A Smart Link Infrastructure for Integrating and Analyzing Process Data

Eric Peukert
SAP SE
Chemnitzer Str. 48
01187 Dresden, Germany
eric.peukert@sap.com

Christian Wartner
InfAI e.V.
Hainstraße 11
04109 Leipzig, Germany
wartner@informatik.uni-leipzig.de

Erhard Rahm
Abteilung Datenbanken - Institut für Informatik
Universität Leipzig
rahm@informatik.uni-leipzig.de

Abstract: An often forgotten asset of many companies is internal process data. From the everyday processes that run within companies, huge amounts of such data is collected within different software systems. However, the value of analyzing this data holistically is often not exploited. This is mainly due to the inherent heterogeneity of the different data sources and the missing flexibility of existing approaches to integrate additional sources in an ad-hoc fashion.

In this paper the Smart Link Infrastructure is introduced. It offers tools that enable data integration and linking to support a holistic analysis of process data. The infrastructure consists of easy to use components for matching schemas, linking entities, storing entities as a graph and executing pattern queries on top of the integrated data. We showcase the value of the presented integration approach with two real world use-cases, one based on knowledge management in manufacturing from the LinkedDesign project and another one based on an agile software development process.

1 Introduction

An often neglected, highly valuable asset of many companies is internal process data. Huge amounts of such data is created while people work along predefined, sometimes implicit processes that are supported by heterogeneous software tools. Examples of such processes are software development, sales activities or customer relationship management. Exemplary data sources can be emails, telephone logs, files in source version control systems, customer interaction protocols or time reports to name a few. If such sources can be properly integrated and linked, it can be used to identify shortcomings of a process and give input for optimization.

However, due to the heterogeneity of the involved tools an integration of these data sources is cumbersome. Even if a single software tool suite is used to support many process steps there is still often a need to flexibly integrate additional data sources that might help to assess the overall quality of a process. Also, processes and tools change over time so

that data source schemas, their relations and links between them suffer from continuous changes.
A number of approaches can be found in literature that simplify the creation of federated data sources [SL90, BLN86], that use schema matching [BBR11, HRO06] or that apply object matching to identify links between entities [NA11, KR10]. However, most of the existing work only focuses on individual aspects of matching and storage and often comes with the restriction of fixed schemas. First approaches try to follow a schema-flexible graph-based integration approach as it is done with BIIIG [PJRR14]. It applies a metadata-based integration of data sources to a unified metadata graph to finally construct an integrated instance graph. However, BIIIG does not provide mechanisms to identify new links at the instance level between data sources.
In this paper the Smart Link Infrastructure for graph-based data integration is introduced. It is based on a schema-flexible integration of data sources into a common information graph. Existing associations from the original data source are stored as edges in the graph. But, in many cases such associations do not exist. The infrastructure supports a ”pay as you go” approach of link generation. Links between nodes of the graph are automatically computed at the instance level based on the analysis of the contents and relations of individual nodes. Nodes in the graph can refer to people, documents, transactions or any other data object that might be generated throughout a complex process. The linking results in additional edges within the information graph which then offers new ways of interaction and analysis. Within the information graph we are able to uniformly navigate, query and analyze the existing information.
For analyzing the graph of nodes and links the infrastructure relies on so called process pattern queries. Such pattern queries are different from classical relational queries. A pattern describes a complex setting involving several types of entities together with a specific set of constraints. A pattern query helps to identify the properties of a project or process that make one project or process more successful than others. For instance, in a software development process different kinds of data entities like tasks, code, bugs and worklogs are typically collected. A flexible tool to integrate and link information created by different software systems promises unprecedented value to software companies by helping to continuously increase the efficiency of their internal development processes. A simple pattern query applied to information from a software development process, for instance, could search for requirements with high complexity that might lead to the involvement of many people or have a high likelihood for bugs.
This paper makes a number of contributions:

- The architecture of the Smart Link Infrastructure for flexible data integration and analysis is introduced together with its core data integration services for schema matching, graph storage and linking.
- The information graph and pattern queries are explained.
- Front-ends for search and analysis that apply a simple but effective drag and drop metaphor are described.
- The infrastructure has been developed within a large EU project and succesfully applied for industrial use cases. We describe two real world use cases in detail to

illustrate the versatility of the Smart Link Infrastructure.

2 Smart Link Infrastructure

2.1 Overview

The architecture of the Smart Link Infrastructure is shown in Figure 1. It consists of three layers. On the top are the user interface components. These are HTML user interfaces supporting administrative tasks, i.e. data import and linking of data records as well as graphical user interfaces for the visualization of graphs, querying and the creation of reports. Since the Smart Link Infrastructure aims at an integration with existing software systems external tools like OneNote (as used in the knowledge management processes described in chapter 3) are supported as well.

