Typing and transformational effects in complex information supply

B. van Gils* H.A. Proper P. van Bommel Th.P. van der Weide
basvg@cs.ru.nl e.proper@cs.ru.nl pvb@cs.ru.nl tvdw@cs.ru.nl

Radboud University Nijmegen, Institute for Computing and Information Sciences
Toernooiveld 1, 6525 ED, Nijmegen, The Netherlands

March 10, 2005

Abstract

Information plays an increasingly important role in our lives. Often we retrieve this information by querying the Web: data resources found on the web may provide the information that we're looking for. This implies that the Web may be seen as an information market: authors supply information and searchers may find it. In this article we present a formal framework for the syntactic aspects of the information market. We explore the information landscape using a modeling approach. An important part of this model is a (syntactic) framework for transformations, which allows us to deal with the heterogeneity of the data resources found on the Web. Last but not least we attempt to give an outline how our framework, which in essence focuses on the data resources on the Web, may lead to a better understanding of how information is supplied via the Web. For this we use an example from the field of information retrieval.

Keywords: Transformation, Information supply, Typing, WWW

1 Introduction

The Web today can be seen as an *information market*, where 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 is of a heterogeneous nature. This heterogeneity is due to at least the following causes:

- 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*.

*Corresponding author, can be reached via telephone at +31-(0)24-3652632 and fax at +31-(0)24-3653356. Postal address: Tournooiveld 1, room A-4031, NL-6525 ED Nijmegen, The Netherlands

The following example illustrates this heterogeneity. Imagine yourselves browsing the Web using a PDA which is connected to the internet by way of 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 to study a complex spreadsheet, hence the form of the resource is off too. This kind of heterogeneity of information supply is likely to cause problems in a retrieval setting, whenever there is a mismatch between the user's wishes on the one hand and the actual form and/or format in which the resources are available on the other hand. In order to investigate the problem area more closely we have developed a conceptual model for information supply (Gils et al., 2003a; Gils et al., 2003b).

In this paper we focus on transformations. Transformations play an important role in several areas. Two main application areas are recognized as follows (see also (Lammel, 2004)). On the one hand, transformations are essential in system development in general, and in transformational specification strategies in particular. On the other hand, when a system has been developed and is being used in a practical context, particular user objects may be available in a heterogeneous environment only (e.g. documents in the context of the WWW). This introduces the need for appropriate transformations.

We hypothesize that having a set of appropriate transformations available, which may be applied automatically by the search infrastructure, will improve upon the current situation. The transformations we refer to here, are transformations of web resources in general; we do not limit ourselves to database transformations (see e.g. our earlier work on transformations in (Bommel et al., 1994; Proper, 1997)). In terms of (Ullman, 1989), the set of resources available on the web can be seen as a (large!) *extensional* database. Use of transformations yields a practically infinite *intensional* database. The *extensional* web thus consists of the resources that are directly available, whereas the *intensional* web also includes the resources that can be derived from the former resources using transformations. As a result, searchers are no longer limited to the *extensional web* when selecting resources. The resources from the much larger *intensional web* can be selected as well.

Resources from the *extensional* web may exhibit several properties of interest, such as its file type and file size, relations to other resources, etc. These properties will be affected by the transformations by which the *intensional* web is be derived from the *intensional* web. In this article, we therefore have a particular interest in the effects that these transformations may (and are desired to have) on the resources.

As most practical sets of transformations of resources can be applied repeatedly, the *intensional* web is practically infinite. This makes that searching can only be done practically if branch-and-bound like optimization strategies are used to reduce search space. We will present a search strategy based that is essentially based on a push-down selection strategy as used in database query optimization (Ullman, 1989):

- First select the resources from the *extensional* web that are topically relevant.
- Use transformations to increase the aptness of the resources.

In selecting topically relevant resources from the *extensional* web, we should aim for a high recall. In other words, it is much less desired to miss potentially relevant resources than it is to

include irrelevant ones. The final selection on topical relevance can still be made at a later stage. This initial selection yields a more workable set of resources, upon which transformations can be applied that tailor the available resources to the specific requirements of the searcher. In other words, the transformations are used to increase the *aptness* of the resources (Gils et al., 2003a). The example given above, already eluded to the fact that *aptness* involves more criteria than just topical relevance.

Several approaches to the study and application of transformations are found in the literature. An important line of research deals with graph transformation, where transformations are defined in terms of rules applied to a source (or parent) graph, resulting in a target (or child) graph. An overview is presented in (Andries et al., 1999). Other lines of research focus on specific application domains, such as bibliographic applications (e.g. (Al-Mishwat, 2000) and biological applications (e.g. (Yang et al., 2005)).

The main contributions of this article are twofold. We start by refining our original conceptual model (Gils et al., 2003a; Gils et al., 2003b) with a distinction between *resource space* (defining the expanse of possible resources) and a *resource base* (representing the set of resources that are available for retrieval at some point in time), we then extend our model with a *typing mechanism*. This typing mechanism is a prerequisite for the second contribution: a formal model for transformations in the information market. In (Gils and Schabell, 2003) we have presented a retrieval architecture that uses transformations.

The remainder of this article is therefore 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 (Gils et al., 2003a). 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, whereas we discuss the effects of transformations in Section 6. Last but not least, in Section 7 we show how transformations can be used in practice. In Section 8 we present our conclusions.

2 The model

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

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 (Berners-Lee, 1994), 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 e.g. (Feng et al., 2001) 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 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*.

As mentioned before, 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* (Conklin, 1987; Bush, 1945), 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

3.1 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 connections: *attributions*, which couple a data value to a data resource, and *relations* between data resources. Figure 1 illustrates the concepts that we use to model resource space[1]. Note that some of the concepts (such as *resource element* and *accessor*) have not yet been defined in the information overview in the previous section. The main purpose of this figure is to provide the reader with a graphical overview of the model throughout the discussion of its formalization in the remainder of this section. Formally, the basic concepts of

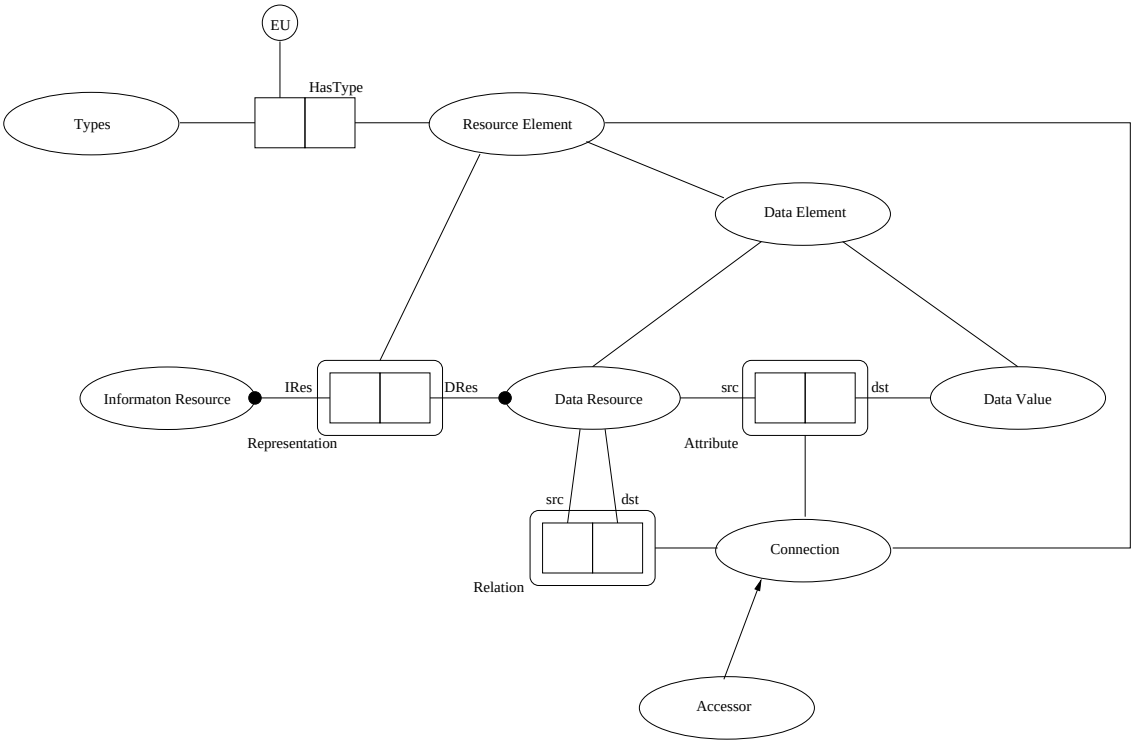


Figure 1: Concepts in resource space

our model are: information resources, representations, data resources, attributions and relations. These basic concepts 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 these sets must be disjoint:

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

[1]The figure uses the *object role modeling* (ORM) notation. The reader is referred to (Halpin, 1995; Hofstede and Weide, 1993) for details.

the data elements. Let $\mathcal{CN}$ be the set of all these connections:

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

Connections can be used to construct *complex data elements*. The intuition is that complex elements can be created by connecting them. A characteristic of complex objects is that their existence depends on the composing data elements, similar to the notion of *aggregation* in modeling languages such as UML (See e.g. (Booch et al., 1999)). This mechanism for creating complex data elements is further elaborated in Section 4.2. The following examples exemplify the main idea: A *Zip*-file consisting of a number of *PostScript* files, is a complex element (a complex data resource, to be precise). An office document containing embedded objects, such as a diagram and tables, is a complex element as well. The connections that are used to *construct* complex data elements are referred to as *accessors*; they provide *access* to the constituent elements of a complex element. Let $\mathcal{AC} \subseteq \mathcal{CN}$ be the set of accessors in resource space. Accessors can be thought of as handles which provide access to the data elements that were used to create complex elements. For example, a *Zip*-file may be regarded as having an accessor (with type *Payload*), which offers access to the files that were used to create this specific *Zip* archive.

Note that not *all* connections in a complex instance have to be accessors. It is, for example, quite possible to store two *Html* files in a *Zip* file that *refer* to each other.

