Bridging the Gap between Comparison and Conforming the Views in View Integration

Peter Bellström

Department of Information Systems
Karlstad University, S-651 88 Karlstad, Sweden
Peter.Bellstrom@kau.se

Abstract. View integration is a complex, error-prone and time-consuming task. Therefore there is a need to decompose the integration methods into smaller well defined phases where different techniques are applied. Most of the methods used today is composed of, or at least is a mixture of, the following four phases: pre-integration, comparison of the views, conforming the views and merging and restructuring, and most of the methods put focus on phase 2 and 3. Despite of this there is a gap between these two phases. To bridge this gap a framework has been developed. In the framework Inference Rules (IR) are used, together with Enterprise Modeling (EM) as the canonical modeling language, to deduce dependencies that are not conflicting from dependencies that are or have been identified as an inter-schema property. Three research questions regarding why, how and when IR should be applied in view integration are analyzed and discussed and it is argued that the framework may not only improve view integration as such but also improve the applied conflict and inter-schema resolution techniques.

1 Introduction

To be able to design a database that is understandable and correct, a database design method has to be used. A database design method has at least three phases: *conceptual database design*, *logical database design* and *physical database design* (e.g. [3, 29]). Conceptual database design is the most critical phase and will probably continue to be so [3, 8]. This is the case since it highly influences the rest of the design and is in this paper seen as divided into two distinct parts: *view design* and *view integration*. Several methods and models, including graphical representations, have been proposed over the last twenty years. The Entity-Relationship (ER) modeling language, with its ER diagrams was first proposed by Chen [7] in the mid 1970's and has since then become one of the most popular and commonly used modeling languages [12, 28] and is almost seen as a de facto standard for conceptual database design [27]. The success of ER mainly depends on four things. Firstly, ER has been extended (generally termed Extended ER) and used in several methods (e.g. [11, 29]). Secondly, ER has concepts naturally occurring in database design [19]. Thirdly, ER is useful for communicating different definitions of data and relationships with the end users [29]. Finally, ER is a good modeling language for defining constructs like the unary and ternary relationships [24]. Although ER has been widely used and has proved to be a good modeling language it has been criticized for its generally use of the relationship construct [11]. ER has also been questioned regarding its use as a canonical modeling language in view integration since it is not clear if a type should be transformed into an entity type or a relationship type [18]. Finally, ER lacks the very important instance-of dependency needed to illustrate classification. A modeling language that may bridge these shortcomings is EM.

In EM a dependency (relationship) is classified as association, composition, aggregation, specialization, generalization or instance-of (see Figure 2). This approach gives the designer an opportunity to define a clear view of the dependencies between the concepts in the views and schema. Another positive aspect with EM is that not only semantic and pragmatic dependencies but also syntactic parts [16] of the future database may be defined and graphically illustrated. The big challenge during conceptual database design is to create a global conceptual schema that is semantically correct, complete, easy to use and comprehensive [24] using the primitives and constructs of the chosen modeling language [28]. One way to achieve this is to involve the end users and define views for each end user or user group, to conduct view design. The views are then integrated, in a task called view integration, into one final and global conceptual schema. View integration, a very critical part of database design [30] is one of two paths in schema integration; database integration is the other one. Schema integration is defined by [2] as ”[...] the activity of integrating the schemas of existing or proposed databases into a global, unified schema.”. View integration is a complex task because different end users may define their own view of the organization, a phenomenon often called semantic relativism [27]. This means that we often have to deal with different representations and interpretations of the same concept.

Although many view integration methods have been proposed during the last twenty years view integration needs continual refinement and reevaluation [30]. This is motivated by the fact that view integration is a complex, time-consuming and error-prone task [22]. Many of the proposed methods use the four phases identified by [2]. However, in this paper it is argued that one phase is still missing henceforward called *pre-conforming the views*. This phase is motivated since a lack of understanding regarding the gap between the second and third phase exists. Earlier methods also lack in the use of IR during conforming the views where conflicts and inter-schema properties are resolved. Addressing the described gap is important because conflict resolution has been ranked as the key issue in view integration [27]. Another motivation to incorporate the proposed phase is the need to decompose view integration into smaller phases not only because it is a complex task but also because there is a need to unambiguously define and describe each phase.

Therefore, the aim of this paper is to present a framework where IR are applied together with EM to bridge the gap between comparison and conforming the view in view integration. The following three research questions are used to reach the aim: W*hy*, *how* and *when* to apply IR in view integration.

The organization of this paper is as follows. Firstly, view integration methods and approaches are described and discussed. Secondly, a subset of EM is discussed in the context of view design and view integration followed by a description of IR. Fourthly, IR are discussed in connection with the three research questions about why, how and when to apply IR in view integration. Finally, the paper is summarized and conclusions are presented.

2 View Integration Methods and Approaches

Several view integration methods and approaches have been proposed over the last twenty years. In the study performed by [2] the authors concluded that a view integration method is composed of, or at least a mixture of, the following four phases: *pre-integration*, *comparison of the schemas*, *conforming the schemas* and *merging and restructuring* (Figure 1).