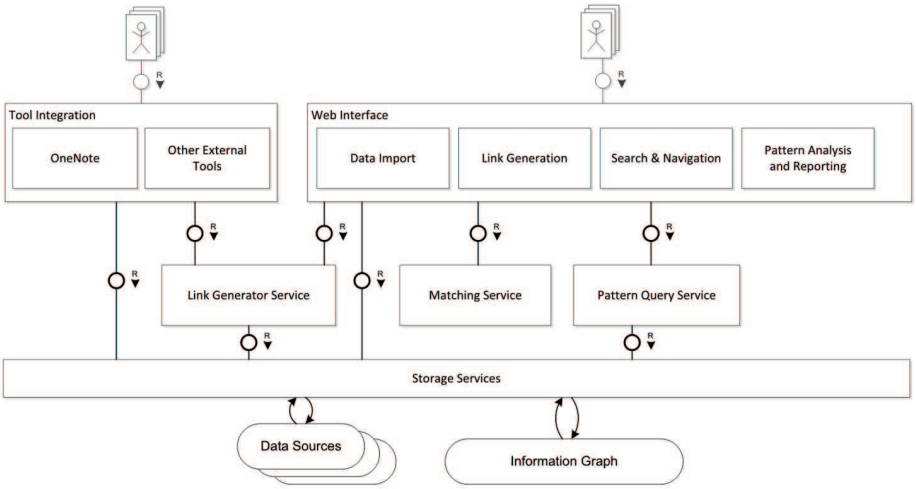

Figure 1: Architecture of the Smart Link Infrastructure

The second layer consists of a collection of services: the Schema Matching Service, the Link Generator Service and the Storage Service. The first two serve data import and management tasks while the Storage Service represents the general interface to the central graph database providing operations to store and query data. The services each expose a REST-based API towards the UI components and external clients. Communication happens largely with JSON messages.
On the bottom of the architecture the data sources can be found. One is the central graph database itself and on the left are the multitude of sources (e.g. databases, file repositories, SPARQL endpoints etc.) from which data is imported. To be able to holistically analyze

and search within data that is distributed across multiple heterogeneous data sources, various data integration approaches are used. There are some similarities to ETL processes that are used when building data warehouses to allow efficient analytic processing, e.g. when data from independently created databases is to be combined to create a unified view that gives new insights for analysts.
The general workflow applied within the Smart Link Infrastructure is depicted in Figure 2. The data integration workflow starts with the data import that is supported by a service providing schema matching methods. In this step relevant data sources can be selected like files, databases, etc. and a mapping of their schema with the schema of the Smart Link Store is created. Based on the mapping, new entities are added to the information graph. The use of the schema matching service is optional and supports administrators during the definition of the mapping. It could simplify subsequent steps if a sound input schema is available.
In the linking step, entity resolution approaches are applied to identify relations between entity sets that originate from different sources. In contrast to standard link approaches, we also support text mining techniques and new mapping types. Text mining helps to deal with unstructured data like text documents whereas new mapping types are needed to determine the type of semantic relations between entities. The linking step is important for defining a well-connected information graph, especially if the imported data sets contain no direct references to each other.
In the center of the Smart Link workflow there is a graph database, the Smart Link Store, which provides the capability to store and query data. In particular, it offers means for executing pattern queries on the information graph that are crucial for the analysis. Each step of the workflow will be detailed in the following chapters. Chapter 3 will introduce two real-world use case scenarios that utilize the Smart Link Infrastructure and give an insight into the actual usage.

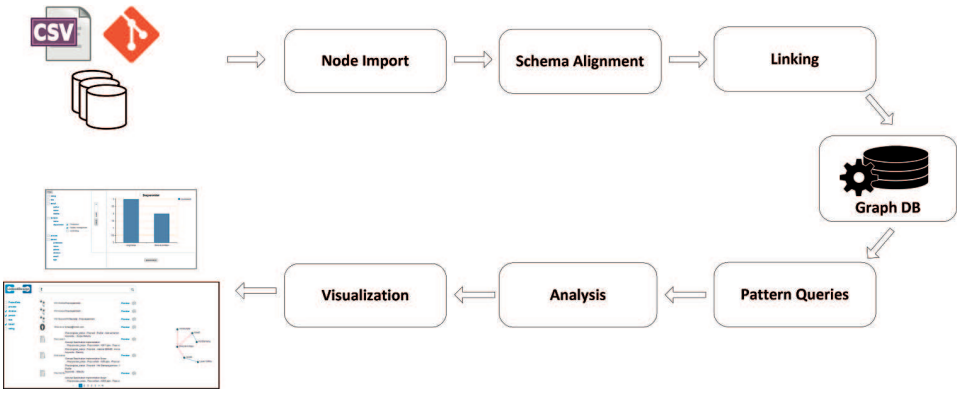

Figure 2: Overview of the Smart Link Infrastructure information flow

2.2 Data Import

In order to pre-fill the Smart-Link Store with entities, an administrative user can use the Data Import Tool. Such pre-filling allows a user to connect to existing data sources such as relational databases, Excel files or data sources that are exposed through a SPARQL endpoint. When integrating a data source into the information graph the schema of the data source needs to be mapped to the existing flexible schema of the information graph. Such a mapping describes how elements of the source schema correspond to elements of the information graph schema. As discussed above, extending the graph schema is not problematic, but existing types and properties should be reused. Adding properties to existing types or adding types is implicitly triggered when new entities are uploaded. Defining the mappings can be complex and time-consuming. It is often done manually, with the help of point and click interfaces. The data import tool of the Smart Link Infrastructure offers a point and click interface similar to existing mapping solutions (see Figure 3). Moreover,