The sources and destinations of connections between data elements are yielded by the functions $\mathsf{Src}, \mathsf{Dst} : \mathcal{CN} \rightarrow \mathcal{DL}$ respectively. As an abbreviation we introduce:

$$\begin{array}{lcl} s \overset{c}{\rightsquigarrow} d & \triangleq & \mathsf{Src}(c) = s \wedge \mathsf{Dst}(c) = d \\ s \rightsquigarrow d & \triangleq & \exists_c \, [s \overset{c}{\rightsquigarrow} d] \end{array}$$

For example, `a.zip` $\rightsquigarrow$ `b.doc` denotes the fact that `a.zip` and `b.doc` are related via some relation, the document is part of the *Zip* archive. Another example is `x.html` $\rightsquigarrow$ *UTF-8*, which denotes that `x.html` uses the *UTF-8* encoding.

Since Src and Dst are total functions it follows that for each connection $c \in \mathcal{CN}$ its source and destination can not be void:

Corollary 1 $c \in \mathcal{CN} \implies \exists_{e_1, e_2 \in \mathcal{DL}} \, [e_1 \overset{c}{\rightsquigarrow} e_2]$

Connectors can not be connected to by yet other connectors. In other words, connectors cannot be nested:

Lemma 1 $e_1 \overset{a}{\rightsquigarrow} e_2 \implies \{e_1, e_2\} \cap \mathcal{CN} = \emptyset$

Proof:

Let $e_1 \overset{a}{\rightsquigarrow} e_2$. From the definition of $\_ \overset{\_}{\rightsquigarrow} \_$ we know that $\mathsf{Src}(a) = e_1 \wedge \mathsf{Dst}(a) = e_2$. From the definitions of Src and Dst we know that $e_1, e_2 \in \mathcal{DL}$. From Axiom 1 it now follows that e_1, e_2 can not be members of $\mathcal{CN}$ and thus that $\{e_1, e_2\} \cap \mathcal{CN} = \emptyset$

□

Recall that *relations* connect data resources to data resources and that *attributions* connect data resources to data values. Hence it follows that the source of a connection is always a data resource, that the destination of a relation is a data resource and that the destination of a attribution is a data value:

Axiom 2 (Attributions) $\forall_{a \in \mathcal{AT}} \, [\mathsf{Src}(a) \in \mathcal{DR} \wedge \mathsf{Dst}(a) \in \mathcal{DV} \,]$

Axiom 3 (Relations) $\forall_{r \in \mathcal{RL}} [\mathsf{Src}(a) \in \mathcal{DR} \wedge \mathsf{Dst}(r) \in \mathcal{DR}]$

The construction of complex instances (by means of accessors) 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 4 (Acyclic construction)
The graph spanned by the relation R defined as $e_1 \,\mathsf{R}\, e_2 \triangleq \exists_{a \in \mathcal{AC}} [e_1 \overset{a}{\rightsquigarrow} e_2]$ is acyclic.

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 $\mathsf{IRes} : \mathcal{RP} \rightarrow \mathcal{IR}$ to be a function yielding the information resource that is associated to a representation and $\mathsf{DRes} : \mathcal{RP} \rightarrow \mathcal{DR}$ to be a function providing the data resource associated to a representation.

Each information resource should have some representation and each data resource should be involved in some representation. The intuition is that each data resource is about at least one information resource (See for example (Huibers et al., 1996)):

Axiom 5 IRes is a surjective function.

Axiom 6 DRes is a surjective function.

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

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

This signature (as well as the above axioms) constrain the extra temporal population of resources; that is: every possible population at any point in time. The following section defines resource*base*. The importance of the distinction between these two is further explained in Section 4.5.

3.2 Resource base

A resource base consists of those resources in resource space that are available at some moment in time. This set spans a sub-space of resource space. Given a resource space Σ, a resource base therefore essentially corresponds to a substructure of Σ that on itself forms a resource (sub)space as well. More formally, a resource base Σ_B is a sub-structure of Σ:

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

such that:

- the $X_B \subseteq X$ for all the base sets,
- $\mathsf{IRes}_B, \mathsf{DRes}_B, \mathsf{Src}_B, \mathsf{Dst}_B$ are restricted to the respective base sets,
- all the above discussed axioms apply to the sub-structure.

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 (and consequently any resource base within this 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 with a typing mechanism.

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

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

The *resource space elements* form the basis for a uniform typing mechanism.

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. (Goguen et al., 1977). 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 (Bruce and Wegner, 1990) 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.

In previous work we presented an example in which we showed how the type *Ascii* can be described by set of *Characters*, the set of *Integers*, the function len that gives the length of an *Ascii*-string and a function char that gives the n'th character of an *Ascii*-string. See (Gils et al., 2003a) for details.

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 (a data resource) 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 $X \subseteq \mathcal{RE}$ is a set of resource elements, in particular one of the basic sets such as $\mathcal{RP}$ and $\mathcal{DR}$, then we will abbreviate $\tau(X)$ as X_τ. The following example illustrates this. Let $\mathcal{DR} = \{e_1, e_2, e_3\}$ be the set of all data resources such that e_1 $\mathsf{HasType}$ t_1, e_2 $\mathsf{HasType}$ t_2 and e_3 $\mathsf{HasType}$ t_2. Then $\mathcal{DR}_\tau = \tau(\mathcal{DR}) = \{t_1, t_2\}$.

Using the definitions of τ it follows that an element may have more than 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 $\tau(E) = \{\mathit{Html}, \mathit{Xml}\}$. We now have:

$$\pi(Html) = \{\texttt{1.htm}\} \qquad \tau(\pi(Html)) = \{Html\}$$
$$\pi(Xml) = \{\texttt{1.htm}, \texttt{2.xml}\} \qquad \tau(\pi(Xml)) = \{Html, Xml\}$$

This example also shows that $\tau(\pi(Html)) \subset \tau(\pi(Xml))$. We will get back to this when we discuss subtyping in Section 4.4.

We assume that all elements have a type:

Axiom 7 (Total typing) $\tau(e) \neq \emptyset$

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

Axiom 8 (Existential typing) $\pi(t) \neq \emptyset$

Note that the above two axioms refer to resource *space* and not to a specific resource base. If B is a resource base, then $\pi(t) \cap B$ can quite well be empty. In resource *space*, however, it does not make sense to allow for types that can not have a population.

As a consequence we can now prove that an element is in the population of its type:

Lemma 2 $e \in \pi(\tau(e))$

Proof:

Let $t \in \tau(e)$. Due to Axiom 7 we know that such a t exists. From the definition of τ it follows that e HasType t. It also follows that $e \in \pi(t)$. Since $t \in \tau(e)$ we have $e \in \pi(\tau(e))$.

□

We can also prove that a type is in the type-set of its population:

Lemma 3 $t \in \tau(\pi(t))$

The proof of this follows that of Lemma 2. Two types are equal if their populations are equal. This axiom is of particular relevance for the definition of subtying in Section 4.4.

Axiom 9 (Equal types) $\pi(s) = \pi(t) \implies s = t$

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

Axiom 10 (Partitioning of types) $\mathcal{RP}_\tau, \mathcal{DR}_\tau, \mathcal{DV}_\tau, \mathcal{AT}_\tau, \mathcal{RL}_\tau$ form a partition of $\mathcal{TP}$

4.2 Typing of connectors

The sources and destinations of connectors ($\mathcal{CN}$) should be reflected at the typing level as well. We can not, however, simply introduce a Src and Dst pendant at the type level. The reason for this is that a single connection type may provide connections between instances of different combinations of types. For example, both a *Zip*-file and a *Tar*-file may have a *Payload* (connector type!) consisting of *Pdf*-files. In one case, the connector type *Payload* connects between *Zip* and *Pdf* and in the other case between *TAR* and *Pdf*. As a result, there is no functional dependency (as e.g. created by the Src and Dst functions) between *connector types* and the underlying types that are connected.

The connections between elements of different types that can be made by a connector type, are therefore formally represented by the *relation*:

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

If $t_1 \stackrel{t}{\rightarrow} t_2$, then the intuition is that complex type t_1 has, via connector type t, at its *base* the type t_2. Note that not all types of resources, such as Zip-*file* and *Multi-part E-mail*, will have a *Payload*. For example, in the case of a type such as *Postal address* it does not make sense to attach it to a connector of type *Payload*. This kind of restriction can also be reflected at the typing level using the above relationship. The actual enforcement of this behavior on the instances, will be enforced by one of the axioms introduced below.

Since *types follow instances*, if $t_1 \stackrel{t}{\rightarrow} t_2$ holds, then there must at least be some instances to make this true, the evidence for this observation:

Axiom 11 (Soundness of $\_ \stackrel{\cdot}{\rightarrow} \_$) $t_1 \stackrel{t}{\rightarrow} t_2 \implies \exists_{e_1 \in \pi(t_1), a \in \pi(t), e_2 \in \pi(t_2)} [e_1 \stackrel{a}{\rightsquigarrow} e_2]$

Note that even though we already enforced total typing (Axiom 7), the above axiom is still needed. Due to total typing, if $t_t \stackrel{t}{\rightarrow} t_2$ then $\pi(t_1)$, $\pi(t)$ and $\pi(t_2)$ are all non-empty. There is however, no way to enforce there to be an instance of $\pi(t)$ that actually bridges (provide evidence) between the populations of t_1 and t_2.

To illustrate why the above axiom is needed, let $t_1 = \mathit{Zip}$, $t = \mathit{Payload}$ and $t_2 = \mathit{File}$, then $t_1 \stackrel{t}{\rightarrow} t_2$ represents the fact that *Zip*-files have a payload consisting of files. Axiom 11 states, then, that at least one *Zip* file exists that has at least one file in its payload.

