Enforcing Data Quality Rules for a Synchronized VM Log Audit Environment Using Transformation Mapping Techniques

Sean Thorpe[1], Indrajit Ray[2], and Tyrone Grandison[3]

[1] Faculty of Engineering and Computing, University of Technology, Kingston, Jamaica
thorpe.sean@gmail.com
[2] Colorado State University, Fort Collins, USA
indrajit@cs.colostate.edu
[3] IBM Almaden Research Silicon Valley, USA
tyroneg@us.ibm.com

Abstract. In this paper we examine the transformation mapping mechanisms required in synchronizing virtual machine (VM) log audit data for the system administrator environment. We explain the formal constraints that are required by the transformation mapping process between the source and target log schemas for these VMs. We discuss the practical considerations of using these formalisms in establishing the suitable data quality rules that provides for security within these abstract domains.

Keywords: Transformation, Mapping, Log, Quality, Audit.

1 Motivation

Often when one thinks about the virtual machine cloud environment one sees economies of scales for the business. To this end enabling security across these domains are critical. The integrity of the disparate data sources however has proven a major interoperability challenge within these hybrid compute clouds. Hence efforts to provide data quality as apart of the VM data centre needs to be meticulous. Customarily data integration systems are automated systems that permit the transformation, integration, and exchange of structured data that has been designed and developed independently. The often subtle and complex interdependencies within data can make the creation, maintenance, and use of such systems quite challenging for database security. When one thinks about these traditional challenges as a new and urgent concern within the cloud computing arena the issues become even more problematic.

There are five (5) sections in this paper. Section 2 looks at the related work. Section 3 looks at the formal transformation mapping constraint definitions. Section 4 provides a case study analysis of the security challenges in the transformation mapping processes for our log auditor deployment. Section 5 summarizes the conclusion and future work in our research.

Á. Herrero and E. Corchado (Eds.): CISIS 2011, LNCS 6694, pp. 266–272, 2011.

2 Related Work

Mapping composition and schema changes are fundamental operations in modeling consistent data quality for the traditional database systems. In this work we reference the semantics for mapping composition that was introduced by Fagin et. al [3]. Schema Mappings, and declarative constraints that model relationships between schemas, are the main enabler of data integration and data exchange. They are used to translate queries over target schema into queries over source schema or to generate executable transformations that produce a target instance from a source instance. These transformations are generated from the logical specification of the mappings. Such schema mappings are normally generated semi-automatically from using well established mapping tools like Clio, BEA Aqualogic [4][5]. The complexity of large schemas, lack schema documentation, and the iterative, semi-automatic process of mapping and transformation generation are common sources of errors. These issues are compounded by the nuisances of mapping tools (which can produce supposedly a wide variety of transformations is far from trivial and often time consuming and expensive). In addition, schema mapping is often done over several data sources that are themselves dirty or inconsistent. We contend that errors caused by faulty data cannot be easily separated from errors caused by an incorrect or inconsistent mapping and/or transformation. In the context of a synchronized VM environment, such inconsistencies are multiplied ten fold.

3 Transformation Mapping Constraints

If this section we identify the types of mapping constraints and how we apply enforcements to over come these constraints for enabling data quality within the synchronized virtual machine environment. We articulate these constraints as a set of formal mathematical definitions.

Definition 1: Rules $R_i.R_j \in \Re$ are mutually inconsistent if

1) $\forall A_k \in A \;\; v\,(R_i.A_k) = v\,(R_j.A_k)$

2) $v\,(R_i.C) \neq v(R_j.C)$

Informally condition 1 in the above definition states that all decision making VM attribute values of R_i are the same as the corresponding attribute values of rule R_j and condition 2 states the category attribute value of rule R_j. Note that one assumes that decision making attributes will be in the same order for all rules, a condition that can easily be satisfied.

