Journal of Universal Computer Science, vol. 16, no. 8 (2010), 1183-1201
submitted: 21/10/09, accepted: 19/8/10, appeared: 28/4/11

Cloud Warehousing

Hui Ma
(Victoria University of Wellington, Wellington, New Zealand
hui.ma@ecs.vuw.ac.nz)

Klaus-Dieter Schewe
(Software Competence Center Hagenberg, Hagenberg, Austria
kd.schewe@scch.at)

Bernhard Thalheim
(Institute of Computer Science, University of Kiel, Germany
thalheim@is.informatik.uni-kiel.de)

Qing Wang
(University of Otago, Dunedin, New Zealand
qing.wang@otago.ac.nz)

Abstract: Data warehouses integrate and aggregate data from various sources to support decision making within an enterprise. Usually, it is assumed that data are extracted from operational databases used by the enterprise. Cloud warehousing relaxes this view permitting data sources to be located anywhere on the world-wide web in a so-called "cloud", which is understood as a registry of services. Thus, we need a model of data-intensive web services, for which we adopt the view of the recently introduced model of abstract state services (AS^{2}s). An AS^2 combines a hidden database layer with an operation-equipped view layer, and thus provides an abstraction of web services that can be made available for use by other systems. In this paper we extend this model to an abstract model of clouds by means of an ontology for service description. The ontology can be specified using description logics, where the ABox contains the set of services, and the TBox can be queried to find suitable services. Consequently, AS^2 composition can be used for cloud warehousing.

Key Words: cloud computing, data warehouse, service-oriented computing, service composition, tenants, service ontology
Category: D.2.10, E.m

1 Introduction

A data warehouse stores data that is used for analytical tasks that support decision making. This is called online-analytical processing (OLAP) and distinguished from online transaction processing (OLTP) in databases. As OLAP tasks do not depend on the latest updates, it is usually assumed that the data are extracted from operational databases and refreshed in larger time intervals. Thus, a data warehouse separates the input from operational databases from output to the OLAP system, which itself is organised by a collection of extended views called datamarts.

Since the early days of data warehousing the emphasis has been on the design of a central data warehouse [Inmon, 1996; Kimball, 1996] using appropriate data models, and the development of corresponding OLAP systems based on views [Lawrence and Rau-Chaplin, 2006; Lewerenz et al., 1999; Thomson, 2002]. An integrated formal approach on the basis of the method of Abstract State Machines [Börger and Stärk, 2003] emphasizing stepwise refinement has been presented in [Zhao and Ma, 2006], and a formalisation of OLAP/OLTP was addressed in [Lenz and Thalheim, 2009]. More recently, the emphasis has shifted to incremental design taking data warehouse and OLAP evolution into account [Ravat et al., 2008; Theodoratos et al., 2001; Theodoratos and Sellis, 1999; Theodoratos and Sellis, 2000; Zhao et al., 2009].

Traditionally, the focus of data warehousing is decision support within an enterprise with little connections to research in emerging areas such as web services [Benatallah et al., 2006; Brenner and Unmehopa, 2007], service-oriented architectures [Brenner and Unmehopa, 2007; Kumaran et al., 2007; Papazoglou and van den Heuvel, 2007], or cloud computing. For instance, decision support could not only be based on data that is collected within an enterprise, but could be based on data that is available on the web, and not only data but also useful functions ranging from simple currency converters to prediction models could be incorporated. That is, data warehousing could exploit functionality that is made available by clouds, i.e. service registries. In this sense, a web data warehouse can be pictured as an aggregation of local components with data and functions that are extracted and integrated from available services. This ressembles the fundamental idea of meme media [Tanaka, 2003].

It is even possible that some data and functionality in a data warehouse and OLAP system is made available as a service. We therefore believe it is worth the effort to investigate the potential of web-based open data warehouses and OLAP systems. In such a system data sources could be located anywhere on the world-wide web. We would like to go even further and propose that web warehousing should exploit data-intensive web services.

Example 1. Let us first look at a data warehouse supporting sales statistics as illustrated in Figure 1 by a traditional ER diagram [Thalheim, 2000]. The schema has the form of a simple star schema with four "dimensions" capturing the location of each sale, represented by the type SHOP, the object of a sale, represented by PRODUCT, the buying CUSTOMER, and the TIME of the sale. Products are grouped into categories, for time the usual grouping into months, quarters and years is available – which is in fact derived data – and for shops a grouping into town, region and state applies. The "fact" type PURCHASE permits capturing quantity (how many of the specific product were bought), sales (how

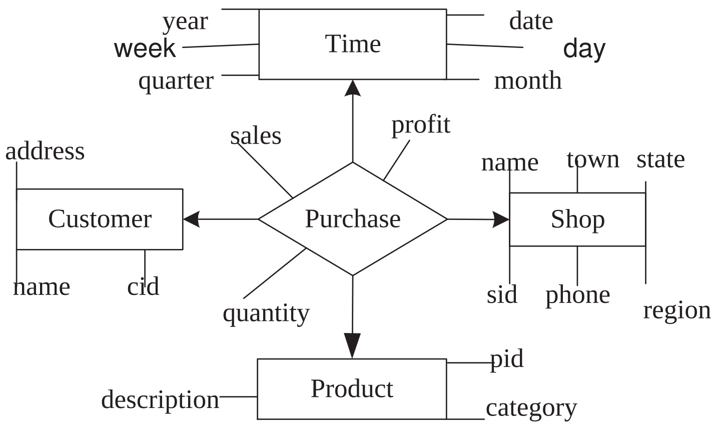

Figure 1: A star schema for a sales statistics data warehouse

much did the customer pay), and profit (what is the difference to the price of the product as paid by the shop).