Conversely, if a connector a exists such that $e_1 \stackrel{a}{\rightsquigarrow} e_2$, then this must be reflected at the typing level. In other words, we know that the types of these instances behave as follows:

Axiom 12 (Completeness of $\_ \stackrel{\cdot}{\rightarrow} \_$) $e_1 \stackrel{a}{\rightsquigarrow} e_2 \implies \exists_{t_1 \in \tau(e_1), t \in \tau(a), t_2 \in \tau(e_2)} [t_1 \stackrel{t}{\rightarrow} t_2]$

Figure 2 illustrates this situation. For any given connector (e.g. a or b) there must be instances attached to this connector such that both the connector and the elements it connects have the proper types. A direct result of Axiom 11, is that Lemma 1 can be translated to types as well. Simply put, if the types t_1 and t_2 are connected via a connection type t, then they can not be connection types themselves:

Corollary 2 (No connector nesting) $t_1 \stackrel{t}{\rightarrow} t_2 \implies \{t_1, t_2\} \cap \mathcal{CN}_\tau = \emptyset$

For a given type, the set of connector types that are attached to it, can be determined by:

$$\mathsf{Conn}(t_1) \triangleq \left\{ t \mid \exists_{t_2} [t_1 \stackrel{t}{\rightarrow} t_2] \right\}$$

To show the proper behavior of this relation we show that if two types have the same outbound connections then this must be reflected by the Conn relation. More formally:

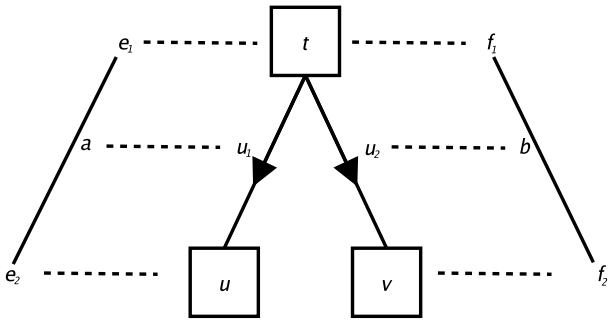


Figure 2: Behavior of connectors and their types

Lemma 4 $\forall_{u,t}\,[s_1 \xrightarrow{u} t \Leftrightarrow s_2 \xrightarrow{u} t] \implies \mathsf{Conn}(s_1) = \mathsf{Conn}(s_2)$

Proof:

Let us suppose that $\forall_{u,t}\,[s_1 \xrightarrow{u} t \Leftrightarrow s_2 \xrightarrow{u} t]$, then:

$\subseteq$ Let $u \in \mathsf{Conn}(s_1)$. From the definition of Conn it follows that there are u, t such that $s_1 \xrightarrow{u} t$. Under the assumption made it follows that $s_2 \xrightarrow{u} t$. From the definition of Conn it follows that $u \in \mathsf{Conn}(s_2)$

$\supseteq$ Due to the symmetry of $\Leftrightarrow$, this follows from the above.

Therefore $\mathsf{Conn}(s_1) = \mathsf{Conn}(s_2)$

□

4.3 Complex data resources

As mentioned before, some data resources can be regarded as being composed of other resource elements. In other words, some 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.

At the instance level, we already introduced a special class of connections, the accessors ($\mathcal{AC}$), to signify that a specific data resource should be regarded as being *composed* of other elements (Section 3.1). A special class of connection types are therefore the *accessor types* $\mathcal{AC}_\tau$. A complex type is a type that has an (outgoing) accessor type associated to it. The set of accessor types that are indeed associated to a type is defined as:

$$\mathsf{Act}(t) \triangleq \mathsf{Conn}(t) \cap \mathcal{AC}_\tau$$

Using the definition of Act we can define the set of complex types to be:

$$\mathcal{TP}_c \triangleq \{t \mid \mathsf{Act}(t) \neq \emptyset\}$$

Since $\mathcal{AC}_\tau \subseteq \mathcal{CN}_\tau$ we can show that the $\_ \xrightarrow{\_} \_$ must behave properly for accessor types as well; similar to what we showed in Lemma 4.

Corollary 3 $\forall_{u,t}\,[s_1 \xrightarrow{u} t \Leftrightarrow s_2 \xrightarrow{u} t] \implies \mathsf{Act}(s_1) = \mathsf{Act}(s_2)$

Thus far we have not associated any semantics to accessors and complex types in terms of their influence on a population in resource space. Complex instances are constructed by means of other instances. Instances of complex types should indeed be constructed by means of other instances. In other words, if an instance has a complex type then it must have at least one accessor with an instance at its base:

Axiom 13 (Complex instances and accessors) $e_1 \in \pi(\mathcal{TP}_c) \implies \exists_{a \in \mathcal{AC}, e_2} [e_1 \overset{a}{\rightsquigarrow} e_2]$

The definition of Act can be overloaded to include the instance level as follows:

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

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.

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*. 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:

$$\begin{aligned}
\pi(\mathcal{DR}) &= \{\texttt{x.zip}, \texttt{a.doc}, \texttt{b.ps}, \texttt{c.pdf}\} \\
\pi(\mathcal{DR}_\tau) &= \{\mathit{Zip}, \mathit{Doc}, \mathit{PostScript}, \mathit{Pdf}, \mathit{file}\} \\
\pi(\mathcal{DV}) &= \{\text{“some comment”}, \text{“secret”}\} \\
\pi(\mathcal{DV}_\tau) &= \{\mathit{String}\} \\
\pi(\mathcal{CN}) &= \{a_1, a_2, a_3, a_4, a_5\} \\
\pi(\mathcal{CN}_\tau) &= \{\mathit{Payload}, \mathit{Comment}, \mathit{Password}\} \\
\pi(\mathcal{AT}) &= \{a_4, a_5\} \\
\pi(\mathcal{AT}_\tau) &= \{\mathit{Comment}, \mathit{Password}\}
\end{aligned}$$

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

Figure 3 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.

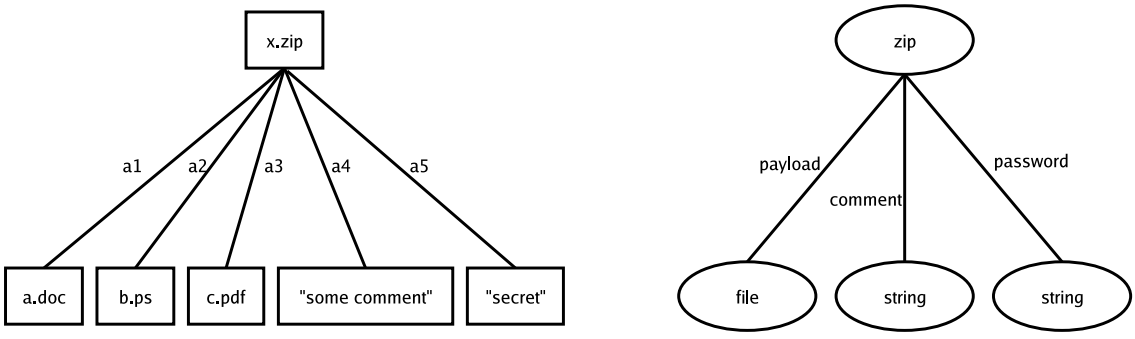


Figure 3: Accessors and typing

[2]The notion of subtyping is introduced in Section 4.4.

4.4 Subtyping

In many modeling languages (such as for example *Object Role Modeling, Entity Relationship Modeling and* UML) a subtyping mechanism is used to denote an *is-a* relation between sets of elements (See e.g. (Halpin, 1995; Chen, 1976; Booch et al., 1999)). For example, the statement "each woman *is a* person" has the same connotation as the statement "woman is subtype of person".

We assume the existence of subtyping for types in our model. Let $\underline{\mathsf{SubOf}} \subseteq \mathcal{TP} \times \mathcal{TP}$ therefore define a subtyping relationship, where $s \,\underline{\mathsf{SubOf}}\, t$ indicates that type s is a subtype of, or equal to type t. Conversely, let $s \,\mathsf{SubOf}\, t$ denote that s is a subtype of t. The mapping to sets of elements is given by the following:

$$\begin{array}{lcl} s \,\mathsf{SubOf}\, t & \triangleq & \pi(s) \subset \pi(t) \\ s \,\underline{\mathsf{SubOf}}\, t & \triangleq & \pi(s) \subseteq \pi(t) \end{array}$$

From these definitions we can prove that:

Corollary 4 $s \,\mathsf{SubOf}\, t \implies s \,\underline{\mathsf{SubOf}}\, t \wedge \neg t \,\underline{\mathsf{SubOf}}\, s$

Furthermore, we know that SubOf is transitive, irreflexive and asymmetric and that $\underline{\mathsf{SubOf}}$ is transitive, reflexive and antisymmetric:

Lemma 5 SubOf is transitive, irreflexive and asymmetric

Lemma 6 $\underline{\mathsf{SubOf}}$ is transitive, reflexive and anti-symmetric

To illustrate what happens with connections (connection types) in case of subtyping consider the following situation (Figure 4): $s \,\mathsf{SubOf}\, t$, $\mathsf{Conn}(t) = \{u\}$ and $\mathsf{Conn}(s) = \{u, v\}$.

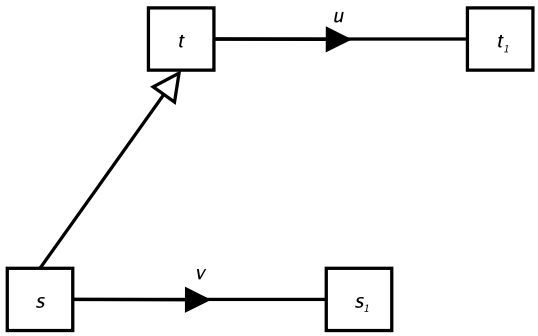


Figure 4: Illustrating the behavior of accessors in case of subtyping

- Suppose we know that $e_1 \overset{a}{\rightsquigarrow} e_2$ such that $e_1 \in \pi(s) \wedge a \in \pi(v) \wedge e_2 \in \pi(s_1)$. Because of subtyping we know that $e_1 \in \pi(t)$. Therefore, at the typing level we know that e_1 *may* have a connector with type u, leading to an instance of type t_1. Without evidence for this at the instance level, we only know that $s \xrightarrow{v} s_1$.
- Suppose we know that $f_1 \overset{b}{\rightsquigarrow} f_2$ and assume that $f_1 \in \pi(s) \wedge b \in \pi(u) \wedge f_2 \in \pi(t_1)$. Based on this information we know that $\exists_{c,f_3} [f_1 \overset{c}{\rightsquigarrow} f_3]$ such that $c \in \pi(v) \wedge f_3 \in \pi(s_1)$, otherwise there would be no proof for $f_1 \in \pi(s)$! This follows from the definition of $\mathcal{CN}$ and Axiom 11. From the definition of $\mathcal{CN}$ it follows that $s \xrightarrow{v} s_1$ and Axiom 11 states that there must be at least one set of instances to provide evidence for this observation at the typing level. We can now conclude that $s \xrightarrow{u} t_1 \wedge s \xrightarrow{v} s_1$.