One has indirect inconsistency if the two rules are present in different policy sets lead to contradictory conclusions.Such inconsistencies are difficult to see because they may not be visible at the time of defining policies and can only be triggered only when some specific event occur. For example on the VM server a Professor T is allowed to create a student exam account and Mary a student is allowed to delete accounts. A policy may state that create and delete operation cannot be performed by the same entity or identity. Inconsistency could occur if Professor T delegates his rights to Mary. However from the perspective of having a synchronized VM log audit policy that can perform data mining on the attribute identities , one could formally define inconsistency in the following manner.

Definition 2: Algorithm 1.0. VM Node Inconsistency Detection Algorithm
Input: Decision tree
Output: Context of VM node inconsistency

```
1:    Let A(bi) be the set of all attributes present in one branch.
2:    Bool consistent = true;
3:     for each branch bi in Decision tree do
4:     if more than 1 category attribute is assigned to terminal
       node bi.tnode then
5:     A(bi) = fetch all attributes of brah(bi);
6:     for each actual rule Ra in the policy set do
7:    if υ(A(Ra)) = υ(A(bi)) then
8:    Highlight: Ra : A1  . . . An → C;
9:    end if
10:   end for
11:    consistent = false;
12:    end if
13:    end for
14:   if consistent = true then
15:   No inconsistency found;
16:    end  if
```

In such a decision tree, each branch *bi* (from the root to a VM terminal node) represents one rule. In order to detect inconsistency, one will apply the above algorithm. First one checks the terminal node of each branch (Lines: 3-4). If any terminal node *tnode* contains more than one category (*C*) attribute value (Line: 4), this means that some rules in the policy set are mutually inconsistent. In order to determine which particular rules in the VM policy are mutually inconsistent, first one fetch all the attributes of the particular branch (Line: 5). After that the algorithm will start searching the attribute-values in the actual VM policy set (Lines: 6-10). All the rules in the policy set that contain those attribute-values will be highlighted as inconsistent (Lines: 7-9). If in a decision tree, no terminal node has more than one category attribute-value then this means that no inconsistency has been found in the policy set (Lines: 14-16).

To eliminate the inconsistency constraint in definitions (1) and (2) above, we define the notion of association to describe a set of associated atomic type schema elements in our transformation mapping mechanism. Intuitively, an association is a query that returns all the atomic type elements in a Log query.

Definition 3: A VM Log mapping system is a triple $< V_{LS}, VT_{LS}, M >$ where V_{LS} and VT_{LS} are the source and target schemas and M is a set of mappings between V_{LS} and VT_{LS}.

Definition 4: A schema element in schema V_{LS} is a path query , that is query of the form:

$$\underline{select}\ e_{n+1}\ \underline{from}\ x_0\ \underline{in}\ L_0\ ,\ x_1\ \underline{in}\ L_1 \ldots\ldots x_n\ \underline{in}\ L_n$$

where each L_k with $k \geq 1$ uses variable x_{k-1}, L_0 is an expression starting at a Log schema root in V_{LS} and expression e_{n+1} uses variable x_n. If the details of the from clause are unimportant, we refer to a Log schema element using the notation select e from L.

Constraints. For Log schema constraints we consider a very general form of referential constraints called nested referential integrity constraints (NRIs) [21] extended to support choice types. NRIs capture naturally relational foreign key constraints.

Definition 5: An association is a query on a VM log schema V_{LS}

$$\begin{array}{l} \underline{select}\ e_{n+1}\ \underline{from}\ x_0\ \underline{in}\ L_0\ ,\ x_1\ \underline{in}\ L_1 \ldots\ldots\ldots x_n\ \underline{in}\ L_n \\ \underline{where}\ e_1 = e^1{}_1\ \underline{and}\ e_2 = e^1{}_2\ \underline{and} \ldots\ldots\ldots\ e_n = e^1{}_n \end{array}$$

Definition 6: A mapping is a constraint for each A^{VLs} exist A^{VTs} with C, where A^{VLs} is an association on a virtual machine log source schema V_{LS} and A^{VTs} is an association on a virtual machine log target schema VT_{LS} and C is a conjunction of equality conditions relating atomic type expressions over V_{LS} with atomic expressions over VT_{LS}.