The annual sales data per region are available through a datamart using a simple typed ASM rule (see [Schewe and Zhao, 2008] for details on typed ASMs):

$$\textsc{Sales} := \pi_{\text{sales:sum(sales), region, year}}(\textsc{Purchase} \bowtie \textsc{Shop} \bowtie \textsc{Time})$$

this view can be extended by standard OLAP service functions such as drill-down, roll-up and slice. Assuming that population data per region can be extraced from another service, e.g. providing a binary relation Population with attributes region and population, a service operation

$$\textsc{Samples} := \pi_{\text{sales, population}}(\textsc{Sales} \bowtie \textsc{Population})$$

provides sample data for linear regression. Taking b_0, b_1 such that

$$\sum_{(p,s)\in\textsc{Samples}} ((b_0 + b_1 \cdot p) - s)^2$$

is minimal, we obtain a service function for forecasting annual sales for all stores depending on the size of the population using an equation

$$\hat{Y}_i = b_0 + b_1 \cdot X_i \ .$$

Here, $\hat{Y}_i$ denotes the predicted value of Y for the observation i, in our case the annual sales, and X_i the value of X for the observation i, in our case the size of the population. $\Diamond$

Recently, we developed the theoretical framework of Abstract State Services (AS^{2}s) [Ma et al., 2008; Ma et al., 2009b], which formalises the notion of data-intensive service, and links service-oriented systems with the fundamental ASM thesis [Gurevich, 2000] and its adaptation to general database transformations [Schewe and Wang, 2008; Wang and Schewe, 2007]. An AS^2 combines a hidden database layer with an operation-equipped view layer, and can be anything from a simple function to a full-fledged Web Information System. A variant of Abstract State Machines (ASMs) can be used to specify and capture AS^{2}s.

In our recent conference publication [Ma et al., 2009a] we demonstrated how open data warehouses and OLAP systems can be considered in a natural way as AS^{2}s, i.e. selected data and functionality of a data warehouse may be made available in the form of a generalised data-intensive web service. Using these services we can then build new services by means of component extraction and recomposition.

Thus, on one side data warehousing will exploit the web and thus become web warehousing, and on the other side web warehouses are themselves made available as services. This constitutes a service-oriented approach to web data warehousing, which at the same time shifts the focus towards more open systems based on model suites [Thalheim, 2008], i.e. sets of models with explicit associations among the individual models. The two layers of an AS^2 plus the AS^2 composition on top of them constitute a three-layer, hierarchically structured model suite.

While AS^{2}s are a suitable abstraction of data-intensive services, it will be crucial for a web-based service-oriented architecture that useful and usable service can be discovered. In order to support the search for such services we propose federations of services together with an ontology that specifies their semantics, i.e. we would lean on the idea of the semantic web. Such an ontology should ideally comprise functional and non-functional characterisations of the services. The functional characterisation could exploit types for input and output, pre- and postconditions, and a description of the application area. The non-functional characterisation could comprise quality-of-service parameters such as performance, costs, availability, etc. This leads to a formal model of a cloud as service registry together with a semantic description.

In the remainder of this paper we develop an abstract model of clouds as federations of services. In Section 2 we briefly review the ASM-based abstract model of Abstract State Services (AS^{2}s), which gives an answer to the question how to specify services. We dispense with all aspects of concrete languages for AS^{2}s (see [Ma et al., 2009b]). In Section 3 we develop a formal model of clouds as federations of services that are in addition equipped with an ontology capturing service descrition. In this way we give an answer to the how to discover services. In Section 4 we address the problem of extracting relevant components of a cloud data warehouse that is available as an AS^2 and recombining these components into a new web-based data warehouse application. In our conclusions in Section 5 we first give a brief summary and place our work into the literature context by looking at related work. We then analyse what has been achieved so far and which open questions remain for future research.

2 An Abstract Model for Services

In this section we first address the question how to specify services in an abstract way. Our goal is to provide a concise formal framework that captures all necessary aspects of data-intensive services rather than focussing on a particular specification language.

As services that are useful for web warehousing will be data-intensive, we first look at databases. Traditional database architecture distinguishes at least three layers: a conceptual layer describing the database schema in an abstract way, a physical layer implementing the schema, and an external layer made out of views. The external layer exports the data that can then be used by users or programs. For our purposes here we can neglect the physical layer, but in order to capture data-intensive services, we complete this architecture by adding operations on both the conceptual and the external layer, the former one being handled as database transactions, whereas the latter ones provide the means with which users can interact with a database.

Starting with the database layer and following the general approach of Abstract State Machines [Gurevich, 2000] we may consider each database computation as a sequence of abstract states, each of which represents the database (instance) at a certain point in time plus maybe additional data that is necessary for the computation, e.g. transaction tables, log files, etc. In order to capture the semantics of transactions we distinguish between a wide-step transition relation and small step transition relations. A transition in the former one marks the execution of a transaction, so the wide-step transition relation defines infinite sequences of transactions. Without loss of generality we can assume a serial execution, while of course interleaving is used for the implementation. Then each transaction itself corresponds to a finite sequence of states resulting from a small step transition relation, which should then be subject to the postulates for database transformations [Wang and Schewe, 2007].

Definition 1. A *database system* DBS consists of a set $\mathcal{S}$ of states, together with a subset $\mathcal{I} \subseteq \mathcal{S}$ of initial states, a wide-step transition relation $\tau \subseteq \mathcal{S} \times \mathcal{S}$, and a set $\mathcal{T}$ of transactions, each of which is associated with a small-step transition relation $\tau_t \subseteq \mathcal{S} \times \mathcal{S}$ $(t \in \mathcal{T})$ satisfying the postulates of a database transformation over $\mathcal{S}$.

A *run* of a database system DBS is an infinite sequence $S_0, S_1, \dots$ of states $S_i \in \mathcal{S}$ starting with an initial state $S_0 \in \mathcal{I}$ such that for all $i \in \mathbb{N}$ $(S_i, S_{i+1}) \in \tau$ holds, and there is a transaction $t_i \in \mathcal{T}$ with a finite run $S_i = S_i^0, \dots, S_i^k = S_{i+1}$ such that $(S_i^j, S_i^{j+1}) \in \tau_{t_i}$ holds for all $j = 0, \dots, k-1$.