- Suppose we know that $g_1 \overset{c}{\rightsquigarrow} g_2$ such that $g_1 \in \pi(t) \wedge c \in \pi(u) \wedge g_2 \in \pi(t_1)$. We may already conclude that $t \xrightarrow{u} t_1$. If we find evidence for $g_1 \overset{d}{\rightsquigarrow} g_3$ such that $d \in \pi(v) \wedge g_3 \in \pi(s_1)$ then we have $g_1 \in \pi(s)$ and may also conclude that $s \xrightarrow{v} s_1$.

This also has consequences for applications that work on complex instances. Consider the situation where t is the *Zip*-type and s extends the standard *Zip* with a password. Take an $e \in \pi(s)$ and consider it from the perspective of type t (this is allowed since $s \,\mathsf{SubOf}\, t$ and thus $\pi(s) \subset \pi(t)$. Similar to what we know from *interfaces* in object orientation (see e.g. (Booch et al., 1999)), the interface of t will simply ignore the additional specificities of the interface of s. In the real world this means that applications may fail if they implement the t-interface and receive an instance built according to the s-interface.

These insights lead to the following: Firstly, connector types and their respective bases are inherited across a subtyping relation. That is, subtypes inherit the connector types associated to their super types (but not vice versa!):

Axiom 14 (Completeness of connections) $s \,\mathsf{SubOf}\, t \wedge t \xrightarrow{u} v \implies s \xrightarrow{u} v$

It follows immediately that:

Corollary 5

- $s \,\mathsf{SubOf}\, t \implies \mathsf{Conn}(t) \subseteq \mathsf{Conn}(s)$
- $s \,\mathsf{SubOf}\, t \implies \mathsf{Act}(t) \subseteq \mathsf{Act}(s)$
- $s \,\mathsf{SubOf}\, t \wedge t \in \mathcal{TP}_c \implies s \in \mathcal{TP}_c$

Following the same line of reasoning the base of connector types must also be complete:

Axiom 15 (Completeness of base) $s \xrightarrow{u} t_1 \wedge t_2 \,\mathsf{SubOf}\, t_1 \implies s \xrightarrow{u} t_2$

For the remainder of the discussion we need to introduce the notion of *type relatedness*. We define that two types are related if their populations overlap:

$$s \sim t \triangleq \pi(s) \cap \pi(t) \neq \emptyset$$

We already discussed subtyping which is one form of type relatedness. From e.g. LISA-D (Hofstede et al., 1993) we know that other forms of type relatedness may exist as well. For example: assume a type (e.g. *Zip*) has two distinct subtypes, each with its own subtype defining rule (e.g. *Zip* with password and *Zip* with comment). It may be possible to have an instance that is in both the subtype (i.e. a specific *Zip* file may have both a password and a comment associated to it).

Using this intuition we can more easily specify the soundness of the base of accessor types:

Axiom 16 (Soundness of Base) $s \xrightarrow{u} t_1 \wedge s \xrightarrow{u} t_2 \implies t_1 \sim t_2$

This allows us to prove that if a subtype and a supertype share an accessor then their bases must at least be type related:

Lemma 7 $s_1 \,\mathsf{SubOf}\, s_2 \wedge s_1 \xrightarrow{u} t_1 \wedge s_2 \xrightarrow{u} t_2 \implies t_1 \sim t_2$

Proof:

Let s_1 SubOf $s_2 \wedge s_1 \xrightarrow{u} t_1 \wedge s_2 \xrightarrow{u} t_2$. From Axiom 14 we know that $s_1 \xrightarrow{u} t_2 \wedge s_1 \xrightarrow{u} t_1$. From Axiom 16 we can now conclude that $t_1 \sim t_2$

□

Note that we did not include a "soundness of complex" axiom. The following example illustrates why such an axiom is not desirable:

- Let $s =$ *Zip*, $t =$*E-mail*, $u =$ *Html* and $v =$ *Payload* ($v \in \mathcal{AC}_\tau$)
- Furthermore, let $s \xrightarrow{v} u$ and $t \xrightarrow{v} u$

In other words, both *Zip* and *E-mail* have an accessor type *Payload* that offers access to a *Html* base. If we were to introduce a soundness axiom for complex we would be forced to conclude that *Zip* and *E-mail* are type related, which is undesirable.

4.5 Typed resource space

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

$$\Sigma^\tau \triangleq \langle \Sigma, \mathcal{TP}, \mathsf{HasType} \rangle$$

The disctinction between resource*space* and resource*base* is important with respect to the observation that, in our model, types follow instances. Without this explicit distinction, if the last instance of a type would be removed then the type would cease to exist. This can be particularly inconvenient in case an application depends on the existence of this type. The world of transformations (as described in Section 5) is a typical example since, as we will show, transformations can be describe in terms of input types and output types.

Similarly to a *resource base*, a *typed resource base* corresponds to those resources *and their types*, that are available at some moment in time. This set spans a sub-space of a typed resource space, consisting of a resource base and its typing. Given a typed resource space Σ^τ, a typed resource base therefore corresponds to a substructure of Σ^τ that on itself forms a typed resource (sub)space as well. More formally, a typed resource base Σ^τ_B is a sub-structure of Σ^τ:

$$\Sigma^\tau_B \triangleq \langle \Sigma_B, \mathcal{TP}_B, \mathsf{HasType}_B \rangle$$

such that:

- the Σ_B is a resource base in resource space Σ,
- $\mathcal{TP}_B \subseteq \mathcal{TP}$, $\mathsf{HasType}_B \subseteq \mathsf{HasType}$
- all the above discussed axioms apply to the sub-structure.

5 Transformations

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

- Suppose we have an image in *Eps* file that we want to view. Unfortunately we don not 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 exist to generate abstracts of these documents.

In other words, transformations help us to have a more flexible view on the information landscape.

In the context of databases, one can distinguish between an extensional database and intentional database (Ullman, 1989; Date, 2003). 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. Translated to our model this means that:

- The resource base is extensional, it consists all the resources that exist in a certain situation
- By means of transformations, new instances (which already existed in resource space) may be arrived at.

Simply put, 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. In Section 5.3 we will present an example. Section 6 discusses the effects of transformations.

5.1 Basic Properties

Before we begin our discussion on transformations, 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 of 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, nor does it present 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 specify what this transformation actually *does*. For any transformation $T \in \mathcal{TR}$, the semantics of T 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 $\overrightarrow{T} \triangleq \mathsf{SEM}(T), T \in \mathcal{TR}$. In logic $M \vDash A$ is often used to denote the fact that M is a model for A. Here we use the symbol to denote something similar: $i \models d$ denotes the fact that data resource d is associated to information resource i via some representation:

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

Simply put: since we consider data resources to be an implementation for an information resource, we can also consider an information resource to be a model for a dataresource.

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: $\mathsf{Input}, \mathsf{Output} : \mathcal{TR} \rightarrow \mathcal{DR}_\tau$. In other words, each transformation has an input type and an output type, which must *at least* exist in resource space[3].

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)

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

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:

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

Simply put this means the following. If a transformation T is applied to a set of data resources E then the transformation will only transform the resources for which it is defined. In other words, $\overrightarrow{T}(E)$ is define as follows

- all data resources in E whose types are in $\mathsf{Input}(T)$ are transformed
- all data resources in E whose types are not in $\mathsf{Input}(T)$ (i.e. the remaining data resources) are left untouched.

[3] We will return to this discussion in Section 6 where we discuss the effects of transformations.

Another property of transformations is the fact that they are closed under composition:

Axiom 19 (Closed under composition)

$$t_1 \xrightarrow{T_1} t_2 \wedge t_2 \xrightarrow{T_2} t_3 \implies \exists_{T_3} [t_1 \xrightarrow{T_3} t_3 \wedge \overrightarrow{T}_3 = \overrightarrow{T}_1 \circ \overrightarrow{T}_2]$$

This axiom states that if the output type of a transformation T_1 equals the input type of another transformation T_2 then the semantics of these two transformations can be combined by applying T_2 after T_1 (resulting in a T_3). Note that there may also be a T_4 such that $e \xrightarrow{T_4} g$ with completely different semantics! The following example illustrates this:

- Let t_1 be the *LaTeX* type, t_2 denotes the *Dvi* type and t_3 denotes the *Pdf* type.
- Let T_1 be the transformation from *LaTeX* to *Dvi*, T_2 be the transformation from *Dvi* to *Pdf* and T_4 be the transformation directly from *LaTeX* to *Pdf*
- Since $\mathsf{Output}(T_1) = \mathsf{Input}(T_2)$ we know from Axiom 19 that there is a $t_1 \xrightarrow{T_3} t_2$ which combines the semantics of T_1 and T_2.
- Note that $\mathsf{Input}(T_3) = \mathsf{Input}(T_4)$, $\mathsf{Output}(T_3) = \mathsf{Output}(T_4)$ but not $\overrightarrow{T}_3 = \overrightarrow{T}_4$.

5.2 Complex Transformations

In the previous subsection 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. A transformation is complex if:

- it operates on (instances of) a complex type
- has an effect on the instances that were used to construct the complex instance (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.

We use $\varrho\_$ to denote such transformation. Let e be an instance with a complex type and $u \in \mathsf{Act}(e)$, then $\varrho_u(e)$ denotes the transformation where the accessor u as well as the instances at its base are removed:

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

In the above definition we have used the following shorthand notation: Let e be a complex instance and t an accessor type, then $e \ltimes t$ denotes the instances at the base of the accessor from e with type t:

$$e \ltimes t \triangleq \left\{ d \mid e \overset{a}{\rightsquigarrow} d \wedge a \in \pi(t) \right\}$$

The intuition behind this shorthand is that $e \overset{a}{\rightsquigarrow} d$ retrieves all data elements that are used in constructing complex a data resource e 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)

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

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, $a \in \mathsf{Act}(\tau(e))$ and $T \in \mathcal{TR}$. Then $\alpha_{a:T}(e)$ denotes the transformation where the instances at the base of e's accessor a are transformed with transformation T:

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

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)

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

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. Let t_1 denote the complex type *E-mail in* UTF-8 *encoding* and let t_2 denote the complex type *E-mail in* UTF-16 *encoding*. Instances of these types have accessors leading to the payload of the E-mail in *UTF-8* and *UTF-16* encoding respectively. 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 $t_1 \xrightarrow{T'} t_2$.