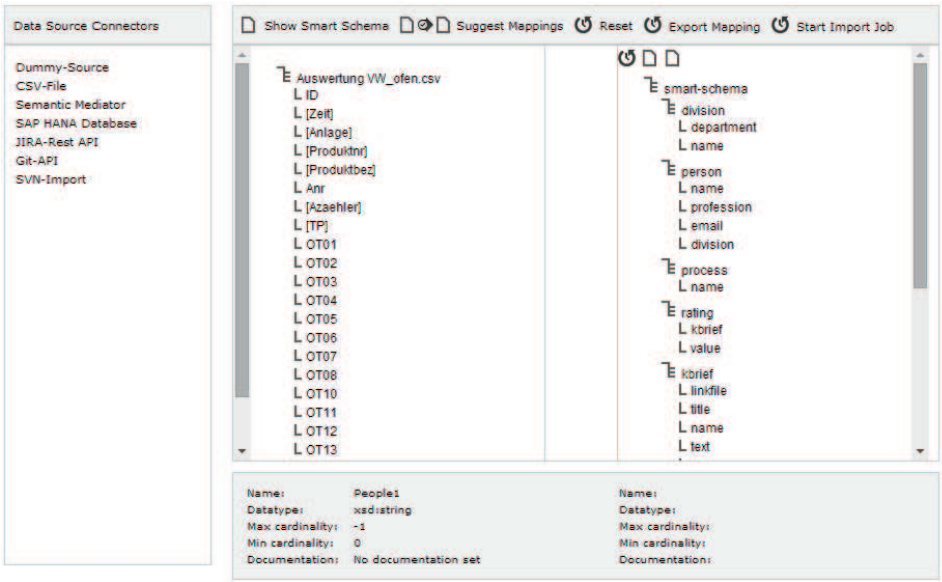

Figure 3: Point and click data import tool

a schema matching service is integrated that is able to compute suggestions for mappings.

2.3 Schema Matching Service

To reduce the manual effort in mapping, a matching service was integrated that semi-automatically computes a mapping suggestion for a user. The matching service contains

a number of matching algorithms and a library of schema importers for different schema types [PER11, DR02]. It takes as input two schemas and computes a mapping suggestion between them. Similarities between source and target elements are computed on metadata level but also on instance level. Since current matching systems are often not robust enough to be able to cope with very heterogeneous source schemas we developed an adaptive matching approach [PER12]. This approach automatically configures a schema matching system process that consists of a set of operators for matching and filtering. Based on measured features of the input schemas and intermediate results so-called matching rules can be defined. These rewrite rules rely on analyzing the input schemas and intermediate results while executing a process and rewrite the process to better fit to the problem at hand.
As was already described the adaptive schema matching approach is crucial to match the heterogeneity of schema types to an integrated schema and to finally integrate those sources within a common information graph. The schema matching service described above implements parts of the adaptive matching system and is therefore able to improve the quality of matching results.

2.4 Link Generation

Creating graphs from a structured data source like a database with well-modeled metadata can be relatively easy. In contrast to that, creating a graph from unstructured sources like document collections or independently created databases requires a component to find links between entities. The Link Generator Service allows the creation, management and execution of workflows, so called linkers, that can determine relations between entities, the relation type and the confidence of these relations. Linkers are related to the entity resolution workflows used in various data integration and data quality related scenarios(e.g. link discovery in the web of linked open data [NKH+13], web data integration [WK11] or classic ETL processes in data warehousing [CP11]). Figure 4 shows how entity resolution workflows look at an abstract level (cf. [Chr12]). Normally they are used for detecting data objects that are equivalent in the real world but have different representations in multiple data sources. The input of an entity resolution workflow are usually sets of data records from one or multiple databases. The operations in the preprocessing step include data transformation steps (e.g. to convert data types or remove special characters) and filter operations or blocking steps to reduce the search space for finding matching objects [DN09]. Match operations are then applied to determine the pairwise similarity of the candidates. Depending on the domain different distance metrics on one attribute or a combination of attributes can be used to determine this likelihood of two entities being equal [EIV07]. The match result usually is an instance-level mapping: a set of correspondences of the type (entity1, entity2, sim) where sim is the confidence of two entities being equal. In general a mapping contains correspondences of a single type, i.e. ‘sameAs’. This resulting mapping is then used to determine a set of matching objects and a set of non-matching object pairs from the input data source. Standard entity resolution workflows can be used to find objects in the graph store that are very similar and merge

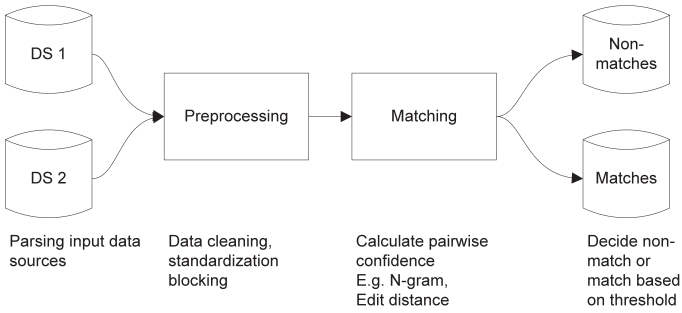

Figure 4: General entity resolution workflow

them under the assumption that these are duplicate nodes or slightly different versions of the same note. These data quality related workflows, however, are not the main focus of the Link Generator service.
Instead of searching for pairs with a high confidence of a 'sameAs' relation, linking workflows aim to search for links between arbitrary objects and their type. In the easiest case a linker can just determine a link between two objects by calculating the similarity between a certain field that exists in two objects. An example of this is a linker that tries to find documents written by the same author which involves the following steps:

1. Parse the Input (e.g. a document with a meta-data field about the author).
2. Preprocessing: Split the author field into words and sort them alphabetically.
3. Matching: Calculate the trigram similarity of the sorted author list.
4. The result is a set of correspondences between documents that in effect have the type 'sameAuthor'.