Views in general are expressed by queries, i.e. read-only database transformations. Therefore, we can assume that a view on a database state $S_i \in \mathcal{S}$ is given by a finite run $S_i = S_0^v, \dots, S_\ell^v$ of some database transformation v with $S_i \subseteq S_\ell^v$ – traditionally, we would consider $S_\ell^v - S_i$ as the view. We can use this to extend a database system by views.

In doing so we let each state $S \in \mathcal{S}$ to be composed as a union $S_d \cup V_1 \cup \dots \cup V_k$ such that each $S_d \cup V_j$ is a view on S_d. As a consequence, each wide-step state transition becomes a parallel composition of a transaction and an operation that switches views on and off.

Definition 2. An *Abstract State Service* (AS^2) consists of a database system DBS, in which each state $S \in \mathcal{S}$ is a finite composition $S_d \cup V_1 \cup \dots \cup V_k$, and a finite set $\mathcal{V}$ of (extended) views. Each view $v \in \mathcal{V}$ is associated with a database transformation such that for each state $S \in \mathcal{S}$ there are views $v_1, \dots, v_k \in \mathcal{V}$ with finite runs $S_d = S_0^j, \dots, S_{n_j}^j = S_d \cup V_j$ of v_j $(j = 1, \dots, k)$. Each view $v \in \mathcal{V}$ is further associated with a finite set $\mathcal{O}_v$ of (service) operations $o_1, \dots, o_n$ such that for each $i \in \{1, \dots, n\}$ and each $S \in \mathcal{S}$ there is a unique state $S' \in \mathcal{S}$ with $(S, S') \in \tau$. Furthermore, if $S = S_d \cup V_1 \cup \dots \cup V_k$ with V_i defined by v_i and o is an operation associated with v_k, then $S' = S'_d \cup V'_1 \cup \dots \cup V'_m$ with $m \geq k-1$, and V'_i for $1 \leq i \leq k-1$ is still defined by v_i.

In a nutshell, in an AS^2 we have view-extended database states, and each service operation associated with a view induces a transaction on the database, and may change or delete the view it is associated with, and even activate other views. These service operations are actually what is exported from the database system to be used by other systems or directly by users.

The abstract handling of service operations that induce transactions avoids the view update problem, which has to be taken into account when dealing with concrete specifications for AS^2s, e.g. using the theory developed in [Hegner, 2008].

A formalisation of database transformations by means of postulates is beyond the scope of this paper and excluded due to space limitations. In a nutshell, the postulates require a one-step transition relation between states (sequential time postulate), states as (meta-finite) first-order structures (abstract state postulate), necessary background for database computations such as complex value constructors (background postulate), limitations to the number of accessed terms in each step (exploration boundary postulate), and the preservation of equivalent substructures in one successor state (genericity postulate) [Ma et al., 2009b; Schewe and Wang, 2008].

Data warehouses and OLAP systems provide aninteresting class of examples of AS^2s. The ASM-based approach in [Zhao and Ma, 2006; Zhao et al., 2009] uses three linked ASMs to model data warehouse and OLAP applications. At its core we have an ASM modelling the data warehouse itself using star or snowflake schemata. This will constitute the database layer of an AS^2. A second ASM would be used for modelling operational databases with rules extracting data from them and refreshing the data warehouse. This ASM is of no further relevance for us and thus can be ignored. A third ASM models the OLAP interface on the basis of the idea that datamarts can be represented as extended views [Lewerenz et al., 1999].

3 An Abstract Model of Clouds

The common understanding of the notion of "cloud" is that it is some kind of service pool, from which services can be extracted and used. Thus, one of the key problems is to discover the services that are needed for a particular application. This should be possible by means of a search engine. Therefore, it is crucial that the service operations including the view defining queries that are made available through some cloud are provided with an adequate description, which will allow a search engine to discover (with some certainty) the required services.

Such a description should at least comprise three parts:

- a *functional* description of input- and output types as well as pre- and post-conditions telling in technical terms, what the service operation will do,
- a *categorical* description by inter-related keywords telling what the service operation does by using common terminology of the application area, and
- a *quality of service* (QoS) description of non-functional properties such as availability, response time, cost, etc.

The QoS description is not needed for service discovery and merely useful to select among alternatives, but neither functional nor categorical description can be dispensed with. However, a functional description alone would be insufficient, as services may have the same functional description, even if they stem from very different application areas. As for the categorical description, the terminology has to be specified. This defines an ontology in the widest sense, i.e. we have to provide definitions of "concepts" and relationships between them, such that each offered service becomes an instantiation of one or several concepts in the terminology. In this way we adopt the fundamental idea of the "semantic web".

In the following we will outline how ontologies can be exploited for service description. We will pay particular attention to description logics without prescribing a particular language. That is, OWL [W3C, 2003] or more expressive description logics [Baader et al., 2003] could be exploited.

3.1 Ontologies

As outlined, the functional, categorical and QoS description of services in a cloud requires the definition of an ontology [Flahive et al., 2006; Flahive et al., 2009; Lanzenberger et al., 2010]. That is, we need a *terminological knowledge* layer (aka TBox in description logics) describing concepts and roles (or relationships) among them. This usually includes a subsumption hierarchy among concepts (and maybe also roles), and cardinality constraints. In addition, there is an *assertional knowledge* layer (aka ABox in description logics) describing individuals. Thus, services in a cloud constitute the ABox of an ontology, while the cloud itself is defined by the TBox.

In principle, instead of TBox and ABox we could use the more classical notions of schema and instance, and exploit any kind of data model. A query language associated with the used data model, could then be used to find the required services. In fact, description logics only provide rather limited logics with respect to expressiveness. There are two major reasons for giving preference to description logics:

1. Description logics use two important relationships, which due to the restrictions become decidable: subsumption and instantiation. Subsumption is a binary relationship between concepts (denoted as $C_1 \sqsubseteq C_2$) guaranteeing that all instances of the subsumed concept C_1 are also instances of the subsuming concept C_2. Instantiation defines a binary relationship between instances in the ABox and concepts in the TBox asserting that an element A of the ABox is an instance of a concept C in the TBox. Subsumption and instantiation together allow us to discover services that are more expressive than needed, but can be projected to a service just as required.