So far we have shown different kinds of transformations between (instances of) types. Even more, we have introduced the notion of subtyping. In many programming languages (some) type casts are executed implicitly by the compiler. For example, the integer 13 is automatically

cast to the float 13.0 when necessary. We propose to explicitly name and call these type casts. As such, type-casting transformations are considered to be first class members of $\mathcal{TR}$. Using these type casts we can *lift* an instance to its supertype; i.e. consider an instance using the interface of its supertype. The point is that the data resource itself does not change. More formally, if $e \in \pi(s)$ then:

$$\iota_s(e) \triangleq e$$

These transformations are applicable for all instances of all types, as long as the types have a supertype:

Axiom 22 (Existence of ι)

$$\textbf{if } s \,\mathsf{SubOf}\, t \textbf{ then } \exists_{T \in \mathcal{TR}} [s \xrightarrow{T} t \wedge \overrightarrow{T} = \iota_s]$$

5.3 Example

In this section we present an example that relies on Axioms 19, 20 and 21. That is, we show a situation where two complex transformations (a *removing* transformation and a *deep* transformation) are concatenated. 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:

$$\begin{aligned}
\tau(\texttt{backup.zip}) &= \mathit{Zip} \\
\mathsf{Act}(\texttt{backup.zip}) &= \{\mathit{Payload}, \mathit{Comment}, \mathit{Password}\} \\
\texttt{backup.zip} \ltimes \mathit{Payload} &= \{\texttt{report.doc}, \texttt{letter.doc}\} \\
\texttt{backup.zip} \ltimes \mathit{Comment} &= \text{“}\texttt{backup}\text{”} \\
\texttt{backup.zip} \ltimes \mathit{Password} &= \text{“}\texttt{secret}\text{”}
\end{aligned}$$

Now, let T_1 be a transformation with $\mathsf{Input}(T) = \mathit{Doc}$ and $\mathsf{Output}(T) = \mathit{Pdf}$. Then, $\alpha_{\mathit{Payload}:T_1}$ is a transformation that transforms the documents in the payload of any *Zip* archive to *Pdf*. Let $\varrho_{\mathit{Password}}$ 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:

$$\begin{aligned}
&\overrightarrow{T} = \alpha_{\mathit{Payload}:T_1} \circ \varrho_{\mathit{Password}} \\
&\overrightarrow{T}(\texttt{backup.zip}) = \texttt{new.zip}
\end{aligned}$$

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

$$\begin{aligned}
\tau(\texttt{new.zip}) &= \mathit{Zip} \\
\mathsf{Act}(\texttt{new.zip}) &= \{\mathit{Payload}, \mathit{Comment}\} \\
\texttt{new.zip} \ltimes \mathit{Payload} &= \{\texttt{report.pdf}, \texttt{letter.pdf}\} \\
\texttt{new.zip} \ltimes \mathit{Comment} &= \text{“}\texttt{backup}\text{”}
\end{aligned}$$

6 Effects of transformations

Transformations can be applied to data resources. The key point about a transformation is that the data resource is somehow changed. For example, the data resource type or its resolution is

altered. In this section we study these effects. This section builds on previous work presented in (Gils et al., 2003a; Gils et al., 2005). The study of specific properties of transformations has been conducted in many other contexts as well. As examples we mention properties concerning clustering (e.g.(Song et al., 2002)) and performance (e.g. (Rahayu et al., 2001)).

The remainder of this section is organized as follows. In Section 6.1 we will start out by defining several classes of effects that a transformation may have and how they are measured in resource space. In Section 6.4 we drill down to resource base. Finally, in Section 7 we show how this framework for transformations can be used to study transformations in practice, i.e. for a real resource base like the Web.

6.1 Classes of effects

The above example shows that a transformation may remove a certain property of a data resource, where a property may be any predicate over Σ^τ . This is not the only possible effect a transformation may have, though. We discern the following 4 classes of effects, when a single instance is concerned:

1. A transformation is *neutral* with regard to some property. For example, a transformation that transforms *Html* files to *Pdf* may be neutral with regard to the price attribute.
2. A transformation may *alter* a certain property. For example, a transformation that lowers the resolution of an image may lower its price too.
3. A transformation may *remove* a certain property. For example, a transformation that transforms *Html* files to *Ascii* may remove all hyperlinks.
4. A transformation may *introduce* a certain property. For example, a transformation may add a password to a *Zip* file.

This is illustrated in Figure 6.1. The left circle depicts the input type of a transformation and the right circle depicts its output type. Some properties are unchanged, which is depicted by the intersection of both circles. Also, some properties are changed, depicted by the *from* and *to* sections of both circles. Finally, properties may be removed or introduced. Let $\mathcal{EC}_i = \{neutral, alter, remove, introduce\}$ be the set of effect classes. At the typing level, the rea-

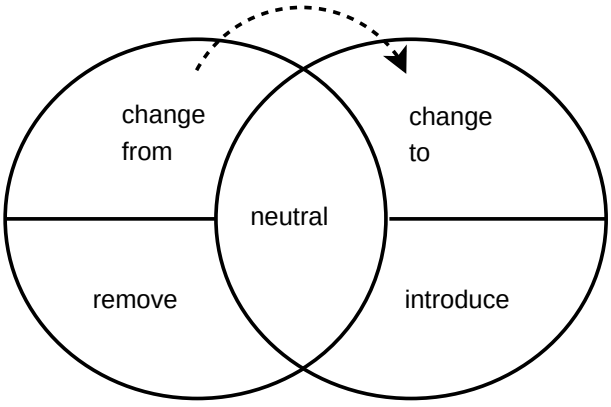


Figure 5: Effects of transformations

soning is slightly more complex. In this case, the effect of a transformation on instances of a

certain data resource type with a certain property is studied. In other words, the conclusion that a transformation has a certain effect at the *instance* level must be generalized to the *type* level. In Section 6.3 we will see how this is done and show that at the type level, the effect class is $\mathcal{EC}_t = \{\textit{neutral}, \textit{remove}, \textit{introduce}, \textit{hybride}\}$.

6.2 Properties

In this section we study the effect that a transformation may have on instances which (may or may not) have certain properties. We consider any binary predicate over Σ^τ to be a property, where the first argument of the predicate concerns a data resource. For example, r *Has hyperlinks* is a unary property, while r *Is based on* s is a binary property. In general, properties can have any arity higher than one. To model this formally, we will presume a property to be represented in general as $\varphi(e, W)$ where e is a data resource and W is a sequence of (zero or more) resource space elements or types. The language for properties is denoted by Φ. If $\varphi(e, W)$ then resource e is said to have property φ with resources W. Note that W must be a sequence, as opposed to an unordered set, since φ resembles the notion of a *predicate*. The ordering of the elements in W matters! An example of a property φ would be the property named φ_p such that:

$$\varphi_p(e, [x, y]) \triangleq \exists_{r \in \mathcal{RP}, a \in \mathcal{AT}} [\mathsf{DRes}(r) = e \wedge r \ \mathsf{HasType} \ x \wedge e \overset{a}{\rightsquigarrow} y \wedge y \ \mathsf{HasType} \ \textit{Encoding}]$$

Given a resource base and an instande (data resource) e the expression $\varphi_p(e, [\textit{Keyword-list}, \textit{UTF-8}])$ may evaluate to either true or false. To be able to evaluate this truth-assignment of φ for a given instance e we use the function $\Gamma : \mathcal{DR} \times \Phi \rightarrow \wp(\mathcal{RE}^*)$:

$$\Gamma(e, \varphi) \triangleq \{W \mid \varphi(e, W)\}$$

In other words, $\Gamma(e, \varphi)$ returns the set of sequences of resource space elements W for which φ is true, given a data resource e. If $\Gamma(e, \varphi)$ returns $\{[\emptyset]\}$ then $\varphi(e, W)$ apparently does not hold. Continuing the previous example where φ_p was defined, if

$$\Gamma(e, \varphi_p) \supset \{[\textit{Keyword-list}, \textit{UTF-8}]\}$$

then apparently data resource e indeed is a *Keyword-list* in the *UTF-8* encoding. Two more examples illustrate how properties and their truth-assignments can be used in practice:

- Let $\varphi_t(e, [t]) \triangleq e \ \mathsf{HasType} \ t$ denote the typing property. For an instance e with $\tau(e) = \{t_1, t_2\}$ we have $\Gamma(e, \varphi_t) = \{[t_1], [t_2]\}$.
- Let $\varphi_t(e, [\,]) \triangleq e \ \mathsf{HasType}$ *Pdf* denote the property *is of type Pdf*. For an instance e with $\textit{Pdf} \in \tau(e)$ we have $\Gamma(e, \varphi_t) = \{[\,]\}$ denoting that e indeed has this property. For an instance f with $\textit{Pdf} \notin \tau(f)$ we have $\Gamma(f, \varphi_t) = \{[\emptyset]\}$ denoting that f does not have this property.