Linking workflows like this are useful to generate certain types of edges for our property graph that connect objects of the same type. However, more complex linking approaches are needed. To enable the functionality needed for the current Smart Link Infrastructure, i.e. the calculation of links between arbitrary data objects representing entities like persons, documents, etc. most of the steps depicted in Figure 4 are extended and an additional, optional, rule application step is added to the process. Figure 5 shows a general linking workflow.

Especially when trying to support search and information discovery in knowledge management processes the data often consists of unstructured text from documents with possibly incomplete metadata. The string similarity of the content of two text documents, for example, does not provide enough information about the relation between them. The only relationship that could be derived from a low lexicographic distance is that the documents have duplicate text or large parts of overlapping text. Named entity recognition, keyword detection, the detection of hyperlinks and mail addresses as well as other text mining

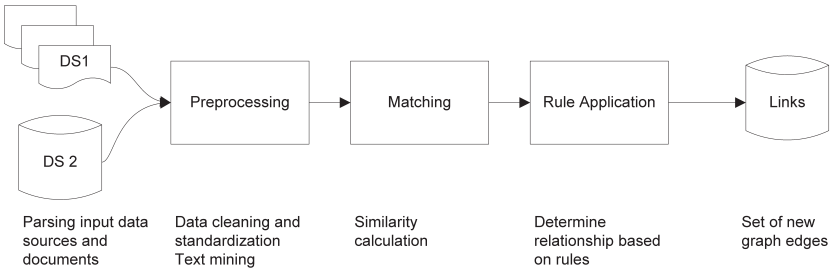

Figure 5: General linking workflow

approaches become important to link documents with semantic relations that go beyond 'sameAs' relations. A linking workflow that relies on text mining and uses a simple rule to decide the relation type could look like this:

1. Parsing of Input data(e.g. text of a PDF file).
2. Preprocessing: Named entity recognition, Keyword detection.
3. Calculation of the similarity (e.g. amount of overlap) of list of keywords, engineering concepts.
4. Application of rules: For example, a high overlap of keywords leads to the relation 'similarContent' and overlapping engineering concepts extracted from the two documents lead to the relation 'sameTopic'.
5. Links, i.e. new edges are inserted into the graph

In addition to the use of text mining approaches linking workflows can also utilize existing edges in the information graph. Since the Smart Link Infrastructure allows the incremental addition of new data sets, it is usually possible to use existing relations. An example would be a relation like 'sameTopic' which is transitive to some degree. Once a linker determines that a new and an already integrated document have a high confidence of such a relation we can infer that the certain relations exist between the new document and the already integrated documents neighborhood. This is similar to data matching approaches that use already existing mappings (e.g., [TR07]) and can be useful to discover and verify certain types of relations as well as to increase the efficiency of a linker. Generally, the workflows of the Link Generator service first produce mappings with instance-level correspondences of the type (entity1 ,entity2, linking method, confidence) representing the used linking method and its resp. confidence (e.g. (document1, document2, trigram-author, 0.8), (document1, document2, keywordoverlap-text, 0.2)). A final rule application step decides based on the applied method and the confidence value what edges are added to the graph. A set of simple conditional rules, e.g. "if confidence (author1, author2) $\geq$ threshold then linktype is sameAuthor" are satisfactory for most use cases, although their definition requires some domain knowledge.

The Link Generator service allows users the definition of suitable workflows (Linkers) for a certain domain.

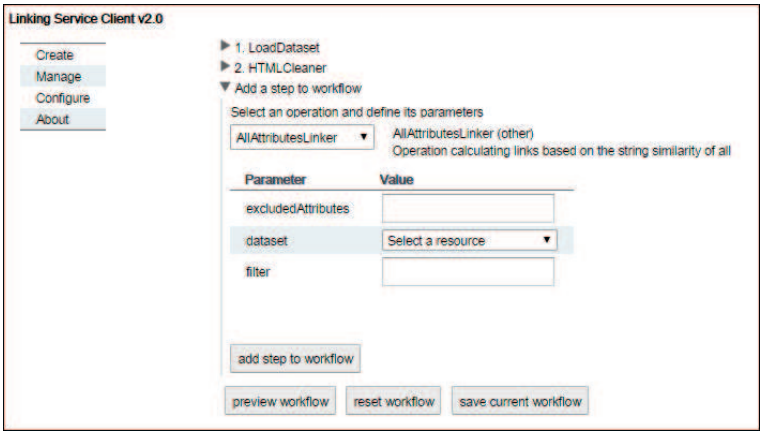

Figure 6: Linking UI

The service offers a variety of preprocessing, text mining, data matching approaches and filtering operations that can be freely combined. These include:

- Preprocessing operations, filter operations to exclude certain entities from subsequent operations,
- Keyword, link, address and named entity detection operations
- Dictionary based unification of terms
- Match approaches like N-gram distance, TF-IDF distance, Edit distance to calculate the confidence of attribute similarity
- Match approaches that determine a confidence value based on the overlap or element similarity of lists
- Match approaches based on already known relations (i.e. the neighborhood of entities)
- Combination operations for aggregating confidence values determined by different approaches
- Relationship detection rules based on individual or combined confidence values

The operation library of the Link Generator service can be extended by operations needed for a certain task at any time. Workflows can be set up through a web based UI (Figure 6) that also allows their management and setup for periodical or triggered execution.

2.5 Smart Link Store

2.5.1 Property Graph Model

The used graph store of the Smart Link Infrastructure relies on a schema-flexible property graph model with a navigational query access as implemented by Neo4J [Neo12] or AIS [RPBL13]. In contrast to widely used triple stores a property graph keeps the object identity intact. This simplifies queries since the entity construction effort can be omitted. The used property graph stores objects together with their properties. Those properties can either be atomic values or associations to other objects in the graph. Each object has a type which is managed in a dedicated type store. The set of properties of an object can be changed at any time. Also, an object type does not fix the set of types used. However, the used graph store permits to dynamically compute statistics about existing types in the store and their used properties together with usage counts. The statistics of property usage could help to filter less frequent associations to get a more stable and accurate global schema. This dynamically computed schema later helps us to generate queries.

2.5.2 Pattern Query and Query Generation

A crucial part for analyzing process data holistically are pattern queries. These are much more intuitive for a user than navigational queries since the user does not need to know the exact topology of the graph which is required for navigational queries. Navigational query access refers to the ability to traverse associations in a graph within the query language. Many graph stores only implement navigational queries such as GREMLIN in Neo4J or WIPE within SAP AIS. Existing pattern query languages include SPARQL for triple stores and Cypher for Neo4J property graphs. Due to the nature of triple stores, SPARQL queries often become rather complex. The Cypher query language for property graphs of Neo4J is much easier to use. Unfortunately it is bound to Neo4J. Typically, results to graph pattern queries are computed by applying maximum common subgraph algorithms which are expensive to execute [McG82]. Since we would like to rely on an existing internal property graph store that only provides navigational query access we decided to implement a pattern query interface on top of the navigational query interface. A pattern query is taken as input and a navigational query is generated.
Figure 7 illustrates the overall process of query generation. As described earlier, the graph model we use allows us to compute a schema dynamically from the instances that are currently in the graph. Before any query is issued to the graph store the current schema is computed. The example assumes that there is a graph with four different kinds of entities which are represented by different shapes and names of nodes. The generated schema only consists of four types of entities including possible associations between instances of these types. A user could create a pattern query by stating that there shall be two nodes directly or indirectly connected and the nodes have to follow certain filter predicates. Such predicates define which nodes from the set of instances of a type should be in the result set.
From the schema graph that was initially created, a minimal connected subgraph for the

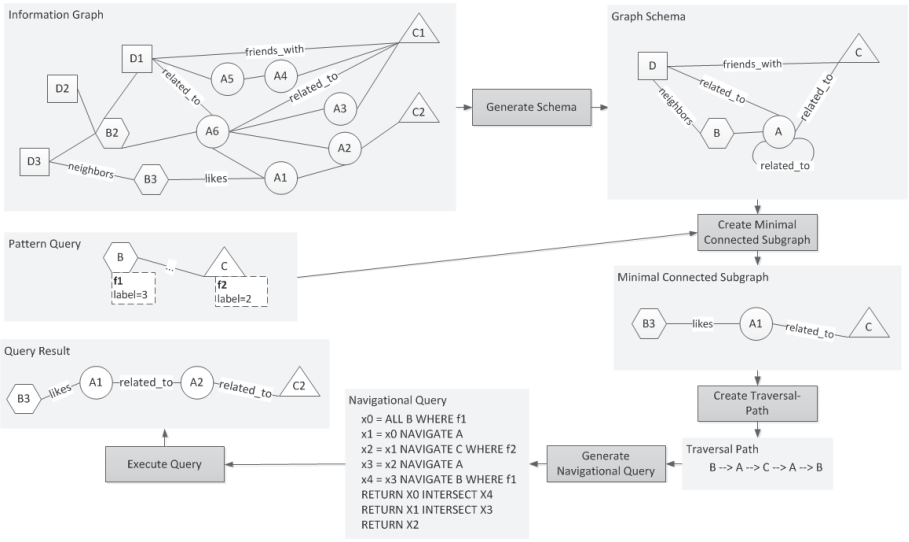

Figure 7: Pattern query generation process

given types is computed. If the types are not connected on schema level, then the query will return an empty answer. In the example, the intermediate type A was added to get a minimal connected subgraph. In order to be able to generate a navigational query on that subgraph a traversal path is needed based on the type graph. In the example the traversal starts at B and walks to C. Note that the walk needs to be done in backwards direction since initial filtering on the B nodes did not involve indirect filtering through the subset of C nodes. Consider that there are B nodes that follow the predicate f1 that have no possible path to any C node that follows predicate f2. After the traversal path has been computed, we can now generate the navigational query using any query language that allows navigating associations in both directions. That means if there is an association from A to B then the classical navigation from A to B is possible. However, in our approach we need to be able to navigate backwards from B to A returning all nodes A that have an association to B. The query language used in the example is a simplified example. At the end of the query the results of the individual navigation steps are intersected to return the correct set of nodes for each node type. In the example, nodes are returned that were originally not requested in the query. This can be filtered out depending on the requirements of the respective application.