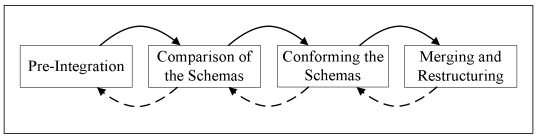

Fig. 1. View integration as described by [2]

The arrows in Figure 1 should be interpreted as follows. Arrows moving from left to right illustrate feed-forward and arrows moving from right to left illustrate feed-back. The arrows also indicate that view integration is an iterative task and it is therefore possible to move back and forth between the four phases. The first phase that should be applied when conducting view integration is *pre-integration*. According to [25], pre-integration has three main tasks that should be carried out: 1) translate all views into a canonical modeling language, 2) check for conflicts and inconsistencies in each view and finally, 3) select integration strategies. In *comparison of the schemas* the main task to perform is to compare the views aiming to identify not only similarities but also differences such as name conflicts, structural conflicts, and inter-schema properties [18]. In *conforming the schemas* the conflicts and inter-schema properties identified in the previous phase are resolved. Conflicts are for instance name conflicts and structural conflicts where the first are often resolved by renaming concept names (e.g. [1, 9]) and the second by restructuring the views or schema (e.g. [19, 27]). Finally, in *merging and restructuring* the views are superimposed, restructured and checked according to one or several quality criteria [2, 3]. To avoid ambiguity regarding the names of the phases and since this paper deals with view integration, they are henceforward called: *pre-integration*, *comparison of the views*, *conforming the views*, and *merging and restructuring*.

Several more specialized view integration approaches and methods have also been proposed. In the rest of this section three such specialized methods are described and discussed with focus on the phases used in each of them. All methods use ER or some similar modeling constructs to graphically define and describe the views and schema. Choosing these three methods is motivated by the fact that ER or some extensions of it has become one of the most commonly and frequently database modeling language used today (e.g. [12, 27, 28]). ER or some extensions of it, has also dominated view integration research since the late 1980's [25]. Although the object-oriented approach has become popular most of the database designers still prefer working with ER [24] or some similar modeling language.

The first method is proposed by [1] and it focuses on ER. The method has three phases. It starts with *conflict analysis* followed by *merging* and ends with *enrichment and restructuring*. The second method is proposed by [19] and it focuses on resolution of structural conflicts in ER. The method has five phases as follows: *resolve naming conflicts*, *resolve structural conflicts*, *merge the schemas*, *create ISA hierarchies and remove inherited attributes* and finally *remove redundant relationship sets and derived attributes*. The third and last method is proposed by [9] and it focuses on resolving conflicts using extended ER. This method is an assertion-based method that only has two phases: *schema comparison* and *schema integration*. When analyzing the example methods and the phases used in each of them it is concluded that all of them have and use similar phases compared with the four phases identified by [2]. This indicates that [2] have had a strong influence on the methods that are developed and used today. However, in this paper it is argued that one phase is still missing. The missing phase that is identified and proposed is called *pre-conforming the views*. This phase is important and needed since it exists a gap between the second phase, identification of conflicts, and the third phase, resolution of conflicts in view integration.

3 Applying EM in View Design and View Integration

Several methods and modeling languages for view design and view integration have been proposed during the last three decades. The main idea is to define and design a conceptual schema relieved from all computer implementation aspects. However the main part of the methods and conceptual modeling languages used today tends to focus on the implementation level and the technical part of the future database [15]. The modeling approach applied in this paper, EM, can be used for modeling and defining not only semantic and pragmatic dependencies but also the syntactic parts of the future database without considering any implementation aspects. Another problem with earlier methods concerns resolution of conflicts in view integration. Many of the earlier proposed methods focus on handling conceptual similarities and differences, resolving synonyms and homonyms with renaming which could compress several concept names into one concept name [4], instead of trying to keep the concept names used in the views and organization as long as possible in view integration.

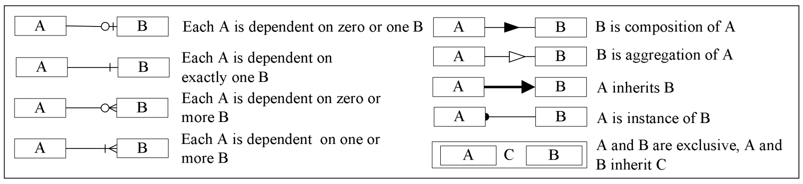
A
B
Each A is dependent on zero or one B
A
B
Each A is dependent on exactly one B
A
B
Each A is dependent on zero or more B
A
B
Each A is dependent on one or more B
A
B
B is composition of A
A
B
B is aggregation of A
A
B
A inherits B
A
B
A is instance of B
A
C
B
A and B are exclusive, A and B inherit C

Fig. 2. Adapted and modified representation of static dependencies in EM [16]

The static dependencies adapted and modified from EM are illustrated in Figure 2.

