Maintaining Semantic Networks: Challenges and Algorithms

Bernd Michelberger
University of Applied Sciences Ravensburg-Weingarten, Weingarten, Germany
bernd.michelberger@hs-weingarten.de

Klaus Ulmschneider
University of Ulm, Institute of Artificial Intelligence, Ulm, Germany
klaus.ulmschneider@uni-ulm.de

Birte Glimm
University of Ulm, Institute of Artificial Intelligence, Ulm, Germany
birte.glimm@uni-ulm.de

Bela Mutschler
University of Applied Sciences Ravensburg-Weingarten, Weingarten, Germany
bela.mutschler@hs-weingarten.de

Manfred Reichert
University of Ulm, Institute of Databases and Information Systems, Ulm, Germany
manfred.reichert@uni-ulm.de

ABSTRACT

Knowledge workers are confronted with a massive load of data from numerous heterogeneous sources, making it difficult for them to identify the information relevant for performing their tasks. Particularly challenging is the alignment of information with business processes. In previous work, we introduced a *Semantic Network* (SN) for bridging this gap, i.e., for identifying relations between information and business processes. What has been neglected so far is the maintenance of an SN in order to keep the SN consistent, complete, and up-to-date. This paper tackles this issue and extends our approach with algorithms dealing with the maintenance of an SN. For this purpose, we identify and classify properties of objects and relations captured in an SN and show how these properties can be maintained. We use a case from the automotive domain in order to demonstrate and validate the feasibility and applicability of our maintenance framework.

Categories and Subject Descriptors

I.2.4 [**Computing methodologies**]: Artificial intelligence —*Knowledge Representation Formalisms and Methods*

General Terms

Algorithms

Keywords

Semantic network, evolution, maintenance

iiWAS '14, December 04 - 06 2014, Hanoi, Viet Nam
Copyright 2014 ACM 978-1-4503-3001-5/14/12 ...$15.00.
http://dx.doi.org/10.1145/2684200.2684287

1. INTRODUCTION

Knowledge workers and decision makers are confronted with a continuously increasing load of data during their daily work. Examples include e-mails, office files, checklists, guidelines, fact sheets, web pages, and best practices. In daily practice, knowledge workers and decision makers are not only interested in getting access to this data, but also require comprehensive and aggregated information when performing specific tasks in a business process context [19]. Handling such information is by far more complex and time-consuming than just storing data because such information can, for example, be incomplete, incorrect, unreliable, unstructured, or outdated [18, 25].

A particular challenge is to align information with business processes and to provide relevant information to knowledge workers and decision makers. In practice, information is not only stored in distributed and heterogeneous sources, but also managed separately from business processes. For example, shared drives, databases, enterprise portals, and enterprise information systems are used to store information. In turn, business processes and their tasks are managed using process-aware information systems [24].

In such an environment, information and business processes are often linked manually and statically, e.g., in enterprise portals. However, it has been shown that such enterprise portals rather contain complex and static content (e.g., long lists of documents, large process maps) confusing users [9]. Therefore, it is challenging for users to identify relations between information and business processes as required when performing a specific process task.

In previous work, we introduced a *Semantic Network* (SN) bridging the gap between information and business processes [16]. Such an SN can be created using a bottom-up approach, i.e., starting with the integration of information and business processes from heterogeneous sources [20]. Following this, the integrated information and business processes are syntactically and semantically analyzed. The resulting SN comprises unified *information objects* (e.g., checklists, guidelines, forms), *process objects* (e.g., pools, lanes, tasks, gateways, events), and semantic *relations* (e.g., "is similar to", "is used by"). More specifically, information objects are needed when performing business processes. In turn, process objects are process elements such as tasks or gateways that guide process-oriented work. Finally, semantic relations allow identifying inter-linked objects in different ways, e.g., information objects referring to the same topic or objects required for performing a particular process task.

An SN constitutes the basis for delivering relevant information to knowledge workers and decision makers (cf. Fig. 1). More specifically, an SN offers an application interface (cf. [8] for details) that may be queried to retrieve required information. Based on a query (e.g., "information objects relevant for creating a review"), the SN automatically delivers respective information objects to users [20].

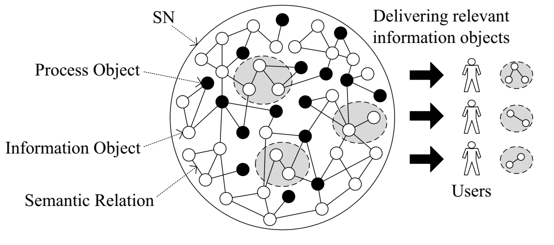


Figure 1: Delivering relevant information objects

In order to provide those information objects to users fitting their demands best, information and process objects in an SN must be up-to-date, complete, and consistent, i.e., an SN must be *maintained* over time. In two case studies and an online survey [10, 17] we have already shown that the maintenance of SNs constitutes a crucial prerequisite in order to continuously provide required information to knowledge workers and decision makers. Maintenance is, however, a non-trivial task. For example, objects may be integrated (e.g., new guidelines are created), updated (e.g., a task is modified), and deleted (e.g., a checklist is no longer valid). Likewise, relations may be established (e.g., two documents have the same author), updated (e.g., two forms become more similar to each other), and deleted (e.g., two documents no longer having the same author). Such changes can happen outside of an SN (e.g., a checklist has changed on a shared drive), which we call *exogenous* changes, or inside an SN over time (e.g., a guideline being out-of-date), which we call *endogenous* changes. Both exogenous and endogenous changes must be properly handled (cf. Fig. 2).

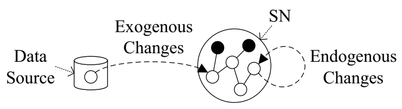


Figure 2: Exogenous and endogenous changes

Picking up the above mentioned challenges on the *maintenance* of SNs, the contribution is as follows: *(i)* We propose a framework for maintaining SNs. Specifically, we show how SNs evolve over time and identify characteristics of object and relation properties and their influence with respect to the maintenance of SNs. *(ii)* We introduce three algorithms dealing with exogenous and endogenous changes of an SN. We examine feasibility and cost of our algorithms by a proof-of-concept implementation and show by an empirical evaluation in the automotive domain that automatic maintenance of SNs is essential and practical at the same time.

The paper is organized as follows: We next introduce the preliminaries (cf. Section 2) and then present the maintenance framework in Section 3. Section 4 validates the cost of the algorithms and presents a case study demonstrating the applicability of the algorithms. Finally, we discuss related work (cf. Section 5) and conclude (cf. Section 6).

