Transformations in Information Supply

B. van Gils, H. A. Proper, P. van Bommel, Th.P. van der Weide

University of Nijmegen* Sub-faculty of Informatics, IRIS Group,
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
basvg@acm.org, erikp@acm.org, pvb@cs.kun.nl, tvdw@cs.kun.nl

PUBLISHED AS:

B. van Gils, H.A. Proper, P. van Bommel, and Th.P. van der Weide. Transformations in information supply. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the Workshop on Web Information Systems Modelling (WISM'04), held in conjunctiun with the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004)*, volume 3, pages 60–78, Riga, Latvia, EU, June 2004. Faculty of Computer Science and Information Technology.

Abstract. In this article, we present a model for transformation of resources in information supply. These transformations allow us to reason more flexibly about information supply, and in particular its heterogeneous nature. They allow us to change the form (e.g. report, abstract, summary) and format (e.g. PDF, DOC, HTML) of data resources found on the Web. In a retrieval context these transformations may be used to ensure that data resources are presented to the user in a form and format that is *apt* at that time.

1 Introduction

The Web today can be seen as an *information market*, on which information supply meets information demand: information is offered via the Web in the form of *resources*, which can be accessed (sometimes at a cost) by anyone interested in these resources. Information supply can be said to be heterogeneous because:

- there are many different ways to represent information. For example using a webpage, a document, an image or some interactive *form*.
- there are many different *formats* that may be used to represent information on the Web. For example, using formats such as PDF, HTML, GIF.

The following example illustrates this heterogeneity. Suppose you are browsing the Web from a PDA over a mobile-phone connection. You are on your way to an important meeting with stockholders of your company and need some last minute information on the price of your stock and that of your most important competitors. Using your favorite search engine you find a large spreadsheet with not only the latest stock price, but also their respective history, several graphs and predictions for the near future. In itself, this is a very useful resource. However, several problems occur at this point. First of all, the document is rather large which is inconvenient because you are on a slow (and possibly buggy) connection. Secondly, it may be that your PDA does not have the proper software to view this spreadsheet. Last but not least, you may not have the time

* The investigations were partly supported by the Dutch Organization for Scientific Research (NWO).

to study a complex spreadsheet, hence the form of the resource is off too. We hypothesize that transformations may cure this type of problems, for example by integrating a "transformation broker" in the retrieval engine in such a way that resources are transformed in a desirable format before sending them back to the user. The transformations in this article are considered in the context of web resources. As such they are not particularly tailored to database transformations (see e.g. our earlier work on transformations in [1, 2]).

The above mentioned forms of heterogeneity may pose problems in a retrieval setting if there is a mismatch between the user's wishes on the one hand and the form and/or format of resources on the other hand. In order to investigate the problem area more closely we have developed a conceptual model for information supply [3, 4]. This model may contribute to more insights in this complex area. Furthermore it is the basis for a prototype implementation of a retrieval engine which we will discuss briefly[1]. The main contribution of this article is twofold. We firstly extend our model with a *typing mechanism*, which is a prerequisite for the second contribution: a formal model for transformations on in the information market. With transformations we will be able to deal with the form/format issues described above.

The remainder of this article is organized as follows: we start by introducing our model for information supply in Section 2. In Section 3 we formalize the (relevant) parts of this model. A more elaborate overview is presented in [3]. Section 4 formally introduces the typing mechanism that we use in our model. This typing mechanism is also the basis for Section 5 in which we discuss transformations in detail. In Section 6 we present our conclusions.

2 The model

In this section we present our model in two steps. We start out by informally introducing our model (Section 2.1) after which we constrain it by presenting its formal properties in Section 3.

2.1 Overview

Our model of information supply is based on the distinction between data and information. The entities found on the Web, which can be identified by means of a URI [6], are *data resources*. These data resources are "information", if and only if they are relevant with regard to a given information need as it is harbored by some user. Data resources may, at least partially, convey the same information for some information need. Hence, we define *information resources* to be the abstract entities that make up information supply. Each information resource has at least one data resource associated to it. Consider for example the situation in which we have two data resources: the painting Mona Lisa, and a very detailed description of this painting. Both adhere to the same information resource in the sense that a person seeking for information on 'the Mona Lisa' will consider both to be relevant.

In a way, data resources *implement* information resources; a notion similar to that reported in [7] where 'facts' in the document subspace are considered to be 'proof' for hypotheses in the knowledge subspace. Note that each data resource may implement the information resource in a different way. One data resource may be a "graphical representation" of an information resource whereas another data resource may be a "textual representation" of the same information resource. We define a *representation* to be the combination of a data resource and an information

[1] For a detailed discussion on this architecture the reader is referred to [5, 3]

resource, and a *representation type* to indicate exactly how this data resource implements the information resource it is associated to. Examples of representation types are: full-content, abstract, keyword-list, extract, audio-only etcetera.

As an example, consider the information resource called Mona Lisa which has two data resources associated to it. One of these resources is a photograph of this famous painting whereas another may be a very detailed description of the Mona Lisa. For the former data resource the representation type would be "graphical full-content" whereas the other would have representation type "description".

