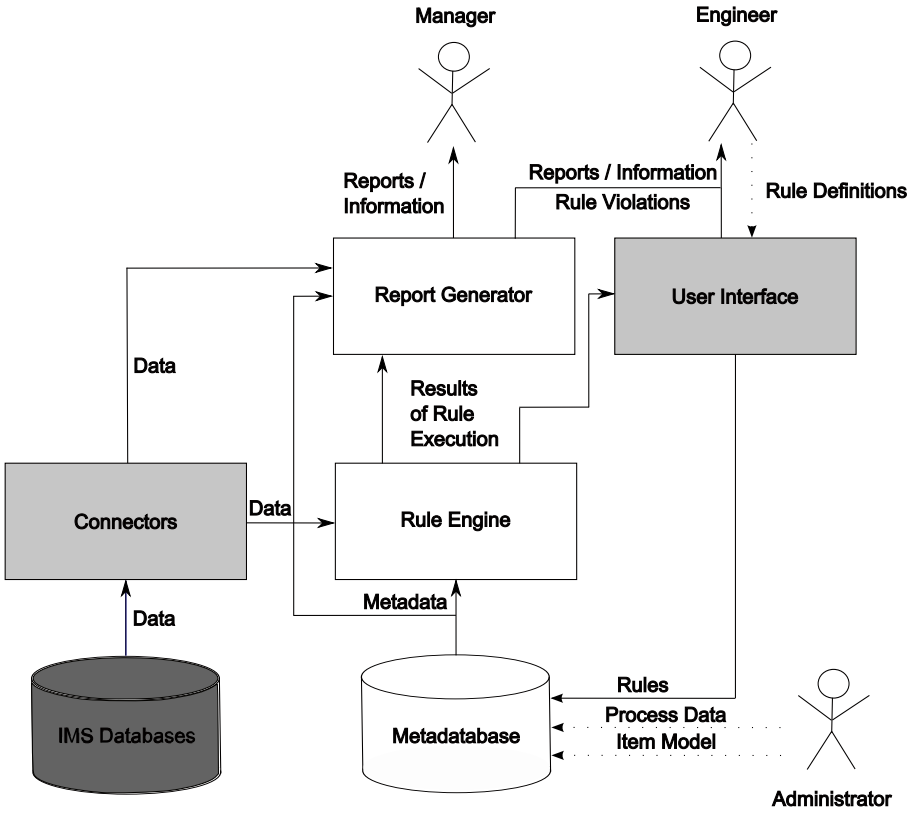

Figure 4. Architecture of the MetaDataRepository (MDR)

Architecture

The architecture of the MDR is depicted in Figure 4. The *IMS Databases* in dark gray together with all *Data* arrows are those parts that are already available in the IMS environment; the *Report Generator*, *Rule Engine* and *Metadatabase* in white with all interior arrows are those parts that lie completely in the responsibility of the MDR-realization-team; the *Connectors* and *User Interface* in light gray together with the outcoming arrows to the managers and the engineers are those parts that have to be built in cooperation with the end users (or IMS administrators regarding the connectors); finally the dotted, incoming arrows regarding *Process Data*, *Item Model* and *Rule Definitions* are those parts that are completely offered by the end users or the MDR administrator, respectively. The architecture is further described in the following. At first, three user groups are identified:

- The managers get information in form of reports about the overall project status.

- The engineers get information in form of reports about the data flows, responsibilities and project status tailored to their task at hand, and are encouraged to define rules that specify correct and consistent engineering data. Appropriate violations of their specified rules are reported back to them, when executing the rules.
- The MDR administrator populates the item model according to our itemtype tree structure with the project-specific attributes, itemtypes and relationships between them. S/he is also able to include process-related data like progress steps and data flows with appropriate master source definitions.

To summarize, the Metadatabase contains metadata about the user-defined rules, metadata about the itemtype tree structure and relationships between the itemtypes or groups, and metadata about the process representing design stages, data flows and responsibilities.
These metadata are processed in the Rule Engine to populate predefined reports in the Report Generator with the results of the rule execution and mark mistakes according to the user-defined rules stated over the User Interface, respectively. Because of the glass box engineering, we have access to the ever-changing engineering data via Connectors to the IMS Databases and are not forced to do a physical data integration. The real data from the IMS Databases provide the input for the Report Generator and the Rule Engine to execute the rule definitions on the engineering data and show the results to the users.