Definition 7: A correspondence is a specification that describes how the value of an atomic target VM log schema element is generated from the VM Log source schema. A correspondence can be represented as simple inter-schema referential constraints. A correspondence from a source element select e^{VLS} from from L^{VLS} to a target element select e^{VTLS} from from L^{VTLS} is an inter-schema NRI for each L^{VLS} exist L^{VTLS} with $e^{VLS} = e^{VTLS}$. Correspondences are implicit within the mappings (and view definitions) can be easily extracted from them.

Definition 8: A VM Log association A is dominated by association **B** (noted as $A \prec B$) if there is a renaming function h from the variables of A to the variables of B such that the from and where clauses of h(A) are subsets, respectively of the from and where clauses of B.

Domination can naturally extend to mappings as follows. VM log Mapping m_1: for each $A^{VLs}{}_1$ exist $A^{VTs}{}_1$ with C_1 is dominated by mapping m_2 for each $A^{VLs}{}_2$ exist $A^{VTs}{}_2$ with C_2 (denoted as $m_1 \prec m_2$) if $A^{VLs}{}_1 \prec A^{VTs}{}_2$ and for every equality condition $e = e^1$ in C_1, $h_1(e) = h_2(e^1)$ is in C_2 (or implied by C_2) where $h_1(e) = h_2(e^1)$ are renaming functions from $A^{VLs}{}_1$ to $A^{VTs}{}_1$ and from $A^{VLs}{}_2$ to $A^{VTs}{}_2$ respectively.

There are three (3) ways in which semantic relationships between schema elements can be encoded. The first is through the structure of the schema. Elements may be related by their placement in the same record type or more generally through parent child relationship in nested schemas. An association containing elements that are related only through the schema structure is referred to as a structural association. Structural associations correspond to the primary paths used in where it is shown that they can be computed by one time traversal over this log schema.

Definition 9: A structural VM Log association is an association

select e_{n+1} *from* x_0 *in* L_0 , x_1 *in* L_1 ……..x_n *in* L_n with no where clause and where the expression L_1 must start at a schema root and every expression L_k , $k > 0$ starts with variable x_{k-1} .

The schema structure encodes a set of VM Log semantic relationships that the VM administrator chose to model explicitly. A second way of encoding semantic associations is in a mapping. A mapping is an encoding of a pair of source and target associations (which may or may not be explicitly present in the VM Log schema structure. A VM Log mapping may expose hidden semantic relations between schema elements. In our case study on same our software prototype of the VM Log auditor acts as such a mapping tool.

Definition 10: Let M be a set of given VM Log schema mappings. A user association is an association that has been provided to the system via a mapping $m \in M$.

Definition 11: A Logical association R is the result of chasing a structural or a user association L with the set S of all the NRIs of the VM Log schema (denoted as $chase_x(L)$)[1].

Definition 12: Chasing is a classical relational method that can be used to assemble elements that are semantically related to each other through constraints. In other words this should be observed as schema constraint against the set of all the VM Log related elements. A chase is a series of chase steps. A chase step of association say R with an NRI Z : for each W exist Y with C, can be applied if, by definition, the association R contains (a renaming of) W but doesn't satisfy the constraint, in which case the Y clause and the C conditions (under respective naming) are added to the association. The chase can be used to enumerate logical join paths, based on the set of dependencies in the VM Log schema. We use a variation of a nested chase that can handle choice types NRIs. We define more formally an extended version of the chase in our next paper.

