Faculty of
Computer Science

universität
wien

Department of Distributed and
Multimedia Systems

Technical Report TR-20090902 September 2009

The Sile Model: An Abstract Model for the Representation of Information Units in Desktop Environments

No Author

The Sile Model: An Abstract Model for the Representation of Information Units in Desktop Environments

Bernhard Schandl

September 2, 2009

Abstract

In this report we discuss a novel data model to represent personal data by describing a number of design considerations and requirements. We introduce, in an informal way, the basic concepts that we include in the model, and give a formal specification of the model elements.

1 Design Considerations

Typically, hierarchical file systems provide relatively weak organizational metaphors. More sophisticated information processing middleware can either be implemented as layers that entirely hide the file system structures from the outside (i.e., applications or end users); an example for this class are relational databases. Alternatively, they can establish data structures (for instance, triple stores) that coexist with file systems; we denote the latter as *hybrid approaches.* There, applications or end users simultaneously operate directly on both the file system and the additional structures. Such approaches for semantic enrichment of files are widely found, either providing very specific functionality (cf. Kante [MH07] or Punakea [Cor]), or constitute an all-embracing generic semantic layer that presents an integrated view on all user data (cf. NEPOMUK [GHM$^+$07] or Haystack [KBH$^+$03]). Without doubt such extended semantic desktop systems increase the data quality and experience for end users [SH08], they suffer from a number of fundamental problems:

- *Identiscription problem.* Hierarchical file systems intermingle the functions of *identification* and *description* of files [OG92]. In the context of a physical device, a file is uniquely identified by the combination of its *path* and its *file name.* However, the path and file name are also used to *describe* the file, and to relate it to other files (by putting them into a common directory). Consequently, when the description of a file is changed (e.g., when a directory is renamed), references to the file become invalid. As a further consequence, it is not possible to attach multiple descriptions to a file, since this would require storing it into multiple directories. This problem is only partially solved by using symbolic links or similar techniques.

- *Mapping problem.* File systems provide a minimal level of descriptive metadata in the form of file names and directories. If a semantic component is established in addition to a file system, it is desirable to include or reflect this information within the semantic layer; e.g., by mapping directory names to ontology classes. Often, such a mapping is not straightforward: since file names and directories usually have no formal semantics [DB99], their meaning cannot directly be captured and mapped to formal constructs.

- *Update problem.* Even if a mapping for a meaningful translation between a hierarchical file system and a semantic layer can be established, there remains the danger of inconsistencies that result from changes in one layer that are not propagated to the other one. To solve this, propagation mechanisms have to be integrated into file system implementations. This constitutes potential performance and security flaws because it requires hooking or modifying code on the operating system level.

- *Portability problem.* Nearly all native file systems support the storage of some form of metadata, e.g., extended attributes, alternate data streams, or resource forks. Since there exists no widely accepted standard for file metadata, this information is often lost when files are transfered across platforms. This may be one reason why such extended metadata facilities are rarely used by applications. A hybrid semantic system operating in parallel to a file system can either ignore platform-specific metadata management facilities (and thus loose the benefit of a tighter integration into the operating system), or use them and provide mappings to the semantic layer, which again raises the Mapping and Update problems.

Any hybrid approach that coexists with hierarchical file system must deal with these issues, which potentially leads to increased complexity and error proneness. Many of these problems could be solved if hierarchical file systems could be disbanded in favor of a commonly accepted semantic file system that allows one to store, annotate, and retrieve arbitrary data objects. Such a system could serve as the common infrastructure for a semantic desktop and its applications.

In the following we discuss requirements and design considerations for a data model of an integrated semantic file system. We emphasize the term *integrated* since many of the problems outlined in the previous section can be avoided by tightly coupling data with descriptive metadata. We outline the different aspects that we considered during the development of the abstract model and the definition of a concrete digital manifestation.

1.1 Identification

To overcome the identiscription problem described above, each file must have an unique, immutable identifier. Such a global unique id can serve as the reference for annotation and linkage of files. In order to support interoperability across sytems the identifier must be unique not only in the context of a single system, but on a global level. URIs [BLFM05] provide a powerful generic mechanism to globally identify resources, and they are designed to be minted without a central authority. As the generic specification for URIs also defines hierarchical

URIs it is additionally straightforward to convert directory and file names to URIs.

1.2 Level of Abstraction

One major strength of file systems is their high level of abstraction: files can be used to store any kind of data and allow for arbitrary formats. It is important for a semantic file system that users and applications are not forced to fit their data into heavily constrained structures. On the other hand, semantics can only be derived from structure, therefore any semantic storage must impose a certain level of rigidness. From observing a variety of applications and systems both in the desktop and the Web domain, we can infer a need for the following structure elements:

- *Concrete and abstract resources.* Current file systems are based on the assumption that every thing has a digital representation (although files can have a content with zero length). This assumption is valid in file system-based environments since the representation of non-digital objects would make sense only if they can be further described, or if they can be brought into relationship with other (digital) objects—both features are, however, not offered by current hierarchical file systems. To facilitate the expression of information about non-digital real-world objects a personal information management system should also be able to represent e.g., persons, events, locations, etc. A semantic file system that supports digital and non-digital objects as well as relationships between them could significantly increase the potential expressivity of desktop applications. Additionally, sharing of data between applications could be improved if common classes of user data (e.g., contacts or appointments) were represented in an application-independent manner.

- *Ontological knowledge.* Semantic annotations, or annotations based on description logic, help to improve the automated processing and retrieval of information. Classes (or *concepts*) are regarded the basic structure element in many knowledge organization systems and ontology languages. To apply them to file systems, and to derive conclusions about files with similar properties, a semantic file system should be able to represent files as class instances and allow to perform reasoning over these data structures. Moreover, ontology classes could be represented as files themselves and could therefore be directly managed by the user, if this functionality is desired in a concrete application scenario.

- *Attributes.* Attributes can be used to describe files in structured form. They allow applications to store information that is not inherent to a file's content. If the meaning of a file's attribute is defined in an ontology it can be used consistently across different applications. However, it would also be possible to attach user-defined attributes in an ad-hoc style; in this case the attributes cannot be automatically processed but still may help the user to organize and retrieve information.

- *Relationships.* Many types of information cannot be attributed to a single file, but are represented by relationships between them; a fact also observed by end users [RSK04]. The World Wide Web is the best example of a knowledge base that gains its power mainly from links between web resources. The way that files are related to each other can be described in more detail by typed relations or by attaching additional information (attributes, other relationships) to them. Therefore, a semantic file system should support (annotated) relationships.

- *Tags.* Tags have become enormously popular through their use in many so-called *Web 2.0* applications. Tags expose a different level of semantic expressivity than classes and attributes, since the relationship between an object and its associated tags, as well as the relationships between tags, are not formally defined. Nevertheless, the popularity of tags shows that they may support the organization of information in a user-centric way and should therefore be supported by a file system targeted towards a semantic desktop.

In addition to the possibility to store and manage these organizational mechanisms, a potential increase of semantic expressivity could be gained from allowing one to define *mappings* between them. Such a mapping could provide additional information hints to the user when searching, and ease the manual annotation process, which often is cumbersome. For instance, one could map a certain attribute value to a tag, and further only use the tag instead of the attribute's name/value pair. This would release users from the need to remember the exact name and value for this attribute and allow for a more intuitive, human-centric application of formal semantics.