2. Concept and role names in the TBox can be subject to similarity search by a search engine. That is, the search engine could produce services that are similar (with a certainty factor) to the ones required with respect to the categorical description, and match the functional description.

Let us now look more closely into one particular description logic in the *DL-Lite* family (see [Baader et al., 2003]). For this assume that C_0 and R_0 represent not further specified specified sets of basic concepts and roles, respectively. Then *concepts* C and *roles* R are defined by the following grammar:

$$\begin{aligned} R &= R_0 \mid R_0^- \\ A &= C_0 \mid \top \mid \geq m.R \text{ (with } m > 0) \\ C &= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \end{aligned}$$

Definition 3. A *terminology* (or TBox) is a finite set $\mathcal{T}$ of assertions of the form $C_1 \sqsubseteq C_2$ with concepts C_1 and C_2 as defined by the grammar above.

Each assertion $C_1 \sqsubseteq C_2$ in a terminology $\mathcal{T}$ is called a *subsumption axiom*. Note that Definition 3 only permits subsumption between concepts, not between roles, though it is possible to define more complex terminologies that also permit role subsumption. As usual, we use the shortcut $C_1 \equiv C_2$ instead of $C_1 \sqsubseteq C_2 \sqsubseteq C_1$. For concepts, $\perp$ is a shortcut for $\neg\top$, and $\leq m.R$ is a shortcut for $\neg \geq m+1.R$. The semantics of a terminology is defined by its models. For this we use structures $\mathcal{S}$.

Definition 4. A structure $\mathcal{S}$ for a terminology $\mathcal{T}$ consists of a non-empty set $\mathcal{O}$ together with subsets $\mathcal{S}(C_0) \subseteq \mathcal{O}$ and $\mathcal{S}(R_0) \subseteq \mathcal{O} \times \mathcal{O}$ for all basic concepts R_0 and basic roles R_0, respectively. $\mathcal{O}$ is called the base set of the structure.

We extend the interpretation of basic concepts and roles and to all concepts and roles as defined by the grammar above, i.e. for each concept C we define a subset $\mathcal{S}(C) \subseteq \mathcal{O}$, and for each role R we define a subset $\mathcal{S}(R) \subseteq \mathcal{O} \times \mathcal{O}$:

$$\mathcal{S}(R_0^-) = \{(y,x) \mid (x,y) \in \mathcal{S}(R_0)\}$$
$$\mathcal{S}(\top) = \mathcal{O}$$
$$\mathcal{S}(\geq m.R) = \{x \in \mathcal{O} \mid \#\{y \mid (x,y) \in \mathcal{S}(R)\} \geq m\}$$
$$\mathcal{S}(\neg C) = \mathcal{O} - \mathcal{S}(C)$$
$$\mathcal{S}(C_1 \sqcap C_2) = \mathcal{S}(C_1) \cap \mathcal{S}(C_2)$$
$$\mathcal{S}(C_1 \sqcup C_2) = \mathcal{S}(C_1) \cup \mathcal{S}(C_2)$$
$$\mathcal{S}(\exists R.C) = \{x \in \mathcal{O} \mid (x,y) \in \mathcal{S}(R) \text{ for some } y \in \mathcal{S}(C)\}$$
$$\mathcal{S}(\forall R.C) = \{x \in \mathcal{O} \mid (x,y) \in \mathcal{S}(R) \Rightarrow y \in \mathcal{S}(C) \text{ for all } y\}$$

Definition 5. A *model* for a terminology $\mathcal{T}$ is a structure $\mathcal{S}$, such that $\mathcal{S}(C_1) \subseteq \mathcal{S}(C_2)$ holds for all assertions $C_1 \sqsubseteq C_2$ in $\mathcal{T}$. A finite model, i.e. a model with a finite base set, is also called *instance* or ABox associated with $\mathcal{T}$.

Example 2. The general part of a service ontology could be defined by a terminology as follows:

$$\begin{gathered}
\text{Service} \sqsubseteq \exists\text{name.Identifier} \sqcap \leq 1.\text{name} \sqcap \exists\text{address.URL} \sqcap \leq 1.\text{address} \sqcap \\
\leq 1.\text{offered_by} \sqcap \exists\text{offered_by.Provider} \sqcap \exists\text{defining.Query} \sqcap \\
\leq 1.\text{defining} \sqcap \exists\text{offers.Operation} \\
\text{Operation} \sqsubseteq \exists\text{associated_with.Query} \sqcap \leq 1.\text{associated_with} \\
\text{Data_Service} \equiv \text{Query} \sqcap \geq 1.\text{defining}^- \\
\text{Functional_Service} \equiv \text{Operation} \sqcap \geq 1.\text{offers}^- \\
\text{Service_Operation} \equiv \text{Data_Service} \sqcup \text{Functional_Service} \\
\text{Service_Operation} \sqsubseteq \exists\text{input.Type} \sqcap \leq 1.\text{input} \sqcap \exists\text{output.Type} \sqcap \leq 1.\text{output} \\
\text{Type} \sqsubseteq \exists\text{name.Identifier} \sqcap \leq 1.\text{name} \sqcap \exists\text{format.Format}
\end{gathered}$$

Here we used capital first letters to indicate concept names, and lower case letters for role names. ◊

3.2 Definition of Clouds

Following our discussion above we can now formally define a cloud as a federation of services together with a descriptive ontology.

Definition 6. A *cloud* is a finite collection $\{\mathcal{A}_i \mid i \in I\}$ of AS^{2}s together with a terminology $\mathcal{T}$, such that the defining queries of views and the associated service operations of these AS^{2}s define an instance of $\mathcal{T}$.