3.1 Why EM should be applied in View Design and View Integration

EM focuses on defining (modeling) and integrating the business processes of the organization in focus [32]. EM can also be described as a generalization and an extension of system analysis and design [16]. The main idea in EM is to define a consistent, coherent and complete specification, a conceptual schema, of the future database [16] using the same schema type [17]. Applying EM during view design and view integration has several advantages if compared with a traditional modeling language. First, EM has a comprehensive set of graphical primitives to use (see Fig. 2). This gives the designer an opportunity to define more detailed dependencies between the concepts. Secondly, in EM every concept is drawn as a box which minimizes the occurrence of a type conflict. Thirdly, in EM both static and dynamic dependencies of the database may be defined and graphically illustrated in one view or schema [5]. This is important since recent studies have indicated and proved that using one view and schema type may result in less confusion for both the designer and the end user than keeping the static and dynamic dependencies separate [31]. Fourthly, in EM a concept may be interpreted differently depending on the dependencies in focus [5]. Finally, applying EM during view design simplifies view integration since the designer is only interested in the semantics of the concepts and the dependencies between them and not in implementation details like ER is in some aspects.

In Figure 3 the concepts *Person*, *Employment* and *Company* are illustrated using both ER and EM. Both views are henceforward shortly described and discussed with the aim to illustrate that EM does not deal with implementation issues which ER does in some cases. A second aim is to illustrate why EM is preferred as a canonical modeling language during both view design and view integration.

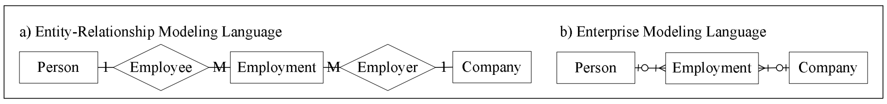
a) Entity-Relationship Modeling Language
Person
1
Employee
M
Employment
M
Employer
1
Company
b) Enterprise Modeling Language
Person
Employment
Company

Fig. 3. Comparison between ER and EM.

The primitives used in Figure 3 (a) are entities represented as rectangles and relationships represented as diamonds. Besides these the cardinality is illustrated as M or 1 where M represents 0, 1 or many and 1 represents 0 or 1. The primitives used in Figure 3 (b) are represented in Figure 2.

Both views in Figure 3 should be interpreted as follows. One *Person* can be employed by many *Companies* and one *Company* can employ many *Persons*. The *Employment* concept is used since applying EM a so called many-to-many dependency (relationship) is divided into two one-to-many relationships. Let us now assume that the designers together with the end users have decided to add two more concepts called *Employee* and *Employer*. *Employee* is defined as a specialization of *Person* and *Employer* is defined as a specialization of *Company*. Adding the concepts is straight forward for the view defined with EM (Figure 3 (b)) but for the view defined with ER (Figure 3 (a)) this is not possible because both *Employee* and *Employer* are used as relationship (dependency) names.

Finally, Figure 4 and Figure 5 both illustrate how EM can be used for view design and view integration where two name conflicts have been identified. The figures also illustrate that the concept names used in both views are retained even after the name conflicts have been resolved. One remark is needed regarding the preservation of concept names. If one or several concepts are found redundant, in [6] called over-specification, during merging and restructuring these may only be removed after carefully considering the consequences of removing a concept and concept name.

Since both Figure 4 and Figure 5 are deeply analyzed, explained and discussed in section 5 there is at this point no need do any further explanations. Just view the left part of Figure 4 and Figure 5 as examples on how to apply EM for view design and view the right part of Figure 4 and Figure 5 as simplification and resolution techniques for conflicts identified in view integration.

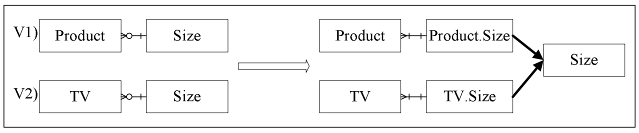

Fig. 4. Homonym conflict before and after resolution (IR 2.6).

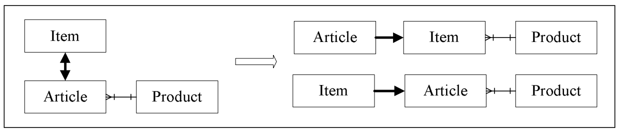

Fig. 5. Synonym conflict before and after resolution (IR 2.2).

4 Four Categories of IR for Static Dependencies

IR should by used to deduce new dependencies from one or several other dependencies and may informally be called reasoning rules. In this paper IR with the following structure are analyzed **if A Dep$_1$ B then A Dep$_2$ B**[1]. Table 1 illustrates how the dependencies are denoted and that this study focuses on static dependencies. Table 2 illustrates the seven examples of IR that are described and discussed. Letters A and B should be interpreted as concepts and arrows as dependencies between the concepts.

[1] Letters A and B should be interpreted as concepts and Dep$_{1-2}$ as a dependencies.

Table 1. Adapted and modified representation of basic static dependencies in EM [14]

Number	Dependency name	Cardinality	Denoted
1	Injection dependency	(0,1; 1,1)	A ===> B
2	Bijection dependency	(1,1; 1,1)	A ←→ B
3	Total functional dependency	(0,*; 1,1)	A → B
4	Surjection dependency	(1,1; 1,*)	A ←→→ B
5	Surjective partial functional dependency	(0,1; 1,*)	A ==>> B
6	Mutual multivalued dependency	(1,*; 1,*)	A ←←→→ B
7	Total multivalued dependency	(0,*; 1,*)	A →→ B
8	Partial injection dependency	(0,1; 0,1)	A |==> B
9	Functional (partial) dependency	(0,*; 0,1)	A |→ B
10	Multivalued (partial) dependency	(0,*; 0,*)	A |→→ B