1.3 Compatibility with File Systems

Although the amount and complexity of data on a user's desktop steadily increases, the organizational metaphors for files have remained unchanged for decades. Since hierarchical file systems are not sufficiently expressive to represent machine-processable annotations, application-dependent parallel structures have been established. For instance, many tools for management of multimedia data (e.g., audio files or photos) extend the file system with application-specific semantics and thus face the same problems as hybrid semantic desktop approaches. These systems often use a mix of the hierarchical file system (e.g., MP3 files in directories named after artists), metadata embedded into files (e.g., ID3 tags), and data in application-specific databases (e.g., e-mail archive files). Naturally, the latter remain hidden to the outside. For user data stored in hierarchical file systems it is important that there exist smooth migration paths that allow users to "glide" into the semantic desktop. When transferring a hierarchical directory structure into a semantically enriched organization paradigm, the following requirements must be considered:

- *Organizational preservation.* No information that was present in the hierarchical system should be lost during the transition. This includes information that is implicitly present in file names and directories.

- *Navigational preservation.* A hierarchical file system provides certain navigation alternatives, and users are presumably familiar with them. For semantic files it must be ensured that similar navigational behavior is still supported. The main goal is that users can actually find objects in locations where they expect them to be.

- *Backwards compatibility.* Although a certain level of backwards compatibility can be achieved by inverting mapping algorithms that ensure organizational preservation, not all information contained in a semantic file system can be translated to the tree model of the hierarchical file system because a tree is not sufficiently expressive to represent certain aspects, (e.g., relationships between files). Maximized backwards compatibility can increase the acceptance of semantic systems on the desktop, especially when users are using legacy software that is not aware of the file system's semantic features. Thus, it is reasonable to invest research effort into the question of representing semantic networks by the means of hierarchical file systems.

1.4 Compatibility with the Web

A large amount of relevant information is available on the World Wide Web and, increasingly, the Semantic Web. Currently we can observe an *information gap* between data stored on a user's local machine on the one hand, and online resources on the other hand. An information architecture that uses a unified mechanism to identify information objects and makes them accessible regardless of their physical location could help to bridge this gap. This bridging could take place in two directions; first, it should allow users to semantically interconnect (and, consequently, retrieve and use) local and web resources, and second, it should be possible to selectively share local resources if needed without much effort, and thus make local information resources a part of the web.

1.5 Why not Plain RDF?

The RDF data model [KC04] is sufficiently generic to represent different organizational elements (including the ones described in Section 1.2), and many of them are already used in web applications. Many tools and libraries for manipulating RDF are available, and the performance of triple stores is steadily increasing (cf., e.g., [AMMH07]), hence RDF seems to be a natural choice for the representation of metadata in file systems.

RDF does not restrict the usage of relationship types; instead, ontology languages are used to do this. This freedom fosters the publication of data on the World Wide Web (as demonstrated, e.g., by the Linked Data initiative[1]) but, on the other hand, imposes restrictions on the applicability of RDF as information exchange format: when different applications access a shared data set they must agree on a common vocabulary, or define potentially complex schema mappings.

Moreover, the RDF data model does not contain the concept of *self-contained information units*. Instead, RDF data can be seen as a continuous stream of relationships between arbitrary resources. This paradigm is fundamentally different from what we can observe in file systems, where information is stored in discrete units, and is also different to the object-oriented modeling paradigm, in which a large share of applications are implemented, and with which many developers are familiar. We also suspect that the flat graph model stands—to a certain extent—in contradiction to the way humans perceive the world:

[1]`http://linkeddata.org`

commonly, objects and their properties are mentally aggregated and treated as integral units. Thus we envision a data model that is as generic and interoperable as RDF but simplifies the representation of object-oriented model elements.

1.6 Conclusions and Design Decisions

The issues discussed in the previous sections have lead us to the specification of a data model for a semantic file system that incorporates aspects from Semantic Web technology (usage of URIs, graph-based metadata structures, usage of ontology elements like classes and properties), object-oriented modeling (integrated view on objects and their properties), Web 2.0 (tags), and file systems (discrete content units).

As an atomic information element we choose to extend the file with semantic annotations of different kinds, and call such objects *siles*. A sile is a discrete unit of information; the content of a sile is—similar to a file—self-contained and does neither depend on any other entity, nor does it make any statements on other entities. To guarantee a unique identity for these discrete information units, we choose to identify a sile by a URI. URIs can take the form of simple names that carry no further semantics (like, for example, a UUID), or can imply mechanisms and protocols for accessing the resource that is identified by a URI (in the case of, e.g., `http:` or `mailto:` URIs). For the sile model we do not establish any constraints on the format of URIs or require a specific URI scheme. Since from our perspective URIs are used as opaque identifiers, we can safely leave the choice of a suitable URI scheme and URI minting algorithm to the concrete implementation.

This design implies that, in contrast to file systems, no *intrinsic semantics* is imposed on sile identifiers. In file systems, the full path of a file is composed of a series of directory names, each of which is chosen by a user and carries implicit or explicit meaning. We unhinge all kinds of contextual meaning from the sile identifier (the URI); instead we define a number of *annotation metaphors* that can be used to express the *extrinsic semantics* of a sile. We have chosen the following set of annotation metaphors to be included in the sile model:

- *Tags.* Tags are simple keywords that consist of a string of arbitrary length and arbitrary format. Tags are usually not chosen from a predefined formal vocabulary, hence the interpretation of a tag is entirely left to the end user. Certain problems arise in systems that use tags: for instance, it is not possible to resolve homonym or synonym conflicts without further analysis. Moreover, tags are always bound to a certain natural language which makes them unfeasible for end users not aware of this language. Nevertheless tags can help to classify and retrieve information in situations where end users follow a common understanding of tags. This is both the case when tags are applied in a single person's data space—we can assume that the person who issues a tag will be able to understand the meaning of this very tag later on—, or in cases where a user group shares a certain, informal vocabulary and has, to a certain extent, a common understanding of its meaning; as it is the case, for instance, in work groups or projects.

- *Categories.* The use of ontologies significantly extends the analysis possibilities that can be applied to documents in general and tags in special (e.g., full text search, string similarity metrics, natural language processing, and statistical methods). Ontologies establish frameworks of formal rules that can be used by reasoners to validate descriptions and to generate new, *implicit* knowledge out of existing information. *Classes* are a core concept in most ontology languages; however we can observe that non-expert users have difficulties in understanding the idea of classes and in perceiving the potentially high complexity of ontology instances. We suspect that one part of this problem is caused by naming: we can observe that the term "category" is often better understood by non-expert users than "class". For our data model we introduce categories as a way to annotate siles but we do not define rules on how categories are to be interpreted by a machine, or which reasoning rules can be applied to them.

- *Attributes.* Attributes are a basic modeling element in many information systems. An attribute describes a specific characteristic of an individual and usually consists of two parts: a *name* part that indicates which characteristic is described by an attribute, and a *value* part that represents the concrete occurrence of this characteristic with respect to the individual. Attributes are a highly generic mechanism for expressing information about individuals and can additionally be used in ontologies to indicate, for instance, class membership or to verify instance equality. As with categories, we do not impose such rules in our data model but leave the definition of such rules up to concrete implementations and applications.

- *Relationships.* Relationships can be considered as attributes whose value is an instance of similar characteristic as the described object. In our model we consider siles as first-class objects, and thus relationships are the subset of a sile's attributes whose value is a sile. By introducing relationships as a separate class of annotations, we emphasize the web-style character of our data model: using relationships, interdependencies between discrete information units are made explicit.