2. PRELIMINARIES

An SN constitutes a labeled and weighted directed graph whose vertices represent objects and the (labeled) edges represent the semantic relations between the objects with weights indicating the relevance of the relations. A weight is expressed in terms of a number ranging between 0 and 1, with 1 indicating the strongest possible semantic relation.

Definition 1. A *Semantic Network SN* is a tuple (V, E, L, W, f_l, f_w), where V is a set of *vertices* such that each $v \in V$ represents an information object or a process object; E is a multiset of *edges* such that each *edge* $e = (v, v') \in E$, $v, v' \in V$ and $v \neq v'$, represents a *relation* between such objects. The function $f_l : E \to L$ labels each edge $e \in E$ with an edge label from the set of labels L. Furthermore, the function $f_w : E \to W$ assigns a weight from the set of weights W to each edge $e \in E$. Given an edge $e = (v, v') \in E$, we call v the *source* and v' the *destination* of e.

Given a vertex $v \in V$, the *internal neighborhood* of v, denoted $\Gamma^-(v)$, is the set of vertices $\{v' \mid (v', v) \in E\}$. Analogously, for $v \in V$, the *external neighborhood* of v, denoted $\Gamma^+(v)$, is the set of vertices $\{v' \mid (v, v') \in E\}$. Then, the *total neighborhood* of $v \in V$ is the union of the internal and external neighborhood of v, denoted $\Gamma(v)$.

The *incoming degree* of a vertex $v \in V$ is the number of incoming edges and the *outgoing degree* is the number of its outgoing edges. The *total degree* of v is the sum of its incoming and outgoing degree.

For example, given two edges $e = (v, v'), e' = (v', v'') \in E$, $v, v', v'' \in V$, we call v an internal neighbor of v' and v'' an external neighbor of v'. Thus, the total degree of v' is 2.

Note that we often refer to vertices as objects (e.g., information and process objects) and to edges as relations of an SN. We next define properties for vertices and edges.

Definition 2. Each vertex $v \in V$ and each edge $e \in E$ has a set of *properties* $P(v)$ and $P(e)$, respectively, where each $p \in P(v) \cup P(e)$ is a pair (key, val). We denote *key* as the *unique name* and *val* as the *value* of p and write $key(v)$ $(key(e))$ to denote *val*.

In order to create an SN, business processes and pieces of information, possibly from different data sources (e.g., process repositories, shared drives), are transformed into process and information objects (cf. Figs. 3(a) and 3(b)), each represented by a vertex and its according properties. The transformation ensures that proprietary formats (e.g., office formats) are converted into a generic and uniform format, which allows for analyzing the objects in the SN.

After that, objects in an SN are syntactically and semantically analyzed to detect their semantic relations (cf. Fig. 3(c)) [8]. First, properties such as authorship are compared (*syntactic analysis*), e.g., to link objects with the same author. Second, the properties of the objects are analyzed (*semantic analysis*). For this purpose, algorithms from the fields of data mining, text mining (e.g., text preprocessing, linguistic preprocessing, clustering, classification, information extraction), pattern-matching, and machine learning (e.g., supervised learning, unsupervised learning, reinforcement learning, transduction) are applied [11, 26] to further classify and group correlated objects.

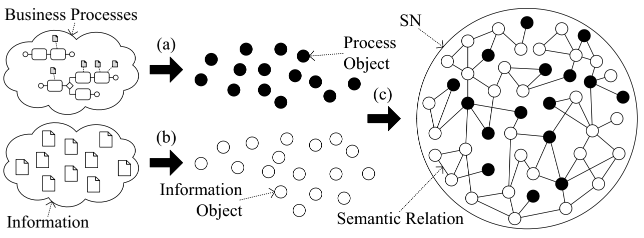


Figure 3: Schematic creation of an SN

Generally, semantic relations in an SN may exist between information objects (e.g., a guideline similar to another one) or process objects (e.g., an event triggering a sub-process). Additionally, semantic relations exist between information and process objects as well (e.g., an instruction required for executing a specific process task). A detailed description of the creation process is given in [20].

3. MAINTAINING SEMANTIC NETWORKS

This section introduces our maintenance framework. First, we show how an SN evolves over time (cf. Section 3.1). Second, we illustrate why properties of objects and relations are important (cf. Section 3.2). Third, we present a collection of algorithms for maintaining SNs (cf. Section 3.3).

3.1 Evolution

Evolution of SNs is driven by exogenous and endogenous changes. We further distinguish between evolution in *depth* and *breadth* [21]: *Depth* is defined by the *size* of all property values in an SN, i.e., the amount of information stored within all objects. *Breadth* is defined as the number of relations in an SN, i.e., the cardinality of the set of edges.

Depth can be increased by adding objects (e.g., new documents on a shared drive), adding properties (e.g., adding keywords to an existing document), or by updating values of properties (e.g., a property is described in more detail) (cf. Fig. 4(a)). In turn, deleting objects and properties decreases the depth of an SN (cf. Fig. 4(b)). Note, that updates of property values can decrease depth as well. Breadth can be increased by adding relations (e.g., a new link between two objects) (cf. Fig. 4(c)), while deleting relations (e.g., two objects no longer have the same author) decreases breadth (cf. Fig. 4(d)). Hence, depth and breadth are indicators for the cost of performing maintenance operations such as adding an object or a relation to an SN (cf. Section 4).

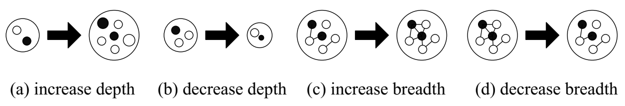


Figure 4: SN evolution in depth and breadth

In order to formalize such maintenance operations in an SN, we next define suitable actions, which can consider both exogenous and endogenous changes.

Definition 3. An *action* changes an SN. Each action a has a set of *parameters* $PA(a)$, where each $pa \in PA(a)$ is a pair (key, val). We call key the *unique name* and val the *value* of pa and write $key(a)$ to denote val. A parameter pa is either *mandatory* or *optional*. If pa with key key is mandatory, then, for each action a, there exists a value val_a such that $(key, val_a) \in PA(a)$.

Typical mandatory parameters of an action are a unique identifier uri (e.g., a URL) and the function $func$ (e.g., add, update, delete) to be executed. Actions are triggered by exogenous or endogenous changes (cf. Fig. 5), e.g., if a document on a shared drive is deleted (an *exogenous* change) or if a document is outdated (an *endogenous* change).