Table 2. Adapted and modified IR [13]

Number	IR	Category
2.1	If A ===> B then A == >> B	Weaker dependency
2.2	If A ◀▬▶ B then A ▬▶ B, B ▬▶ A	Weaker dependency
2.3	If A ▬▶ B, B ▬▶ C then A ▬▶ C	Inheritance dependency
2.4	If A ▬▶ B, B --< C then A --< C	Inheritance dependency
2.5	If A → B, B → C then A → B.C, B.C ▬▶ C	Semantic quality improvement
2.6	If A → B then A.B ←→→ A, A.B ▬▶ B	Semantic quality improvement
2.7	If A ≤ B, B ▬▶ C then A ≤ C	Classification dependency

In Table 2 a few of the IR have additional dependencies. These are explained when they first occur in the IR. The IR in Table 2 has been chosen since they exemplify IR of four different categories: 1) *IR and weaker dependencies*, 2) *IR and inheritance dependencies*, 3) *IR and semantic quality improvement*, and 4) *IR and classification dependencies*. Although other IR exists (e.g. [13, 14]) and new IR may be developed, the main idea in this paper is to illustrate how to apply IR together with EM to bridge the gap between comparison and conforming the views in view integration. In the rest of this section each category is therefore described and discussed.

Each IR rule is illustrated and described in Section 5.2 using both the syntax illustrated in Table 1 and the primitives adapted from EM illustrated in Figure 2.

4.1 IR and Weaker Dependencies

Often the designer of the views does not know if an instance of a concept exists or even how many instances there could exist for one specific concept: the exact cardinality is not known. The designer then often chooses to use a zero in the cardinality. A zero in the cardinality results in a weaker dependency and in some situations even a ‘many cardinality’ could be viewed as a weaker dependency because the exact number of instances of a concept is not known. A weaker dependency could also be viewed as a special case of a stronger dependency [13] and can be derived through IR. Examples of rules that use weaker dependencies are found in IR 2.1 and IR 2.2 in Table 2. In IR 2.2 two additional dependencies are used. The thick arrow (‘▬▶’) should to be interpreted as inherits and the double sided thick arrow

(‘◀▬▶’) should be interpreted as mutual inheritance. Mutual inheritance is also used to illustrate that two concepts are synonymous [5]. *Remark 4.1(a)*: When applying the mutual inheritance dependency to illustrate synonyms the two concepts are equivalent according to their names. None of the concept names are therefore preferred before the other.

4.2 IR and Inheritance Dependencies

Inheritance is a useful dependency to apply in view design and view integration because often a need to describe hierarchies exists. This could be illustrated through generalization and specialization of concepts in the views and schema. Therefore it is important to understand how and when inheritance should be used and what are actually inherited. Examples of IR that use the inheritance dependency are found in IR 2.3 and IR 2.4 in Table 2. In IR 2.4 an additional dependency is used. The dependency that appear as a thin arrow (‘--<’) should be interpreted as composition. Composition is useful when defining that a concept is composed of several other concepts [16]. *Remark 4.2(a):* Composition is a stronger dependency than aggregation. Composition illustrates that a concept is composed of several other concepts but when the composed concept is removed all concepts that it is composed of are also removed. *Remark 4.2(b):* Composition is just one example of dependency that is inherited through the inheritance dependency.

4.3 IR and Semantic Quality Improvement

Often there is a need to improve the semantic quality in a view or a schema. This occurs because ambiguity often exists between the concept names. Semantic quality improvement has been given different names in different methods and models such as sharpening meaning [20]. Examples of IR that can be applied for semantic quality improvement are found in IR 2.5 and IR 2.6 in Table 2. In both IR the dot (‘.’) notation is used. In this context it is used to prefix concept names to unambiguously identify a concept [13]. This means that one concept name is compounded from several concept names with a dot between them. The dot notation can also be used as a component for resolution of homonym conflicts [5]. Another application of the quality improvement rules is as a technique to prevent an impoverishment of the concept names used in the views and schema. Impoverishment of concept names in view design and view integration is further discussed and criticized in [4].

4.4 IR and Classification Dependencies

One important dependency that traditional methods and models such as ER often are missing is the instance-of dependency. This dependence is needed to illustrate classification. Instance-of is closely related to the inheritance dependency. The difference between them is that ‘inherits’ deals with concepts (classes) and instance-of deals with instances of concepts (classes) called objects in object-oriented methods and models (e.g. [20]). In EM both these dependencies are important and could be used in the views and schema. The motivation behind this is that sometimes there exists a need to define different levels of abstraction (e.g. model, meta-model) of the database in one view or schema [5]. One example of IR that can be applied for classification dependencies is found in IR 2.7 in Table 2. In IR 2.7 an additional dependency is used ‘less than or equal to’ (‘≤’) which should be interpreted as instance-of. *Remark 4.4(a):* If we have the dependency Sony_TV_32001 ≤ Sony TV then the instance-of dependency should be interpreted as Sony_TV_32001 is an instance-of Sony TV.