Many different types of data resources can be distinguished on the Web today, such as documents in different formats (HTML, PDF, etc.), databases and interactive Web-services. This is reflected in our model by the fact that each data resource has a *data resource type*. Furthermore, data resources may have several attributes such as a price or a measurement for its quality. Such attributes can be defined in terms of an *attribute type* and the actual value that a data resource has for this given attribute type.

Last but not least, data resources can be interrelated. The most prominent example of this interrelatedness on the Web is the notion of *hyperlinks* [8, 9], but other types of relations between data resources exist as well. Examples are: an image may be *part of* a webpage and a scientific article may *refer to* other articles.

The following summarizes our model:

- Information Resources have at least one Data Resource associated to them;
- A Representation denotes the unique combination of an Information Resource and a Data Resource;
- Representations have at least one Representation Type;
- Data Resources have at least one Data Resource Type;
- Data Resources are related via Relations with a source and a destination;
- Relations have at least one Relation Type;
- Data Resources may have attributed values which are typed;
- An Attribute denotes the combination of a Data Resource and a Data Value;
- Attributes have at least one Attribute Type.

3 Formalization of resource space

As discussed in the previous sections, *resource space* consists of two types of resources: *information resources* and *data resources*. Information resources form an abstract landscape presenting the "semantics"; the "things we know something about". Data resources, on the other hand, are information that is "physically" stored in one way or the other. The *representations* relation, as discussed above, forms a bridge between these two worlds. Furthermore, in the data resource world we distinguish two types of relations: *attributions*, which couple a data value to a data resource, and *relations* between data resources. Formally, the basic concepts of our model are: information resources, representations, data resources, attributions and relations. They are represented by the following sets:

information resources: $\mathcal{IR}$	representations: $\mathcal{RP}$
data resources: $\mathcal{DR}$	attributions: $\mathcal{AT}$
data values: $\mathcal{DV}$	relations: $\mathcal{RL}$

Because we consider these to be elementary (for example, it does not make sense if something is a relation and at the same time also a data resource), these sets must be disjoint:

Axiom 1 (Disjoint Base Sets) $\mathcal{IR}, \mathcal{RP}, \mathcal{DR}, \mathcal{DV}, \mathcal{AT}, \mathcal{RL}$ are disjoint sets

Collectively, the data values and data resources are referred to as data elements:

$$\mathcal{DL} \triangleq \mathcal{DR} \cup \mathcal{DV}$$

Attributions connect data values to their respective data resources, and relations are used to interconnect data resources. Hence, attributions and relations form all possible *connections* between the data elements. Let $\mathcal{CN}$ be the set of all these connections:

$$\mathcal{CN} \triangleq \mathcal{RL} \cup \mathcal{AT}$$

The sources and destinations of connections between data elements are yielded by the functions $\mathsf{Src}, \mathsf{Dst} : \mathcal{CN} \to \mathcal{DL}$ respectively. Since these are total functions it follows that if a $c \in \mathcal{CN}$ exists then its source and destination can not be void. Even more, we state that the source and destination can not be the same element:

Axiom 2 (Source and Destination of connections)

$$c \in \mathcal{CN} \implies \exists_{e_1, e_2 \in \mathcal{DL}} [\mathsf{Src}(c) = e_1 \wedge \mathsf{Dst}(c) = e_2 \wedge e_1 \neq e_2]$$

The destination of an attribution should be a data value:

Axiom 3 (Attribute Values)

$$\forall_{a \in \mathcal{AT}} [\mathsf{Src}(a) \in \mathcal{DR} \wedge \mathsf{Dst}(a) \in \mathcal{DV}]$$

Similarly, the destination of a relation should be a data resource:

Axiom 4 (Relations)

$$\forall_{r \in \mathcal{RL}} [\mathsf{Src}(a) \in \mathcal{DR} \wedge \mathsf{Dst}(r) \in \mathcal{DR}]$$

As an abbreviation we introduce:

$$\begin{aligned} s \overset{c}{\rightsquigarrow} d &\triangleq \mathsf{Src}(c) = s \wedge \mathsf{Dst}(c) = d \\ s \rightsquigarrow d &\triangleq \exists_c \left[s \overset{c}{\rightsquigarrow} d \right] \end{aligned}$$

For example, `a.zip` $\rightsquigarrow$ `b.doc` denotes that `a.zip` and `b.doc` are related via some relation (for example, the document may be part of the ZIP archive). Another example is `x.html` $\rightsquigarrow$ UTF-8, which denotes that `x.html` uses the UTF-8 encoding.

Recall that a representation is the combination of an information resource and a data resource. They form the bridge between the abstract world of information resources and the concrete world of data resources. Hence we define $\mathsf{IRes} : \mathcal{RP} \to \mathcal{IR}$ to be a function yielding the information resource that is associated to a representation and $\mathsf{DRes} : \mathcal{RP} \to \mathcal{DR}$ to be a function providing the data resource associated to a representation.