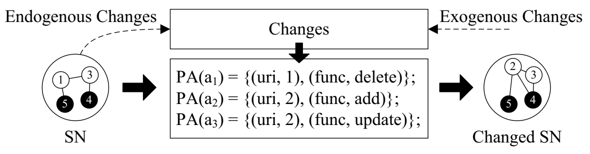

Endogenous Changes
Changes
Exogenous Changes
PA(a₁) = {(uri, 1), (func, delete)};
PA(a₂) = {(uri, 2), (func, add)};
PA(a₃) = {(uri, 2), (func, update)};
SN
Changed SN

Figure 5: Actions changing an SN

For example, consider an engineer in the automotive domain conducting a review of product requirements documented in functional specifications. The goal is to improve as well as to approve such specifications. Due to a revision of the review process, an employee from the quality management department replaced an outdated review template. Thus, an action a_1 was triggered with $uri(a_1)$ = "H:/templates/review-v1.xls" and $func(a_1)$ = "delete". Thereby, another action a_2 was triggered with $uri(a_2)$ = "H:/templates/review-v2.xls" and $func(a_2)$ = "add". However, based on new guidelines the engineer noticed that the template was incomplete (e.g., a required question was missing). Therefore, the engineer adapts the template. Thus, another action a_3 is triggered with $uri(a_3)$ = "H:/templates/review-v2.xls" and $func(a_3)$ = "update". Thereby, we ensure by actions that an SN is maintained.

3.2 Property Classification

As mentioned above, maintaining SNs requires not only to consider the object-relation-level, but also the properties of objects and relations. Furthermore, if, for example, the author of a document has changed, it is not necessary to update the entire object but only relevant parts, i.e., the value of the property *author* and according relations. One can observe that some properties change over time (e.g., *filesize*), whereas others do not (e.g., *uri*). This can be exploited in maintenance algorithms by focusing only on properties that are relevant for a particular operation. To capture this, we categorize properties as follows:

Definition 4. Properties are classified into two categories: *existence* and *mutability*. *Existence* expresses whether a property is *mandatory* or *optional*, where a property p with key key is mandatory for the vertices V (edges E) of an SN if, for each $v \in V$ ($e \in E$), there exists a value val_v (val_e) such that $(key, val_v) \in P(v)$ ($(key, val_e) \in P(e)$) and it is optional otherwise. *Mutability* expresses whether a property's value is *dynamic* or *static*, where p is *dynamic* if val in (key, val) can change over time and it is *static* otherwise.

For example, for $v \in V$, typical mandatory properties are a unique identifier *uri*, a data source *source*, a creation date *cdate*, a modification date *mdate*, and a content *cont*. The categories, existence and mutability, can be combined into a matrix comprising four blocks (a, b, c, d) to which we assign the properties (cf. Fig. 6).

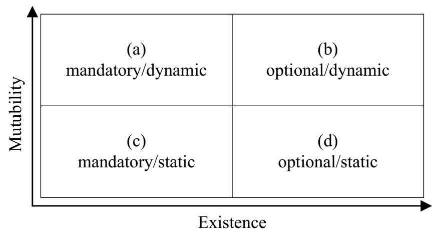


Figure 6: Property classification

In the following, we illustrate the assignment of individual properties to different blocks with examples:

(a) *mandatory/dynamic:* Some properties are always part of an SN and are dynamic. For example, the modification date *mdate* changes with every update of an object or a relation. The content *cont* or the total degree *deg* of an object can change over time as well. Therefore, these properties are mandatory and dynamic.

(b) *optional/dynamic:* The *title* of a document can change over time, but some filetypes (e.g., a text file) do not have a title. Therefore, the *title* of a document can be considered as optional and dynamic.

(c) *mandatory/static:* An identifier *uri* or a creation date *cdate* exists for all objects and relations and therefore is mandatory. Since these properties do not change over time, they can be considered as static.

(d) *optional/static:* If a property does not change over time and does not exist for every object or relation it is called optional/static, e.g., the *filetype* of a document.

Based on the property classification we infer the following for adding, updating, and deleting elements of an SN: One must ensure that static/mandatory properties (c) are given as a minimum requirement when adding elements (cf. Fig. 7(a)). When deleting elements one has to consider properties within all blocks (a, b, c, d) (cf. Fig. 7(b)). When executing updates, only properties which are not assigned to the mandatory/static block (a, b, d) must be considered (cf. Fig. 7(c)). Note that the grey background color in Fig. 7 indicates affected blocks for each function (cf. Section 3.3).

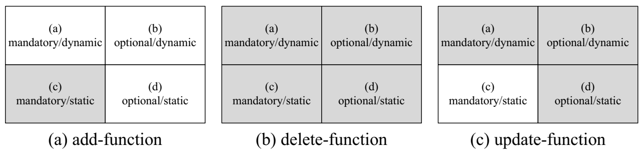


Figure 7: Property classification and functions

3.3 Algorithms

We next specify functions (*add*, *delete*, and *update*) used in our algorithms that perform maintenance operations.

The *add*-function adds a vertex v_{add} and its properties to a Semantic Network SN and determines which semantic relations exist between v_{add} and existing vertices. As mentioned above, mandatory/static properties are the minimum input parameter for the *add*-function.

```
Function add($SN, v_{add}$)
  Require: $SN = (V, E, L, W, f_l, f_w)$ an SN,
  $v_{add}$ the vertex to be added incl. its properties $P(v_{add})$;
  Ensure: $SN$ is updated;
  $V := V \cup \{v_{add}\}$;
  foreach $v \in V$ do
      if $uri(v) \neq uri(v_{add})$ then
          $E := E \cup \{$new edge/s between $v$ and $v_{add}\}$;
```

The *delete*-function deletes a vertex v_{del} in a Semantic Network SN including existing semantic relations between v_{del} and its total neighborhood. Note that the function considers all blocks of the property classification, i.e., all properties of v_{del} are deleted.

```
Function delete($SN, v_{del}$)
  Require: $SN = (V, E, L, W, f_l, f_w)$ an SN,
  $v_{del}$ the vertex to be deleted incl. its properties $P(v_{del})$;
  Ensure: $SN$ is updated;
  $v :=$ get $v \in V$ s.t. $uri(v) = uri(v_{del})$;
  $E := E \setminus \{(v, \gamma), (\gamma, v) \mid \gamma \in \Gamma(v)\}$;
  $V := V \setminus \{v\}$;
```