We do not predefine types or processing rules for categories, attributes, or relationships, but leave this to the concrete applications that make use of siles; an example of such such an application is described in [Sch06]. However we want to give our system the possibility to express interdependencies between different types of sile annotations, because we see the need of information management systems to maintain a certain level of data integrity. There exist a magnitude of mechanisms to express such information (e.g., ontologies, schema descriptions, or abstract modelling languages like UML). Similar to our vision of siles as generalized view on personal information, we use the term *spect*[2] to denote collections of interdependency rules between categories, attributes, and relationships.

We deliberately choose not to include a representation of any kind of *hierarchies* (like file system directories) in the model, since we are interested in the applicability and impact of the metaphors mentioned above *in the absence* of directories. Additionally, hierarchical structures can be simulated using relationships between objects, as shown e.g., in [SH09]. We are aware of the fact

[2]The name *"spect"* is derived from the term "spectacles", i.e., a means to provide the user with a clearer view on things.

that this decision is contradictory to the requirement of compatibility with hierarchical file systems (cf. Section 1.3), thus hierarchical collections may be added to the data model at a later point in time.

By making information semantics extrinsic and representing them in a unified manner, a storage system for semantically annotated objects can act as a shared information infrastructure for different applications and services. Usually, applications will operate on a limited set of objects; e.g., ones that are available on a user's local machine, or ones that are stored on the working group's file server. In the following we use the term *repository* to denote a logically closed context of information interpretation. Such a context consists of a set of siles, including their associated content and annotations. The interpretation of the siles' annotations is only valid within the context of a given repository; a different repository may make completely different statements about siles and apply different rules on how annotations may be combined. The connections between such distinct repositories are established using *referenced siles*, i.e., pointers that refer to siles within an external repository. The identity of (and therefore, the connection to) a sile is established by a unique opaque identifier, the sile URI.

In terms of a physical unit, a repository can be regarded similar to the concept of a volume in a file system, or the concept of a host on the World Wide Web. In terms of a logical unit, it can be regarded as a collection of siles that share a logical context, e.g., the siles that have been created by a specific user, or the siles that are of relevance in the context of a research project. We do not define constraints on the inner structure of a repository; instead we define a set of operations that a repository must be able to execute and treat it in other respects as a black box.

2 Data Model

In the previous section we have informally introduced the concepts of a sile, its characteristics, and the various annotation classes that can be applied to siles. In the following we give a formal notion of siles and annotations by defining them in terms of sets. We start this by introducing a number of symbols that we use throughout this section.

Definition 1 [Symbols]

Let Σ denote the set of all siles in the university of discourse. Let $\mathbb{LIT}$ denote the set of all *string literals* which are finite sequences of characters from an *literal alphabet* α, and $\mathbb{B}$ the set of all *content literals* which are finite sequences of characters from an *content alphabet* β. Further, let $\mathbb{URI}$ denote the set of all *Uniform Resource Identifiers (URIs)*[3]. Let $\mathbb{T}$ denote the set of all *tags*, $\mathbb{T} \subseteq \mathbb{LIT}$, Let $\mathbb{C}$ denote the set of all *categories*, $\mathbb{C} \subseteq \mathbb{URI}$, and let $\mathbb{A}$ denote the set of all *attributes*, $\mathbb{A} = \mathbb{URI} \times \mathbb{LIT} \times \mathbb{URI}$. and let $\mathbb{L}$ denote the set of all *slinks*, $\mathbb{L} = \mathbb{URI} \times \mathbb{URI}$. Let $\mathbb{ANN}$ denote the set of *annotations*, $\mathbb{ANN} = \mathbb{T} \cup \mathbb{A} \cup \mathbb{C} \cup \mathbb{L}$, and let $\mathbb{SP}$ denote the set of all *spects*, as described in Section 1.6. Finally, let $\mathbb{ENT} = \Sigma \cup \mathbb{ANN} \cup \mathbb{SP}$ denote the set of all *entities*. Using this vocabulary, we can describe the data structures, constraints, and operations that constitute our data model.

[3]URIs are treated as opaque identifiers and should be formatted according to [BLFM05].

Definition 2 [Siles]

A sile $s \in \Sigma$ is a six-tuple $s = (u_s, b_s, T_s, C_s, A_s, L_s)$. $u_s \in \mathbb{URI}$ denotes the *URI* (Uniform Resource Identifier) of sile s, $b_s \in \mathbb{B} \cup \bot$ denotes the sile's *binary content*, $T_s \subseteq \mathbb{T}$ is the set of the sile's associated *tags*, $C_s \subseteq \mathbb{C}$ is the set of the sile's associated *categories*, $A_s \subseteq \mathbb{A}$ the set of the sile's associated *attributes*, and $L_s \subseteq \mathbb{L}$ is the set of the sile's associated *slinks*. The set $ANN_s = T_s \cup A_s \cup C_s \cup L_s$ subsumes all annotations that are associated to s.

A sile $s \in \Sigma$ is uniquely identified by its URI u_s, hence the sile's URI u_s is a functional property of the sile:

$$\forall s_i, s_j \in \Sigma : u_{s_i} = u_{s_j} \longleftrightarrow s_i = s_j$$

This equality is the sole criterion that allows one to decide whether two given sile entities actually are *equal*. Especially does the sile model neither state that siles that share the same annotations (as described below) are considered equal, nor that siles are considered equal if they have equal content.

Within the sile model, we do not impose constraints on the structure or the nature of a sile's binary content. Especially we do not define rules that state how the content is to be interpreted, or how one can deduce annotations from analyzing the content. Sile content may also be of arbitrary length, including zero.

Two examples of siles, an email message and a file, are depicted in Figure 1.

Definition 3 [Tags]

As annotations we denote the organizational metaphors of the sile data model that describe siles and bring them into context. As described above, annotations can be of four types: *tags*, *categories*, *attributes*, and *slinks*.

A tag is described and identified by a string literal and carries no further machine-processable semantics. Hence a tag associates a sile with a plain literal string, and it is sufficient to describe a tag $t \in \mathbb{T}$ by such a simple literal: $\mathbb{T} \subseteq \mathbb{LIT}$.

We consider two tags t_a and t_b as equal if their plain literal strings are equal, which in turn is the case if *(i)* they are of the same length, $length(t_a) = length(t_b)$, and *(ii)* each character on position i of tag t_a is equal to the character on position i of tag t_b, $char(t_a, i) = char(t_b, i), 1 \leq i \leq length(t_a)$.

Definition 4 [Categories]

As described before, a category annotation is a reference to an abstract concept entity that may carry machine-processable semantics, which in turn may be used to validate a data model or to enrich a set of annotations with implicit (derived) annotations. To ensure uniqueness, a category c is identified by a URI: $c \in \mathbb{C}$, $\mathbb{C} \subset \mathbb{URI}$.

It is obvious that categories are considered as equal if their URIs are equal[4]. We externalize any further details regarding the nature of categories, including semantic relationships to other categories, attributes and slinks, and annotation aspects, like e.g., human-readable labels or comments, from categories; instead, spects (see below) and domain-specific extensions can be employed for this purpose.