In sum, we define resource space to be defined by the following signature:

$$\Sigma_r \triangleq \langle \mathcal{IR}, \mathcal{RP}, \mathcal{DR}, \mathcal{RL}, \mathcal{AT}, \mathcal{DV}, \mathsf{IRes}, \mathsf{DRes}, \mathsf{Src}, \mathsf{Dst} \rangle$$

4 Typing mechanism for descriptive elements

Before we are able to discuss transformations on data resources, we first need to introduce a typing mechanism on resource space. This typing mechanism allows us to limit the applicability of transformations to specific types of resources. In this section we therefore aim to extend resource space Σ_r with a typing mechanism.

All elements in resource space can be typed. Let $\mathcal{RE}$ therefore be the set of all elements in resource space:

$$\mathcal{RE} \triangleq \mathcal{IR} \cup \mathcal{RP} \cup \mathcal{DR} \cup \mathcal{RL} \cup \mathcal{AT} \cup \mathcal{DV}$$

The *resource space elements* form basis for a uniform typing mechanism. Data resources are allowed to have a type that is either "basic" or "complex". This is explained in more detail in Section 4.2.

Let $\mathcal{TP}$ to be the set of all types and $\mathsf{HasType} \subseteq \mathcal{RE} \times \mathcal{TP}$ be the relation for typing descriptive elements in our model. Our typing mechanism is inspired by *abstract data types* as introduced in e.g. [10]. This implies that we can perform operations on the instances of these types. Note that such a strategy can deal with both static as well as dynamic resources. For example, the approach as described in [11] actually uses many-sorted algebra's to formalize the behavior of objects as used in object-oriented approaches. In the case of data resources, examples of these operations/methods are:

- give me the first byte,
- give me the n'th character,
- (in the case of an XML document) give me the first node in the DOM-tree.

4.1 Types and population

Given some element from resource space, we can use $\mathsf{HasType}$ to determine the set of types of this element. For example, the types of a given file may be XML, SGML and *file* or, the type of a relation may be "part of" or "refers to". Conversely, we can also determine the set of elements of a given type. Formally, we use the functions τ and π respectively to yield these sets:

$$\tau(e) \triangleq \{t \mid e \ \mathsf{HasType}\ t\} \qquad \pi(t) \triangleq \{e \mid e \ \mathsf{HasType}\ t\}$$

These functions may be generalized to sets of elements and types respectively:

$$\tau(E) \triangleq \bigcup_{e \in E} \tau(e) \qquad \pi(T) \triangleq \bigcup_{t \in T} \pi(t)$$

If X is one of the base sets, such as $\mathcal{RL}$, $\mathcal{DR}$, then we will abbreviate $\tau(X)$ as X_τ.

Using the definitions of τ it follows that an element may have more then one type. An example from the domain of data resources illustrates this. Suppose that $E = \{\texttt{1.htm}, \texttt{2.xml}\}$ such that `1.htm` $\mathsf{HasType}$ HTML, `1.htm` $\mathsf{HasType}$ XML and `2.xml` $\mathsf{HasType}$ XML. In this case $\tau(E) = \{\text{HTML}, \text{XML}\}$. We now have:

$$\begin{array}{ll} \pi(\text{HTML}) = \{\texttt{1.htm}\} & \tau(\pi(\text{HTML})) = \{\text{HTML}\} \\ \pi(\text{XML}) = \{\texttt{1.htm}, \texttt{2.xml}\} & \tau(\pi(\text{XML})) = \{\text{HTML}, \text{XML}\} \end{array}$$

This example also shows that $\tau(\pi(\text{HTML})) \subset \tau(\pi(\text{XML}))$. We will get back to this when we discuss subtyping in Section 4.3.

We assume that all elements have a type:

Axiom 5 (Total typing) $\tau(e) \neq \emptyset$

Conversely, in our model we presume types to exist only when they have a population:

Axiom 6 (Existential typing) $\pi(t) \neq \emptyset$

In the approach we take, typing of resource space is derived *from* the available resources. In other words, a new *type* of resources can only be introduced to the model if and only if instances (data elements) of this (new) type exist. This is particularly convenient since our model has to "fit" on an existing situation: the Web. In the case of database design, for example, the opposite holds: first the schema is defined, then it is populated with instances.

The partitioning of elements from resource space over $\mathcal{IR}, \mathcal{RP}, \mathcal{DR}, \mathcal{DV}, \mathcal{AT}, \mathcal{RL}$ should be obeyed by their types as well:

Axiom 7 $\mathcal{IR}_\tau, \mathcal{RP}_\tau, \mathcal{DR}_\tau, \mathcal{DV}_\tau, \mathcal{AT}_\tau, \mathcal{RL}_\tau$ form a partition of $\mathcal{TP}$

4.2 Complex data resources

Data resources may depend on the existence of other elements from resource space. For example, some data resource may be constructed in terms of other data resources, and/or it may have some data value associated to it as an attribute. A data resource that is dependent on the existence of other elements is called a complex data resource.

In the case of a data resource which is considered to be (partially) constructed by means of other data resources, we are essentially dealing with a subset of the relations in $\mathcal{RL}$ which we regard as being *compositions*. Let therefore $\mathcal{CM} \subseteq \mathcal{RL}$ be the set of relations that are considered to be compositions of complex data resources.