2.6 Data Analysis

The Smart Link infrastructure offers tools for search and analysis within the integrated information graph. The information graph with its links can help to improve the ranking

(see PageRank from Google) of a key-word based search result. Moreover, the graph that can be found in the vicinity of a node is displayed in addition to the search result and entities can be filtered by facets. A screenshot of the developed search interface can be found in Figure 8.

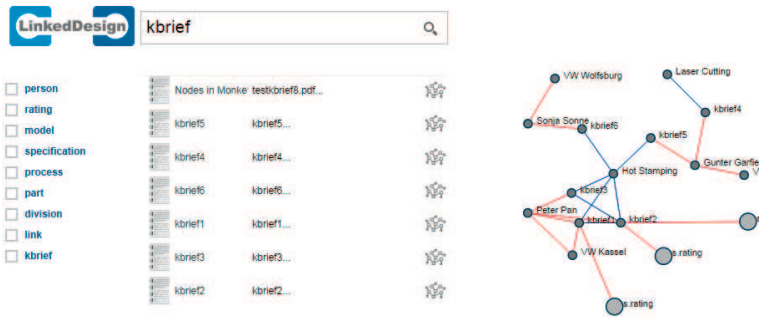

Figure 8: Search UI

The analysis of the information graph is driven by pattern queries and a simple drag and drop metaphor for creating charts. Figure 9 shows a screenshot of the analysis tool. It supports a user to graphically create pattern queries. By dragging and dropping dimensions and measures onto the axes of an empty chart, a new report can be created. In addition, users can define functions over measures to compute complex measures on top of the pattern query result.

3 Case Studies

3.1 Software Development Process

The Smart Link Infrastructure was used in a proof of concept implementation to analyze process data that is created within an agile software development process. In order to get an idea what data sources were integrated the development process is briefly described. The development is organized as an agile project based on the SCRUM method. A project is split into short sprints of 1 or 2 weeks. Requirements are collected within a so called backlog. At the beginning of a sprint, tasks (issues) are taken from the backlog, their effort is estimated and the issue is assigned to the sprint in a so called planning meeting. The development team takes issues and marks them as resolved when finished. Software code and documents are created and pushed to special code management systems. After each sprint the completed issues are reviewed and the next sprint is planned. There are many tools such as Agilefant [Agi] or JIRA [Atl] that try to support such processes. These tools offer support for issue creation, assignment to developers as well as time and status reporting. For the actual development and code management, source version control systems can

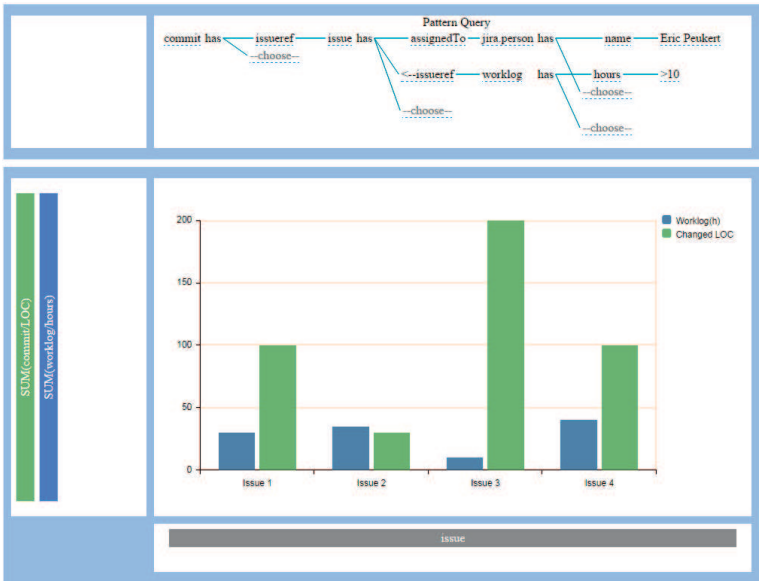

Figure 9: Analysis UI

be used such as CVS, SVN, Perforce or Git. Throughout the process, meetings are organized between team members and emails are exchanged, that contain discussions about the current development tasks. Continuous integration and test systems take developed code and run tests. Committed code is analyzed and executed which results in error/test reports that provide the developer with feedback about the quality of the committed code. In later stages, bugs are filed and attached to certain code fragments. For each of these process steps separate software systems and data stores are typically used. However, it would be valuable to be able to link tasks to actual commits, corresponding files and test reports. In the proof of concept the data sources were integrated with the help of the Smart Link Infrastructure in a common information graph that links requirements with tasks, people, code commits and test reports. (see Figure 10).