6.3 Effects: instance and type level

Using this Γ relation, it is straightforward to find out the effect class of a transformation $T \in \mathcal{TR}$ with regard to a specific φ. Let $\mathsf{Effect} : \mathcal{TR} \times \mathcal{DR} \times \Phi \rightarrow \mathcal{EC}_i$ be the function that finds the effect class of a transformation $T \in \mathcal{TR}$ on a data resource $e \in \mathcal{DR}$ with regard to a property φ. This can be achieved by comparing the sets of objects that make Γ true for both the input and the output instance of the transformation:

- If these sets are equal, then for this (input) instance, the transformation is neutral with regard to this specific φ. For example, in case of the typeping property φ_t defined above and both the input instance and output instance have the same types then the transformation is neutral with regard to data resource types.
- If the input set is a subset of the output set, then the transformation, for this (input) instance, apparently is introducing with regard to this φ. In case of this means that the input type of the transformation is a subtype of its output type (a direct consequence of the definition of SubOf).
- Similarly, if the output set is a subset of the input set, then the transformation, for this (input) instance is removing with regard to this φ. In case of the typing property φ_t, this means that the output type of the transformation is a subtype of its input type.
- If neither of the above applies then, for this (input) instance, the transformation is said to be altering with regard to this φ. We describe this as follows: let e be the input instance of the transformation and $\overrightarrow{T}(e) = f$ be the output instance of the transformation T. In case of the typing property φ_t this implies the following:
  - The sets $\tau(e)$ and $\tau(f)$ overlap such that $\tau(e) \not\subseteq \tau(f) \wedge \tau(f) \not\subseteq \tau(e)$. For example, e and f do have a supertype in common (both are files) but apart from that they are completely different.
  - The sets $\tau(e)$ and $\tau(f)$ are disjoint. This implies that e and f have no (super)type in common.

Summarizing, the effect of transformation T on data resource e with regard to property φ is the following:

$$
\begin{array}{l}
\mathsf{Effect}(T, e, \varphi) \triangleq \\
\quad \textbf{if} \quad \Gamma(e, \varphi) = \Gamma(\overrightarrow{T}(e), \varphi) \quad \textbf{then} \quad \textit{neutral} \\
\quad \textbf{if} \quad \Gamma(e, \varphi) \subset \Gamma(\overrightarrow{T}(e), \varphi) \quad \textbf{then} \quad \textit{introduce} \\
\quad \textbf{if} \quad \Gamma(e, \varphi) \supset \Gamma(\overrightarrow{T}(e), \varphi) \quad \textbf{then} \quad \textit{remove} \\
\quad \textbf{else} \quad \textit{alter}
\end{array}
$$

A similar line of reasoning explains how the definition of Effect may be generalized to the typing level such that $\mathsf{Effect} : \mathcal{TR} \times \mathcal{DR}_\tau \times \Phi \rightarrow \mathcal{EC}_t$. In this case the effect that a transformation has on a certain property must be analyzed for *all instances* of a certain type.

- If a transformation is neutral with regard to a φ for all instances of a given data resource type then, at the typing level, the transformation is said to be neutral with regard to this specific φ.
- It seems apparent that, at the type level, a transformation is introducing for a given φ if the transformation is introducing for every instance of this type. A simple example shows that this not the case. Consider the property "has relations of type *Hyperlink*" and a transformation T. Furthermore let e be an instances that, indeed, has hyperlinks. Then: $\mathsf{Effect}(T, e, \varphi) =$ *remove*. If the transformed instance $\overrightarrow{T}(e) = f$ is transformed again, using the same transformation then $\mathsf{Effect}(T, f, \varphi) =$ *neutral* since f did not have any hyperlinks in the first place.

  Therefore the rule must be: if a transformation is introduction with regard to a φ for at least one instance and neutral for all others, then at the typing level the transformation is said to be introducing with regard to this specific φ.

- For similar reasons, if a transformation is removing with regard to a φ for at least one instance and neutral for all others, then at the typing level the transformation is said to be removing for this specific φ.
- Again, it may seem that at the typing level a transformation is altering with regard to a property if it is altering for all instances of this type. However, this is not the case. Other situations may occur also, for example: a transformation may be introducing for one instance, and altering for another. This occurs when a transformation sets the version attribute to the value 2.6, regardless of the fact that data resource already had a version attribute. If it did, the transformation is likely to be altering for this property. If it did not, the transformation would be introducing. In this case, we are indecisive about the effect that a transformation has on a certain property.

Summarizing, the effect of a transformation T with regard to a property φ, considered at the *type level* is the following:

$$
\begin{array}{l}
\mathsf{Effect}(T, t, \varphi) \triangleq \\
\quad \textbf{if} \quad \forall_{e \in \pi(t)} \left[\Gamma(e, \varphi) = \Gamma(\overrightarrow{T}(e), \varphi) \right] \quad \textbf{then} \quad \textit{neutral} \\
\quad \textbf{if} \quad \forall_{e \in \pi(t)} \left[\Gamma(e, \varphi) \subseteq \Gamma(\overrightarrow{T}(e), \varphi) \right] \quad \textbf{then} \quad \textit{introduce} \\
\quad \textbf{if} \quad \forall_{e \in \pi(t)} \left[\Gamma(e, \varphi) \supseteq \Gamma(\overrightarrow{T}(e), \varphi) \right] \quad \textbf{then} \quad \textit{remove} \\
\quad \textbf{else} \quad \textit{hybride}
\end{array}
$$

6.4 Transformations, resource space and resource base

In this section we will discuss how the above can be applied to the resource space and resource base. In this context, the key distinction between them is that:

- in the resource base we can observe the effect a transformation has on a specific instance
- in the resource space we can observe the total effect a transformation may have.

We will start at the resource base level and then generalize to the resource base level. Recall that the signature of the typed resource base is given by the following signature:

$$\Sigma_B^\tau \triangleq \langle \mathcal{IR}_B, \mathcal{RP}_B, \mathcal{DR}_B, \mathcal{RL}_B, \mathcal{AT}_B, \mathcal{DV}_B, \mathcal{AC}_B, \mathsf{IRes}_B, \mathsf{DRes}_B, \mathsf{Src}_B, \mathsf{Dst}_B, \mathcal{TP}_B, \mathsf{HasType}_B \rangle$$

and that its population can be represented in terms of this signature. Any given population can be represented in terms of this signature, even the population consisting of a single data resource and its associated properties. For example: A person called John Doe has a webpage in *Html* format. The current version of his webpage is 2.4. Furthermore, this webpage does not have any

links to other sites. The following population exemplifies this:

$$
\begin{array}{ll}
\mathcal{IR} & = \{\text{John Doe}\} \\
\mathcal{RP} & = \{r\} \\
\mathcal{DR} & = \{\texttt{john-doe.html}\} \\
\mathcal{RL} & = \emptyset \\
\mathcal{AT} & = \{a\} \\
\mathcal{DV} & = \{2.4\} \\
\mathcal{AC} & = \emptyset \\
\mathsf{IRes} & = \{\langle r, \text{John Doe}\rangle\} \\
\mathsf{DRes} & = \{\langle r, \texttt{john-doe.html}\rangle\} \\
\mathsf{Src} & = \{\langle a, \texttt{john-doe.html}\rangle\} \\
\mathsf{Dst} & = \{\langle a, 2.4\rangle\} \\
\mathcal{TP} & = \{\text{webpage-of}, \textit{Html}, \text{version}\} \\
\mathsf{HasType} & = \{\langle r, \text{webpage-of}\rangle, \langle \texttt{john-doe.html}, \textit{Html}\rangle, \langle a, \text{version}\rangle\}
\end{array}
$$

The effect that a specific transformation has on *this* particular instance may be measure by transforming the instance and comparing the two signatures. For example, let $T \in \mathcal{TR}$ and $\mathsf{Input}(T) = \textit{Html}, \mathsf{Output}(T) = \textit{Ascii}$ such that $\overrightarrow{T}(\texttt{john-doe.html}) = \texttt{john-doe.txt}$. The following denotes this new data resource and its properties:

$$
\begin{array}{ll}
\mathcal{IR} & = \{\text{John Doe}\} \\
\mathcal{RP} & = \{r\} \\
\mathcal{DR} & = \{\texttt{john-doe.txt}\} \\
\mathcal{RL} & = \emptyset \\
\mathcal{AT} & = \{a\} \\
\mathcal{DV} & = \{2.4\} \\
\mathcal{AC} & = \emptyset \\
\mathsf{IRes} & = \{\langle r, \text{John Doe}\rangle\} \\
\mathsf{DRes} & = \{\langle r, \texttt{john-doe.html}\rangle\} \\
\mathsf{Src} & = \{\langle a, \texttt{john-doe.html}\rangle\} \\
\mathsf{Dst} & = \{\langle a, 2.4\rangle\} \\
\mathcal{TP} & = \{\textit{Document-about}, \textit{Ascii}, \textit{Version}\} \\
\mathsf{HasType} & = \{\langle r, \textit{Document-about}\rangle, \langle \texttt{john-doe.txt}, \textit{Ascii}\rangle, \langle a, \textit{Version}\rangle\}
\end{array}
$$

The effects of the actual transformation can now be studied by comparing these two signatures. More specifically, the effects can be found by computing $\mathsf{Effect}(T, \texttt{john-doe.html}, \varphi)$ for different φ:

- The input instance and output instance have different data resource types. More specifically, we transform an instance from type *Html* to type *Ascii*. For $\varphi_t(e, [t]) \triangleq e\ \mathsf{HasType}\ t$, then $\mathsf{Effect}(T, \texttt{john-doe.html}, \varphi) = \textit{alter}$.
- Conform with Axiom 17, both the input instance and output instance of the transformation are attached to the same information resource; the input instance has representation type *Webpage-of* and the output instance has representation type *Document-about*. The property would be:
$$\varphi(e, [\,]) = \exists_{r \in \mathcal{RP}} [\mathsf{DRes}(r) = e \wedge r\ \mathsf{HasType}\ \textit{Document-about}]$$
Which leads to: $\mathsf{Effect}(T, \texttt{john-doe.html}, \varphi) = \textit{alter}$.
- Both the input instance and output instance have an attribute of type *Version*. The attributed value is 2.4 in both cases. Hence, the property would be:
$$\varphi(e, [v]) = \exists_{a \in \mathcal{AT}} [a\ \mathsf{HasType}\ \textit{version} \wedge \mathsf{Src}(a) = e \wedge \mathsf{Dst}(e) = v]$$
Since the truth-assignment for both instances is the same (i.e. $\Gamma(\texttt{john-doe.html}, \varphi) = \Gamma(\texttt{john-doe.txt}, \varphi) = \{[2.4]\}$) we know that $\mathsf{Effect}(T, \texttt{john-doe.html}, \varphi) = \textit{neutral}$

Instances in the resource *space*, by definition, reflect the full extent of a given (data resource type) in the sense that every possible property of a type has an instance. By locating the instance in resource space that has every possible property, applying the transformation on this instance and comparing the two signatures the total effect of a signature.