The compositions, in conjunction with the attributions, are the only ways of constructing complex data resources. The compositions and attributions used to construct the complex data resources are referred to as accessors; they offer *access* to the underlying composing elements. We define the set of accessors formally as:

$$\mathcal{AC} \triangleq \mathcal{CM} \cup \mathcal{AT}$$

The types of complex data resources, the underlying data resources/values, and the composition/attribution relations between them, have a special relationship: at the instance level, accessors can be thought of as "handles" which provide access to the that data elements were used to create the instance of a complex type. At the typing level, these "handles" are reflected by the accessor types. For example, a ZIP-file may have an accessor (with type "payload"), which offers access to the files that were used to create this specific ZIP archive.

The construction of instances of complex types is restricted in the sense that cyclic behavior is forbidden: it is considered illegal if an instance a is used to construct b while at the same time b is used to construct a:

Axiom 8 (Acyclic construction) The relation R defined as $e_1 \,\mathsf{R}\, e_2 \triangleq \exists_{a \in \mathcal{AC}} \left[e_1 \overset{a}{\rightsquigarrow} e_2 \right]$ is acyclic.

Not all types of complex data resources, such as "ZIP-file" and "multi-part E-mail", will have a "payload". For example, in the case of a complex type such as "postal address" it does not make sense to use an accessor of type "payload" on its instances. This kind of restriction must be reflected at the typing level, and pertains to the fact that only instances of a specific type may be involved in an accessor. To formally represent this, we introduce the relation:

$$\_ \stackrel{\cdot}{\rightarrow} \_ \subseteq \mathcal{TP} \times \mathcal{AC}_\tau \times \mathcal{TP}$$

If $s \stackrel{u}{\rightarrow} t$, then the intuition is that complex type s has, via accessor type u, at its *base* the type t.

As an example, let $t_1 = \text{ZIP}$, $a =$ 'payload' and $t_2 =$ *file*, then $t_1 \stackrel{a}{\rightarrow} t_2$ represents the fact that ZIP-files have a payload consisting of files.

Using the definition of $\_ \stackrel{\cdot}{\rightarrow} \_$ we define the set of complex types to be:

$$\mathcal{TP}_c \triangleq \left\{ t_1 \in \mathcal{TP} \;\middle|\; \exists_{a \in \mathcal{AC}_\tau, t_2 \in \mathcal{TP}} \left[t_1 \stackrel{a}{\rightarrow} t_2 \right] \right\}$$

At the instance level, accessors should behave as stipulated at the type level:

Axiom 9 (Correct types)

$$e_1 \stackrel{a}{\rightsquigarrow} e_2 \implies \exists_{t_1 \in \tau(e_1), t \in \tau(a), t_2 \in \tau(e_2)} \left[t_1 \stackrel{t}{\rightarrow} t_2 \right]$$

The set of accessors that is associated to a complex type is defined by:

$$\mathsf{Acc}(t_1) \triangleq \left\{ t \in \mathcal{AC}_\tau \;\middle|\; \exists_{t_2} \left[t_1 \stackrel{t}{\rightarrow} t_2 \right] \right\}$$

This definition can be generalized to the instance level:

$$\mathsf{Acc}(e) \triangleq \bigcup_{t \in \tau(e)} \mathsf{Acc}(t)$$

Note that it may be the case that some of the accessor types in an instance of a complex type are unused. For example, not every ZIP file has a comment or a password associated to it.

If two complex types have the same set of accessor types, such that the types at the base of these accessor types are the same, then the two complex types are really the same:

Axiom 10 (Equality of Complex Types) If $\mathsf{Acc}(s_1) = \mathsf{Acc}(s_2)$, then:

$$\forall_{u \in \mathsf{Acc}(s_1), t \in \mathcal{TP}} \left[s_1 \stackrel{u}{\rightarrow} t \Leftrightarrow s_2 \stackrel{u}{\rightarrow} t \right] \implies s_1 = s_2$$

As an example of how the accessor mechanism works in practice, consider the following example: suppose `x.zip` is a ZIP-file, while it's payload consists of three files, `a.doc`, `b.ps` and `c.pdf`. They can be accessed via their respective accessors a_1, a_2 and a_3 which all have accessor type "payload". Note that this accessor type is really a composition ($\mathcal{CM}$)! Furthermore, there is a comment and a password attached to the ZIP-file which are accessed via accessors a_4 and a_5

which have accessor types "comment" and "password" respectively. These accessor types originate from attributions. More formally:

$$\begin{aligned}
\pi(\mathcal{DR}) &= \{\texttt{x.zip}, \texttt{a.doc}, \texttt{b.ps}, \texttt{c.pdf}\} \\
\pi(\mathcal{DR}_\tau) &= \{\text{ZIP}, \text{DOC}, \text{PS}, \text{PDF}, \textit{file}\} \\
\pi(\mathcal{DV}) &= \{\text{"some comment"}, \text{"secret"}\} \\
\pi(\mathcal{DV}_\tau) &= \{\textit{String}\} \\
\pi(\mathcal{CM}) &= \{a_1, a_2, a_3\} \\
\pi(\mathcal{CM}_\tau) &= \text{"payload"} \\
\pi(\mathcal{AT}) &= \{a_4, a_5\} \\
\pi(\mathcal{AT}_\tau) &= \{\text{"comment"}, \text{"password"}\}
\end{aligned}$$

Note that for $a \in \{a_1, a_2, a_3\}$ it holds that ZIP $\xrightarrow{a}$ *file*[2]. Similarly, for $a \in \{a_4, a_5\}$ it holds that ZIP $\xrightarrow{a}$ *String*.