5 Applying IR in View Integration

This section presents the main ideas behind the framework developed to bridge the gap between comparison and conforming the views in view integration. The framework is composed of IR applied with EM as a canonical modeling language in view integration. To reach this framework three research questions have been asked and analyzed regarding *why*, *how* and *when* IR should be applied in view integration using EM as a canonical modeling language. In section 5.1 conflicts and inter-schema properties are described and discussed in the context of view integration and the *why* question is raised and discussed. In section 5.2 IR are described, discussed and illustrated using not only the primitives adapted from EM but also using the syntax illustrated in Table 1. The *how* question is also raised and discussed. Section 5.3 includes a discussion about *when* to apply IR in view integration. Together these three subsections (5.1-5.3) build and illustrate the developed framework.

5.1 Why IR should be applied in View Integration

Two of the most complex phases in view integration are *comparison of the views* and *conforming the views*. During comparison of the views both differences such as conflicts and inter-schema properties and similarities are identified. This phase has been called the most important [25], difficult [10] and challenging [18] phase in view integration. During conforming the views the conflicts and inter-schema properties identified in the previous phase are resolved. Therefore this phase has been mentioned as a critical issue [19] and a key issue [27] in view integration. Most of the earlier view integration methods also put focus on these two phases. As mentioned earlier while analyzing and comparing two views several conflicts and inter-schema properties may often be detected. What types of conflict that can occur depend on the chosen modeling language and the level of abstraction integration is to be performed at (e.g. conceptual or logical level). In this paper the conflict classification proposed by [2] is used as a starting point. This is motivated since it is well known and gives an illustrating picture over the conflicts that may occur during view integration performed at the conceptual level. The conflict classification can also be used as a checklist and as a reference while applying and discussing IR in the context of view integration. Conflicts are by [2] classified as either *name conflicts* or *structural* *conflicts*. Name conflicts are further divided into *homonyms*, which occur if one name is used for two or more concepts and *synonyms*, which occur if two or more names are used for one concept. Structural conflicts are further divided into *type conflicts*, which occur if different modeling constructs are used to define the same concept. Applying the static part of EM the type conflict is minimized because every concept is modeled as a box [5]. *Dependency conflicts* occur if different dependencies are defined between the same concepts and *key conflicts* occur if different keys are defined for the same concept. Finally, *behavioral conflicts* occur if different policies for insertion and deletion are defined for the same concept [2]. Apart from these conflicts, there exists a phenomenon called inter-schema properties. These properties concern concepts that are not exactly the same but have certain constraints in common [18]. After the conflicts and the inter-schema properties have been identified the next step is to resolve them. Resolution and simplification of conflicts and inter-schema properties are a very complex task and therefore it is desirable to apply techniques that may help to resolve or at least simplify the problems. IR are one such technique and should therefore be used in view integration.

5.2 How to Apply IR in View Integration

As also mentioned earlier IR should be applied in view integration as rules for simplifying or resolving identified conflicts and inter-schema properties. The examples used in this subsection are applications of the IR in Table 2. The examples are both illustrated graphically using the primitives adapted from EM and illustrated using the syntax in Table 1. In this paper IR are used as a technique to reasoning with the end users about conflict and inter-schema simplification and resolution. Therefore IR may informally be called reasoning rules. Although there exists similar reasoning about applying rules for conflict resolution (e.g. [25]) these discussions are very broad and not detailed, while the discussion in this paper, especially this subsection, answer the how question at a very detailed level. The examples are scenarios that may occur in an organization conducting view integration in conceptual database design. In all figures the abbreviations V1 and V2 are used. These should be interpreted as "view one" and "view two" which are views prepared for integration. Let us now work through the examples found in Figure 4-10 starting with Figure 6.

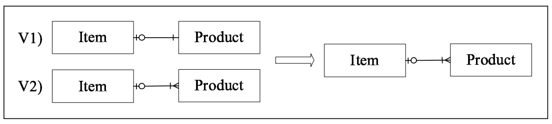

Fig. 6. Pre-conforming views with IR 2.1.

When comparing the views in Figure 6 a *dependency conflict* is identified. In V1 the dependency is defined as **Item ===> Product** and in V2 as **Item ==>> Product**. Applying IR 2.1 the conflict can be simplified or resolved as follows: **If Item ===> Product then Item ==>> Product**. The dependency in V1 is altered to be the same as in V2. The same resolution technique, to introduce a semantic weaker dependency, is described by [23] as a general resolution technique for simple dependency conflicts such as the one illustrated in Figure 6. One remark regarding the use of IR 2.1 is required. The differences in cardinality may also indicate a homonym conflict. It is therefore very important to analyze not only the concept names and dependencies but also the concepts surroundings before deciding to apply this rule. However, using it as a communication tool the rule can be very useful when trying to identify and resolve problems such as dependency conflicts and homonyms.

Let us now analyze the situation in Figure 5. In the left view we have the following dependencies **Article ◄▬► Item, Product ←→→ Item**. This illustrates that *Item* and *Article* is synonyms and that each *Item* is dependent on one single *Product* and that each *Product* may be associated with one or more *Items*. Applying IR 2.2 the synonyms can be simplified as following: **if Item ◄▬►Article then Article ▬►Item, Item ▬► Article.**

It can be concluded that IR for semantic weaker dependencies can be used not only to simplify and resolve *dependency conflicts* (IR 2.1) but also to simplify or resolve *synonym conflicts* (IR 2.2) where synonyms are defined as **A ◄▬►B if and only if A ▬►B, B ▬► A**.