7 Towards implementation: transformation selection

When defining a transformational approach, we need to consider formal as well as practical aspects (see e.g. (Polo et al., 2002) for a database-oriented approach). In this section we consider transformation selection, in order to illustrate the application of our transformations in a practical context.

In previous sections we have described a model for resource space as well as a framework for transformations. These transformations may have effects on properties of the resources. In Section 7.1 we will discuss how these effects can be learned in a real application. In Section 7.2 we will focus on the basic patterns that we observe when looking at transformations, whereas patterns for choosing between several transformation paths is the focus of Section 7.3. Last but not least, in Section 7.4 we will discuss the foundations for a transformation selection algorithm.

7.1 Learning the effects of a transformation

In resource base, the effects of transformations and properties of types can be *learned* in the following manner:

- Initially, it is assumed that a transformation is *neutral* with regard to every property, similar to the notion of being innocent until proven otherwise.
- After a transformation is performed on an instance (from resource base!), the properties of the input instance and the output instance are compared to study the effects:
  - We may discover a new property of a type. For example: before the transformation was executed we did not have a single instance of the *Pdf* type with a price but after the transformation we do. This implies that the next time we compose a transformation (See Axiom 19) involving the *Pdf*-type we can use this additional knowledge.
  - We may discover that a transformation is not neutral with regard to some property; i.e. it may alter, remove or add certain properties. For example, we may learn that a transformation from *Html* to *PostScript* removes all hyperlinks.

Being able to reason about transformations at this level is particularly convenient when composing transformations. In any practical situation one would want to know that it is possible to perform a certain transformation rather than knowing it is composed of several others. Being able to compose transformations automatically, based on the knowledge about its effects, is therefore an important feature.

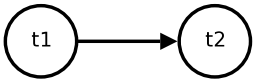
Recall that initially, we assume that transformations are neutral with regard to all properties. This may not be the case and therefore it may not be possible to judge with 100% certainty what a transformation will do and if a certain (composed / complex) transformation is possible. These effects will have to be learned as time progresses, thus improving the quality of the transformation framework.

7.2 Basic patterns of transformations

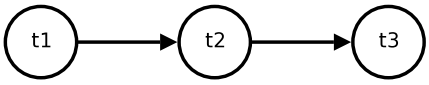
Transformations have an input type and an output type. Formally, if $t_1, t_2 \in \mathcal{TP}$ and $T \in \mathcal{TR}$ then $t_1 \xrightarrow{T} t_2$ denotes the fact that with transformation T, instances of type t_1 can be transformed into instances of type t_2. Furthermore, the transformation may have some effect on a property φ. For example: let $\varphi(e, [t]) \triangleq e$ HasType t be the type property of a data resource and $e \in \mathcal{DR}$ be a data resource. Then $\mathsf{Effect}(T, e\varphi) = \textit{neutral}$ denotes that transformation T is neutral with regard to the HasType property for this specific instance. In Section 6.3 we have described how these effects can be determined at both the instance level and the type level.

Given the above, the two basic patterns are:

- A transformation from a type to a (different) type: $t_1 \xrightarrow{T} t_2$. For example: a transformation *from Html to Pdf*. A special case would be a a transformation from a type to the same type: $t_1 \xrightarrow{T} t_1$. For example: a transformation *from Html to Html* that removes all hyperlinks. This pattern is illustrated in the following diagram:



- A transformation from a type to different type, combined with a transformation to (yet) another type: $t_1 \xrightarrow{T_1} t_2 \wedge t_2 \xrightarrow{T_2} t_3 \wedge T_3 = T_1 \circ T_2$. For example, a transformation *from Html via PostScript to Pdf*. A special case would be a transformation from a type to a different type, combined with a transformation back to the first type: $t_1 \xrightarrow{T_1} t_2 \wedge t_2 \xrightarrow{T_2} t_1 \wedge T_3 = T_1 \circ T_2$. For example, a transformation *from Xml via Html to Html*. Such a transformation may be used to achieve a certain effect on a property. This pattern is illustrated in the following diagram:



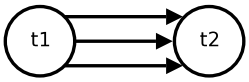
Using these two patterns a directed graph of transformations can be created: the nodes are data resource types and the arcs are transformations between these types.

7.3 Patterns for transformation selection

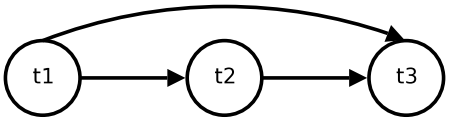
The transformation selection problem is to find the right path from an input type to an output type, which has to do with the properties that must or must not hold after the transformation has been executed. In this section we will present the selection patterns that may occur in a transformation graph.

Selection implies that a choice can be made. In other words, there must be at least two paths from a certain node to another. We observe the following selection patterns:

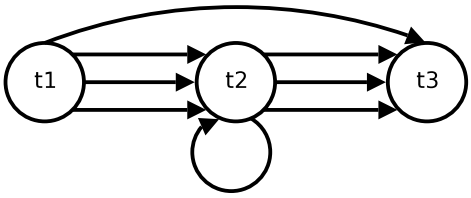
- Two or more different paths exist from one type to another. In this case there are one ore more paths from the input type to the output type. Each path may have different characteristics in terms of what it does to properties. This pattern is illustrated in the following diagram:



- Another option is that there exist both a direct and an indirect path from a given input to an output type. This pattern is illustrated in the following diagram:



- These patterns can, of course, be combined over and over to form even more complex structures. For example the situation where one can choose between many possible paths from the input type via an intermediate type to an output type, as well as a direct transformation from the input type to the output type. This is illustrated in the following diagram:



Using combinations of these patterns over and over results in the (directed) transformation graph. The actual choice is based on properties: which properties must hold after the transformation and which must not hold. The (characteristics of the) algorithm for doing this selection is described in the next section.

7.4 Transformation selection

We have stated before that transformations can be used to improve the aptness of a data resource for users. In this section we discuss the issue of how to decide which transformation does this best. The metric for assessing the quality of a transformation is based on properties. At a high level of abstraction the decision boils down to comparing the properties that a data resource is expected to have after a transformation with the properties desired by the user. In earlier work (see (Gils et al., 2005)) we presented a specific algorithm that took not only the properties into account, but also the length of the transformation path (in the directed graph where data resource types are nodes and transformations are the arcs). In this section we ignore these path-lengths for the time being.

The first step in this decision process is to determine the expected properties of a data resource after it is transformed (possibly by a composed transformation!). This can be achieved by calculating the expected truth assignment for every possible property after each possible transformation on a given data resource.

Recall that the the truth assignment for a property, given a data resource, is a sequence of resource space elements. Hence, the he truth assignment for every possible property is given by the function $\Psi : \mathcal{DR} \rightarrow (\Phi \rightarrow \wp(\mathcal{RE}^*))$. This relation is defined as follows:

$$\Psi(e) \triangleq \{ \langle \varphi, \Gamma(e, \varphi) \rangle \mid \varphi \in \Phi \}$$

The following example illustrates the use of this relation. Let Φ be the language of all properties such that there are exactly two properties φ_1 and φ_2. Furthermore, let e be a data resource of type t. The first property is made true for e by exactly a resource space element w_1 as well as element w_2 and the second property is made true by exactly the sequence of resource space elements $[w_3, w_4]$. Then we know that:

$$\Psi(e) = \{\langle\varphi_1, \{[w_1], [w_2]\}\rangle,\ \langle\varphi_2, \{[w_3, w_4]\}\rangle\}$$

This shorthand notation allows us to easily decide on the effects of a transformation by comparing the truth assignment of the properties of a data resource with the truth assignment of the properties of the transformed data resource. Continuing the previous example, if T_1 is a transformation (with $\mathsf{Input}(T_1) = t$) such that

$$\Psi(\overrightarrow{T}_1(e)) = \{\langle\varphi_1, \{[w_1]\}\rangle,\ \langle\varphi_2, \{[w_3, w_5]\}\rangle\}$$

Then we can conclude that this transformation is *removing* for property φ_1 and *removing* for φ_2. This shorthand, together with the learned effects of transformations (see Section 7.1) provide the machinery for estimating the expected truth assignment for a data resource after it is transformed. For example, if T_2 a transformation (with $\mathsf{Input}(T_2) = t$) for which we know that $\mathsf{Effect}(T_2, t, \varphi_2)$ then we know that the data resource resulting from $\overrightarrow{T}_2(e)$ will not have property φ_2; its truth assignment will be void (i.e. $\Gamma(\overrightarrow{T}_2(e), \varphi_2) = \{[\emptyset]\}$).

The second step is to compare the expected truth assignment of properties for the transformed data resource to the wishes of the user with regard to these properties. Let $\mathcal{U} \subseteq \Phi \times I\!R$ denote a preference assignment (as a real number) with regard to properties.

A positive preference assignment indicates that the property must be present, a negative assignment indicates that the property must not be present and a zero-assignment denotes indifference. For example:

$$\mathcal{U} = \{\langle\varphi_1, 10\rangle, \langle\varphi_2, -5\rangle\}$$

denotes that φ_1 must be present and φ_2 must not be present. Evenmore, the magnitude of the preference assignment that this specific user finds the prences of φ_1 twice as important as the absence of φ_2. This step is completed by comparing the truth assignment of properties for every possible transformation to $\mathcal{U}$. This is illustrated in the following example. Let

- $e, f \in \mathcal{DR}$ be data resources
- *Doc*, *Pdf* $\in \mathcal{DR}_\tau$ be two data resource types
- $r_1, r_2 \in \mathcal{RL}$, $T \in \mathcal{TR}$ such that $e \overset{r_1}{\rightsquigarrow} f$ and $\overrightarrow{T}(e) \overset{r_2}{\rightsquigarrow} f$
- *Hyperlink*, *Refernece* $\in \mathcal{RL}_\tau$ be two relation types
- The language for properties is $\Phi = \{\varphi_t, \varphi_s, \varphi_r\}$ such that
  - $\varphi_t(e, [t]) \triangleq e$ HasType t is the typing property
  - $\varphi_s(e, [\,]) \triangleq e$ HasType *Doc* is a more specific typing property for it tests whether an instance has the specific data resource type *Doc*.
  - $\varphi_r(e, [\,] \triangleq \exists_r [\mathsf{Src}(r) = e \wedge r$ HasType *Hyerlink*$]$ is the property denoting that a resource has outgoing hyperlinks.