Figure 1 provides a graphical depiction of the above sketched situation. The left-hand side of the figure is at the instance level, whereas the right-hand side is at the typing level.

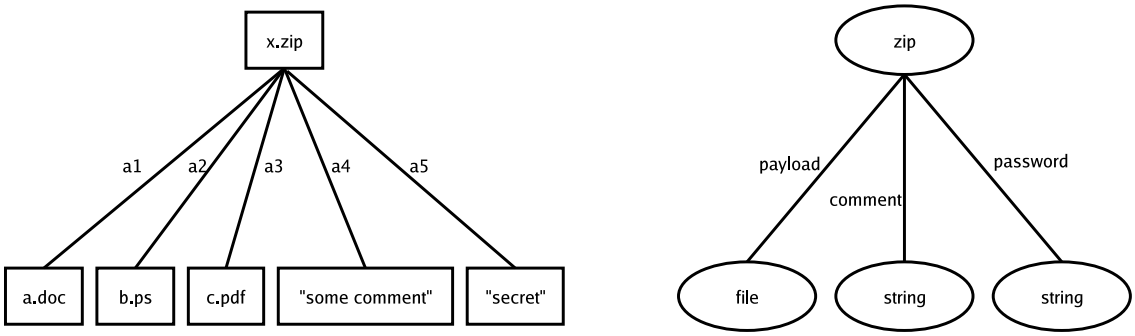

Fig. 1. Accessors

4.3 Subtyping

We assume the existence of subtyping. Let $\underline{\text{SubOf}} \subseteq \mathcal{TP} \times \mathcal{TP}$ therefore define a subtyping relationship, where $s \,\underline{\text{SubOf}}\, t$ indicates that type s is a subtype of, or equal to type t. Based on this definition, we introduce the notion of *proper subtypes*:

$$s \,\text{SubOf}\, t \;\triangleq\; s \,\underline{\text{SubOf}}\, t \wedge \neg t \,\underline{\text{SubOf}}\, s$$

We presume $\underline{\text{SubOf}}$ to be transitive, reflexive and antisymmetric:

Axiom 11 (Behavior of $\underline{\text{SubOf}}$)

$$t \in \mathcal{TP} \implies t \,\underline{\text{SubOf}}\, t$$

$$t \,\underline{\text{SubOf}}\, s \wedge s \,\underline{\text{SubOf}}\, t \implies t = s$$

$$s \,\underline{\text{SubOf}}\, t \,\underline{\text{SubOf}}\, u \implies s \,\underline{\text{SubOf}}\, u$$

[2] The notion of subtyping is introduced in Section 4.3.

From this we can prove:

Lemma 1 SubOf is irreflexive, asymmetric and transitive

At the instance level, if $s \underline{\mathsf{SubOf}}\, t$ then the population of s must be a subset of, or equal to the population of t:

Axiom 12 (Population and SubOf)

$$s \underline{\mathsf{SubOf}}\, t \implies \pi(s) \subseteq \pi(t)$$

From this we can prove that:

Lemma 2 $s \,\mathsf{SubOf}\, t \implies \pi(s) \subset \pi(t)$

For example, if XML SubOf SGML and $x \in \pi(\text{XML})$ then Lemma 2 states that also $x \in \pi(\text{SGML})$. Recall that, at the instance level, the type of a resource can be seen as the interface with which instances can be accessed. Hence, an instance with type XML can also be accessed via an "SGML-interface".

If a complex type has a subtype then the accessor types of the supertype are inherited:

Axiom 13 (Inherritance of accessor types)

$$s_1 \underline{\mathsf{SubOf}}\, s_2 \implies \mathsf{Acc}(s_1) \subseteq \mathsf{Acc}(s_2)$$

This axiom forbids the situation that a type with 2 accessor types is a subtype of another type with 3 accessor types (the converse is allowed, and is akin to specialization in object-orientation).

Types that are at the base of a specific accessor type (in the context of a single complex type) should be subtypes:

Axiom 14 (Subtyping of accessor bases)

$$s \xrightarrow{u} t_1 \wedge s \xrightarrow{u} t_2 \implies t_1 \underline{\mathsf{SubOf}}\, t_2 \vee t_2 \underline{\mathsf{SubOf}}\, t_1$$

Even more, the set of types that are at the base of an accessor type comprises all relevant super types:

Axiom 15 (Inclusion of super types)

$$s \xrightarrow{u} t_1 \wedge t_1 \,\mathsf{SubOf}\, t_2 \implies s \xrightarrow{u} t_2$$

If a complex type has a subtype then the underlying base types must obey this subtyping as well:

Axiom 16 (Base types obey subtyping)

$$s_1 \underline{\mathsf{SubOf}}\, s_2 \wedge s_1 \xrightarrow{u} t_1 \wedge s_2 \xrightarrow{u} t_2 \implies t_1 \underline{\mathsf{SubOf}}\, t_2$$