The next situation to analyze is illustrated in Figure 7. In V1 we have the following dependency **LCD TV ▬►Flat Screen TV** and in V2 the following **TV ▬► Product**.

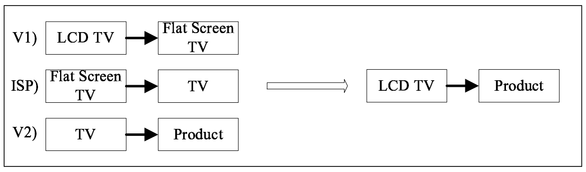

Fig. 7. Pre-conforming views with IR 2.3.

When comparing the views an *inter-schema property* is identified. In V1 *LCD TV* is a specialization of *Flat Screen TV* and in V2 *TV* is a specialization of *Product*. This indicates that an inheritance hierarchy exists between the views. *Flat Screen TV* is a specialization of *TV* which also makes it a *Product*. Applying IR 2.3 the inter-schema property can be simplified or resolved as follows: **If LCD TV ▬► Flat Screen TV, Flat Screen TV ▬► TV[2], TV ▬► Product then LCD TV ▬► Product**.

Let us now continue with the situation in Figure 8. The dependencies in V1 and V2 can be described as **LCD TV ▬► Flat Screen TV** (V1) and **TV --< Integrated Home Cinema Package** (V2). When comparing the views an *inter-schema property* is identified. In V1 *Flat Screen TV* is defined as a generalization of *LCD TV* and in V2 *TV* is defined as a part of (composition) *Integrated Home Cinema Package*. *Flat Screen TV* is a specialization of *TV* which also makes it a part of *Integrated Home Cinema Package*. Applying IR 2.4 the inter-schema property can be simplified or resolved as follows: **If LCD TV ▬► Flat Screen TV, Flat Screen TV ▬► TV[3],**

[2] This inheritance dependency is the inter-schema property, marked ISP in Fig. 7, identified between Flat Screen TV and TV.

[3] This inheritance dependency is the inter-schema property, marked ISP in Fig. 8, identified between Flat Screen TV and TV.

TV --< Integrated Home Cinema Package then LCD TV --< Integrated Home Cinema Package.

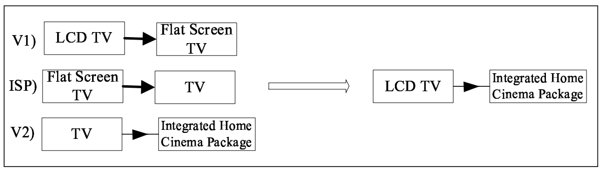
V1)
LCD TV
Flat Screen TV
ISP)
Flat Screen TV
TV
V2)
TV
Integrated Home Cinema Package
LCD TV
Integrated Home Cinema Package

Fig. 8. Pre-conforming views with IR 2.4.

It can be concluded that inference rules for inheritance dependencies (IR 2.3 and IR 2.4) can be used to simplify and resolve *inter-schema properties*.

The next situation to analyze is illustrated in Figure 9. In V1 we have the following dependency **Product → Type** (V1) and in V2 the following **Type → Family** (V2).

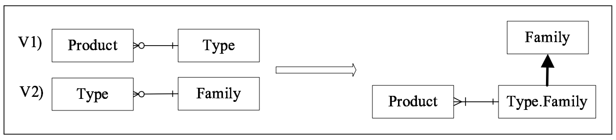

Fig. 9. Pre-conforming views with IR 2.5.

When comparing the views in Figure 9 neither a conflict nor an inter-schema property is identified. However ambiguity exists regarding the use of concept names. Each *Product* is dependent on one single *Type* and each *Type* is dependent on one single *Family*. One way to integrate the views is to merge them over the *Type* concept. If that is the situation there is a need to improve the semantic quality and at the same time eliminate the ambiguity pointed out earlier. This can be achieved applying IR 2.5 as follows: **If Product → Type, Type → Family then Product → Type.Family, Type.Family ▬► Family**. As also illustrated in the integrated schema in Figure 9 the dependencies and the *Type* concept are improved regarding the semantic quality.

Let us compare the views in Figure 4. In V1 the following dependency is defined **Product → Size** and in V2 the following **TV → Size**. The similarity and at the same time the difference between the views are identified in the use of concept names. In this case it indicates a *homonym conflict*. The *Size* concept is used differently and therefore a sharpening of meaning, a semantic quality improvement, is needed. This can be achieved applying IR 2.6 as follows: **If TV → Size then TV.Size ←→→ TV, TV.Size ▬►Size**. To end up with the schema in Figure 4 IR 2.6 also has to be applied for V1 and then the views can be merged over the *Size* concept.

It can be concluded that IR for semantic quality improvement can be used not only to simplify or resolve *homonym conflicts* (IR 2.6) but also to *improvement semantic quality* (IR 2.5). A deeper discussion about identifying and resolving homonym conflicts applying EM is given in [5].

The seventh and last situation to analyze is illustrated in Figure 10. In V1 the following dependency is defined **STV32001 ≤ Sony LCD TV** and in V2 the following **TV ▬▶ Product**.