Given the truth assignment for properties, as well as a vector of preference assignments $\mathcal{U}$ we can now come to a decision with regard to the *aptness* of e and $\overrightarrow{T}(e)$ for this $\mathcal{U}$. Let:

- $\Psi(e) = \{\langle\varphi_t, \{[Doc]\}\rangle,\ \langle\varphi_s, \{[\,]\}\rangle,\ \langle\varphi_r, \{[\,]\}\rangle\}$
  in other words, e is of type *Doc* and indeed has an outgoing hyperlink.
- $\Psi(\overrightarrow{T}(e)) = \{\langle\varphi_t, \{[Pdf]\}\rangle,\ \langle\varphi_s, \{[\emptyset]\}\rangle,\ \langle\varphi_r, \{[\emptyset]\}\rangle\}$
  in other words, $\overrightarrow{T}(e)$ is of type *Pdf* and has no outgoing hyperlinks (instead the relation can, for example, be of type *Reference*).
- $\mathcal{U} = \{\langle\varphi_t, 0\rangle,\ \langle\varphi_s, -10\rangle,\ \langle\varphi_r, 5\rangle\}$
  in other words, the user refuses data resources of type *Doc* and would prefer them to have outgoiing hyperlinks.

We can now simply calucate the quality/aptness score of both data resources with regard to the users wishes. The quality score of $e = -10 + 5 = -5$. Sincie e is of type *Doc* it receives a penalty of -10 and since it has outgoing hyperlinks it receives a positive score of 5. Similarly, the quality score of $\overrightarrow{T}(e) = 10 - 5 = 5$. Clearly the transformation increases the aptness given the current preferences of the user.

7.5 In practice

Systems that deploy a transformation selection strategy face the problem of implementing an algorithm to do so. A number of choices have to be made while doing so:

- What to do with cyclic transformation graphs: a depth-first exhaustive search may get stuck and loop forever.
- What to do with very large transformation graphs: it may be impractical to perform an exhaustive search in real time; it may simply take too long to calculate all possible paths. A solution to this problem might be to adopt a penalty-based approach:
  - long paths are penalized. This implies that each step receives a step-penalty
  - after each step the penalty of the current path is calculated by comparing the expected truth assignment of properties with the desired properties by the user.

  if the current penalty exceeds a predefined penalty the path will no longer be considered.
- Even if the transformation graph is relatively small then an important question is: is the user willing to wait while the algorithm executes as oposed to simply accepting the untransformed resource or going for the first possible transformation path that is found.

8 Conclusion

With the aparent rise of the Web as a medium for *information supply* it becomes increasingly important to deal with its heterogeneity (heterogeneity in terms of the *form* and *format* of resources). In this article we zoom in on *search* tools on the Web and hypothesize that merely looking at topicality when selecting resources that conform to an information need is not enough. Because of the heterogeneous nature of the Web, we propose to use *aptness* instead.

Searching for apt resources for a given information need (as opposed to topically relevant resources) implies that the topicality of a resource, its form and its formt must be taken into account. To explore a system that for such a task we have taken the following steps. We firstly

studied the resources that are available to us online (The resources that are available in a given situation are together called a resource base. All potentional resource bases are called resource space). This has resulted in a formal model for resource space (Section 3). The heterogeneity of resource space (and thus all potential resource basis) comes to the for in the typing mechanism for our model (Section 4). In our model we follow a *types follow population* approach, meaning that the types are derived from the instances (instead of defining the types first and fitting the population to the prescribed scheme as is common in e.g. database design).

The major contribution of this article, though, is the transformation framework as described in Section 5 for it is the basis for dealing with heterogeneity in information supply be (search) tools on the Web. The simplest definition of a transformation, in this context, is a piece of software that transforms instances (of a certain type) to instances (of, potentially, another type). We describe what transformations are, what their characteristics are and how they can be composed to make complex transformations. As for the search architecture: in Section 6 we extend the transformation framework with a mechanism to measure the effect that a transformation may have on (properties of) resources in a given resource base. Properties of resources are, in essense, any relation, attribution or type that such a resource may have. These properties can either be neutral to such a property (both the input instance and the output instance have this property), it can remove a property (the input instance has it, but the output instance does not), it can introduce a property (the input instance does not have it, but the output instance does) or it may alter the property (both instances have it, but their respective *values* for this property are different).

The data resource types combined with the transformations form a directed graph: the types are the nodes and the transformations are the edges. For each edge we (may or may not) know the effect on properties.

These effects, which may be learned from a real resource base, provide a good starting point for aptness-based search on the Web. We propose a *push-down selection*-like scheme. As soon as a searcher's information need is known (i.e. the query) the first step is to select the topically relevant resources. In the second step we try to upgrade the *aptness* of resources by means of transformations. That is: we know from the query which properties instances must have (the topic being one of them) and try to transform the relevant resources so that they will have these properties.

In Section 7.4 we have described how transformations can be selected to estimate which (composed) transformations are "best" at performing the desired transformation on an input resource. In theory this can be done by comparing the properties desired by the user with the expected properties of data resources after a transformation has taken place. In real applications several issues have to be resolved such as how to deal with cyclic transformation graphs and (very) large graphs: it may simply take too much time to do an exhaustive search of all possible paths. A penalty-based approach (penalizing long paths and undesireable manipulation of properties by transformations) may resolve these issues.

A lot of work remains to be done in this area. We are currently working on an Web retrieval system (i.e. a search engine) that combines the push-down selection mechanism with the transformation selection algorithm. That is, we are working on a search engine for *aptness*-based search.

References

Al-Mishwat, A. T. (2000). Zgeoref: a program for transformation of georef database bibliographical extracts to reference formats suitable for earth science journals. *Computers & Geosciences*, 26(5):607–611.

Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.-J., Kuske, S., Plump, D., Schürr, A., and Taentzer, G. (1999). Graph transformation for specification and programming. *Science of Computer Programming*, 4(1):1–54.

Berners-Lee, T. (1994). Universal resource identifiers in www. Technical Report RFC 1630, IETF Network Working Group, http://www.ietf.org/rfc/rfc1630.txt.

Bommel, P. v., Kovács, G., and Micsik, A. (1994). Transformation of database populations and operations from the conceptual to the internal level. *Information Systems*, 19(2):175–191.

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modelling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA. ISBN 0-201-57168-4

Bruce, K. and Wegner, P. (1990). An algebraic model of subtype and inheritance. In Bancilhon, F. and Buneman, P., editors, *Advances in Database Programming Languages*, ACM Press, Frontier Series, pages 75–96. Addison-Wesley, Reading, Massachusetts.

Bush, V. (1945). As we may think. *The Atlantic Monthly*, 176(1):101–108.

Chen, P. (1976). The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36.

Conklin, J. (1987). Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41.

Date, C. J. (2003). *An introduction to Database Systems*. Addison Wesley, Bosten, Massachusets, USA, 8th edition. ISBN: 0-321-18956-6

Feng, L., Hoppenbrouwers, J., and Jeusfeld, M. (2001). Towards knowledge-based digital libraries. *SIGMOD Record 30*, 1:41–46.

Gils, B. v., Proper, H., and Bommel, P. v. (2003a). A conceptual model for information suppy. *Data & Knowledge Engineering*, 51:189–222.

Gils, B. v., Proper, H., and Bommel, P. v. (2003b). Towards a general theory for information supply. In Stephanidis, C., editor, *Proceedings of the 10th International Conference on Human-Computer Interaction*, pages 720–724, Crete, Greece, EU. ISBN 0805849300

Gils, B. v., Proper, H., Bommel, P. v., and Vrieze, P. d. (2005). Transformation selection for aptness-based web retrieval. In Williams, H. and Dobbie, G., editors, *Sixteenth Australasian Database Conference (ADC2005)*, volume 39 of *Conferences in Research and Practice in Information Technology Series*, pages 115–124, Sydney, New South Wales, Australia. Australian Computer Society. ISBN 1-920682-21-X

Gils, B. v. and Schabell, E. (2003). User-profiles for information retrieval. In *Proceedings of the 15th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'03)*, Nijmegen, The Netherlands.

Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B. (1977). Initial algebra semantics and continuous algebras. *Journal of the ACM (JACM)*, 24(1):68–95. ISSN: 0004-5411.

Halpin, T. (1995). *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia, 2nd edition.

Hofstede, A. t., Proper, H., and Weide, T. v. d. (1993). Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523.

Hofstede, A. t. and Weide, T. v. d. (1993). Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100.

Huibers, T. W. C., Lalmas, M., and Rijsbergen, C. J. v. (1996). Information retrieval and situation theory. *ACM SIGIR Forum*, 30(1):11–25. ISSN: 0163-5840

Lammel, R. (2004). Transformations everywhere. *Science of Computer Programming*, 52(1-3):1–8.

Polo, M., Gomez, J. A., Piattini, M., and Ruiz, F. (2002). Generating three-tier applications from relational databases: a formal and practical approach. *Information and Software Technology*, 44(15):923–941.

Proper, H. (1997). Data Schema Design as a Schema Evolution Process. *Data & Knowledge Engineering*, 22(2):159–189.

Rahayu, J. W., Chang, E., Dillon, T. S., and Taniar, D. (2001). Performance evaluation of the object-relational transformation methodology. *Data & Knowledge Engineering*, 38(3):265–300.

Song, J.-W., Whang, K.-Y., Lee, Y.-K., Lee, M.-J., Wook-Shin, H., and Park, B.-K. (2002). The clustering property of corner transformation for spatial database applications. *Information and Software Technology*, 44(7):419–429.

Ullman, J. (1989). *Principles of Database and Knowledge-base Systems*, volume I. Computer Science Press, Rockville, Maryland. ISBN 0716781581

Yang, S., Bhowmick, S. S., and Madria, S. (2005). Bio2x: a rule-based approach for semi-automatic transformation of semi-structured biological data to xml. *Data & Knowledge Engineering*, 52(2):249–271.