From the above Axiom, in combination with Axiom 10, it follows that if two complex types are proper subtypes, then there is at least one accessor type whose base types show this proper subtyping:

Lemma 3 $s_1, s_2 \in \mathcal{TP}_c \wedge s_1 \,\mathsf{SubOf}\, s_2 \implies \exists_{u,t_1,t_2} \left[s_1 \xrightarrow{u} t_1 \wedge s_2 \xrightarrow{u} t_2 \wedge t_1 \,\mathsf{SubOf}\, t_2 \right]$

4.4 Typed resource space

In sum, we define a typed resource space to be defined by the following signature:

$$\Sigma_r^{\mathcal{T}} \triangleq \langle \Sigma_r, \mathcal{TP}, \mathcal{CM}, \mathsf{HasType} \rangle$$

5 Transformations

In this section we introduce *transformations*, a way to change the nature / structure of instances. These transformations can be very used in practice to solve several problems. For example:

- Suppose we have an image in EPS file that we want to view. Unfortunately we don't have a viewer for this file-type. We do have a viewer for JPEG files, though. By means of a transformation we may be able to transform the EPS file to JPEG and thus access the information we need.
- Managers of large organizations often have to read many lengthy reports. Because of time constraints it is not always possible to read all these reports. Again, transformations may help. Transformations exists to generate abstracts of these documents.

In other words, transformations help us to have a more flexible view on the information landscape. In generl, one can distinguish between an extensional database and intentional database [12, 13]. The extensional database corresponds to the a set of basic facts known about the world, whereas the intentional database represents the facts that may be derived from the extensional database by applying inference rules. The transformations can be regarded as inference rules on the extensional database (information supply as we know it), resulting in a larger intentional database.

The remainder of this section is organized as follows. In Section 5.1 we define what transformations are and show their basic properties. Section 5.2 elaborates and presents complex transformations.

5.1 Basic Properties

Recall that IRes finds the unique information resource associated to a representation, and that DRes finds the unique data resource associated to a representation. Essentially, a representation is information represented on a medium, and the representation type expresses how / to what extent this is done.

As was stated before, with transformations we can transform data resources. This paper does not present a language for specifying what a specific transformation does / a language for composing transformations. We focus on general properties of transformations and, hence, view them as a "black box" for the time being.

Let $\mathcal{TR}$ be the set of all transformations. The semantics of a transformation $T \in \mathcal{TR}$ is given by the function:

$$\mathsf{SEM} : \mathcal{TR} \rightarrow (\mathcal{DR} \rightarrowtail \mathcal{DR})$$

In other words, transformations transform a representation to another. As an abbreviation, let $\overrightarrow{T} \triangleq \mathsf{SEM}(T), T \in \mathcal{TR}$. Furthermore, let $i \models d$ denote that data resource d is associated to information resource i via some representation:

$$i \models d \triangleq \exists_{r \in \mathcal{RP}} [\mathsf{IRes}(r) = i \wedge \mathsf{DRes}(r) = d]$$

If a data resource is transformed, then the resulting data resource is associated to the same information information resource as the original information resource.

Axiom 17 ($\mathcal{IR}$ neutral transformations)

$$i \models d \wedge \overrightarrow{T}(d) = d' \implies i \models d'$$

Any given transformation has a fixed input and output for which it is defined, similar to the notion of mathematical functions having a domain and a range: $\mathsf{Input}, \mathsf{Output} : \mathcal{TR} \to \tau(\mathcal{DR})$. Let $t \xrightarrow{T} u$ denote the fact that transformation $T \in \mathcal{TR}$ can be applied on instances of type t and results in instances of type u:

$$t \xrightarrow{T} u \triangleq \mathsf{Input}(T) = t \wedge \mathsf{Output}(T) = u$$

Any given transformation is only defined for all instances that are of the correct input-format. Even more so, it can only produce instances of its output-format:

Axiom 18 (I/O of Transformations)

$$\text{if } t \xrightarrow{T} u \text{ then } T : \pi(t) \rightarrowtail \pi(u)$$

This allows us to define how a transformation T can be applied to a set of data resources. Let $E \subseteq \mathcal{DR}$ be a set of data resources. Then:

$$\overrightarrow{T}(E) \triangleq \left\{ e \;\middle|\; e \in E \wedge e \notin \mathsf{Input}(T) \right\} \cup \left\{ \overrightarrow{T}(e) \;\middle|\; e \in E \wedge e \in \mathsf{Input}(T) \right\}$$

Another property of transformations is the fact that they are transitive:

Axiom 19 (Transitivity of Transformations)

$$e \xrightarrow{T_1} f \wedge f \xrightarrow{T_2} g \implies \exists_{T_3} \left[e \xrightarrow{T_3} g \wedge T_3 = T_1 \circ T_2 \right]$$

This property can be used to transform data resources into an appropriate format even is there is no 1-step transformation is available. It is, for example, possible to generate an abstract of a large ASCII-file and transform that to PS by sequencing the two transformations.