The *update*-function takes a vertex v_{upd} as input, which is used to update the vertex v in the SN that is identified by the same uri as v_{upd}. The function also adds, deletes, and updates semantic relations between the updated vertex v and existing vertices. Note that mandatory/static properties are not considered in case of objects.

```
Function update($SN, v_{upd}$)
  Require: $SN = (V, E, L, W, f_l, f_w)$ an SN,
  $v_{upd}$ the vertex used to update the SN incl. its
  properties $P(v_{upd})$;
  Ensure: $SN$ is updated;
  $P(v_{upd}) := \{p \in P(v_{upd}) \mid p$ is not mandatory/static$\}$;
  $v :=$ get $v \in V$ s.t. $uri(v) = uri(v_{upd})$;
  foreach $(key, val) \in P(v)$ do
      if $(key, val_{upd}) \in P(v_{upd})$ then
          $val := val_{upd}$;
          $P(v_{upd}) := P(v_{\mathrm{upd}}) \setminus \{(key, val_{upd})\}$;
      else
          $P(v) := P(v) \setminus \{(key, val)\}$;
  $P(v) := P(v) \cup P(v_{upd})$;
  foreach $v' \in V$ do
      if $v' \in \Gamma(v)$ then
          $E :=$ update edge/s betw. $v'$ and $v$;
          $E := E \setminus \{$obsolete edge/s between $v'$ and $v\}$;
      if $uri(v') \neq uri(v)$ then
          $E := E \cup \{$new edge/s between $v'$ and $v\}$;
```

Based on these functions, we propose three algorithms for maintaining SNs. The maintenance is based on two main principles: the *push-* and the *pull-principle*. The former can be applied to both exogenous and endogenous changes, whereas the latter can only be applied to exogenous changes.

With the *push-principle*, the data source pushes information and business processes automatically to an SN when they are added, updated, or deleted within the data source. However, with regard to exogenous changes, prerequisite is that the data source is able to send notifications if information and/or business processes have been changed. Regarding endogenous changes, the prerequisite is that the SN detects changes automatically and triggers respective actions.

With the *pull-principle*, an SN gathers information and business processes from a data source. Such a maintenance process is triggered by time-based schedulers, i.e., the SN is maintained at a certain point in time. The principle is used for data sources which are not capable of sending change notifications (e.g., a document has changed) to the SN.

However, the use of a specific principle depends on the capabilities of a data source (whether the data source is able to send change notifications to the SN or not). For example, for an enterprise information system which is capable of sending notifications we use the push-principle whereas for a shared drive we use the pull-principle.

For each of these two principles, we introduce a corresponding algorithm (cf. Fig. 8). Prerequisite for both principles is that the SN has access to underlying data sources. In case of exogenous changes the SN transforms information and business processes into a uniform format. In case of endogenous changes no transformation is necessary.

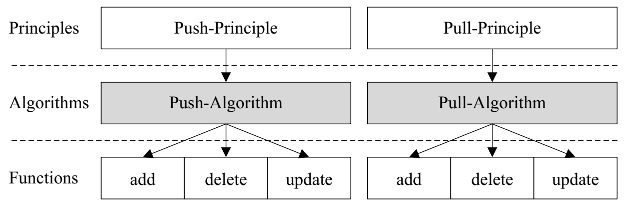


Figure 8: Push- and Pull-Algorithm

3.3.1 Push-Algorithm

The Push-Algorithm deals with changes of an SN based on the push-principle, e.g., a policy is no longer valid in 2014 and the corresponding object in the SN has to be maintained accordingly. Thus, the maintenance of the SN is triggered by an action that is applied to the SN by the Push-Algorithm.

The algorithm works as follows: In the add and update case, we create a vertex v including its properties from the data source affected by the action a (i.e., based on the *uri* of the action). After that, we call the corresponding function. In the delete case, we identify the corresponding vertex $v \in V$ based on the *uri* of the action and call the according function. Hence, the Push-Algorithm applies endogenous and exogenous changes to the SN by actions.

3.3.2 Pull-Algorithm

The Pull-Algorithm deals with changes of an SN based on the pull-principle, i.e., data has changed in the data source and needs to be gathered by the SN. For example, documents on a shared drive are updated and, therefore, respective changes have to be made in the SN. As aforementioned, the maintenance of the SN is triggered by a scheduler.

Algorithm 1: Push-Algorithm

```
Require: $SN = (V, E, L, W, f_l, f_w)$ an SN,
$a$ an action;
Ensure: $SN$ is updated;
switch $func(a)$ do
    case $add$
        $v$ := create a vertex incl. its properties from
            the data source affected by the $uri$ of $a$;
        $add(SN, v)$;
        break;
    case $update$
        $v$ := create a vertex incl. its properties from
            the data source affected by the $uri$ of $a$;
        $update(SN, v)$;
        break;
    case $delete$
        $v$ := get $v \in V$ s.t. $uri(v) = uri(a)$;
        $delete(SN, v)$;
```

The algorithm works as follows: First, we create a set of vertices V_{ds} from a data source ds. After that, for each vertex $v \in V$ in the SN that was created from ds (property $source(v)$), we check if a corresponding vertex $v_{ds} \in V_{ds}$ exists. If this is the case, we check if v_{ds} is newer than v (e.g., by comparing the creation and/or modification dates). If v is out-of-date, it is updated with the properties of v_{ds} by calling $update(SN, v_{ds})$. After that, v_{ds} is removed from V_{ds}. If no corresponding vertex exists in the data source, we delete the vertex (v) in the SN using the $delete(SN, v)$ function. Finally, we add each remaining vertex $v_{ds} \in V_{ds}$ from the data source to the SN by calling $add(SN, v_{ds})$, which leaves the SN synchronized with the data source. Hence, the Pull-Algorithm allows for maintaining the SN at a certain point in time. Note that the process has to be repeated for each data source that has to be synchronized.

Algorithm 2: Pull-Algorithm

```
Require: $SN = (V, E, L, W, f_l, f_w)$ an SN,
$ds$ the data source;
Ensure: $SN$ is updated;
$V_{ds}$ := create a set of vertices incl. their properties
    from the data source $ds$;
foreach $v \in V$ s.t. $source(v) = ds$ do
    $v_{ds}$ := get $v_{ds} \in V_{ds}$ s.t. $uri(v_{ds}) = uri(v)$;
    if $v_{ds}$ then
        if $v_{ds}$ is newer than $v$ then
            $update(SN, v_{ds})$;
        $V_{ds} := V_{ds} \setminus \{v_{ds}\}$
    else
        $delete(SN, v)$;
foreach $v_{ds} \in V_{ds}$ do
    $add(SN, v_{ds})$;
```

3.3.3 Partial-Pull-Principle and Algorithm

In practice, SNs can comprise a large amount of objects and relations. Maintaining SNs using the pull-principle can, therefore, be a very time-consuming task. In a specific work context, a user might, however, only be interested in a selected part of the SN. For example, during a review, review templates, review minutes, existing reviews, or even results of a real-time evaluation (e.g., prioritization of projects in a workshop) are of great importance, while checklists and best practices for performing effective project management are less interesting. Thus, it is sufficient to maintain only these objects and relations that are relevant to the user when querying the SN. To capture this, we introduce a further principle, called the *partial-pull-principle*, where the SN gathers only information and business processes from data sources as requested by a user. These (and only these objects) are then updated on demand.