[4]According to [BLFM05], URI equality is defined as string equality applied to the URIs' absolute forms

Example. A number of email messages on a personal desktop computer can be regarded as siles. By default, email messages carry unique *message-id*s which constitute the respective sile URIs. The bodies of the messages can be regarded as sile contents; the mail folder where the message is stored and its read/unread status can be attached to the message as tag. Metadata about the message (like the subject, sender and recipient, and the date of delivery) can be represented as attributes and slinks.
Such mail messages may be represented as follows[a]:

$$
\begin{aligned}
s_1 = (\ & \texttt{msg:0BBF7468-7C34-4587-97E0-D8DB9E8CBDD9@univie.ac.at},\\
& \texttt{"Sehr geehrte Damen und Herren, [...]"},\\
& \{\texttt{"INBOX"}, \texttt{"unread"}\},\\
& \{\texttt{nmo:Email}\},\\
& \{(\texttt{nmo:subject}, \texttt{"LV-Evaluierung SS 2009"}, \texttt{xsd:string}),\\
& (\texttt{nmo:receivedDate}, \texttt{"2009-04-05T16:54"}, \texttt{xsd:dateTime})\},\\
& \{(\texttt{nmo:from}, \texttt{mailto:wolfgang.klas@univie.ac.at}),\\
& (\texttt{nmo:to}, \texttt{mailto:bernhard.schandl@univie.ac.at})\} \quad)
\end{aligned}
$$

Similarly, a file in a file system may be represented using the sile model, whereas its URI is constructed using its absolute path[b]. In this example, no tags or slinks are attached, and the set of attributes is restricted to the file name and its creation date:

$$
\begin{aligned}
s_2 = (\ & \texttt{urn:uuid:c6b18466-eab4-4943-8699-62eb6dc229d9},\\
& \texttt{"\textbackslash documentclass\{report\} [...]"},\\
& \{\},\\
& \{\texttt{nfo:FileDataObject}\},\\
& \{(\texttt{nfo:fileName}, \texttt{"file:///Users/bs/work/phd/phd.tex"}, \texttt{xsd:string}),\\
& (\texttt{nfo:fileCreated}, \texttt{"2008-11-07T12:41"}, \texttt{xsd:dateTime})\},\\
& \{\} \quad)
\end{aligned}
$$

[a]For the examples we use prefixed QName notation ([BHLT06], Section 4) for URIs; annotation URIs refer to the NEPOMUK Semantic Desktop ontologies [MSSvE07].

[b]Note that we consider the usage of mutable file paths as URIs (which are meant to be immutable) as harmful practice [SH09].

Figure 1: Representation of mail messages and files using the sile model

Definition 5 [Attributes]
In contrast to tags and categories, attributes are tuples of three elements: the *attribute name* identifies the characteristic that is described by the attribute. The *attribute value* represents the actual manifestation of the attribute with respect to the sile, and the *attribute data type* identifies a rule set that describes the lexical value space (i.e., the set of literal strings that are valid for this attribute) as well as the intended interpretation of the value literal.

We use URIs to describe the attribute name and the attribute data type; hence the set of all possible attributes is $\mathbb{A} = \mathbb{URI} \times \mathbb{LIT} \times \mathbb{URI}$, and an attribute a can be written as 3-tuple: $a = (an_a, av_a, at_a)$, with $an_a \in \mathbb{URI}$, $av_a \in \mathbb{LIT}$, and $at_a \in \mathbb{URI}$.

Again, we do not associate to attributes formal rules regarding the interpretation and restriction of attribute names or attribute values and data types. For the former we outsource this description to spects and application-specific extensions of the data model; for the latter we follow the RDF semantics regarding data types which are defined in Section 5 of [Hay04].

Definition 6 [Slinks]
A slink $l \in \mathbb{L}$ relates a sile to another sile, whereas the nature of the slink is identified by the *slink name*. A slink can be regarded as a directed labeled edge in a graph, where siles form the graph nodes. Each slink leads from a *source sile* to a *target sile*, whereas the slink is attached to the source sile and carries a reference URI to the target sile. We use a URI for the name (the label) of the slink. Therefore, the set of all slinks $\mathbb{L}$ is the cartesian product of the set of all URIs and the set of all siles: $\mathbb{L} = \mathbb{URI} \times \mathbb{URI}$, and a slink l can be written as $l = (ln_l, ld_l)$, with $ln_l, ld_l \in \mathbb{URI}$.

As mentioned in the previous section, slinks and attributes share some common properties. Slinks can be seen as a special case of attributes whose value is a URI that references a sile. For this reason consider slinks first-class annotations, since they help to construct a *web* of siles and allow—in a metaphorical sense—to interconnect the otherwise separate data units.

Definition 7 [Spects]
Spects are used to define *applicability rules* for annotations. Applicability rules restrict the set of possible relationships between siles and annotations, and define when a given constellation can be regarded as consistent. Such consistency rules can be employed by a repository implementation *(1)* to ensure the internal consistency of its data model, *(2)* to infer new (implicit) annotations from existing ones, and by a client application in order *(3)* to restrict the set of possible annotation opportunities presented to the end user.

A spect $sp_i \in \mathbb{SP}$ defines four classes of applicability rules; *Category Hierarchy Rules*, *Attribute Applicability Rules*, *Slink Domain Applicability Rules*, and *Slink Range Applicability Rules.*

Category Hierarchy Rules define subsumption rules between categories; in this sense, categories can be interpreted as classes from set theory. A category hierarchy rule $chr \in sp_i$ defines a relationship between two categories; thus it can be written as two-tuple $chr \in \mathbb{C} \times \mathbb{C}$. The semantic interpretation of a category hierarchy rules is as follows: if a sile s is annotated with category c_1, and the category hierarchy rule $chr_i = \{c_1, c_2\}$ exists in any known spect sp_i, sile s is also annotated with category c_2:

$$\forall s \in \Sigma, sp_i \in \mathbb{SP} : c_1 \in C_s \wedge \{c_1, c_2\} \in sp_i \longrightarrow c_2 \in C_s$$

Category hierarchy rules are *transitive*:

$$\forall sp_i \in \mathbb{SP} : \{c_1, c_2\}, \{c_2, c_3\} \in sp_i \longrightarrow \{c_1, c_3\} \in sp_i$$

Attribute Applicability Rules establish formal relationships between categories and attribute names: an Attribute Applicability Rule between an attribute name an and a category c states that a sile that is annotated with attribute an is also annotated with category c:

$$\forall s \in \Sigma, sp_i \in \mathbb{SP} : (an, av, at) \in A_s \wedge \{an, c\} \in sp_i \longrightarrow c \in C_s$$

A *Slink Domain Applicability Rule* between a slink name ln and a category c defines that a sile that is annotated with a slink with name ln it is also annotated with category c:

$$\forall s \in \Sigma, sp_i \in \mathbb{SP} : (ln, ld) \in L_s \wedge \{ln, c\} \in sp_i \longrightarrow c \in C_s$$

A *Slink Range Applicability Rule* between a slink name ln and a category c defines that a sile that is the destination of a slink with name ln is also annotated with category c:

$$\forall s, \overline{s} \in \Sigma, sp_i \in \mathbb{SP} : (ln, s) \in L_{\overline{s}} \wedge \{ln, c\} \in sp_i \longrightarrow c \in C_s$$

Examples for all classes of rules in the context of email messages are given in Figure 2.

Definition 8 [Repositories]
As described in Section 1.6, a repository is a closed context of interpretation, within which a set of entities (siles, annotations, spects) are considered to be valid. A repository indicates thus a set of entities that are considered as a separate, self-contained, and logically consistent knowledge corpus.

A repository $R \in \rho$ (where ρ denotes the set of all repositories) can be written as a 7-tuple $R = (\Sigma_{Rh}, \Sigma_{Rr}, \mathbb{T}_R, \mathbb{C}_R, \mathbb{A}_R, \mathbb{L}_R, \mathbb{SP}_R)$. It consists of a set of hosted siles Σ_{Rh} and a set of referenced siles Σ_{Rr}, which together form the set of the repository's known siles $\Sigma_R = \Sigma_{Rh} \cup \Sigma_{Rr}$. It further consists of sets of tags ($\mathbb{T}_R \subseteq \mathbb{T}$), categories ($\mathbb{C}_R \subseteq \mathbb{C}$), attributes ($\mathbb{A}_R \subseteq \mathbb{A}$), and slinks ($\mathbb{L}_R \subseteq \mathbb{L}$). Additionally, it consists of a set of spects $\mathbb{SP}_R \subseteq \mathbb{SP}$ that define the applicability rules that this repository applies to annotations, as described before.