In principle, this definition permits any kind of terminology, as long as it deals with service operations. However, as outlined above we expect the terminology $\mathcal{T}$ of a cloud to provide the functional, categorical and QoS description of its offered services.

Functional Description. The functional description of a service operation consists of input- and output-types as already indicated in Example 2, and pre- and postconditions. For the types we need a type system with base types and constructors. For instance, the following grammar

$$t = b \mid \mathbb{1} \mid (a_1 : t_1, \dots, a_n : t_n) \mid \{t\} \mid [t] \mid (a_1 : t_1) \oplus \cdots \oplus (a_n : t_n)$$

describes (the abstract syntax of) a type system with a trivial type $\mathbb{1}$, a non-further specified collection of base types b, and four type constructors $(\cdot)$ for record types, $\{\cdot\}$ for finite set types, $[t]$ for list types, and $\oplus$ for union types. Record and union types use field labels a_i.

The semantics of such types is basically described by their domain, i.e. sets of values $dom(t)$. Usally, for a base type b such as *Cardinal*, *Decimal*, *Float*, etc. the domain is some commonly known at most countable set with a common presentation. The domain of the trivial type contains a single special value, say $dom(\mathbb{1}) = \{\bot\}$. For constructed types we obtain the domain in the usual way:

$$\begin{aligned} dom((a_1 : t_1, \dots, a_n : t_n)) &= \{(a_1 : v_1, \dots, a_n : v_n) \mid a_i \in dom(t_i) \\ &\qquad \text{for } i = 1, \dots, n\} \\ dom(\{t\}) &= \{A \mid A \subseteq dom(t) \text{ finite}\} \\ dom([t]) &= \{[v_1, \dots, v_k] \mid v_i \in dom(t) \text{ for } i = 1, \dots, k\} \\ dom((a_1 : t_1) \oplus \cdots \oplus (a_n : t_n)) &= \bigcup_{i=1}^{n} \{(a_i : v_i) \mid v_i \in dom(t_i)\} \end{aligned}$$

In particular, a union type $(a_1 : \mathbb{1}) \oplus \cdots \oplus (a_n : \mathbb{1})$ has the domain $\{(a_1 : \bot), \dots, (a_n : \bot)\}$, which can be identified with the set $\{a_1, \dots, a_n\}$, i.e. such types are in fact enumeration types.

It is no problem to add the specification of types to the general service terminology as outlined in Example 2 thereby defining part of the functional description.

Example 3. We can extend the terminology in Example 2 by the following axioms for types:

$$\text{Type} \equiv \text{Base\_type} \sqcup \text{Trivial\_type} \sqcup \text{Composed\_type}$$
$$\text{Composed\_type} \equiv \text{Record} \sqcup \text{Set} \sqcup \text{List} \sqcup \text{Union}$$
$$\text{Record} \sqsubseteq \forall \text{component.Field}$$
$$\text{Field} \sqsubseteq \exists \text{field\_name.Identifier} \sqcap \leq 1.\text{field\_name} \sqcap \exists \text{type.Type} \sqcap \leq 1.\text{type}$$
$$\text{Union} \sqsubseteq \forall \text{component.Field}$$
$$\text{Record} \sqcap \text{Union} \sqsubseteq \bot$$
$$\text{Set} \sqsubseteq \exists \text{component.Type} \sqcap \leq 1.\text{component}$$
$$\text{List} \sqsubseteq \exists \text{component.Type} \sqcap \leq 1.\text{component}$$
$$\text{Set} \sqcap \text{List} \sqsubseteq \bot$$

The specification of composed types impacts directly on the format, which is defined by field names and the format for the component type(s). Nevertheless, this constraint can be handled by the specification of ABox assertions. ◊

In addition to the types, the functional description of a service operation includes pre- and postconditions, which are defined by (first-order) predicate formulae. These formulae may contain further functions and predicates, which are subject to further (categorical) description.

Example 4. The terminology in Examples 2 and 3 can be further extended by the following axioms:

$$\text{Service\_Operation} \sqsubseteq \forall \text{pre.Condition} \sqcap \leq 1.\text{pre} \sqcap \exists \text{post.Condition} \sqcap \leq 1.\text{post}$$
$$\text{Condition} \sqsubseteq \text{Formula} \sqcap \forall \text{uses.}(\text{Predicate} \sqcap \text{Function})$$
$$\text{Predicate} \sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \neg \geq 1.\text{out}$$
$$\text{Function} \sqsubseteq \exists \text{in.Type} \sqcap \leq 1.\text{in} \sqcap \exists \text{out.Type} \sqcap \leq 1.\text{out}$$

This would complete the functional part of the terminology. ◊

Quality of Service. Quality of service properties are availablity, cost, response time, etc. These can be added directly by means of roles to the concept Service_Operation.

Example 5. By means of the axiom

$$\begin{gathered}\text{Service\_Operation} \sqsubseteq \exists \text{cost.Amount} \sqcap \exists \text{availability.Times} \\ \sqcap \text{response\_time.Duration}\end{gathered}$$

the terminology in Example 2 will be extended by a QoS description. $\Diamond$

Categorical Description. The core of the service description by means of the terminology of a cloud is the categorical description, which refers to the standard terminology of the application area, and relates the used notions to each other. There are no general requirements for the categorical description, as it depends completely on the application domain. However, it will always lead to subconcepts of the concept Service_Operation plus additional concepts and roles. It will also add more details to the predicates and functions used in the pre- and postconditions.

