Designing Uniform Database Representations for Cloud Data Interchange Services

Alina Andreica
Babes-Bolyai University, Cluj-Napoca, Romania

Keywords: Equivalence and Simplification Algorithms, Data Interchange, Cloud Services, Database Representation, Pattern Matching, Software Design.

Abstract: The paper proposes design principles for data representation in cloud data interchange services among various information systems. We apply equivalence algorithms and canonical representation in order to ensure the uniform representation in the cloud database. The solution we describe, proposed to be provided within cloud architectures, brings important advantages in organizational communication and cooperation, with important societal benefits. The generic design principles we apply bring important advantages in the design of the cloud interchange services.

1 INTRODUCTION AND WORKING FRAMEWORK

Within the framework of software design principles, based on systematic techniques, abstract patterns and adequate tools for problem solving, we propose means of using simplification and equivalence algorithms for modelling data representation. We apply these techniques in designing data interchange service among various information systems within cloud environments.

Equivalence algorithms can be implemented in an abstract manner, based on category theory (Andreica et al, 2012). Applying generic techniques is useful both for design reasons and for solving specific problems based on mathematical models (Andreica et al, 2012).

Interoperability is the capability of different systems to share functionalities or data (Olmedilla et al, 2006). System interoperability has been dealt with by means of various models (Morris et al, 2004) and has been extensively researched for business processes (Ziemann, 2010).

In (Andreica et al, 2015) we overview interoperability layers, principles and tools. Ones of the most relevant are: The Electronic Data Interchange (EDI) model (Adams et al, 2002), the XML standard (XML standard, 2015), RosettaNet (Rosetta, 2015), ebXML (OASIS, 2015) standards. Knowledge discovery, inference, logic are enabled by semantic interoperability.

Knowledge sharing over computer information systems, a major task for ensuring interoperability, is based on the Conceptual Knowledge Processing paradigm (Stumme and Wille, 2000).The Open Internet of Things standards (OpenIoT, 2015) may also be used as an efficient framework for data interchange.

Within section 2 we address means of implementing simplification and equivalence algorithms on various entities, including hierarchical structures. In section 3 we propose techniques for solving specific pattern matching problems using equivalence algorithms. Section 4 addresses principles of data interchange between information systems using a cloud database that retains data in a canonical representation. We propose structures for the entities and attributes to be used in the local and cloud databases. Conclusions reveal the most important topics presented in the paper and future research and development directions.

2 EQUIVALENCE ALGORITHMS

Within this section we present means of implementing equivalence algorithms (Buchberger and Loos, 1982) on various entities, including hierarchical structures. We use the implementation framework that we have introduced in (Andreica et al, 2012).

An equivalence relation '~' verifies reflexivity,

Andreica, A.
Designing Uniform Database Representations for Cloud Data Interchange Services.
DOI: 10.5220/0006354205820587
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 554-559
ISBN: 978-989-758-243-1

symmetry, transitivity properties (Buchberger and Loos, 1982).

Since the data volume that has to be processed is usually large, the most efficient way of organizing it is using a relational database, in which entities are retained in dedicated tables. Database structuring principles for processing equivalent entities are introduced in (Andreica et al, 2012).

We also use tables for retaining the entities on which equivalence algorithms are applied and we process entities belonging to the dedicated tables that retain those entities based on the following principle:

$$\text{IsSpecificEntity(d)} := \begin{cases} true, d \in Tbl\_Entity[\text{id\_entity}] \\ false, otherwise \end{cases}$$

Hierarchical data structures are often necessary to be processed in a database; such structures may be retained in relational databases by means of ascendant / successor pointers in dedicated tables (Andreica et al, 2010). Principles for retaining and processing hierarchical structures and a comparison of their processing techniques are presented in (Andreica et al, 2010).