The linking service of the Smart Link Infrastructure created a number of meaningful links between different types of objects. A linking between commits and issues was possible since some developers put contents from the issue description and issue names into the commit message. Commits are also linked to people based on user ids, emails and names. Some objects are also linked by date which helps to assign commits to sprints. Issues are also linked to people which finally forms a nice information graph of the development process at hand. With the help of the analysis tools the development process could now be analyzed. It was now possible to observe interesting properties of the process that could serve as input for improving the process in future. For instance, times of high workload could easily be identified. Even though the actual estimated times for issues remained similar for all sprints the actual logged working times and committed lines of code showed

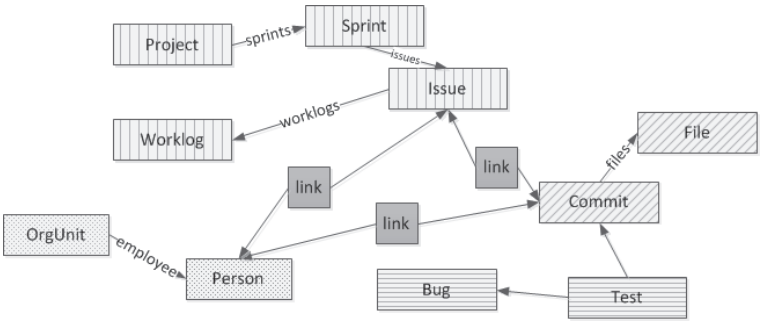

Figure 10: Linked data sources in the information graph

some strong peaks. This happened often before milestone deadlines. However, according to the scrum method, such peaks should be avoided since they could put too much pressure onto a development team. Moreover, the tools helps to measure the efficiency of a team and could give an indicator to the complexity of a software system. When a software is newly build, many lines of code are initially created in very short times. Unfortunately, the number of changed lines per logged hour of work reduces over time since the complexity of change increases. The exemplary analysis from Figure 9 showed a comparison of changed lines of code to actually logged working hours for a selected set of issues. With additional data sources like vacation times or calendar data one could certainly make further helpful observations.

In summary, the integrated information graph provided many insights on the running process that would otherwise be hidden. The lightweight integration approach of the Smart Link Infrastructure helped a lot since there was no need to define a global schema upfront. The sources could simply be integrated by letting the automatic linking identify correspondences.

3.2 Smart Link Infrastructure for Knowledge Management Processes

To demonstrate the wide spectrum of application cases the Smart Link Infrastructure was applied to analyze and support a knowledge management process within an oil platform construction company. In particular the linking component was tightly integrated into a document authoring tool (Microsoft OneNote) that is typically used within that company.

The Smart Link infrastructure is able to automatically link a document to an existing corpus of documents while it is created or edited. Found links are dynamically inserted into the currently edited document and also within related documents of the corpus. Links can be created between documents of the same kind but also to external sources such as taxonomies of terms or other records such as people, files on a share or a source version control system. This approach does not require the user to collect links to entities from each of these possible information sources manually or to manually insert them into the

document at hand. Given that these information sources are known, all such links are added automatically.

The prototype consists of an extension (Linking Plugin) to Microsoft OneNote. While editing textual content the Linking Plugin extracts the content and triggers a linking workflow of the Linking Service. The Linking Service computes links from the newly created document to existing documents or to other sources. Each linker can produce different types of links such as *references, is-related, is-contained-in, created*. As described above, the linking service can be extended by further linkers and workflows.

The linking result is filtered and ranked based on the computed confidence values. Selected links are finally inserted in the currently edited document. Links can be inserted as separate sections, but could also be added inline to the text if specific keywords relate to an entity in one of the attached information sources. Changes to the text or referenced documents can also trigger a removal of previously computed links. Generated links could be marked as being strong or less strong. Also the age of the link could be visualized. The longer a link is part of a document the more time is should take to remove the link. It should be shown as deprecated so that a user can accept removal manually. Linked documents are also changed to also link to the newly created document. The linking process is dynamic so that by writing new content the dynamic links can change. The user has the option to manually remove link suggestions or to reset linking parameters so that only links with higher confidence are added to the document.

3.2.1 Link Generation

For generating links the linking component that was described above is used. In the following we introduce a few interesting linkers. A *file reference linker* looks up file names and paths in existing file-shares to recognize file-names inside the text at hand. Also names of other documents mentioned within the text can serve as indication for creating reference links. To identify related documents text-similarity measures like TF-IDF or N-Gram are applied. These measures attach a confidence to a link that can be used for filtering and ranking. The linking quality can be improved by extracting most representative keywords from the text and to compute similarity based on comparing keyword-sets. A number of techniques propose to identify structure of documents, apply stemming to normalize and tokenize longer terms. Such preprocessing can help to improve linking quality.

The computed links between documents, authors, files and other entities build a graph of entities in the background. A neighborhood linker uses this graph to identify indirectly related documents and further information such as the authors of such documents.

3.2.2 Search and Analysis

Through the above linking process a complex graph of documents and other data entities is created.

