View integration

Herman Balsters, Maarten Fokkinga, and Maurice van Keulen
University of Twente, Dept INF
{balsters,fokkinga}@cs.utwente.nl, M.van.Keulen@bigfoot.com

Working draft — Version of December 10, 1999

Problem statement Suppose that we are to build *one* information system (database, say) for *several* users that each have their own view of the world. The idea is to *integrate* the views of all individual users into one view (which is satisfactory for all users), and build the information system for that view. The problem that we are faced with is this:

What is the/a formal definition of *view integration* that serves this purpose?

We tackle this question, dealing with integration on the schema level as wel as on the world/extension level, and we will make explicit various assumptions. It remains to be investigated whether our notion of view integration (with the motivation above) is also suitable for federated databases and datawharehousing.

Preliminaries As a starting point we assume that a view V consists of an ER schema S together with some constraints C; thus $V = (S, C)$. (Part of C may have a graphical notation, this is particularly true for the so-called cardinality or multiplicity constraints.) We assume that it is clear that, and how, an ER schema S defines a logical language $\mathcal{L}(S)$ (consisting of *terms* and *predicates*, collectively called *expressions*). For example, $\mathcal{L}(S)$ is the language of set expressions (the Z notation) with the entity and relation names of S as given set names, where the elements of these sets have the attributes as given in the schema. The constraint C does not have an influence on $\mathcal{L}(S)$. A query q is a term in the language $\mathcal{L}(S)$. We do not consider updates.

A *world* for $V = (S, C)$ is on the hand a mathematical structure w consisting of (at least) a so-called entity extension and relation extension for each entity and relation type mentioned in S, and on the other hand an interpretation I of $\mathcal{L}(S)$ into w. We use the conventional notation "$w \models \phi$" to denote the truth value of predicate ϕ when interpreted into w (thereby not mentioning the interpretation I explicitly). An *instance* or *materialisation* or *model* of V is a world w for V that satisfies all constraints of V, that is, $w \models C$. The set of all *instances* of V is denoted $[\![V]\!]$, so $[\![V]\!] = \{w : \textit{world for } V \mid w \models C\}$.

If w is a world for $V = (S, C)$, then we write $\mathcal{L}_w(S)$ for the language that is obtained from $\mathcal{L}(S)$ by adding constants for the elements of the sets in w. From now on we consider queries and answers to be terms in the language $\mathcal{L}_w(S)$, where an answer is a term built entirely from constants. To express the semantics of a query, one can consider $w \models q{=}a$ for various answers a. However, since '$q{=}a$' is a predicate, we can abstract away from queries and answers (which are terms) and consider predicates ϕ only. This generalisation is not too far, since arbitray predicates can already be represented by queries (if the language is

sufficiently expressive), via the mapping $\phi \mapsto \{1 \mid \phi\}$: the answer set is non-empty if and only if the predicate ϕ holds.

Assumption Our investigation does not consider the important, hard, practical problem of removing conflicts and anomalies between the two views: inconsistencies, synonyms, homonyms, and so on. So, when integrating two views V_1 and V_2, we assume that the same names have the same semantics, both intuitively and formally. Specifically, if E is an entity type in both views, then in each world w for V_1 (not necessarily instance of V_1) we have that the entity extension for E in w is also the entity extension for E in some world for V_2, and conversely.

View integration Our first tentative definition for view integration reads as follows. View V is, intuitively, the integration of views V_1 and V_2 if, firstly, both V_1 and V_2 can be mapped into V without loss of information, and secondly, if another V' has this property too, then V in turn can be mapped into V' without loss of information. Writing $\preceq$ for "can be mapped, without loss of information, into", the tentative definition reads formally:

$$V_1 \preceq V \wedge V_2 \preceq V \wedge (\forall\, V' \mid V_1 \preceq V' \wedge V_2 \preceq V' \bullet V \preceq V')$$

that is, V is the least upper bound of V_1 and V_2 wrt $\preceq$

In order for this definition to make sense, we should provide a definition for $\preceq$ and prove that 'the least upper bound' exists. The definition is given below; I doubt whether lub's exists...

Now we make the following assumption (leading to a relaxation of the definition of view integration). In practice, a view is used to express a *part* of the world formally. So, inherently, the user knows that the world is larger than just the part he is interested in. Therefore, the user will (must!) consider any upperbound V' of his view to be as good as the view V he has in mind, since *view V can be mapped without loss of information into view V'*. Consequently, we may relax the proposed definition somewhat:

view V is *an* (rather than *the*) integration of V_1 and V_2 if: $\quad V_1 \preceq V \quad \wedge \quad V_2 \preceq V$.

In this case, there is no need for lub's to exist.

Regarding relation $\preceq$, we propose to define $V \preceq V'$ ("V can be mapped into V' without loss of information") as follows:

there exists a (type-preserving) function f mapping $\mathcal{L}(S)$ into $\mathcal{L}(S')$, and mapping $[\![V]\!]$ into $[\![V']\!]$, such that

(*) for all instances $w \in [\![V]\!]$ and predicates ϕ in $\mathcal{L}_w(S)$, $w \models \phi \Rightarrow f(w) \models f(\phi)$.