In contrast to the other principles, the partial-pull-principle is completely *user-driven* because it is triggered by a user request, whereas the push- and pull-principle are *machine-driven*, e.g., through notifications from other enterprise information systems or schedulers.

Based on the partial-pull-principle, we introduce a third algorithm (cf. Fig. 9) as a lightweight version of the Pull-Algorithm. It does not maintain the entire SN, but only the parts which are relevant for a given request.

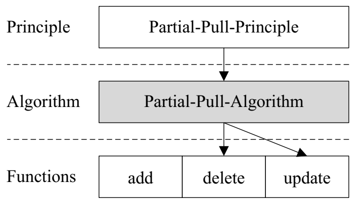


Figure 9: Partial-Pull-Algorithm

The algorithm works as follows: First, we create a set of vertices V_{ds} from affected data sources according to the user request *req*. Then, for each vertex $v \in V$ affected by *req* we retrieve the corresponding vertex v_{ds} from the affected data sources. Thus, if a corresponding vertex v_{ds} is in the data source, we check if v_{ds} is newer than v (e.g., by comparing the creation and/or modification dates) and if v is out-of-date, it is updated with v_{ds} by calling $update(SN, v_{ds})$. If there is no corresponding vertex in the data source, we delete the vertex (v) in the SN using the $delete(SN, v)$ function. Hence, the Partial-Pull-Algorithm allows for maintaining parts of an SN based on a user request and ensures that all requested objects including their properties are synchronized with affected data sources.

4. VALIDATION

To prove that our algorithms are able to successfully maintain SNs, we implemented them and evaluated their performance in consideration of depth and breadth. We further evaluated the necessity and application of our algorithms in a real-world case study in the automotive domain.

Our validation is guided by three research questions:

RQ1 *Is automatic maintenance of SNs feasible in consideration of exogenous and endogenous changes?*

RQ2 *How do depth and breadth affect the runtime of maintenance operations?*

RQ3 *How essential is automatic maintenance of SNs?*

Algorithm 3: Partial-Pull-Algorithm

```
Require: $SN = (V, E, L, W, f_l, f_w)$ an SN,
$req$ the request to an SN;
Ensure: $SN$ is partially updated;
$V_{req}$ contains the requested vertices;
$V_{ds}$ := create a set of vertices from the
        data sources affected by $req$;
foreach $v \in V$ affected by $req$ do
    $v_{ds} :=$ get $v_{ds} \in V_{ds}$ s.t. $uri(v_{ds}) = uri(v)$;
    if $v_{ds}$ then
        if $v_{ds}$ is newer than $v$ then
            $update(SN, v_{ds})$;
            $v_{upd} :=$ get $v' \in V$ s.t. $uri(v') = uri(v_{ds})$;
            $V_{req} := V_{req} \cup \{v_{upd}\}$;
        else
            $V_{req} := V_{req} \cup \{v\}$;
    else
        $delete(SN, v)$;
```

4.1 Implementation and Configuration

Driven by these research questions, we implemented a prototype installed on a laptop with an Intel quad-core CPU, 16 GB RAM, and a Windows 7 64-bit operating system. The prototype is a web-based Java (JRE7)/Scala (2.10)[1] application realized as a 3-tier architecture.

The *presentation layer* uses the web application framework Play,[2] the Twitter Bootstrap framework,[3] the Data-Driven Documents (D3) library,[4] jQuery,[5] HTML 5 templates, and Cascading Style Sheets (CSS) 3. We created our SNs using the semantic middleware iQser GIN Server (2.0.0.36) [26]. In addition, we developed several Java open-source plugins[6] on the *logic layer*. The *data layer* is realized by a Lucene search index[7] and a MySQL database.[8]

Based on our property classification (cf. Section 3.2) we configured objects and relations of our prototype. As mandatory/static object properties we choose *cdate*, *uri*, and *source* (cf. Fig. 10(a)). Optional properties are *filetype* and *title*. While the *filetype* does not change over time (static), the *title* can vary (dynamic), but it might not be available for every *filetype* (e.g., text file) and, therefore, it is optional. Mandatory dynamic properties are the *buzz* (activity on object), *cont*, *deg*, and *mdate*.

Analogous to object properties, *uri* and *cdate* are mandatory for relations (cf. Fig. 10(b)). However, relations have additional mandatory properties such as the *source vertex* and *destination vertex*. The *label* of an edge is also mandatory, while its relevance (*weight*) can vary, e.g., changing *cont* may affect the *weight* of “is similar to” relations. Additionally, the *reason* of a relation, which describes why a relation has been established (e.g., a particular method which detected a “is similar to” relation), is static and optional.

[1]http://www.scala-lang.org/
[2]http://www.playframework.com/
[3]http://getbootstrap.com/
[4]http://d3js.org/
[5]http://jquery.com/
[6]http://sourceforge.net/directory/?q=nipro
[7]http://lucene.apache.org/
[8]http://www.mysql.com/

We split our validation into two parts: The first part addresses the technical perspective (cf. Section 4.2), i.e., the influence of depth and breadth on our algorithms, whereas the second part evaluates their application in an empirical case study in the automotive domain (cf. Section 4.3).

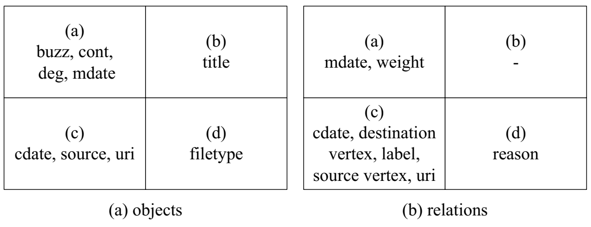


Figure 10: Property classification implementation

Therefore, we created two SNs based on the configuration mentioned above: one with synthetic text files to evaluate the runtime behavior of the algorithms and one with real-world documents from the automotive domain to evaluate the functional perspective of our maintenance framework. Both SNs were created with the following relations: "is author of", "has same title as", and "is similar to". The successful implementation and initial tests have already shown that automatic maintenance of SNs is feasible in general. Therefore, we now examine the runtime in consideration of depth and breadth. Finally, we evaluate the necessity and practicability of automatic maintenance of SNs in a business environment.

4.2 Technical Validation

In order to address RQ1 and RQ2, we created SNs containing 5, 50 and 500 objects once with small (1 KB) and once with larger (100 KB) files. To obtain comparable results for our measurements, all objects within an SN were identical (e.g., identical property *cont*). Therefore, we simulated the worst-case scenario regarding performance: Every object was connected with every other object in each SN yielding 40, 4,900 and 499,000 relations. Note, only "is similar to" and "is author of" relations were detected since text files do not have the property *title* and therefore no "has same title as" relations were recognized. Based on the initial SNs, we performed operations (add, update, delete) with a single object using our Push- and Pull-Algorithm (cf. Section 3.3). Each combination of these dimensions represented one case to be examined with regard to small and large files (36 cases in total, e.g., updating an object with a size of 1 KB through the Push-Algorithm in an SN containing 5 objects).