Definition 13: Let V_{LS} and VT_{LS} be the pair of VM Log source and target schemas and M a set of VM Log mappings between them. Consider C to be the set of correspondences specified by mappings in M. A semantically valid VM Log mapping is an expression of the form for each A^{VLs} exist A^{VTs} with D , where A^{VLs} and A^{VTs} are logical associations in the source and target schema correspondingly, and D is the conjunction of the conditions of the correspondences in C that are covered by the pair $< A^{VLs}$, $A^{VTs} >$ (provided that at least one such correspondence exist.) A correspondence v : for each L^{VLS} exist L^{VTLS} with D is covered by the associations $< A^{VLs}$, $A^{VTs} >$ if $L^{VLS} \lhd A^{VLs}$ and $L^{VTLS} \lhd A^{VTs}$.

Definition 14: Given a source and target VM Log schema V_{LS} and VT_{LS} along with the set of mappings M from V_{LS} to VT_{LS} , a VM Log mapping universe $U^M_{V LS, VTLS}$ is the set of all semantically valid mappings.

4 Case Study Evaluation

At the University of Technology (UTECH) we demonstrate the design of a synchronized virtual machine log auditor using VMWare essx3i data centre. Our goal

in this research was to guarantee data consistency between the source and target schemas as a data quality concern. We achieve this by applying transformation mapping to our log auditor. The auditor runs a LOADER script, which happens to be an Oracle stored procedure that reads the source schema after we had successfully mapped it to our staging database. At the staging area, the LOADER script scans the column values of the source and matches its attribute parameters before copying it to the auditor's target database. However there are cases when an associatively mapped VM instance, shows different transformation attributes from what is registered in the auditor's logical target schema. This observation reflects changes in the source schema composition due to adaptation at the source VM host. For example we noted this as a concern, when data sets were mapped from a new test VMware host in the production centre. The CPU World ID strings appeared to have had a concatenated string length as compared to it's original attribute values on the current production host VM. This resulted in null column value update on the target. We treat this as a referential integrity constraint error at the target, and record such transformation events as a new and unknown within the VM. Realistically, this may be considered an inconclusive observation, as the attribute type is legitimate, it just had an unknown format to our auditor's target schema. In this context these results will subject the auditor to its own lack of integrity, due to human misunderstanding of these database constraints or some other anomaly. Hence we have started to look at the various intelligent adaptation and inference mechanisms to handle this log schema concern. One approach is the use of chasing methods in our transformation, by running a parser that periodically checks column formats and updates the auditor of any semantic changes. Ongoing work explores this chasing technique not only as a function of our static log monitoring auditor but also as dynamic monitoring parameter for the synchronized virtual machine environment. We still don't have any conclusive empirical study on these transformations, as our work is still in its preliminary stages. We however are testing application log profiles to determine the different real issues of these transformation formatting arguments.

5 Conclusion and Future Work

We have presented a formal theoretical approach on how we can establish virtual machine log synchronization using transformation mapping mechanisms. We relate from our case study deployment of our log auditor prototype how these transformation mapping mechanisms prove critical to data quality assurance and invariably the security within such logical domains. Our ongoing work assesses the transformation mapping issues for both the homogenous and heterogeneous VM cloud environments.

References

1. Grandison, T., Maximillen, E.M., Thorpe, S., Alba, A.: Towards a Formal Cloud Computing definition. In: Proceedings of IEEE Services (July)
2. Thorpe, S., Ray, I.: Global Virtual Machine Policy Auditor. CSU PhD Discussion Forum (September 2010)

3. Fagin, R., Kolatis, P., Miller, R.J., Popa, L.: Data Exchange Semantics and Query Answering. Theory of Computer Science 336(1), 89–124
4. Miller, R.J., Haas, L.M., Hernandez, M.: Schema Mapping as a Query Discovery. In: VLDB, pp. 77–88 (2000)
5. Chiticariu, L., Tan, W.: Debugging Schema Mappings with routes. In: VLDB, pp. 79–90 (2006)
6. Van den Bussche, J., Vansummeren, S., Vossen, G.: Towards Practical Meta Querying. Information Systems 30(4), 317–332 (2005)