A sile may be *hosted* in a given repository, or it may be *referenced* which means that it is interpreted as a pointer to a sile that is hosted in a different repository. We distinguish these two classes of siles based on the presence or absence of content: in the case of hosted siles, a content is present (although it may be of zero length); in the case of referenced siles, no content is present. Instead, the content may be retrieved by dereferencing the sile URI[5].

[5]The procedure of dereferencing URIs is out of the scope of the abstract sile model, and depends on the format of the sile URI: for `http` URIs, dereferencing means to establish a connection to a remote HTTP server and to retrieve the content using e.g., a HTTP `GET` call. `imap` URIs may be dereferenced by establishing a connection to the respective IMAP server, and so forth. More details on the process of dereferencing in the context of the World Wide Web are given in [JW05], Section 3.1.

Example. A spect SP_{mail} describing the annotation vocabulary for e-mail communication may define the following rules. It defines a number of category hierarchy rules, amongst them the relationship between abstracts messages, email messages, and attachments:

$$\begin{aligned} CHR_{mail} &= \{(\texttt{nmo:Email}, \texttt{nmo:Message}), \\ &\quad (\texttt{nmo:Message}, \texttt{nie:InformationElement}), \\ &\quad (\texttt{nfo:Attachment}, \texttt{nie:InformationElement}), \ldots\} \end{aligned}$$

Using the following attribute applicability rules, attribute names are related to category names:

$$\begin{aligned} AAR_{mail} &= \{(\texttt{nmo:sentDate}, \texttt{nmo:Message}), \\ &\quad (\texttt{nmo:receivedDate}, \texttt{nmo:Message}), \\ &\quad (\texttt{nmo:messageSubject}, \texttt{nmo:Message}), \ldots\} \end{aligned}$$

Slink domain and range applicability rules define the relationship between category names and slink names:

$$\begin{aligned} SDAR_{mail} &= \{(\texttt{nmo:from}, \texttt{nmo:Message}), \\ &\quad (\texttt{nmo:to}, \texttt{nmo:Message}), \\ &\quad (\texttt{nmo:hasAttachment}, \texttt{nmo:Message}), \\ &\quad (\texttt{nmo:cc}, \texttt{nmo:Email}), \ldots\} \end{aligned}$$

$$\begin{aligned} SRAR_{mail} &= \{(\texttt{nmo:from}, \texttt{nco:Contact}), \\ &\quad (\texttt{nmo:to}, \texttt{nco:Contact}), \\ &\quad (\texttt{nmo:hasAttachment}, \texttt{nfo:Attachment}), \\ &\quad (\texttt{nmo:cc}, \texttt{nco:Contact}), \ldots\} \end{aligned}$$

The complete spect consists of all the following definitions:

$$SP_{mail} = CHR_{mail} \cup AAR_{mail} \cup SDAR_{mail} \cup SRAR_{mail}$$

Figure 2: A spect defining rules for the relationship between email categories and attributes

Thus, the set of all siles that exist within a given repository R (denoted by Σ_R, $\Sigma_R \in \Sigma$) can be separated into two subsets, the set of all siles that are hosted by this repository Σ_{Rh} and the set of all siles that represent references to siles hosted in other repositories Σ_{Rr}. Thus, the following rules hold for all siles stored within a repository:

$$\forall s \in \Sigma_R : b_s \neq \perp \longleftrightarrow s \in \Sigma_{Rh}, s \notin \Sigma_{Rr}$$

$$\forall s \in \Sigma_R : b_s = \perp \longleftrightarrow s \in \Sigma_{Rr}, s \notin \Sigma_{Rh}$$

The sets of hosted and referenced siles are disjunct,

$$\forall R \in \rho : \Sigma_{Rh} \cap \Sigma_{Rr} = \emptyset$$

and no siles other than hosted or referenced ones exist in a repository:

$$\forall R \in \rho : \Sigma_R \setminus (\Sigma_{Rh} \cup \Sigma_{Rr}) = \emptyset$$

We interpret the term "repository" in a broad manner: every system whose information can be represented within the sile model can be referred to as repository. Since siles are identified by URIs, every piece of information that can be identified by an URI can potentially become a sile, and its physical location can be regarded as sile repository.

The design of the sile model and the concept of repositories provide the possibility to make assertions about the same information on different places: while repository R_1 may hold the actual content of a sile, repository R_2 may hold annotations that have been extracted by analyzing the sile content, and repository R_3 may store user-defined annotations (e.g., tags) for the sile. While these data may share no common semantics, and their repositories may apply different storage technologies, these different pieces of information are still connected through the unique identifier of the sile, its URI.

3 A Query Framework for Siles

The definitions given in the previous section describe the *static data model* for siles. This model represents a framework that structures siles, their contents, and their annotations for further processing. In the following we describe generic abstract operators that use the elements from the sile data model. These operators are designed to be simple to understand and use, but can be combined and nested in order to formulate complex operations and queries.

In the following, we define three operator types:

- *Entity Extraction Operators* extract information parts out of entities; i.e., they provide access to the parts of entity tuples;
- *Entity Existence Operators* indicate whether a specific information entity exists; i.e., whether a tuple (or a combination of tuples) exists that represents a specific information constellation; and
- *Sile Selection Operators* select, from a set of siles, a subset that fulfils certain criteria.

The combination of these operators allows us to model complex expressions over the sile data model that can be used to retrieve information, decide whether siles fulfil certain required information constellations, and restrict sets of siles based on these decisions.

3.1 Prerequisites

Definition 9 [Basic Definitions]
Let $\mathbb{BOOL} = \{true, false\}$ denote the Boolean set, and let $\mathcal{P}(A) = \{x \mid x \subseteq A\}$ denote the powerset (i.e., the set of all subsets) of A.

3.2 Entity Extraction Operators

Entity extraction operators extract specific information from an entity. As each entity is described by several different characteristics, we need these extraction operators to be able to process these individual items.

Definition 10 [Sile Extraction Operators]
As described before, a sile s can be written as 6-tuple $s = (u_s, b_s, T_s, A_s, C_s, L_s)$. We define the following operators that extract the individual parts of these tuples as follows:

$$uri : \Sigma \mapsto \mathbb{URI}, uri(s) = u_s$$

$$content : \Sigma \mapsto \mathbb{B}, content(s) = b_s$$

$$tags : \Sigma \mapsto \mathcal{P}(\mathbb{T}), tags(s) = T_s$$

$$attributes : \Sigma \mapsto \mathcal{P}(\mathbb{A}), attributes(s) = A_s$$

$$categories : \Sigma \mapsto \mathcal{P}(\mathbb{C}), categories(s) = C_s$$

$$slinks : \Sigma \mapsto \mathcal{P}(\mathbb{L}), slinks(s) = L_s$$

We additionally introduce one extraction operator that returns all sile annotations, regardless of which type they are:

$$annotations : \Sigma \mapsto \mathcal{P}(\mathbb{ANN}), annotations(s) = T_s \cup A_s \cup C_s \cup L_s$$