In (Andreica et al, 2012) and (Andreica et al, 2010) we present means of processing hierarchical structures at database level, using a dedicated table for retaining the corresponding entities and a 4 pointers representation technique: ascendant, descendant, predecessor (same level), successor (same level).

In (Andreica et al, 2012) we discuss the case study of equivalent disciplines and in (Andreica, 2016) – the example of an expert system for plant therapy

In (Andreica et al, 2010) we detail the above mentioned principles for processing modules of didactic activities. The implementation uses stored procedures parameterized with the level value. The system uses a MS SQL database, the hierarchical structures which model curricula information being processed by means of stored procedures – see (Andreica et al, 2010) for details.

Postorder type n-ary tree evaluation algorithms using the above described tree representation are implemented in order to parse the hierarchy of entities.

Some efficiency studies we have performed on processing hierarchical structures at database level are presented in (Andreica et al, 2010).

In the hierarchical entity structure, leaf entities are retained in dedicated tables, based on the principle stated below:

$$\text{IsLeafEntity(m)} := \begin{cases} true, \forall d \in m : IsSpecificEntity[\text{d}] \\ false, otherwise \end{cases}$$

For example, in (Andreica et al, 2012) we process hierarchies of modules in which all non-leaf modules consist only of modules.

Let d1, d2 be two entities. We use the notation '~' for describing the equivalence of the two entities d1 ~ d2; this relation may have various significances in various case studies – see (Andreica et al, 2010), (Andreica et al, 2012). In each case, we have to check whether the relation is an equivalence one since by verifying if it complies reflexivity, symmetry, transitivity properties.

The canonical representative of an entity equivalence class is important since it will be further used in pattern matching rules – see section 3.

We implemented the simplification algorithm for determining the canonical representative for a given entity class (Andreica et al, 2015). Based on this algorithm, we may also test the equivalence of two entities by verifying they have the same canonical representative.

By generically denoting with '≈' an equivalence relation for categories of entities, we may state that:

$$e1 \approx e2 \Leftrightarrow (\forall d1 \in e1, \exists!\ d2 \in e2 : d1 \sim d2\ ) \wedge (\forall d2 \in e2, \exists!\ d1 \in e1 : d1 \sim d2\ )$$

For a leaf category of entities e, we consider Canonic (e) = {Canonic(d) | d∈e} – the set of canonical representative for the contained entities. It can be shown that two leaf equivalent entities have the same sets of canonical representatives.

For a category of entities we can recursively compute its canonical representative set as:

$$\text{Canonic(e)} = \begin{cases} \{\text{Canonic(d)} \mid \text{d} \in e\}, IsLeafEntity[e] \\ \{\text{Canonic(ed)} \mid \text{ed} \subset e\}, otherwise \end{cases}$$

Intuitively, the canonical set for a category of entities is obtained by "flattening" its category sub-tree and computing the union set of all canonical sets corresponding to its descendant leaf entities. Generically, we may state:

$$\text{Canonic (e)} = \{\text{Canonic(d)} \mid \text{d} \in \text{e}\}$$

3 PATTERN MATCHING PRINCIPLES

Within this section we refer to mappings between two elements as correspondences between the two elements that may be occur in various cases, usually named as pattern matching.

We implement pattern matching rules for equivalent entities by reducing the mapping between

two elements, belonging to the two equivalence classes that are to be mapped, to mapping their canonical representatives, as described below:

Let $e_i \in E$ class of equivalent entities, $em_j \in EM$ class of equivalent mapped entities, e_0 – the canonical representative of class E and em_0 – the canonical representative of class EM. Then we reduce a mapping of two entities e_i , em_j belonging respectively to the equivalence classes E, EM to the mapping between the two canonical representatives $e_0 \in E$, $em_0 \in EM$ – see Figure 1:

e_i -> em_j → e_0 -> em_0, where $e_i \in E$, $em_j \in EM$

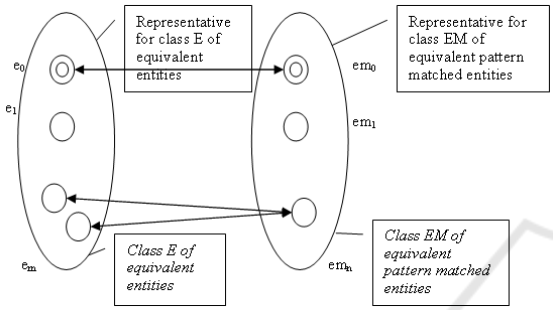

Figure 1: Pattern Matching Scheme for Equivalence Classes.

We may as well use the equivalent mappings:

e_i -> em_0, where $e_i \in E$,

(any element of E may be mapped into the canonical element of EM)

or

e_0 -> em_j , where $em_j \in EM$

(there is a mapping between the canonical element of E and any element of EM)

For the case of equivalence classes with hierarchical representations – see Figure 2 – the canonical representatives are the roots of the corresponding trees (Figure 2). Parsing algorithms for finding the canonical representatives generally use the ascendant pointer – see section 2.

We can use the above described rules in managing pattern matching problems on equivalence classes that occur in the design of expert systems.

We note that for the database representation it is important to improve table access speed table by indexing the tables in respect with the search id; this principle is very useful to be applied as well in managing hierarchical representations at database level, which are frequently processed in order to find the canonical representative.

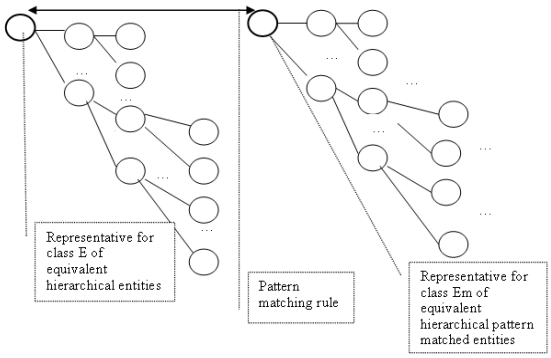

Figure 2: Pattern Matching on Hierarchical Structures of Equivalence Classes.

4 PRINCIPLES FOR BUILDING THE CLOUD UNIFORM DATABASE REPRESENTATION

The data interchange architecture (Andreica et al, 2015) provides data exchange services between various information systems or entities using cloud services and a cloud database for mapping and handling the exchanged information – see Figure 3. Data exchange may be performed both in XML relational database formats; for XML format, the corresponding database representation (Andreica et al, 2012) is generated into the cloud database. The following sequences are pursued: data to be exchanged is marked in the source database using dedicated tables and columns, mapped into the cloud database, sent and retained into the cloud database – see the structure proposed below. For the destination system / database, which sends data requests, a data mapping is also performed and required data is sent from the cloud database into the destination one.

The data exchange may use multi-criteria agents implemented in the cloud environment both for performing necessary mappings and for handling communication.

The *Cloud Data Interchange Services* will be designed for supporting automatic data exchange in various fields, with important communication efficiency benefits (Andreica et al, 2015).

The design principles are based on agent system development, the communication between agents being designed as a multi-agent system, since the multi-agent architecture (Weiss, 1999) provides many advantages, such as: decentralization, extensibility, robustness, maintainability, flexibility.

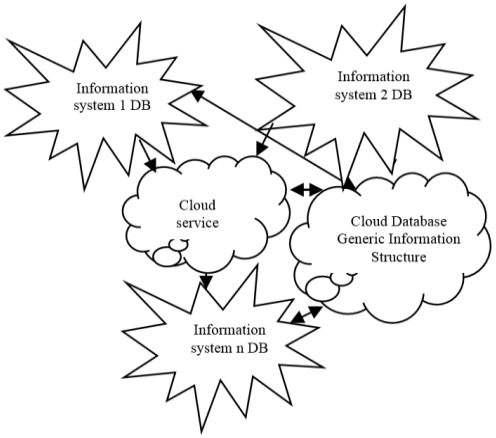

Figure 3: Data Interchange Model Using Cloud Services.