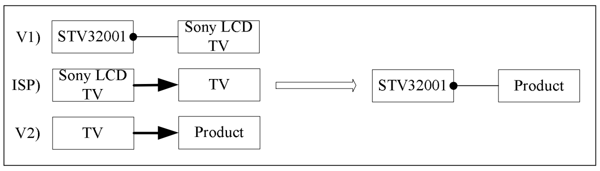

Fig. 10. Pre-conforming views with IR 2.7.

When comparing the views an *inter-schema property* is once again identified. In V1 *STV32001* is an instance-of *Sony LCD TV* and in V2 is *TV* a specialization of *Product*. An inheritance hierarchy is identified between the views: *Sony LCD TV* in V1 is a specialization of *TV* in V2. Applying IR 2.7 the inter-schema property can be simplified or resolved as follows: **If STV32001 ≤ Sony LCD TV, Sony LCD TV ▬▶ TV**[4]**, TV ▬▶ Product then STV32001 ≤ Product**.

It can be concluded that IR for classification dependencies can be used to simplify and resolve *inter-schema properties* (IR 2.7).

5.3. When to Apply IR in View Integration

This section finalizes the description and discussion about the developed framework by focusing on the new identified and proposed phase. In the developed framework IR should be applied to simplify and resolve conflicts and inter-schema properties in view integration. View integration is often seen as an important and critical part of conceptual database design and therefore needs continual refinement and reevaluation [30]. Two serious weaknesses have in this paper been identified in the earlier proposed integration methods. The first is the gap between comparison and conforming the views and the second is the lack of explicitly mentioning the use of IR as such but also an own phase. The absence of mentioning IR as a conflict and inter-schema property simplification or resolution technique is even more surprising. When IR are mentioned (e.g. [25]) they are treated superficially and incorporated in other phases which makes them complex and hard to handle. One specific phase for the use of IR is therefore proposed. The new phase is placed between *comparison of the views* and *conforming the views* and is henceforward called *pre-conforming the views*. Figure 11 illustrates the phases in view integration as described and proposed in this paper.

[4] This inheritance dependency is the inter-schema property, marked ISP in Fig. 10, identified between Sony LCD TV and TV.

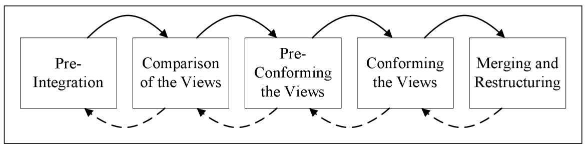

Fig. 11. View integration process as described and proposed in this paper.

Figure 11 should be interpreted from left to right and the arrows indicate that it is possible to go back to an earlier phase and do adjustments. The use of pre-conforming the views is motivated because there is a need not only to decompose the integration methods into smaller phases that are unambiguous regarding their content but also to bridge the gap between comparison and conforming the views. Therefore it is important to incorporate pre-conforming the views as a standard phase into view integration and in it apply IR to simplify and resolve conflicts and inter-schema properties.

6. Summary and Conclusions

In this paper a framework that bridges the gap between comparison of the views and conforming the views in view integration has been developed. The three research questions about *why*, *how* and *when* IR should be applied during view integration have been raised and discussed. The question regarding *why* IR should be applied has at least two possible answers. First because there exist a gap between the two most complex and important phases, *comparison of the views* and *conforming the views*, in view integration. Therefore a need for techniques that can be used to solve, or at least simplify, the conflicts and inter-schema properties exists and IR are one such technique. Secondly, because the integration methods used today needs continual refinement and reevaluation as pointed out by [30]. The reason behind this is the complexity of the view integration task and because the used primitives and modeling languages are continually developed and refined. The question about *how* IR should be applied also has at least two possible answers. First IR should be applied to deduce new dependencies from two or more dependencies where a conflict has been identified. Secondly, IR should to be applied to deduce new dependencies from two or more dependencies where an inter-schema property has been identified. The question regarding *when* to apply IR has in this study one obvious answer. In a new phase in this paper called *pre-conforming the views*. This new phase should be incorporated between *comparison of the views* and *conforming the views*. Pre-conforming the views should first of all be used to bridge the gap between phase 2 and phase 3 which are the two most complex phases in view integration. By incorporating *pre-conforming the views* as a standard phase in view integration the task for each phase can be defined and described more precisely. To answer these three questions the static dependencies of EM [16] has been applied including a deeper discussion about why to apply EM in view design and view integration. Analysis using both the primitives in EM and the syntax for IR has been conducted, illustrated and discussed.

One important observation of this work is that using the developed framework where EM is treated as the canonical modeling language and IR are used to deduce dependencies name conflicts (synonyms and homonyms) can be resolved without loosing any of the concept names. Retaining the concept names may counteract language impoverishment in the global schema. Finally, by combining EM as a canonical modeling language during view design and view integration and using IR for pre-conforming the views the designers can focus on the semantics of each concept and the dependencies between the concepts instead of dealing with implementation issues of the future database.