For each case, the shown numbers in the diagrams (cf. Figs. 11-13) constitute averages over three warm runs. Warm runs were chosen to ensure comparability of the measured values since the iQser GIN Server [26] performs several initial background tasks on start-up. Note the logarithmic scale used in the diagrams.

Fig. 11 shows that objects can be added to an SN in linear time. Compared to the number of objects within an SN, the depth has a bigger influence on the runtime as shown by the comparison with objects of different size (1 KB vs. 100 KB). Detecting relations between the added and existing objects is polynomial in the number of vertices. However, note that the actual performance of the algorithms also depends on the property values of vertices. For example, the value of *cont* (i.e., the *size* of the property in bytes) affects the analysis of similarity relations between the added and existing vertices.

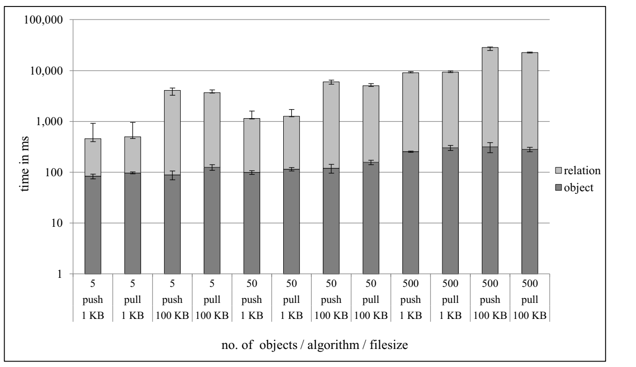


Figure 11: Effect of depth and breadth on additions

As shown in Fig. 12 update operations perform similarly. In contrast to the integration of objects, however, some operations are not required (e.g., for mandatory static properties). Comparisons between existing properties take place instead (e.g., to check whether a property has changed). Nevertheless, we could not detect significant differences concerning cost between add and update operations on identical initial situations.

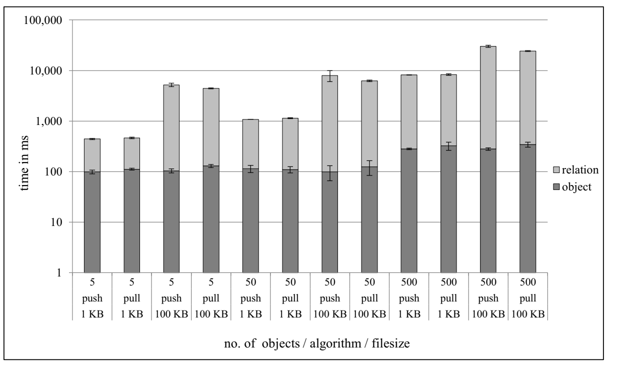


Figure 12: Effect of depth and breadth on updates

In contrast to add and update operations, delete operations perform differently. While the deletion of objects can be done in linear time, the deletion of relations varies significantly with the size of the objects. The reason is that all references, which form the basis of a relation (e.g., extracted concepts and co-occurrences of a specific object in case of "is similar to" relations) must be also deleted. Note that such "housekeeping" tasks are controlled by the iQser GIN Server. In turn, this might be the reason for differences in measured values resulting in unexpected outcomes. For example, the cost for deleting an object with a size of 1 KB out of 5 objects was higher than deleting an object with a size of 100 KB out of 5 objects (cf. Fig. 13). Furthermore, measurements with the programming language Java can be less accurate compared to native programming languages (e.g., the Java garbage collector cannot be disabled).

Despite the fact that the implemented technology might cause inaccuracies in measurements, we were still able to verify that maintenance cost with our algorithms is highly dependent on depth and breadth (RQ2). However, external components (e.g., a component for the semantic analysis of object properties) and their computation cost can influence the performance of maintenance operations significantly. In contrast, the consideration of our property classification (cf. Section 3.2) can have positive effects on performance if only necessary operations (e.g., only updating properties which are not mandatory/static) are executed.

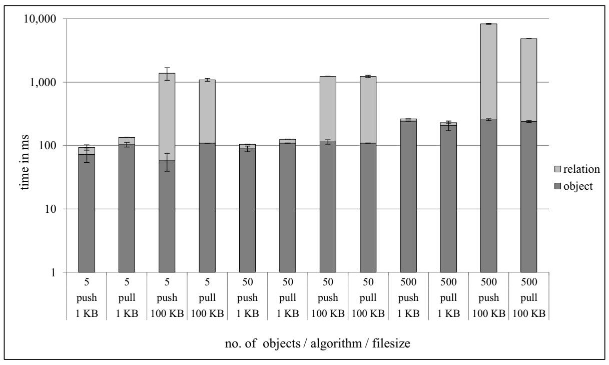


Figure 13: Effect of depth and breadth on deletes

Altogether, the Push- and the Pull-Algorithm both ensure a synchronized SN over affected data sources after completion. Thereby, automatic maintenance is feasible (RQ1).

4.3 Empirical Validation

As proposed by Yin [27] and Kitchenham et al. [12], we performed a case study to evaluate RQ3 in a "typical" project setting with knowledge workers and decision makers from several innovation departments in the automotive domain. Thus, all participants were involved in knowledge-intensive business processes. The participants were selected based on their expert knowledge regarding the considered case. No participant was a member of the research team.

The case study was performed in July 2014 with five decision makers and six knowledge workers from a large automotive manufacturer. It was conducted based on a questionnaire comprising 60 questions.[9] Each interview lasted around 90 minutes.

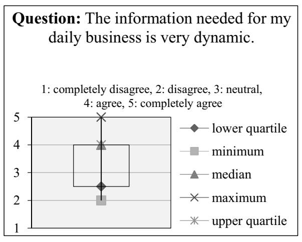
First, the participants had to answer general questions about their current work environment and information handling within their processes. Afterwards they had to perform tasks with our prototype and answer questions based on these tasks with respect to maintenance (i.e., add, update, delete, and search for objects and validate property values as well as detected relations). The underlying corpus (data sources) contained 333 internal documents from their field of interest (e.g., scientific papers from departments dealing with technology monitoring and technology development). Therefore, the users were familiar with the information represented in the SN and able to judge the containing information and its quality on a certain level.

In order to generalize the results and to gain further insights, we asked the users some concluding questions when all tasks had been completed. More specifically, RQ1 was addressed from the user's perspective by the tasks with associated questions, whereas RQ3 was investigated by the introducing and concluding questions.