Example 6. Let us look again at the regression in Example 1, for which the categorical description may consist of the following axioms:

$$\begin{gathered}\text{Annual\_Sales\_Q} \sqsubseteq \text{Data\_Service} \sqcap \exists \text{output.Sales\_Stats} \\ \text{Sales\_Stats} \sqsubseteq \text{Set} \sqcap \exists \text{component.Sales\_Record} \\ \text{Sales\_Record} \sqsubseteq \text{Record} \sqcap \text{component.Sales} \sqcap \exists \text{component.Region} \\ \sqcap \exists \text{component.Year} \\ \text{Population\_Q} \sqsubseteq \text{Data\_Service} \sqcap \exists \text{output.Population\_Stats} \\ \text{Samples} \sqsubseteq \text{Data\_Service} \sqcap \exists \text{uses.Annual\_Sales} \sqcap \exists \text{uses.Population\_Stats} \\ \text{Regression} \sqsubseteq \text{Service\_Operation} \sqcap \exists \text{requires.Samples} \sqcap \exists \text{produces.Sales\_Estim}\end{gathered}$$

This specification is of course incomplete, but it shows how to proceed. That is, annual sales are defined by a data service, i.e. a query that produces a relation with at least three attributes Sales, Region and Year. Similarly, population is defined by a data service, and samples result from combining both (using a join). The regression service operation requires these samples and produces a sales estimate. $\Diamond$

The terminology of a cloud enables the discovery of services, as it can be queried. For this we assume that the terminology $\mathcal{T}$ is complete with respect to subsumption, i.e. whenever $C_1 \sqsubseteq C_2$ is implied by the axioms in $\mathcal{T}$, it is added to $\mathcal{T}$. Implication is defined in the usual way, i.e. $\mathcal{T} \models C_1 \sqsubseteq C_2$ iff $\mathcal{S}(C_1) \subseteq \mathcal{S}(C_2)$ holds for every model of $\mathcal{T}$.

We further assume that the assertions in the ABox have been completed as well, i.e. whenever $C_1(o)$ is such an assertion and the axiom $C_1 \sqsubseteq C_2$ is in $\mathcal{T}$, then also $C_2(o)$ is asserted in the ABox. With these assumptions we can treat the ABox as a simple relational database with unary and binary relation symbols for the concepts and roles of the terminology $\mathcal{T}$, respectively. Then any query language for relational databases, calculus, algebra, SQL, fixed-point logic, DATALOG, etc. can be used to query a cloud.

However, the success of a search depends on the compatibility of the ontologies used by the service seeker and the service provider. If there is no common understanding of the terms used used in the terminology, in particular in the categorical description, the service seeker may apply a service, which does not deliver the required functionality.

4 Extraction and Composition of Services

In the previous two sections we developed a general model for clouds as service federations, thus answering how services can be specified and discovered. In this section we address the related question how services that have been identified for solving an application problem can be composed. For this we only have to consider the composition of AS^{2}s. We follow and generalise [Ma et al., 2009b] and discuss how to extract components from AS^{2}s, and how to recompose them to form new AS^{2}s. The application to cloud data warehouses is a special case.

4.1 Component Extraction

In order to extract components from an AS^2 we first build a subset $\mathcal{V}' \subseteq \mathcal{V}$ of the set of views, and for each view $v \in \mathcal{V}'$ we restrict the service operations to a subset $\mathcal{O}'_v \subseteq \mathcal{O}_v$. These subset restrictions obviously produce an AS^2 with the same underlying database system as before.

In a second step we actually restrict the views $v \in \mathcal{V}'$ themselves by defining a view p_v on top of it, i.e. p_v is a database transformation that will transform a state V into a state $V \cup V'$. Practically speaking, service extraction can only be performed by service users, and they only have access to the view layer, not to the underlying database system. Nevertheless, by forgetting the original view V, the composed database transformation $p_v \circ v$ defines a view on top of the original database system transforming states $S \in \mathcal{S}$ into states $S \cup V'$.

Furthermore, $o \in \mathcal{O}'_v$ still induces the same transaction, and if v would be replaced by $\{v_1, \dots, v_k\}$, then $p_v \circ v$ would have to be replaced by $\{p_{v_i} \circ v_i \mid i \in \{1, \dots, k\}, v_i \in \mathcal{V}'\}$. In this way, the collection of views p_v defines an AS^2 with the same underlying database system as before. We will call this an AS^2 component.

Definition 7. Let $\mathcal{A} = (DBS, \mathcal{V}) = (\mathcal{S}, \tau, \{\tau_t\}_{t\in\mathcal{T}}, \{(v, \{o_1, \dots, o_{n_v}\})\}_{v\in\mathcal{V}})$ be an AS^2. A *component* of $\mathcal{A}$ is an AS^2 $(\mathcal{S}, \tau, \{\tau_t\}_{t\in\mathcal{T}}, \{(p_v \circ v, \{o'_1, \dots, o'_{n'_v}\})\}_{v\in\mathcal{V}'})$ with $\mathcal{V}' \subseteq \mathcal{V}$ and $\{o'_1, \dots, o'_{n'_v}\} \subseteq \{o_1, \dots, o_{n_v}\}$.

Note that in order to define a component of an AS^2 $\mathcal{A}$, it is not necessary to know anything about the DBS of $\mathcal{A}$, as only the views and associated service operations are affected. In a practical sense this reflects the request that the database layer of an AS^2 should be hidden.

The functions p_v that are used to define a component primarily constitute views over views, but can be interpreted in the most general way, as they are fully under control of those extracting components. In particular, functions extracted from any other AS^2 can be used.

4.2 AS^2 Composition

In order to recombine the data extraction and service operations from several AS^{2}s it is natural to use functional composition, if input and output are compatible. In addition, we may use aggregation operations and other locally defined auxiliary functions. For instance, if q_{v_1} and q_{v_2} are defining queries of two views v_1 and v_2 resulting in relations, we can aggregate them by building the natural join of the corresponding results. We denote this view by $v_1 \bowtie v_2$, and call it an *aggregated view* of $\{v_1, v_2\}$ with aggregate functions $\{\bowtie\}$. More general, any functional composition of given views with other functions defines an aggregated view, provided the views have to be executed first.

Similarly, any such functional composition (without the restriction that views have to come first) can define a new service operation. This leads to the following definition of an aggregated AS^2.