5.2 Complex Transformations

In the previous section we presented a framework for transformations and showed how transformations can be composed by sequencing them using the $\circ$ operator. In this section we discuss a more complex way of composing transformations, relying heavily on the accessor types presented in previous sections. We define a transformation to be complex if the transformation operates on instances that were used to create an instance of a complex type (that is, instances at the *base* of an instance of a complex type). There are two types of complex transformations which, like all transformations, may be sequenced using the $\circ$ operator.

The first complex transformation is used to *remove* an accessor and the instance(s) at its base. For example, it may be desirable to remove a comment from a ZIP-file, or to remove an attachment from an E-mail. Such transformation:

– takes an instance with a complex type as input;
– removes a specified accessor and its base from an instance with a complex type;
– leaves other accessors (and their bases) untouched.

More formally, Let e be an instance with a complex type and $a \in \mathsf{Acc}(\tau(e))$:

$$\varrho_a(e) = e' \triangleq e' \ltimes a = \emptyset \wedge \forall_{b \neq a} [e \ltimes b = e' \ltimes b]$$

In the above definition we have used the following shorthand notation:

$$c \ltimes t \triangleq \left\{ d \;\middle|\; c \overset{a}{\rightsquigarrow} d \wedge a \in \pi(t) \right\}$$

The intuition behind this shorthand is that $c \ltimes ad$ retrieves all data elements that are used in constructing complex data resources c via accessors of type t.

This type of transformations can be performed on each instance with a complex type, since such an instance *must* have at least one accessor. If the last accessor of an instance is removed then ϱ is said to destruct the instance.

Axiom 20 (Existence of ϱ)

$$\begin{array}{ll} \textbf{if} & t \in \mathcal{TP}_c, a \in \mathsf{Acc}(T) \\ \textbf{then} & \exists_{T \in \mathcal{TR}} \left[t \xrightarrow{T} t \wedge \overrightarrow{T} = \varrho_a \right] \end{array}$$

The second class of complex transformations does a little more work; they are *deep* transformations in the sense that instances at the base of a complex type are transformed. For example, all DOC files in a ZIP archive may be transformed to PDF. These transformations:

– takes an instance with a complex type as input, and returns an instance with a (possibly different) complex type;
– Transform the instances a the base of an accessor;
– leave other accessors (and their bases) untouched.

More formally, Let e be an instance of a complex type, $a \in \mathsf{Acc}(\tau(e))$ and $T \in \mathcal{TR}$:

$$\alpha_{a:T}(e) = e' \triangleq e' \ltimes a = T(e \ltimes a) \wedge \forall_{b \neq a} [e \ltimes b = e' \ltimes b]$$

These transformations are defined for all types t_1, t_2 as long as they have the same accessor types. Even more, transformation T must at least be defined for the instances at the base of the specified accessor:

Axiom 21 (Existence of α)

$$\begin{array}{ll} \textbf{if} & \mathsf{Acc}(t_1) = \mathsf{Acc}(t_2) \wedge a \in \mathsf{Acc}(t_1) \wedge \\ & \exists_{b_1, b_2} \left[t_1 \xrightarrow{a} b_1 \wedge t_2 \xrightarrow{a} b_2 \wedge b_1 \xrightarrow{T} b_2 \right] \\ \textbf{then} & \exists_{T' \in \mathcal{TR}} \left[t_1 \xrightarrow{T'} t_2 \wedge \overrightarrow{T'} = \alpha_{a:T} \right] \end{array}$$

To illustrate how such a deep transformation can be used to transform an instance from complex type t_1 to complex type t_2, consider the following situation. t_1 is the format for an E-mail for which the body is in UTF-8 encoding, and t_2 has its body in UTF-16 encoding. That is, t_1 has an accessor with type UTF-8 and some text formatted accordingly at its base and the same accessor has, in the context of type t_2, accessor type UTF-16. If T is a transformation capable of transforming text in UTF-8 encoding to UTF-16 encoding then Axiom 21 dictates that a T' must exist such that $t_1 \xrightarrow{T'} T_2$.

5.3 Example

In this section we present an example that relies on Axioms 19, 20 and 21. Consider the following: Let `backup.zip` be a ZIP archive. Two files (`report.doc` and `letter.doc`) form the payload of this archive. Also, a comment ("`backup`") and a password ("`secret`") are associated to it. In other words:

$$\begin{aligned}
\tau(\texttt{backup.zip}) &= \text{ZIP} \\
\mathsf{Acc}(\texttt{backup.zip}) &= \{\text{payload}, \text{comment}, \text{password}\} \\
\texttt{backup.zip} \ltimes \text{payload} &= \{\texttt{report.doc}, \texttt{letter.doc}\} \\
\texttt{backup.zip} \ltimes \text{comment} &= \text{“}\texttt{backup}\text{”} \\
\texttt{backup.zip} \ltimes \text{password} &= \text{“}\texttt{secret}\text{”}
\end{aligned}$$

Now, let T_1 be a transformation with $\mathsf{Input}(T) = \text{DOC}$ and $\mathsf{Output}(T) = \text{PDF}$. Then, $\alpha_{\text{payload}:T_1}$ is a transformation that transforms the documents in the payload of any ZIP archive to PDF. Let $\varrho_{\text{password}}$ be a transformation that removes the password of a ZIP archive.