Definition 11 [Annotation Extraction Operators]
For the atomic annotation types (tags and categories) we define two auxiliary operators that return the annotation's identifying characteristic (i.e., the tag label or the category URI, respectively). Since tags and categories only consist of one element, the definition of these extraction operators is straightforward:

$$text : \mathbb{T} \mapsto \mathbb{LIT}, text(t) = t$$

$$catname : \mathbb{C} \mapsto \mathbb{URI}, catname(c) = c$$

For the annotation types that are not atomic (attributes and slinks) we need operators to extract their parts. For attributes we define the following extraction operators:

$$attname : \mathbb{A} \mapsto \mathbb{URI}, attname(a) = an_a$$

$$attvalue : \mathbb{A} \mapsto \mathbb{LIT}, attvalue(a) = av_a$$

$$atttype : \mathbb{A} \mapsto \mathbb{URI}, atttype(a) = at_a$$

Similarily, for slinks we define:

$$slinkname : \mathbb{L} \mapsto \mathbb{URI}, slinkname(l) = ln_l$$

$$slinkdst : \mathbb{L} \mapsto \mathbb{URI}, slinkdst(l) = ld_l$$

Finally, we define a generic URI extraction operator $uri : \mathbb{ANN} \mapsto \mathbb{URI} \cup \{\bot\}$ that can be applied to all types of annotations:

$$uri(e) = \begin{cases} catname(e) & \text{if } e \in \mathbb{C} \\ attname(e) & \text{if } e \in \mathbb{A} \\ slinkname(e) & \text{if } e \in \mathbb{L} \\ \bot & \text{otherwise} \end{cases}$$

We can see that the *uri* operator returns a URI that can be used to identify the nature of the annotation for all types of annotations except tags. The result of this operator can, in general, *not* be used to compare annotations for equality, since for this all characteristics of an annotation must be considered.

To accomplish this, we introduce the annotation generic comparison operator $equals : \mathbb{ANN} \times \mathbb{ANN} \mapsto \mathbb{BOOL}$ that indicates whether two annotations are equal:

$$equals(e_1, e_2) := \begin{cases} true & \text{if } e_1, e_2 \in \mathbb{T} \wedge text(e_1) = text(e_2) \\ & \text{or } e_1, e_2 \in \mathbb{C} \wedge catname(e_1) = catname(e_2) \\ & \text{or } e_1, e_2 \in \mathbb{A} \wedge attname(e_1) = attname(e_2) \wedge \\ & \quad attvalue(e_1) = attvalue(e_2) \wedge atttype(e_1) = atttype(e_2) \\ & \text{or } e_1, e_2 \in \mathbb{L} \wedge slinkname(e_1) = slinkname(e_2) \wedge \\ & \quad slinkdst(e_1) = slinkdst(e_2) \\ false & \text{otherwise} \end{cases}$$

We also introduce a generic comparison operator $equalsIgnore : \mathbb{ANN} \times \mathbb{ANN} \mapsto \mathbb{BOOL}$ that compares annotations without considering certain components, i.e., the value and data type in the case of attributes, and the destination sile in the case of slinks. For tags and categories, *equalsIgnore* returns the same result as *equals*:

$$equalsIgnore(e_1, e_2) = \begin{cases} equals(e_1, e_2) & \text{if } e_1, e_2 \in \mathbb{T} \cup \mathbb{C} \\ true & \text{if } e_1, e_2 \in \mathbb{A} \wedge attname(e_1) = attname(e_2) \\ & \text{or } e_1, e_2 \in \mathbb{L} \wedge slinkname(e_1) = slinkname(e_2) \\ false & \text{otherwise} \end{cases}$$

3.3 Entity Existence Predicates

In comparison to the entity extraction operators which return specific parts of entities, i.e., siles or annotations, in the following we discuss *existence predicates.* These predicates indicate whether a specific data constellation is given in the context of interpretation, and correspondingly return a Boolean value (*true* or *false*). The context of interpretation Γ depends on the application: it may be a single repository R ($\Gamma = R$) or an arbitrary number of repositories $R_1, R_2, \dots, R_n$, in which case the operators consider the union of all their annotations ($\Gamma = \bigcup_{i=1}^{n} R_i$). The context of interpretation can then be written as 7-tuple that subsumes all elements of the considered repositories R_i, $i = 1 \dots n$:

$$\begin{aligned} \Gamma &= (\Sigma_{\Gamma h}, \Sigma_{\Gamma r}, \mathbb{T}_\Gamma, \mathbb{C}_\Gamma, \mathbb{A}_\Gamma, \mathbb{L}_\Gamma, \mathbb{SP}_\Gamma) \\ &= (\bigcup_{i=1}^{n} \Sigma_{R_i h}, \bigcup_{i=1}^{n} \Sigma_{R_i r}, \bigcup_{i=1}^{n} \mathbb{T}_{R_i}, \bigcup_{i=1}^{n} \mathbb{C}_{R_i}, \bigcup_{i=1}^{n} \mathbb{A}_{R_i}, \bigcup_{i=1}^{n} \mathbb{L}_{R_i}, \bigcup_{i=1}^{n} \mathbb{SP}_{R_i}) \end{aligned}$$

In the following, a $\cdot_\Gamma$ index indicates that all operators are defined with respect to a given context of interpretation Γ.

Definition 12 [Sile Existence Predicate]
We start with the very basic definition of a sile existence predicate, $exists_\Gamma : \Sigma \mapsto \mathbb{BOOL}$ which returns whether a given sile exists in the context of interpretation, either in the form of a hosted or a referenced sile:

$$exists_\Gamma(s) = \begin{cases} true & \text{if } \exists R, R \in \Gamma \mid s \in \Sigma_R \\ false & \text{otherwise} \end{cases}$$

Definition 13 [Annotation Existence Predicates]
We can now define predicates that indicate whether a specific combination of entities (i.e., siles and annotations) is present in the context of interpretation. As the most generic predicate, $hasAnnotation_\Gamma : \Sigma \times \mathbb{ANN} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with a specific annotation by using the *annotations* operator:

$$hasAnnotation_\Gamma(s, a) = \begin{cases} true & \text{if } exists_\Gamma(s) \wedge a \in annotations(s) \\ false & \text{otherwise} \end{cases}$$

Also we define a predicate $existsAnnotation_\Gamma : \mathbb{ANN} \mapsto \mathbb{BOOL}$ that indicates whether there exists any sile in the context of interpretation that is associated with a specific annotation:

$$existsAnnotation_\Gamma(a) = \begin{cases} true & \text{if } \exists s \mid exists_\Gamma(s) \wedge hasAnnotation_\Gamma(s, a) = true \\ false & \text{otherwise} \end{cases}$$

Additionally, we can define such annotation existence predicates for specific types of annotations. As tags and categories are atomic annotations, there is no need for type-specific definitions; instead we can directly reuse the already defined $hasAnnotation_\Gamma$ operator to define $hasTag_\Gamma$ and $hasCategory_\Gamma$,

$$hasTag_\Gamma : \Sigma \times \mathbb{T} \mapsto \mathbb{BOOL}, hasTag_\Gamma(s, t) = hasAnnotation_\Gamma(s, t)$$

$$hasCategory_\Gamma : \Sigma \times \mathbb{C} \mapsto \mathbb{BOOL}, hasCategory_\Gamma(s, c) = hasAnnotation_\Gamma(s, c)$$

as well as $existsTag_\Gamma$ and $existsCategory_\Gamma$:

$$existsTag_\Gamma : \mathbb{T} \mapsto \mathbb{BOOL}, existsTag_\Gamma(t) = existsAnnotation_\Gamma(t)$$

$$existsCategory_\Gamma : \mathbb{C} \mapsto \mathbb{BOOL}, existsCategory_\Gamma(c) = existsAnnotation_\Gamma(c)$$