The agent architecture includes self-adapting communicating objects, which work on distributed datasets, supporting both the exchange and the analysis of distributed sources (Andreica et al, 2015).

Multi-agent systems (MAS) are appropriate for modelling heterogeneous interactions, based on flexible autonomous actions aiming at achieving specific goals. Ontologies and agent technologies may be combined in order to successfully enable heterogeneous knowledge sharing (Andreica et al, 2015).

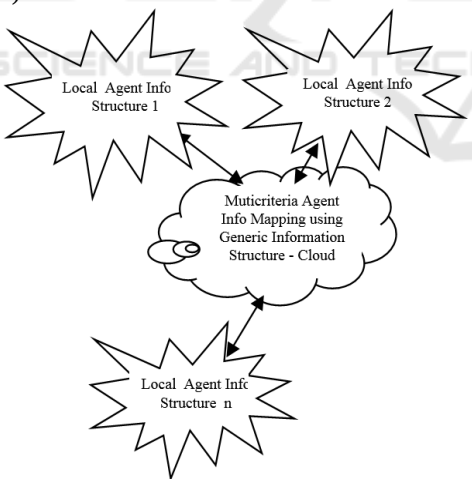

Figure 4: Agent Architecture Model for Data Interchange in Cloud Frameworks.

We use the multi-agent model proposed in (Faulkner et al, 2014) for modelling the agent interaction. The agent architecture implements local agents for gathering data from the communicating systems. Within the cloud environment, a multi-criteria agent handles information mapping between source and destination systems, using a generic information structure defined in the cloud database, as a canonical representation – see Figure 4.

This agent can be viewed as a mediator type of agent. A mediator may be defined as a system which refines, in a specific way, information from one or more sources (Wiederhold , 1992). A mediator embeds the knowledge which is necessary for processing a specific type of information and may also convert data to a common format (Chawathe et al , 1994). This mediator agent definition (Andreica et al, 2015) frame is given in Definition 1.

Agent:{ Local
Interface:
Data[require(*local_representation*)]
…
Effector[provide(*exchanged_items*)]
KnowledgeBase:
Source_Exchanged_Data
LocalDB_into_CloudDB
…
Capabilities:
Handle_Exchange_Request
… }

Definition 1: Mediator definition frame

These mediators model the interface between the destination systems and the cloud environment, having queries as inputs and returning objects via the interface layer. The proposed mediators are primarily hosted in the cloud environment, but they can also be used within client information systems (Andreica et al, 2015). They send dedicated queries in order to obtain appropriate objects from the remote sources, via the cloud environment.

In order to enable the exchange of various data between two information systems, using dedicated databases, with different structures, we set the canonical representation on the cloud database. The mapping process is user assisted since it requires human input.

We further propose a database mpdel for processing the entity equivalence from tables in various local databases using a uniform caninical representation in the clod database.

Let $\{E_1, E_2, ..., E_m\}$ be the class of entities to be processed from a local database:

$$E_1\ [A_1^{E1}, A_2^{E1}, ..., A_{n1}^{E1}]$$
$$E_2\ [A_1^{E2}, A_2^{E2}, ..., A_{n2}^{E2}]$$
$$....$$
$$E_m\ [A_1^{Em}, A_2^{Em}, ..., A_{nm}^{Em}]$$

We note $min=min\{Card(E_1), ..., Card(E_m)\}$.

If $min < n$ ie. $\exists i \in \{1,...,m\}$: $Card(E_i)<n$, there exist entities E_i, $i \in \{1,...,m\}$ which do not represent

all attributes that are taken into account into the canonical representation. In this case, we consider the corresponding attributes from the entities E_i, $i \in \{1,...,m\}$ to be Null.