Definition 8. Let $\mathcal{A}^i = (\mathcal{S}^i, \tau^i, \{\tau^i_t\}_{t\in\mathcal{T}^i}, \{(v^i, \{o^i_1, \dots, o^i_{n_{v^i}}\})\}_{v^i\in\mathcal{V}^i})$ be AS^2 components ($i = 1, \dots, n$). An AS^2 $\mathcal{A} = (\mathcal{S}, \tau, \{\tau_t\}_{t\in\mathcal{T}}, \{(v, \{o_1, \dots, o_{n_v}\})\}_{v\in\mathcal{V}})$ is an *aggregation* of $\mathcal{A}^1, \dots, \mathcal{A}^n$ with a set of local functions $\mathcal{F}$ iff each view $v \in \mathcal{V}$ is an aggregated view of $\bigcup_{i=1}^{n} \mathcal{V}^i$ with the aggregate functions

$$\mathcal{F} \cup \bigcup_{i=1}^{n} \bigcup_{v^i\in\mathcal{V}^i} \{o^i_1, \dots, o^i_{n_{v^i}}\},$$

and each service operation $o \in \mathcal{V}$ is composed out of $\bigcup_{i=1}^{n} \bigcup_{v^i\in\mathcal{V}^i} \{o^i_1, \dots, o^i_{n_{v^i}}\} \cup \mathcal{F} \cup \bigcup_{i=1}^{n} \{q_v \mid v \in \mathcal{V}^i\}$.

5 Evaluation and Conclusions

In this paper we combined our ASM-based approach to the dynamic design of data warehouses and OLAP systems [Zhao et al., 2009] with our recent foundational approach to data-intensive software services [Ma et al., 2009b]. Technically, a data warehouse and OLAP system can be considered as an instantiation of an abstract state service. We elaborated this view and provided examples to illustrate this idea. In this way we obtain some form of decision support services. Components of such services can be extracted and recombined with other service components to define other services. Thus, also the data warehouse and OLAP system itself can benefit from other services, in particular other decision support services. This constitutes an approach to web warehousing and interoperability focusing on decision support extending our previous work in [Ma et al., 2009a].

Furthermore, we extended our work in the direction of search for services. For this we picked up the idea from [Ma et al., 2009c] of clouds as federations of services that are in addition equipped with a detailed ontology of the services. We argued that such an ontology should comprise three parts: a functional part comprising types and pre- and postconditions for the service operations, a quality-of-service part comprising non-functional properties such as availability, performance, etc., and a categorical part comprising the concepts and relationships of the application domain. Using description logics for such ontologies permits searching for services by querying the ABox of the terminology. With this extension we obtain a model of data warehousing in the cloud.

Our work stands out in the service-oriented and cloud computing community, as we are aiming at solid formal foundations addressing the key questions instead of ad-hoc technological solutions for isolated problems. For instance, the idea of using service-orientation to build web applications is also stressed in [Zaupa et al., 2008], and the composition of complex web applications leading to so-called mashups is addressed in [Ikeda et al., 2009]. With [Altenhofen et al., 2008] we have in common that we exploit ASMs in our work, but we do not start with an ASM-based specification language. Instead we started from the sequential ASM thesis [Gurevich, 2000], which we first extended to a thesis for database transformations, which are a key notion in the formal model of AS^{2}s. None of the mentioned related articles address the particular area of data warehousing.

Leaving the technical aspects aside we argued that cloud web warehousing makes sense, while traditionally, warehousing is considered an intra-enterprise activity. We provided examples supporting this generalised view. However, even if the focus remains on enterprise-centric warehousing, our service-based approach can be useful, e.g. for enterprises with a variety of diverse activities or international branches, as it enables a separation between decision support on various levels.

What is still missing in our formal model of clouds is a treatment of service queries. That is, how can we decompose a request for a particular service into elementary services that are then searched for in the clouds. Furthermore, in the context of such search we will have to exploit the quality-of-service properties to make an optimal selection among alternative solutions.

References

[Altenhofen et al., 2008] Altenhofen, M., Friesen, A., and Lemcke, J. (2008). ASMs in service oriented architectures. *Journal of Universal Computer Science*, 14(12):2034–2058.

[Baader et al., 2003] Baader, F. et al., editors (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

[Benatallah et al., 2006] Benatallah, B., Casati, F., and Toumani, F. (2006). Representing, analysing and managing web service protocols. *Data and Knowledge Engineering*, 58(3):327–357.

[Börger and Stärk, 2003] Börger, E. and Stärk, R. (2003). *Abstract State Machines*. Springer-Verlag, Berlin Heidelberg New York.

[Brenner and Unmehopa, 2007] Brenner, M. R. and Unmehopa, M. R. (2007). Service-oriented architecture and web services penetration in next-generation networks. *Bell Labs Technical Journal*, 12(2):147–159.

[Flahive et al., 2006] Flahive, A., Apduhan, B. O., Rahayu, J. W., and Taniar, D. (2006). Large scale ontology tailoring and simulation in the semantic grid environment. *International Journal of Metadata, Semantics and Ontologies*, 1(4):265–281.

[Flahive et al., 2009] Flahive, A., Taniar, D., Rahayu, J. W., and Apduhan, B. O. (2009). Ontology tailoring in the semantic grid. *Computer Standards & Interfaces*, 31(5):870–885.

[Gurevich, 2000] Gurevich, J. (2000). Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111.

[Hegner, 2008] Hegner, S. J. (2008). Information-optimal reflections of view updates on relational database schemata. In Hartmann, S. and Kern-Isberner, G., editors, *Foundations of Information and Knowledge Systems – 5th International Symposium (FoIKS 2008)*, volume 4932 of *Lecture Notes in Computer Science*, pages 112–131. Springer-Verlag.

[Ikeda et al., 2009] Ikeda, S., Nagamine, T., and Kamada, T. (2009). Application framework with demand-driven mashup for selective browsing. *Journal of Universal Computer Science*, 15(10):2109–2137.