In the proof of concept we also added facilities for users to rate created content and give feedback. Moreover the viewer tracks when certain documents where last viewed and

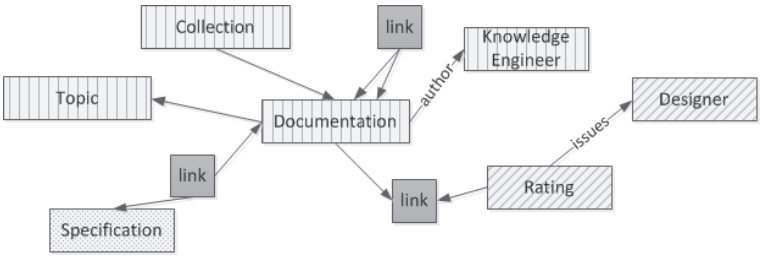

Figure 11: Document graph

edited. As a result of combining these data sources, an analysis of the age and maturity of existing knowledge documents can be performed. If legally permitted one could also measure the documentation quality and relevance of content from individual authors. The authors can be clustered by their expertise for certain topics which helps users to identify experts.
Moreover, searching over the document base improves with the help of a PageRank-like algorithm to measure the importance of linked documents.

4 Conclusions & Outlook

The paper introduced the Smart Link Infrastructure that can be used to integrate and analyze highly heterogeneous data sources that result from implicit or explicit working processes into a common information graph. Data import is supported by schema matching and links between entities of the information graph are generated by linking workflows. The integrated data of the information graph can be queried with the help of pattern queries and reports can be generated with the help of a drag and drop metaphor. Also, a search interface supports faceted search over the information graph and subgraphs can be visualized and used for navigation.
Two real world use cases proof that the Smart Link Infrastructure can be applied for solving a broad range of data integration problems that occur when integrating process data.
Future work will focus on pattern mining and query pattern recommendation techniques to simplify the construction of relevant pattern queries for a user.

5 Acknowledgement

This paper was partly funded by the European Commission through Project LinkedDesign (No. 284613 FoF-ICT-2011.7.4). LinkedDesign is a European project aiming to create a platform that addresses the problem of a missing integrated, holistic view of data across the whole product life cycle in the current manufacturing ICT landscape. The Smart Link

Infrastructure is the data integration and analysis component of the LinkedDesign platform and provides tools to integrate and link data objects from different product life-cycle phases into a property graph to enable search and analysis in order to solve problems in various use case scenarios.

References

[Agi] Agilefant. http://agilefant.com/. [last visited 10.10.2014].

[Atl] Atlassian. https://www.atlassian.com/software/jira. [last visited 10.10.2014].

[BBR11] Z. Bellahsene., A. Bonifati, and E. Rahm. *Schema Matching and Mapping*. Springer, 2011.

[BLN86] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[Chr12] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.

[CP11] Alfredo Cuzzocrea and Laura Puglisi. Record linkage in data warehousing: State-of-the-art analysis and research perspectives. In *Proc. of 22nd International Workshop on Database and Expert Systems Applications (DEXA), 2011*, pages 121–125. IEEE, 2011.

[DN09] Uwe Draisbach and Felix Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, pages 51–56, 2009.

[DR02] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proc. 28th Intl. Conference on Very Large Databases (VLDB)*, pages 610–621, 2002.

[EIV07] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[HRO06] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data Integration: The Teenage Years. In *Proc. of the 32. International Conference on Very Large Data Bases(VLDB)*, pages 9–16. VLDB Endowment, 2006.

[KR10] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[McG82] J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23–34, 1982.

[NA11] Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES: a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the 22. international joint conference on Artificial Intelligence*, pages 2312–2317. AAAI Press, 2011.

[Neo12] Neo4j. Neo4j - The World's Leading Graph Database, 2012.

[NKH+13] Axel-Cyrille Ngonga Ngomo, Lars Kolb, Norman Heino, Michael Hartung, Sören Auer, and Erhard Rahm. When to reach for the cloud: Using parallel hardware for link discovery. In *The Semantic Web: Semantics and Big Data*, pages 275–289. Springer, 2013.

[PER11] Eric Peukert, Julian Eberius, and Erhard Rahm. AMC - A Framework for Modelling and Comparing Matching Systems as Matching Processes. In *Proc. Int. Conf. on Data Engineering (ICDE)*, pages 1304–1307, 2011.

[PER12] Eric Peukert, Julian Eberius, and Erhard Rahm. A Self-Configuring Schema Matching System. In *Proc. Int. Conf. on Data Engineering (ICDE)*, pages 306–317, 2012.

[PJRR14] Andre Petermann, Martin Junghanns, Müller Robert, and E. Rahm. Graph-based Data Integration and Business Intelligence with BIIIG. In *Proc. of the 40. International Conference on Very Large Data Bases(VLDB)*, 2014.

[RPBL13] Michael Rudolf, Marcus Paradies, Christof Bornhövd, and Wolfgang Lehner. The Graph Story of the SAP HANA Database. In *Proc. of 15. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, pages 403–420, 2013.

[SL90] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[TR07] Andreas Thor and Erhard Rahm. MOMA-A Mapping-based Object Matching System. In *Proc. of 3rd Conference on Innovative Data Systems Research (CIDR)*, pages 247–258, 2007.

[WK11] Christian Wartner and Sven Kitschke. PROOF: Produktmonitoring im Web. In *Proc. of 14. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW)*, pages 722–725, 2011.