If we want to transform `backup.zip` such that the documents in its payload are transformed to PDF and its password is removed then we can achieve this as follows:

$$\begin{aligned}
T &= \alpha_{\text{payload}:T_1} \circ \varrho_{\text{password}} \\
\overrightarrow{T}(\texttt{backup.zip}) &= \texttt{new.zip}
\end{aligned}$$

The result of this transformation is a new archive `new.zip` such that:

$$\begin{aligned}
\tau(\texttt{new.zip}) &= \text{ZIP} \\
\mathsf{Acc}(\texttt{new.zip}) &= \{\text{payload}, \text{comment}\} \\
\texttt{new.zip} \ltimes \text{payload} &= \{\texttt{report.pdf}, \texttt{letter.pdf}\} \\
\texttt{new.zip} \ltimes \text{comment} &= \text{“}\texttt{backup}\text{”}
\end{aligned}$$

5.4 Open issues

In this section we have presented a theoretical framework for transformations and their basic properties. This framework allows us to reason more efficiently about the information that is supplied to us via the Web. In [5] we have presented a retrieval architecture called *Vimes* that makes use of these transformations in a retrieval-setting. The main idea behind *Vimes* is that data resources on the Web may be transformed in a format suitable for the user.

What is missing still, though, is a mechanism to examine the effects of transformations, *and transformation paths* in particular. Suppose a transformation from p to q is needed, and two sequences of transformations are possible to achieve this. Which sequence is "best"? Based on which properties / quality attributes can such a decision be made? Devising a mechanism is part of future research.

6 Conclusion

In this article we set out to do two things: present a formal model for information supply, the totality of information available to us via the Web, and present a framework of transformations to add flexibility to this model.

The basic model stems from earlier work [3, 4] with basic elements: data resource, information resource, representation, value, attribution and relation. In this article we extended it with an extensive typing mechanism with an explicit distinction between *basic* and *complex* types. An instance is said to be complex if other instances (data resources or attributed values) were used to construct it. An example is a ZIP-file with several documents, a password and a comment associated to it.

For our transformation framework we defined that transformations work on data resources. A distinction is made between the *semantics* of a transformation (pertaining to its signature), and its actual *application* on real instances. Transformations have a fixed input type and output type similar to mathematical functions having a domain and a range.

We distinguish two types of transformations: transformations on instances of simple types and deep transformations (which operate on instances used to construct the instance of a complex type). Furthermore, a property of all transformations is that they may be transitively nested.

Even though our model covers both "simple" and "complex" transformations, much work needs to be done still. First of all, the *effects* of transformations must be studied still. That is, by performing a transformation on a data resource it's (perceived) quality may change. Even more so, if a transformation from one type to another may be achieved via two possible sequences of transformation, a choice must be made: which one is the best, and why? Last but not least, a *language* to constrain our model and transformations in this model must be developed still. Last but not least, we are currently working on an implementation of our transformation framework and refer the interested reader to [5, 3] for details.

References

1. Bommel, P.v., Kovács, G., Micsik, A.: Transformation of database populations and operations from the conceptual to the internal level. Information Systems **19** (1994) 175–191.
2. Proper, H.: Data Schema Design as a Schema Evolution Process. Data & Knowledge Engineering **22** (1997) 159–189.
3. Gils, B.v., Proper, H., Bommel, P.v.: A conceptual model for information suppy. Technical Report NIII-R0313, Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU (2003) Accepted for publication in Data & Knowledge Engineering.
4. Gils, B.v., Proper, H., Bommel, P.v.: Towards a general theory for information supply. In Stephanidis, C., ed.: Proceedings of the 10th International Conference on Human-Computer Interaction, Crete, Greece, EU (2003) 720–724. ISBN 0805849300
5. Gils, B.v., Schabell, E.: User-profiles for information retrieval. In: Proceedings of the 15th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'03), Nijmegen, The Netherlands (2003).
6. Berners-Lee, T.: Universal resource identifiers in www. Technical Report RFC 1630, IETF Network Working Group, http://www.ietf.org/rfc/rfc1630.txt (1994).
7. Feng, L., Hoppenbrouwers, J., Jeusfeld, M.: Towards knowledge-based digital libraries. SIGMOD Record 30 **1** (2001) 41–46.
8. Conklin, J.: Hypertext: An Introduction and Survey. IEEE Computer **20** (1987) 17–41.
9. Bush, V.: As we may think. The Atlantic Monthly **176** (1945) 101–108.
10. Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. Journal of the ACM (JACM) **24** (1977) 68–95 ISSN: 0004-5411.
11. Bruce, K., Wegner, P.: An algebraic model of subtype and inheritance. In Bancilhon, F., Buneman, P., eds.: Advances in Database Programming Languages. ACM Press, Frontier Series. Addison-Wesley, Reading, Massachusetts (1990) 75–96.
12. Ullman, J.: Principles of Database and Knowledge-base Systems. Volume II. Computer Science Press, Rockville, Maryland (1989). ISBN 071678162X
13. Date, C.J.: An introduction to Database Systems. 8th edn. Addison Wesley, Bosten, Massachusets, USA (2003). ISBN: 0-321-18956-6