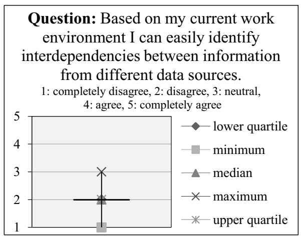
[9]Available at http://nipro.hs-weingarten.de/casestudy

The initial questions about information handling in the users' current work environment revealed that, with the exception of personal contacts (e.g., in meetings, phone calls), information is mostly handled and accessed in a digital way. Information is, however, mostly not well-structured and often distributed in different sources (e.g., information systems or shared drives), which makes it difficult to search for it. These statements are endorsed by the fact that a significant amount of information is stored in files.

Given the fact of enormous file usage, a consequential challenge working with files in a business environment is keeping track of changes (cf. Fig. 14). This is mainly substantiated by the dynamics and amount of information available as well as a lack of technological assistance (e.g., search over different data sources, notifications on changed content).



(a) Information dynamics



(b) Information overview

Figure 14: Information characteristics

In order to address these challenges (e.g., dynamics of information in distributed sources, heterogeneous filetypes), we introduced our prototype (which captures such dynamics) and asked the participants to perform tasks with the offered graphical user interface (e.g., check if property values were updated, validate the correctness of a relation).

In the concluding questions all participants stated that every task with our algorithms could be completed successfully. Since objects, relations, and properties were adapted based on exogenous and endogenous changes, we can confirm that automatic maintenance of SNs is feasible in practice.

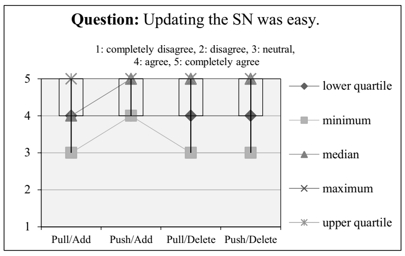


Figure 15: Simplicity of maintenance for users

However, some users could not recognize any relation between the document they added to the SN and others. While no "is author of" and "has same title as" relation existed, the "is similar to" relations were not shown. For more sophisticated user experience we filtered these relations according to a hardcoded weight-threshold with respect to the overall amount of "is similar to" relations. Nevertheless, we could verify with according database entries that the relations were set and take this as an input for further user interface development by allowing users to set the threshold dynamically.

Additionally, we asked the users about their preference between push and pull. All users preferred the Push-Algorithm since the effects of their tasks are reflected immediately in the user interface. By contrast, the Pull-Algorithm always has a delay since it is triggered at a certain point in time. A synchronization every three minutes caused obviously too much delay for some users, even when performing other tasks in between. However, the Partial-Pull-Algorithm, which addresses this downside, received positive feedback from the participants. Note that the Partial-Pull Algorithm only considers objects currently existing in an SN.

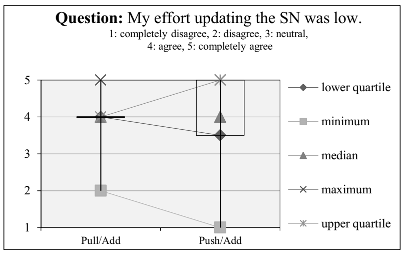


Figure 16: Maintenance effort for users

We interpret the disadvantage of the Pull-Algorithm as a confirmation of RQ3 (automatic maintenance is essential) and recommend the Push-Algorithm if technology permits offering an improved real-time-experience to users. For the Pull-Algorithm, the update intervals can be shortened, depending on the number of files in a data source. Nevertheless, the delay of the Pull-Algorithm, as observed by the users, is caused by the configured time interval of implemented schedulers. However, its performance compared to the Push-Algorithm is almost identical (cf. Section 4.2).

Concluding, we asked the users about their impression concerning benefits for their daily work in consideration of SN maintenance. All users stated that integrating and connecting distributed information was easy when using the prototype (cf. Fig. 15) and could be done with reasonable effort (cf. Fig. 16). Additionally they confirmed the benefit of their daily work (cf. Fig. 17).

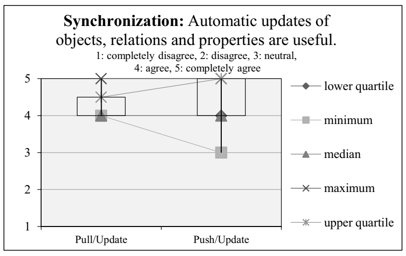


Figure 17: User perspective on auto. maintenance

More than 90% of the participants also stated that an SN provides an enhanced overview on information and that a maintained SN is desirable. In particular, this would be a benefit compared to distributed data sources. One user summarized: "Maintenance and corresponding updates should be automatic. No user interaction for maintenance should be required. Users should focus on working with the SN." Asking about use cases for a maintained SN, both, knowledge workers and decision makers recognized further potentials of the SN to support their daily work. For example, an expert search for people assigned to objects (e.g., authors) and their relations with each other as well as decision support scenarios (e.g., detecting patterns within the SN).

4.4 Discussion

We have shown that automatic maintenance of SNs with regard to both exogenous and endogenous changes is feasible with acceptable cost (RQ1). Furthermore, we examined the effect of depth and breadth on the runtime of the proposed algorithms (RQ2). Both algorithms performed satisfactorily in terms of adding, updating, and deleting objects. The cost of detecting relations, however, varies widely when expensive linguistic and/or statistical algorithms are used.

All users appreciated a unified single point of information access, which is up-to-date and makes distributed information searchable. Many of the interviewed users already experienced a well maintained SN as an enabler for various use cases (e.g., expert search, gap analysis). In particular, knowledge workers and decision makers can benefit from maintained SNs. Both groups emphasized that relations between objects can be interesting for various purposes (e.g., navigation within an SN or decision support).

Our case study results on the deficits of information handling in current business environments (cf. Section 4.3) have shown that it becomes increasingly crucial to provide up-to-date, integrated, and homogenous views on enterprise information. The empirical validation confirms that there is a high demand for a single point of information access in knowledge-intensive processes. Finally, we verified that a maintained SN is not only essential for such processes, but a practicable solution as well (RQ3).

5. RELATED WORK

Generally, various types of SNs exist. Examples include *associative networks* [5], *topic networks* [22], *fact networks* [23], or *ontologies* [13]. They usually have a special field of application (e.g., semantic search, reasoning). All of them (including our approach) have in common, that they have to be maintained. Depending on the expressiveness, however, maintenance effort varies widely. Usually the higher the expressiveness, the higher the creation and maintenance effort [23]. In practice, most of the above mentioned SNs have to be maintained manually. However, several semi-automatic maintenance approaches exist such as ontology maintenance using textual analysis [7], ontology mapping maintenance support [3], or SN integrity maintenance [2].