Information

The report format for the management will be predefined because their information need is manageable and barely changing. The information about amounts of completed items per itemgroup and corresponding responsibilities and design stages can be derived from the IMS databases and the grouping information available in the item model and progress steps in the process data.

The report format without rule involvement for the engineers can also be predefined because these reports will show static information about tree structures of itemtypes, progress steps, transfers and responsibilities.

Reports with rule involvement shall be partly configurable by the engineers, because, when inspecting wrong attribute values of specific items, the engineers often want to add attribute columns to get a complete overview about the affected item. The rule violations will be presented with prioritized correctness values to support the engineers in their problem-solving decision.

Finally, all reports will be presented in the well-known table format and shall be storable to disk.

Rule Logic

One of the most difficult problems lies in choosing an appropriate representation of rules. The rules need to have sufficient expressive power, yet be easy to use for domain experts and reasonably easy to interpret automatically. In the following sections, we describe the main elements of our rules - the rule types, the rule tree and the interval scaling method - and how they might be manipulated from a graphical user interface.

Rule Types

We found that rules can be classified in three families, depending on the data elements they need to access. Fortunately, the architecture described above allows us to implement them in a very similar way.

- Intraitem Rules: This type of rules refers to the attribute of a single tuple T consisting of attributes A_1 to A_m. An example is: "If A_i has a value of more than 100, then A_j must have a value of either 'C' or 'D'". More formally:

$$\text{cond}\left(T(A_i)\right) \rightarrow \text{cond}\left(T(A_j)\right), \qquad i, j \in \{1, ..., m\}, i \neq j$$

- Interitem Rules within the same IMS: These rules refer to multiple tuples in either one or several tables in one IMS. This requires two steps:

    1. Finding corresponding tuples
    2. Applying a rule to a set of corresponding tuples.

    For simplification, we join corresponding tuples together, forming a new tuple that contains all the attributes of the constituting tuples. Then we can apply rules in the same way as for intraitem tuples. While there may be complex scenarios that cannot be handled by this approach, it works well for all the quality problems we have described above.

For example, consider tuple T_I of itemtype I with attributes A_1 to A_m and tuple T_J of itemtype J with attributes B_1 to B_n and assume A_1 and B_1 to be appropriate identifiers (like the plant engineering identification number KKS (VGB PowerTech e.V., 2010)) and A_2 pointing to B_1 for connecting adjacent items:

1. $\left(T_I(A_2)\right) == \left(T_J(B_1)\right) \rightarrow T_{I,J}(A_1, ..., A_m, B_2, ..., B_n)$
2. $\text{cond}\left(T_{I,J}(A_i)\right) \rightarrow \text{cond}\left(T_{I,J}(B_j)\right), \qquad i \in \{3, ..., m\}, j \in \{2, ..., n\}$

- Interitem Rules between different IMSs: This is the most complex type of rules. It is similar to the class of interitem rules with one exception: The corresponding tuples may be stored in different information systems. This problem is solved by the connectors in our architecture as shown in Figure 4 and thus reducible to interitem rules within the same IMS.

Rule Tree

To gain a single correctness value out of a complete rule as mentioned above, the rule can be translated into a so-called rule tree. For example, consider the following interitem rule:

For valves connected to pipes the out-diameter of the valve must be equal to the diameter of the pipe and the maximal_pressure of the valve must be not less than the maximal pressure of the pipe and if the pipe has safetyclass 2, the valve must have either safetyclass 2, 3 or 4.

After finding corresponding tuples via identifying equal KKSs in the valve's *hostline* attribute and the pipe's *KKS* attribute, the resulting rule can be translated into the rule tree depicted in Figure 5.

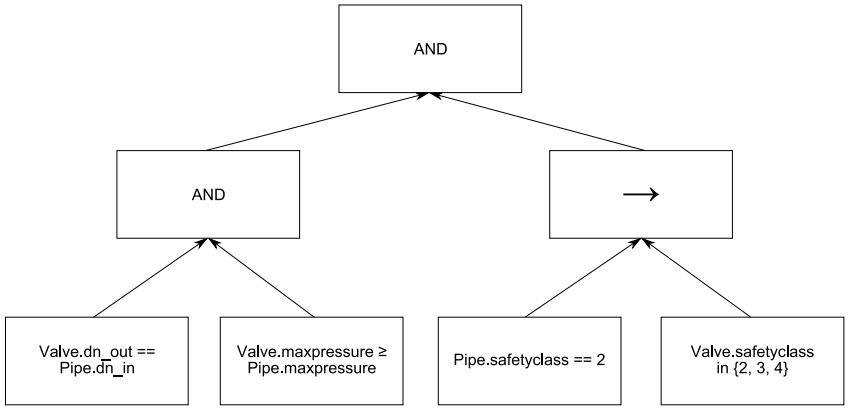

Figure 5: Rule Tree example

Interval-scaling

Because of the fact that we want to provide the user not only with Boolean correctness results, we have to introduce functions that map deviations according to their severity to interval-scaled correctness values.