If min > n, there exist entities E_i, $i \in \{1,...,m\}$ which contain attributes that are not represented into the canonical representation.

We further discuss the attribute equivalence from a local database, ie. the columns contained in the entity tables.

We note with A_1 the canonical representative for the class $\{A_1^{E1}, A_1^{E2}, ... A_1^{Em}\}$ containing attributes from the entities $\{E_1, E_2, ..., E_m\}$

We note with A_2 the canonical representative for the class $\{ A_2^{E1}, A_2^{E2}, ... A_2^{Em}\}$ containing attributes from the entities $\{E_1, E_2, ..., E_m\}$

....

We note with A_n the canonical representative for the class $\{ A_n^{E1}, A_n^{E2}, ... A_n^{Em}\}$ containing attributes from the entities $\{E_1, E_2, ..., E_m\}$

Then the entity $E_0[A_1, A_2 , ..., A_n]$ will be the canonical representative for the class of entities $\{E_1, E_2, ..., E_m\}$

When mappings between the entities and attributes from the various information systems databases are performed, we create equivalence tables in each **local information system database** containing:

EntityEquiv*[id_entities, LocalEntity_TableName, CloudEntity_TableName, Obs]*
AttributeEquiv*[id_attributes, id_entities, LocalAttribute, CloudAttribute, Type, Matched, Permissions, UpdatedFrom, Date, Significance, Obs]*

where
Type – is the type of the attribute
Matched is a logical value, representing whether the matching was performed
Permissions $\in$ {R, W} representing the permissions on the attribute, Read or Write
Significance is the significance of the Attribute
Obs = observations

UpdateList*[id_attributes, UpdatedFrom, Date, Obs]*
where
UpdatedFrom = the source database from which the update was performed (via the cloud database)
Date = the date of the update

Observations:

1. Back-ups for local databases are obviously recommended. In the cases in which local database administrators choose, dedicated tables or even databases (copies) may be used for the interchange process and then synchronizations can be managed locally.
2. In the case in which more cloud environments are used, the id of the cloud database has also to be retained.

The **cloud database** will contain the following equivalence structure:

CloudEntitiesEquiv*[id_ent, source, LocalEntity_TableName, CloudEntity_TableName, Significance, Obs]*

CloudAttributeEquiv*[id_attributes, id_entities, LocalAttribute, CloudAttribute, Type, Significance, Matched, Permissions, Obs]*

CloudUpdateList*[id_attributes, UpdatedFrom, Date, Obs]*

// the LocalAttribute with id_entities id is updated in the corresponding local database from the source specified in *UpdatedFrom*, at the specified *Date*

The fields have the same significances as described above, but are retained in the cloud database. The Date of the update is the date when the data is sent into a local database within the interchange process.

Observations:

1. The cloud table **CloudEntitiesEquiv** contains an entry for each entity / table from the local databases which are involved in the interchange process.
2. The cloud table **CloudAttributeEquiv** contains an entry for each atttribute from the local databases which are involved in the interchange process.

5 CONCLUSION AND FUTURE WORK

The paper proposes data representation and design principles for performing data interchange between various information systems databases by means of cloud services.

Simplification and equivalence algorithms are used in order to ensure canonical data representation in the cloud database and data correspondence in the data exchange process. We propose a specific structure for the entities / tables and attributes that are to be exchanged in the local and cloud databases.

The generic manner in which we implement simplification and equivalence algorithms on various

entities, including hierarchical ones, represented at database level, ensures generality and applicability in various cases. Pattern matching rules and canonical representatives are used in the cloud database.

We reveal the advantages of applying the algebraic equivalence algorithm and of applying canonical representatives' properties in solving pattern matching problems and designing data interchange services.