[Inmon, 1996] Inmon, W. (1996). *Building the Data Warehouse*. Wiley & Sons, New York.

[Kimball, 1996] Kimball, R. (1996). *The Data Warehouse Toolkit*. John Wiley & Sons.

[Kumaran et al., 2007] Kumaran, S. et al. (2007). Using a model-driven transformational approach and service-oriented architecture for service delivery management. *IBM Systems Journal*, 46(3):513–530.

[Lanzenberger et al., 2010] Lanzenberger, M., Sampson, J., and Rester, M. (2010). Ontology visualization: Tools and techniques for visual representation of semi-structured meta-data. *Journal of Universal Computer Science*, 16(7):1036–1054.

[Lawrence and Rau-Chaplin, 2006] Lawrence, M. and Rau-Chaplin, A. (2006). Dynamic view selection for OLAP. In Tjoa, A. M. and Trujillo, J., editors, *Data Warehousing and Knowledge Discovery – DaWaK 2006*, volume 4081 of *Lecture Notes in Computer Science*, pages 33–44. Springer-Verlag.

[Lenz and Thalheim, 2009] Lenz, H.-J. and Thalheim, B. (2009). A formal framework of aggregation for the OLAP-OLTP model. *Journal of Universal Computer Science*, 15(1):273–303.

[Lewerenz et al., 1999] Lewerenz, J., Schewe, K.-D., and Thalheim, B. (1999). Modelling data warehouses and OLAP applications using dialogue objects. In Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., and Métais, E., editors, *Conceptual Modeling – ER'99*, volume 1728 of *LNCS*, pages 354–368. Springer-Verlag.

[Ma et al., 2008] Ma, H., Schewe, K.-D., Thalheim, B., and Wang, Q. (2008). Abstract state services. In Song, I.-Y. et al., editors, *Advances in Conceptual Modeling – Challenges and Opportunities, ER 2008 Workshops*, volume 5232 of *LNCS*, pages 406–415. Springer-Verlag.

[Ma et al., 2009a] Ma, H., Schewe, K.-D., Thalheim, B., and Wang, Q. (2009a). A service-oriented approach to web warehousing. In Kotsis, G., Taniar, D., Pardede, E., and Khalil, I., editors, *Proceedings of the 11th International Conference on Information Integration and Web-based Applications and Services (iiWAS 2009)*, pages 94–101. ACM and Austrian Computer Society.

[Ma et al., 2009b] Ma, H., Schewe, K.-D., Thalheim, B., and Wang, Q. (2009b). A theory of data-intensive software services. *Service-Oriented Computing and Applications*, 3(4):263–283.

[Ma et al., 2009c] Ma, H., Schewe, K.-D., and Wang, Q. (2009c). An abstract model for service provision, search and composition. In Kirchberg, M. et al., editors, *Proc. 2009 IEEE Asia-Pacific Services Computing Conference (IEEE APSCC 2009)*. IEEE.

[Papazoglou and van den Heuvel, 2007] Papazoglou, M. P. and van den Heuvel, W.-J. (2007). Service oriented architectures: Approaches, technologies and research issues. *VLDB Journal*, 16(3):389–415.

[Ravat et al., 2008] Ravat, F., Teste, O., Tournier, R., and Zurfluh, G. (2008). Algebraic and graphic languages for olap manipulations. *International Journal of Data Warehousing and Mining*, 4(1):17–46.

[Schewe and Wang, 2008] Schewe, K.-D. and Wang, Q. (2008). A customised ASM thesis for database transformations. submitted for publication.

[Schewe and Zhao, 2008] Schewe, K.-D. and Zhao, J. (2008). Typed abstract state machines for data-intensive applications. *Knowledge and Information Systems*, 15(3):381–391.

[Tanaka, 2003] Tanaka, Y. (2003). *Meme Media and Meme Market Architectures*. IEEE Press, Wiley-Interscience, USA.

[Thalheim, 2000] Thalheim, B. (2000). *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag.

[Thalheim, 2008] Thalheim, B. (2008). Model suites. In *Second International Workshop on Knowledge Cluster Systems*, pages 20–40. IOS Press.

[Theodoratos et al., 2001] Theodoratos, D., Dalamagas, T., Simitsis, A., and Stavropoulos, M. (2001). A randomized approach for the incremental design of an evolving data warehouse. In *Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling*, volume 2224 of *LNCS*, pages 325–338. Springer-Verlag.

[Theodoratos and Sellis, 1999] Theodoratos, D. and Sellis, T. (1999). Dynamic data warehouse design. In *Data Warehousing and Knowledge Discovery – DaWaK'99*, volume 1676 of *LNCS*, pages 1–10. Springer-Verlag.

[Theodoratos and Sellis, 2000] Theodoratos, D. and Sellis, T. (2000). Incremental design of a data warehouse. *Journal of Intelligent Information Systems*, 15(1):7–27.

[Thomson, 2002] Thomson, E. (2002). *OLAP Solutions: Building Multidimensional Information Systems*. John Wiley & Sons.

[W3C, 2003] W3C (2003). Web ontology language (OWL). http://www.w3c.org//OWL/.

[Wang and Schewe, 2007] Wang, Q. and Schewe, K.-D. (2007). Axiomatization of database transformations. In *Proceedings of the 14th International ASM Workshop (ASM 2007)*, University of Agder, Norway.

[Zaupa et al., 2008] Zaupa, F. et al. (2008). A service-oriented process to develop web applications. *Journal of Universal Computer Science*, 14(8):1368–1387.

[Zhao and Ma, 2006] Zhao, J. and Ma, H. (2006). ASM-based design of data warehouses and on-line analytical processing systems. *Journal of Systems and Software*, 79(5):613–629.

[Zhao et al., 2009] Zhao, J., Schewe, K.-D., and Köhler, H. (2009). Dynamic data warehouse design with abstract state machines. *Journal of Universal Computer Science*, 15(1):355–397.