For attributes and slinks, we must go into more detail since it should be possible to query for siles based on each individual part of an annotation. Hence, we define three variants of the $hasAttribute_\Gamma$ predicate:

- $hasAttributeName_\Gamma : \Sigma \times \mathbb{A} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with an attribute that has given name, ignoring the attribute value and the attribute data type:

$$hasAttributeName_\Gamma(s,a) = \begin{cases} true & \text{if } \exists a' \mid exists_\Gamma(s) \wedge a' \in A_s \\ & \wedge\ attname(a) = attname(a') \\ false & \text{otherwise} \end{cases}$$

- $hasAttributeValue_\Gamma : \Sigma \times \mathbb{A} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with an attribute that has a given value and data type, ignoring the attribute name:

$$hasAttributeValue_\Gamma(s,a) = \begin{cases} true & \text{if } \exists a' \mid exists_\Gamma(s) \wedge a' \in A_s \\ & \wedge\ attvalue(a) = attvalue(a') \\ & \wedge\ atttype(a) = atttype(a') \\ false & \text{otherwise} \end{cases}$$

- $hasAttributeNameValue_\Gamma : \Sigma \times \mathbb{A} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with an attribute whose name, value, and data type are equal to the respective elements of the specified attribute. This operator is equal to the $hasAnnotation_\Gamma$ operator when applied to an attribute:

$$hasAttributeNameValue_\Gamma(s,a) = \begin{cases} hasAnnotation_\Gamma(s,a) & \text{if } a \in \mathbb{A} \\ false & \text{otherwise} \end{cases}$$

As an alias, we also define the predicate $hasAttribute_\Gamma : \Sigma \times \mathbb{A} \mapsto \mathbb{BOOL}$ as being equal to $hasAttributeNameValue_\Gamma$:

$$hasAttribute_\Gamma(s,a) = hasAttributeNameValue_\Gamma(s,a)$$

Correspondingly, we can define three variants of the $hasSlink_\Gamma$ predicate that consider the separate parts of slink annotations:

- $hasSlinkName_\Gamma : \Sigma \times \mathbb{L} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with a slink that has a given name, ignoring the destination sile:

$$hasSlinkName_\Gamma(s,l) = \begin{cases} true & \text{if } \exists l' \mid exists_\Gamma(s) \wedge l' \in L_s \wedge slinkname(l) = slinkname(l') \\ false & \text{otherwise} \end{cases}$$

- $hasSlinkDestination_\Gamma : \Sigma \times \mathbb{L} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with a slink with a given destination sile, ignoring the slink name:

$$hasSlinkDestination_\Gamma(s,l) = \begin{cases} true & \text{if } \exists l' \mid exists_\Gamma(s) \wedge l' \in L_s \wedge slinkdst(l) = slinkdst(l') \\ false & \text{otherwise} \end{cases}$$

- $hasSlinkNameDestination_\Gamma : \Sigma \times \mathbb{L} \mapsto \mathbb{BOOL}$ indicates whether a sile is annotated with a slink that has a given name and destination sile:

$$hasSlinkNameDestination_\Gamma(s,l) = \begin{cases} true & \text{if } \exists l' \mid exists_\Gamma(s) \wedge l' \in L_s \\ & \wedge\, slinkname(l) = slinkname(l') \\ & \wedge\, slinkdst(l) = slinkdst(l') \\ false & \text{otherwise} \end{cases}$$

In analogy to $hasAttribute_\Gamma$ we define the predicate $hasSlink_\Gamma : \Sigma \times \mathbb{S} \mapsto \mathbb{BOOL}$ which is an alias for $hasSlinkNameDestination_\Gamma$:

$$hasSlink_\Gamma(s,l) = hasSlinkNameDestination_\Gamma(s,l)$$

Based on the slink existence predicates $hasSlinkName_\Gamma$ and $hasSlink_\Gamma$, we can define two predicates that indicate whether siles are slinked to each other. We can define $areDirectedRelated_\Gamma : \Sigma \times \Sigma \mapsto \mathbb{BOOL}$ that indicates whether a sile s_s is annotated with a slink to another sile s_d, whereas u_{s_d} denotes the URI of sile s_d:

$$areDirectedRelated_\Gamma(s_s, s_d) = \begin{cases} true & \text{if } \exists l \mid exists_\Gamma(s_s) \wedge exists_\Gamma(s_d) \\ & \wedge\, l \in slinks(s_s) \wedge slinkdst(l) = u_{s_d} \\ false & \text{otherwise} \end{cases}$$

In addition to the *directed* variant we also define an *undirected* variant $areRelated_\Gamma : \Sigma \times \Sigma \mapsto \mathbb{BOOL}$ that indicates whether two siles are related, regardless of the direction of the slink:

$$areRelated_\Gamma(s_a, s_b) = \begin{cases} true & \text{if } areDirectedRelated_\Gamma(s_a, s_b) = true \\ & \vee\, areDirectedRelated_\Gamma(s_b, s_a) = true \\ false & \text{otherwise} \end{cases}$$

3.4 Sile Selection Operators

Based on the existence operators, we can define operators that *select* siles based on specific criteria. Selection operators are always applied to a base set of siles and return a subset of this set. This subset contains only siles that fulfill specific criteria.

The definition of the selection operators based on the existence predicate is straightforward; basically it is constituted by wrapping each annotation existence predicate by an operator that returns all siles $s_i \in \Sigma$ for which the respective existence predicate is *true*. Thus we give here only a list of all operators without further explanation.

$TagSiles : \mathcal{P}(\Sigma) \times \mathbb{T} \mapsto \mathcal{P}(\Sigma)$
$TagSiles(S, t) = \{s_i \in S \mid hasTag(s_i, t) = true\}$

$CategorySiles : \mathcal{P}(\Sigma) \times \mathbb{C} \mapsto \mathcal{P}(\Sigma)$
$CategorySiles(S, c) = \{s_i \in S \mid hasCategory(s_i, c) = true\}$

$AttributeNameSiles : \mathcal{P}(\Sigma) \times \mathbb{A} \mapsto \mathcal{P}(\Sigma)$
$AttributeNameSiles(S, a) = \{s_i \in S \mid hasAttributeName(s_i, a) = true\}$

$AttributeValueSiles : \mathcal{P}(\Sigma) \times \mathbb{A} \mapsto \mathcal{P}(\Sigma)$
$AttributeValueSiles(S, a) = \{s_i \in S \mid hasAttributeValue(s_i, a) = true\}$

$AttributeNameValueSiles : \mathcal{P}(\Sigma) \times \mathbb{A} \mapsto \mathcal{P}(\Sigma)$
$AttributeNameValueSiles(S, a) = \{s_i \in S \mid hasAttributeNameValue(s_i, a) = true\}$

$AttributeSiles : \mathcal{P}(\Sigma) \times \mathbb{A} \mapsto \mathcal{P}(\Sigma)$
$AttributeSiles(S, a) = AttributeNameValueSiles(S, a)$

$SlinkNameSiles : \mathcal{P}(\Sigma) \times \mathbb{L} \mapsto \mathcal{P}(\Sigma)$
$SlinkNameSiles(S, l) = \{s_i \in S \mid hasSlinkName(s_i, l) = true\}$

$SlinkDestinationSiles : \mathcal{P}(\Sigma) \times \mathbb{L} \mapsto \mathcal{P}(\Sigma)$
$SlinkDestinationSiles(S, l) = \{s_i \in S \mid hasSlinkDestination(s_i, l) = true\}$