The data interchange model we present provides important practical advantages for increasing organizational competitiveness, with a significant societal impact on institutional and entities' cooperation, efficient information access and management for various stakeholders. A relevant advantage of the solution is its flexibility and efficiency in information exchange (only relevant data is exchanged), with minimal resources involved and significant security benefits.

Future work is related to further development and implementation of the above described techniques.

REFERENCES

Andreica, A, Stuparu, D, and Miu, C. (2012). *Applying Mathematical Models in Software Design*, 2012 IEEE 8th International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, Proceedings of ICCP 2012, IEEE, Ed: Ioan Alfred Letia, p.87-90

Olmedilla, D., Saito, N., and Simon, B. eds. (2006). "Educational Technology & Society", Special Issue on Interoperability of Educational Systems, vol 9

Andreica, A, Stuparu, D, and Miu, C., (2010). "Design Techniques in Processing Hierarchical Structures at Database Level", *Proceedings of Iadis Information Systems 2010*, Porto, 18-20 March 2010 , IADIS Press, Ed: M Nunes, P Isaias, P Powell, p. 483-488

Ziemann, J. (2010). "*Architecture of Interoperable Information Systems - An Enterprise Model-Based for Describing and Enacting Collaborative Business Processes*", Logos Verlag, Berlin

Andreica, A, Covaci, F. and Küng, J. (2015). *A Generic Model for Cloud Data Interchange*, Proceedings of 14h RoEduNet International Conference - IEEE, Craiova, 24-26 September 2015, IEEE Computer Society, p. 138-142

Adams, S, Hardas, D, Iossein, A. and Kaiman,. C. (2002) *BizTalk Unleashed*. Indianapolis, Indiana: Sams Publishing. p. 966

XML standard, http://www.w3.org/TR/xml11/#charsets , retreived Nov 2015

Rosetta – Rosettanet Overview: Clusters, Segments, and PIPs (ver 02.13.00), 2011, retrieved December 2015 http://www.rosettanet.org/TheStandards/RosettaNetStandards/PIPOverview/tabid/3482/Default.aspx

OASIS – OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features, 2007. http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.pdf, retreived Dec 2015

Stumme, G., Wille, R. (2000). „*Begriffliche Wissensverarbeitung / Conceptual Knowledge Processing*", Springer Verlag,

OpenIoT - Open Internet of Things architecture https://github.com/OpenIotOrg/openiot/wiki/OpenIoT-Architecture , retreived Dec 2015

Buchberger B. and Loos, R. (1982) *Algebraic Simplification*, Computing, Suppl. 4, Springer Verlag, p.11-43

Andreica, A (2016) *Applying Equivalence Algorithms in Solving Pattern Matching Problems. Case Study for Expert System Design*, Proceedings of International Conference on Theory and Practice in Modern Computing- TPMC, July 1-4, 2016, Portugal

Andreica, A, Stuparu, D. and Mantu, I. (2005). "Symbolic Modelling of Database Representations", *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 2005*, IEEE Press, p 59-62

Morris, E., Levine, L, Meyers, C., Place, P. and Plakosh. D. (2004) "*System of Systems Interoperability (SOSI): Final Report*", Carnegie Mellon Univ., Software Engineering Institute, http://www.sei.cmu.edu/reports/04tr004.pdf , accessed May 2016

Weiss, G. E, (1999). "*Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*", MIT,

Faulkner, S, Kolp, M, Nguyen, Tai, Coyette, A, Do, T. (2014). "*Information Integration Architecture Development: A Multi-Agent Approach*", retr Nov 2016

Wiederhold, G, (1992). "*Mediators in the Architecture of Future Information Systems*", IEEE Computer, 25:38-49

Chawathe, S, Garcia-Molina, H, Hammer, J,.Ireland, K, Papakonstatinou, Y, Ullman, J, Widom,. J, (1994). "*The TSIMMIS Project: Integration of Heterogeneous Information Sources*", , retreived Jan 2015 http://ilpubs.stanford.edu:8090/66/1/1994-32.pdf