In turn, further approaches such as *data warehousing* [14], *enterprise architecture management* (EAM) [6], or *search engines* [15] have to deal with maintenance as well. However, manual effort is usually necessary (e.g., schema mapping) [1]. For example, the literature study of Farwick et al. [4] confirms that there exists no related work addressing the problem of automated EA maintenance.

Our SN represents information in a unified, integrated, user-, and machine-interpretable form. Unlike existing maintenance approaches, our approach focuses on the integration of distributed and heterogeneous information and business processes including the detection of their interdependencies. We showed that no manual effort is needed to maintain such an SN, but manual effort can increase the expressiveness, i.e., better conceptualization and delimitation of concepts (e.g., for enhanced detection of relations).

6. SUMMARY AND OUTLOOK

This paper presented a framework for maintaining SNs, i.e., adapting exogenous and endogenous changes. Therefore, we presented three algorithms. Furthermore, our property classification allows maintaining SNs efficiently.

Our validation confirms that our maintenance framework is able to keep SNs synchronized with underlying sources in reasonable time. Moreover, we applied our algorithms to a real-world case, i.e., validated them based on an implementation in the automotive domain.

In future, we will improve our algorithms to support self-learning components with a focus on endogenous changes.

7. REFERENCES

[1] P. Bernstein and L. Haas. Information integration in the enterprise. *Communications of the ACM*, pages 72–79, 2008.

[2] T. Čapek. Semantic network integrity maintenance via heuristic semi-automatic tests. In *Proc. of the RASLAN Workshop 2009*, pages 63–67. Masaryk University, 2009.

[3] D. Dinh, J. Dos Reis, C. Pruski, M. Da Silveira, and C. Reynaud-Delaître. Identifying relevant concept attributes to support mapping maintenance under ontology evolution. *J. of Web Semantics: Science, Services and Agents on the World Wide Web*, 2014.

[4] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke. Requirements for automated enterprise architecture model maintenance - a requirements analysis based on a literature review and an exploratory survey. In *Proc. of the 13th Int'l Conf. on Enterprise Information System (ICEIS'11)*, pages 325–337. SciTePress, 2011.

[5] N. V. Findler. *Associative Networks: The Representation and Use of Knowledge of Computers.* Academic Press, 1979.

[6] R. Fischer, S. Aier, and R. Winter. A federated approach to enterprise architecture model maintenance. In *Proc. of the 2nd Int'l Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07)*, pages 9–22. Springer, 2007.

[7] Y. Gargouri, B. Lefebvre, and J.-G. Meunier. Ontology maintenance using textual analysis. In *Proc. of the 7th World Multi-Conf. on Systemics, Cybernetics and Informatics (SCI'03)*. International Institute of Informatics and Systemics, 2003.

[8] M. Hipp, B. Michelberger, B. Mutschler, and M. Reichert. A framework for the intelligent delivery and user-adequate visualization of process information. In *Proc. of the 28th Symp. on Applied Computing (SAC'13)*, pages 1383–1390. ACM Press, 2013.

[9] M. Hipp, B. Mutschler, B. Michelberger, and M. Reichert. Navigating in process model repositories and enterprise process information. In *Proc. of the 8th Int'l Conf. on Research Challenges in Information Science (RCIS'14)*. IEEE Computer Society, 2014.

[10] M. Hipp, B. Mutschler, and M. Reichert. On the context-aware, personalized delivery of process information: Viewpoints, problems, and requirements. In *Proc. of the 6th Int'l Conf. on Availability, Reliability and Security (ARES'11)*, pages 390–397. IEEE Computer Society, 2011.

[11] A. Hotho, A. Nürnberger, and G. Paaß. A brief survey of text mining. *LDV Forum*, 20(1):19–62, 2005.

[12] B. Kitchenham, L. Pickard, and S. L. Pfleeger. Case studies for method and tool evaluation. *IEEE Software*, 12(4):52–62, 1995.

[13] L. W. Lacy. *OWL: Representing Information Using the Web Ontology Language.* Trafford Publishing, 2005.

[14] J. Lechtenbörger. *Data warehouse schema design.* Akademische Verlagsgesellschaft AKA, PhD Thesis, University of Münster, 2001.

[15] D. Lewandowski. Web searching, search engines and information retrieval. *Information Services & Use*, 18(3), 2005.

[16] B. Michelberger, B. Mutschler, M. Hipp, and M. Reichert. Determining the link and rate popularity of enterprise process information. In *Proc. of the 21st Int'l Conf. on Cooperative Information Systems (CoopIS'13)*, pages 112–129. Springer, 2013.

[17] B. Michelberger, B. Mutschler, and M. Reichert. On handling process information: Results from case studies and a survey. In *Proc. of the 2nd Int'l Workshop on Empirical Research in Business Process Management (ER-BPM'11)*, pages 333–344. Springer, 2011.

[18] B. Michelberger, B. Mutschler, and M. Reichert. Towards process-oriented information logistics: Why quality dimensions of process information matter. In *Proc. of the 4th Int'l Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'11)*, pages 107–120. Koellen-Verlag, 2011.

[19] B. Michelberger, B. Mutschler, and M. Reichert. A context framework for process-oriented information logistics. In *Proc. 15th Int'l Conf. on Business Information Systems (BIS'12)*, pages 260–271. Springer, 2012.

[20] B. Michelberger, B. Mutschler, and M. Reichert. Process-oriented information logistics: Aligning enterprise information with business processes. In *Proc. of the 16th Int'l EDOC Conf. (EDOC'12)*, pages 21–30. IEEE Computer Society, 2012.

[21] S. Oertelt and K. Ulmschneider. Prozessintegrierter Einsatz virtueller Methoden im strategischen Technologie- und Innovationsmanagement. In *Proc. of KnowTech - Wissensmanagement und Social Media - Markterfolg im Innovationswettbewerb*, pages 485–498. GITO, 2013.

[22] J. Park and S. Hunting. *XML Topic Maps: Creating and Using Topic Maps for the Web.* Addison-Wesley Professional, 2002.

[23] K. Reichenberger. *Kompendium semantische Netze: Konzepte, Technologie, Modellierung.* Springer, 2010.

[24] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies.* Springer, 2012.

[25] J. Rowley. The wisdom hierarchy: representations of the DIKW hierarchy. *J. of Information Science*, 33(2):163–180, 2007.

[26] J. Wurzer. New approach for semantic web by automatic semantics. In *Proc. of the 2nd European Semantic Technology Conf. (ESCT'08)*, 2008.

[27] R. K. Yin. *Case Study Research: Design and Methods.* Sage Publications, 2009.