[Consequently, we have $f(w) \models f(C) \wedge C'$.]

Function f can be very wild: it may do renaming, double all numbers, add superflous attributes, remove derivable attributes, replace relations by entities and conversely, and so on. What matters is that f also translates all observations ϕ while preserving the meaning. When an application has been built on view V, and later is forced to use V', then it should invoke f so as to translate everything to the new view.

Note: should '$\Rightarrow$' in (*) be replaced by '$\Leftrightarrow$'?

Integrating materialisations Above we have defined the integration of two views, implying that each view instance *on its own* can be transformed into an instance of the integrated view, without loss of information. Now, we also need to say how the instances of the the two views *jointly* can be mapped into the integrated view. This is needed to "fill the database" according to the world conceptions of both users jointly. In doing so, it may happen that valid predicates (e.g., the constraint!) of a given instance fail to be valid in the integrated instance: indeed, the world changes, and some information loss may occur.

Here are two proposals:

> Let V be an/the integration of V_1 and V_2, say via f_1 and f_2, respectively.
> Then, for any $w_1 \in [\![V_1]\!]$ and $w_2 \in [\![V_2]\!]$, take
>
> $$w = f_1(w_1) \cup f_2(w_2), \quad \text{or}$$
> $$w = \text{'natural full outer join' of } f_1(w_1) \text{ and } f_2(w_2)$$

Here, the union $\cup$ is meant to denote union of 'equally named' entity extensions in the two worlds, union of 'equally named' relation extensions of the two worlds, and incorporation of all the extensions of the two worlds. Likewise, the natural full outer join has to be taken per relation. A proper formalisation requires more detail about the ER schemas S, the induced language $\mathcal{L}(S)$ and the $\mathcal{L}(S)$-related mathematical structures w. Futhermore, with the join-version, partiality (null values) are going to play a role, and that might complicate matters a lot.

Note also that "the integrated instance" is *not* determined uniquely: there may be several mappings f_1 for $V_1 \preceq V$, and several mappings f_2 for $V_2 \preceq V$.

Theorem: Let V be an/the integration of V_1 and V_2, say via f_1 and f_2, respectively. Let $w_1 \in [\![V_1]\!]$ and $w_2 \in [\![V_2]\!]$, and w defined as above. Then, even though $w_1 \models f_1(C_1)$, it is not necessarily true that $w \models f_1(C_1)$.

In other words, relaxation of constraints is, in general, unavoidable when integrating materialisations. Even though each user *on his own* can transform his view and materialisation into V, they cannot do so jointly without making concessions to each other: some loss of information may occur!

Still to be investigated Here are some points for further research.

About the definition of $\preceq$:

1. Is *injectivety* of f a necessary and sufficient condition for (*)? Remember that a function is injective if and only if it has an inverse (partial function).

2. Under what condition does '$\Leftrightarrow$' hold in (*) instead of only '$\Rightarrow$'?

3. It seems reasonable to require that the language mapping part of f is a homomorphism, that is, the image of a composed expression is built from the images from the constituent parts. In particular, it is *very* reasonable to require f to be a homomorphism for the logical operations: $f(\phi \wedge \psi) = f(\phi) \wedge f(\psi)$, and so on.

About integration of materialisations:

4. What are the weakest, additional, constraints for views V_1 and V_2, such that materialisation integration *does* preserve the entire constraint?

5. Let V be an/the integration of V_1 and V_2, say via f_1 and f_2, respectively. Let $w_1 \in [\![V_1]\!]$ and $w_2 \in [\![V_2]\!]$, and w defined as above (in particular, via the join-method). Is it true that $w \models f_1(C_1) \vee f_2(C_2)$? What can we say when C_1 and C_2 are tuple constraints. There seems to be no stronger, true property of $f_1(C_1)$ and $f_2(C_2)$ for class constraints C_1 and C_2. A proper fomalisation requires to introduce structure in the language, and distinguish between tuple predicates and class predicates.

6. What is the "smallest" world that can be considered an integration of w_1 and w_2? The one in the second alternative (in the definition of 'integrating materialisations')?

   About the applicability

7. What about updates?

8. What about federated databases, and datawarehousing?

   Miscellanea

9. Make 'world' into a synonym for 'instance, materialisation, model', and use another word for what currently is called 'world'.

10. Work out a (surprising) example.

11. The references below need to be read and compared with our approach.

References

[1] S.S. Chawathe, H. Garcia-Molina, and J. Widom. A toolkit for constraint management in heterogeneous systems. In *12th International Conference on Data Engineering, New Orleans*, Montvale, NJ, 1996. IEEE Press.

[2] M.P. Reddy, B.E. Prasad, and A. Gupta. Formulating global integrity constraints during derivation of global schema. *Data and Knowledge Engineering*, 16:241–268, 1995.

[3] Mark Vermeer. *Semantic Interoperability for Legacy Databases.* PhD thesis, University of Twente, Enschede, Netherlands, 1997.