At first we have to measure the deviation with similarity functions. Appropriate distance measures for strings in the data quality context are for example mentioned in (Klier, 2008), where also interval-scaling by means of one exponent is described. In contrast to that approach, we want to give more power to the user by allowing him/her to define various supporting points; i.e. considering the scale of deviation, the engineer can mark what deviation corresponds to what correctness value. As an example, Figure 6 shows a possible deviation scale from 0 to 500 where the Engineer has decided that a deviation of 500 corresponds to fully incorrect and a deviation of 5 as half correct.

Formally, the interval-scaled measure for a deviation x can be calculated by the following function with x_0 as the subinterval's starting point, x_1 the subinterval's endpoint and $f(x_0)$, $f(x_1)$ the corresponding points on the y-axis [0,1] according to the marked supporting points:

$$f(x) = f(x_0) - \frac{x - x_0}{x_1 - x_0}\left(f(x_0) - f(x_1)\right), \qquad x_0 < x < x_1, x_0 \geq 0$$

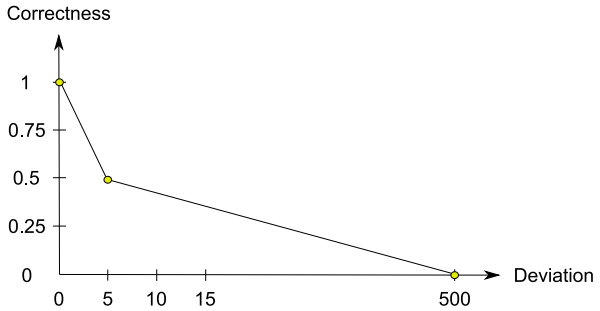

Figure 6: Interval-Scaling example

Rule Representation on the User Interface

As we want users to define the rules by themselves, they need to be represented in the user interface in an intuitive way. There are different options for this representation which may be more or less appropriate depending on the specific rule and the experience of the user. We describe shortly two possible representations together with their advantages and disadvantages.

Plain Text

A rule tree can easily be serialized to a plain text representation, much like any mathematical formula. This is the internal representation of rules in the MDR. Providing the engineers with the possibility to directly enter plain text rules via the user interface would offer maximal flexibility, but is too cumbersome, complex and non-intuitive for the engineers resulting in poor usability.

Graph (Boxes + Arrows)

A very intuitive way to represent a rule is decomposing it into its components which are then correctly combined building a directed graph. The nodes correspond to boxes and the edges to arrows. Using this way, in general, we offer the engineers the possibility to define a kind of rule tree: The leaf nodes would correspond to attributes. The internal nodes correspond to predefined operators and similarity functions the engineers can choose. This kind of representation is easy to learn and use. It can become tedious for huge rules with a lot of components, but in our environment the rule size will be surely manageable.

The rule representation on the user interface must be closely discussed with the engineers to find the best-usable solution for them. Different possibilities will be presented to them to get feedback, and their expectations will be checked with them.

CONCLUSION

This paper proposes a metadata-based approach to support a continuous data quality management process in concurrent engineering environments. It outlines the basic information flow in long-lasting engineering projects that aim at building on explicit order a huge physical object consisting of many composite items. Furthermore, it uncovers DQ problems arising out of lacking information support and wrong or inconsistent data values. The static environment in such projects poses the need for an adaptable, IMS neutral and usable DQ-solution.

Existent DQ-tools focus on data cleaning, data enrichment and duplicate elimination or on profiling activities that are limited to SQL-like checks in form of range or format definitions and simple Boolean checks on functionally dependent attributes of only one itemtype. Additionally, they are cumbersome to use for engineers and offer no possibility to deposit important engineering information in form of itemtype tree structures and process-related metadata.

The MetaDataRepository fills in this blank in offering reports for the engineers and the management via a report generator, encouraging the engineers to incorporate their knowledge in rule definitions via a usable user interface and empowering an administrator to insert process related and itemtype-specific metadata. That way, rule violations can be easily reported back to the engineers and assessed by means of interval-scaled results.