$SlinkSiles : \mathcal{P}(\Sigma) \times \mathbb{L} \mapsto \mathcal{P}(\Sigma)$
$SlinkSiles(S, l) = \{s_i \in S \mid hasSlink(s_i, l) = true\}$

$DirectedRelatedSiles : \mathcal{P}(\Sigma) \times \Sigma \mapsto \mathcal{P}(\Sigma)$
$DirectedRelatedSiles(S, src) = \{s_i \in S \mid areDirectedRelated(src, s_i) = true\}$

$RelatedSiles : \mathcal{P}(\Sigma) \times \Sigma \mapsto \mathcal{P}(\Sigma)$
$RelatedSiles(S, d) = \{s_i \in S \mid areRelated(s_i, d) = true\}$

Each of these operators can be applied to a given base set of siles S and returns a result set of siles. This base set could, for instance, be derived from the given context of interpretation: based on the selection of the base set, either only hosted siles ($S = \Sigma_{\Gamma h}$), referenced siles ($S = \Sigma_{\Gamma r}$), or the full set of siles $S = \Sigma_\Gamma = \Sigma_{\Gamma h} \cup \Sigma_{\Gamma r}$ can be used as base set. Alternatively, the operators can be arbitrarily nested in order to form more expressive queries. The following set of boolean operators can be used to state combinations of selection operators:

$$\begin{aligned} \mathit{and} : \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \mapsto \mathcal{P}(\Sigma) &\qquad \mathit{and}(S_1, S_2) = S_1 \cap S_2 \\ \mathit{or} : \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \mapsto \mathcal{P}(\Sigma) &\qquad \mathit{or}(S_1, S_2) = S_1 \cup S_2 \\ \mathit{not} : \mathcal{P}(\Sigma) \mapsto \mathcal{P}(\Sigma) &\qquad \mathit{not}(S) = \{s_i \mid s_i \in \Sigma \wedge s_i \notin S\} \end{aligned}$$

4 Summary

In this section we have discussed the abstract sile model. Its basic constituents are *siles*, which are units of digital contents, and different types of *annotations* that can be attached to siles: *tags* are plain, unstructured keyword strings; *categories* are formally specified classes which are identified by a unique id; *attributes* are typed name/value pairs, and *slinks* are labelled connections between siles. Furthermore we have defined *spects* that are a lightweight notion for ontological knowledge, and our understanding of a *repository*; i.e., a logically closed unit that hosts a set of siles.

The presented query framework for sile data covers all static elements of the sile data model, which were introduced in the previous chapter. It allows one to formulate expressions that evaluate the state of siles and their annotations, i.e., tags, categories, attributes, and slinks. As such it is appropriate to model information needs that arise in concrete applications, and it is suitable to retrieve sile entities that fulfil certain criteria.

However, the model exposes the following limitations:

1. *Restricted Domain* — The query algebra can be only applied to siles, not to other elements of the sile model. It can not be used to query for annotations; e.g., it is not possible to retrieve a list of all tags, or to query which rules are defined within a spect.

2. *No Joins* — The query algebra defines no possibility to join objects; e.g., it is not possible to query for siles that share common, equal annotations.

3. *Unspecified Data Type Semantics* — Attribute annotations contain a URI that identifies the attribute's data type, i.e., the way its value has to be interpreted. Since the query algebra abstracts over concrete data types, it does not define semantics for this interpretation; for instance, it does not specify an ordering for data type values, or arithmetic operators.

4. *No Aggregate Functions* — The algebra does not specify aggregate functions, thus it is not possible to e.g., count the number of siles that fulfil a certain criterion.

We are aware of the fact that the lack of these features may cause problems for certain information needs, and plan to further extend the query language in the future in order to cover additional use cases. Nevertheless we do not want to abandon our goal of providing a model and a query algebra that are simple to understand and to use, and this goal must be considered when designing model extensions.

References

[AMMH07] Daniel J. Abadi, Adam Marcus, Samuel Madden, and Katherine J. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *Proceedings of the 33rd International Conference on Very Proceedings of the 33rd International Conference on Very Large Database (VLDB 2007)*, pages 411–422, 2007.

[BHLT06] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. *Namespaces in XML (Second Edition) (W3C Recommendation 16 August 2006).* World Wide Web Consortium, 2006. Available at `http://www.w3.org/TR/REC-xml-names/`.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986).* Network Working Group, January 2005.

[Cor] NudgeNudge Corp. Punakea – Make Tags come True.

[DB99] John R. Douceur and William J. Bolosky. A Large-scale Study of File-system Contents. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 59–70, New York, NY, USA, 1999. ACM Press.

[GHM$^+$07] Tudor Groza, Siegfried Handschuh, Knud Moeller, Gunnar Grimnes, Leo Sauermann, Enrico Minack, Cedric Mesnage, Mehdi Jazayeri, Gerald Reif, and Rosa Gudjonsdottir. The NEPOMUK Project - On the Way to the Social Semantic Desktop. In Tassilo Pellegrini and Sebastian Schaffert, editors, *Proceedings of I-Semantics' 07*, pages pp. 201–211. JUCS, 2007.

[Hay04] Patrick Hayes. *RDF Semantics (W3C Recommendation 10 February 2004).* World Wide Web Consortium, 2004.

[JW05] Ian Jacobs and Norman Walsh. *Architecture of the World Wide Web, Volume One (W3C Recommendation 15 December 2004).* World Wide Web Consortium, 2005. Available at `http://www.w3.org/TR/webarch/`.

[KBH$^+$03] David Karger, Karun Baksxhi, David Huynh, Dennis Quan, and Vineet Sinha. Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data. In *Proceedings of the 2nd Biennal Conference on Innovative Data Systems Research (CIDR 2005)*, 2003.

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004).* World Wide Web Consortium, 2004.

[MH07] Knud Möller and Siegfried Handschuh. Towards a Light-Weight Semantic Desktop. In Siegfried Handschuh and Gerald Reif, editors, *Proceedings of the Semantic Desktop Design Workshop at ESWC 2007*, pages 36–46, 2007.

[MSSvE07] Antoni Mylka, Leo Sauermann, Michael Sintek, and Ludger van Elst. NEPOMUK Ontologies. Technical report, NEPOMUK Project Consortium, 2007. Available at `http://www.semanticdesktop.org/ontologies`.

[OG92] James W. O'Toole and David K. Gifford. Names Should Mean What, Not Where. In *Proceedings of the 5th ACM SIGOPS European Workshop on Models and Paradigms for Distributed Systems Structuring*, pages 1–5, New York, NY, USA, 1992. ACM Press.

[RSK04] Pamela Ravasio, Sissel Guttormsen Schär, and Helmut Krueger. In Pursuit of Desktop Evolution: User Problems and Practices with Modern Desktop Systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 11(2):156–180, 2004.

[Sch06] Bernhard Schandl. SemDAV: A File Exchange Protocol for the Semantic Desktop. In *Proceedings of the Semantic Desktop and Social Semantic Collaboration Workshop*, volume 202, Athens, GA, USA, November 2006. CEUR Workshop Proceedings.

[SH08] Leo Sauermann and Dominik Heim. Evaluating Long-Term Use of the Gnowsis Semantic Desktop for PIM. In *The Semantic Web — ISWC 2008*, volume 5318 of *LNCS*, pages 467–482. Springer, 2008.

[SH09] Bernhard Schandl and Bernhard Haslhofer. The Sile Model – A Semantic File System Infrastructure for the Desktop. In *Proceedings of the 6th European Semantic Web Conference (ESWC 2009), Heraklion, Greece*, 2009.