A prototype implementation of the approach is in progress. Its usability is verified with end users. Meanwhile, user interaction models are developed in close cooperation with the engineers, in order to achieve both interfaces that meet the users' needs and user acceptance of the tooling.

REFERENCES

1. Barateiro, J. and Galhardas, H. (2005) A Survey of Data Quality Tools, *Datenbank-Spektrum*, 14, 5, 15-21.
2. Batini, C. and Scannapieco, M. (2006) Data Quality, Concepts, Methodologies and Techniques, *Springer*.
3. Becker, D., McMullen, W. and Hetherington-Young, K. (2007) A Flexible and Generic Data Quality Metamodel, *Proceedings of the 12th International Conference on Information Quality*, 50-64.
4. Dubbel, H. (2004) Taschenbuch für den Maschinenbau, *Springer*.
5. English, L. P. (1999) Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits, Wiley, New York.
6. Fan, W., Geerts, F. and Jia, X. (2009) Conditional Dependencies: A Principled Approach to Improving Data Quality *British National Conference on Databases (BNCOD09)*, 8-20.
7. Heinrich, B., Klier, M. and Kaiser, M. (2009) A Procedure to Develop Metrics for Currency and its Application in CRM, *ACM Journal of Data and Information Quality*, 1, 1, 1-28.
8. Helfert, M. and Herrmann, C. (2002) Proactive Data Quality Management for Data Warehouse Systems - A Metadata based Data Quality System, *Proceedings of the 4th International Workshop on Design and Management of Data Warehouses*, 97-106.
9. International Organization for Standardization (2007) – ISO/TS 15926-4 Industrial automation systems and integration — Integration of life-cycle data for process plants including oil and gas production facilities — Part 4: Initial reference data, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41329.
10. Kahn, B. K., Strong, D. M. and Wang, R. Y. (2002) Information Quality Benchmarks: Product and Service Performance, *Communications of the ACM*, 45, 4, 184-192.
11. Kim, W., Choi, B., Hong, E., Kim, S. and Lee, D. (2003) A Taxonomy of Dirty Data, *Data Mining and Knowledge Discovery*, 7, 81-99.
12. Klier, M. (2008) Metriken zur Bewertung der Datenqualität - Konzeption und praktischer Nutzen, *Informatik Spektrum*, 31, 3, 223-236.
13. Oliveira, P., Rodrigues, F. and Henriques, P. (2005) A Formal Definition of Data Quality Problems, *Proceedings of the 10th International Conference on Information Quality (ICIQ05)*, 13-26.
14. Oracle (2007) Oracle Data Profiling and Oracle Data Quality for Data Integrator (Getting Started Guide), *Product Information*, http://www.oracle.com/technology/products/oracle-data-quality/pdf/oracledq_gs_guide.pdf.
15. Prasad, B. (1999) System Integration Techniques of Sharing and Collaboration among Work-groups, Computers and Processes, *Journal of Systems Integration*, 9, 115-139.
16. Prasad, B., Morenc, R. S. and Rangan, R. M. (1993) Information Management for Concurrent Engineering: Research Issues, *Concurrent Engineering*, 1, 1, 3-20.
17. Scannapieco, M., Missier, P. and Batini, C. (2005) Data Quality at a Glance, *Datenbank-Spektrum*, 5, 14, 6-14.
18. Strong, D. M., Lee, Y. W. and Wang, R. Y. (1997) 10 Potholes in the Road to Information Quality, *Computer*, 30, 8, 38-46.
19. UniservGmbH (2009) Uniserv Data Quality Solutions, *Product Information,* http://www.uniserv.com/en/products/data-quality-solutions.php.
20. VGB PowerTech e.V. (2010) Kraftwerk-Kennzeichensystem – KKS, *Online Information*, http://www.vgb.org/db_kks.html
21. Wang, R. Y. and Strong, D. (1996) Beyond Accuracy: What Data Quality Means to Data Consumers, *Journal of Management Information Systems*, 12, 4, 5-33.
22. Wang, R. Y., Ziad, M. and Lee, Y. W. (2000) Data Quality, *Springer - Kluwer Academic*.
23. Yassine, A. and Braha, D. (2003) Complex Concurrent Engineering and the Design Structure Matrix Method, *Concurrent Engineering*,11, 3, 165-176.