References

1. Batini, C., Lenzerini, M.: A Methodology for Data Schema Integration in the Entity Relationship Model. IEEE Transactions on Software Engineering 10(6) (1984) 650–664
2. Batini, C., Lenzerini, M., Navathe, B.L. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18(4) (1986) 323–363
3. Batini, C., Ceri, S., Navathe, S.: Conceptual Database Design – An Entity-Relationship Approach. Benjamin-Cummings, Redwood City, California (1992)
4. Bellström, P., Carlsson, S.: Towards an Understanding of the Meaning and the Contents of a Database through Design and Reconstruction. In: Vasilecas, O. et al. (eds.): Proc. 13th ISD Conference (2004) 283–293
5. Bellström, P.: Using Enterprise Modeling for Identification and Resolution of Homonym Conflicts in View Integration. In: Vasilecas, O. et al. (eds.): Information Systems Development Advances in Theory, Practice and Education. Springer (2005) 265–276
6. Bellström, P., Jakobsson, L.: Towards a Generic and Integrated Enterprise Modeling Approach to Designing Databases and Software Components. In: Nilsson, A.G. et al. (eds.): Advances in Information Systems Development: Bridging the Gap between Academia and Industry. Springer, (2006) 635–646
7. Chen, P.: The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems 1(1) (1976) 9–36
8. Dey, D., Storey, V.C., Barron, T.M.: Improving Database Design through the Analysis of Relationships. ACM Transactions on Database Systems 24(4) (1999) 453–486
9. Dupont, Y.: Resolving Fragmentation Conflicts in Schema Integration. In: Loucopoulos, P. (ed.): Proc. 13th ER Conference. Springer-Verlag, (1994) 513–532
10. Ekenberg, L., Johannesson, P.: A Formal Basis for Dynamic Schema Integration. In: Thalheim, B. (ed.): Proc. 15th ER Conference. Springer, (1996) 211–226
11. Engels, G., Gogolla, M., Hohenstein, U., Hulsmann, K., Lohr-Richter, P., Saake, G., Ehrich, H.D.: Concepual modelling of Database Applications Using an Extended ER Model. Data & Knowledge Engineering 9 (1992) 157–204
12. Fahrner, C., Vossen, G.: A Survey of Database Design Transformations Based on the Entity-Relationship Model. Data & Knowledge Engineering 15 (1995) 213–250
13. Gustas, R.: Semantic and Pragmatic Dependencies of Information Systems. Monograph, Technologija Kaunas (1997)
14. Gustas, R.: Integrated Approach for Modelling of Semantic and Pragmatic Dependencies of Information Systems. In: Ling, T.W. et al. (eds.): Proc. 17th ER Conference. Springer-Verlag, (1998) 121–134

15. Gustas, R., Gustiené, P.: Extending Lyee Methodology Using the Enterprise Modelling Approach. In: Fujita, H., Johannesson, P. (eds.): Proc. 1st SoMeT Workshop. IOS Press, Amsterdam (2002) 273–288
16. Gustas, R., Gustiené, P.: Towards the Enterprise Engineering Approach for Information System Modelling Across Organisational and Technical Boundaries. In: Camp, O. et al. (eds.): Enterprise Information Systems V. Kluwer Academic Publishers, Netherlands, (2004) 204–215
17. Gustas, R., Jakobsson, L.: Enterprise Modelling of Component Oriented Information System Architechtures. In: Fujita, H., Gruhn V. (eds.): Proc. 3rd SoMeT Workshop. IOS Press, (2004) 88–102
18. Johannesson, P.: Schema Integration, Schema Translation, and Interoperability in Federated Information Systems. PhD thesis, Department of Computer & Systems Sciences, Stockholm University, Royal Institute of Technology, No 93-010-DSV, Edsbruk (1993)
19. Lee, M.L., Ling, T.W.: A Methodology for Structural Conflict Resolution in the Integration of Entity-Relationship Schemas. Knowledge and Information System 5(2) (2003) 225–247
20. Martin, J., Odell, J.J.: Object Oriented Methods A Foundation, Prentice Hall, New Jersey (1998)
21. Navathe, S.B., Gadgil, S.U.: A Methodology for View Integration in Logical Database Design. In: Proc.8th VLDB Conference. (1982) 142–164
22. Navathe, S., Elmasri, R., Larson, J.: Integrating User Views in Database Design. IEEE Computer 19(1) (1986) 50–62
23. Parent, C., Spaccapietra, S.: Issues and Approaches of Database Integration. Communications of the ACM 41(5es) (1998) 166–178
24. Shoval, P., Shiran, S.: Entity-Relationship and Object-Oriented Data Modeling – an Experimental Comparison of Design Quality. Data & Knowledge Engineering 21 (1997). 297–315
25. Song, W.: Schema Integration – Principles, Methods, and Applications. PhD thesis, Department of Computer & Systems Sciences, Stockholm University, Royal Institute of Technology, No 95-019, Edsbruk (1995)
26. Spaccapietra, S., Parent, C.: Conflicts and Correspondence Assertions in Interoperable Databases. ACM SIGMOD RECORD 20(4) (1991) 49–54
27. Spaccapietra, S., Parent, C.: View Integration: a Step Forward in Solving Structural Conflicts. IEEE Transactions on Knowledge and Data Engineering 6(2) (1994) 258–274
28. Storey, V.C., Thompson, C.B., Ram, S.: Understanding Database Design Expertise. Data & Knowledge Engineering 16 (1995) 97–124
29. Teorey, T.J., Yang, D., Fry, J.P.: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. Computing Surveys 18(2) (1986) 197–222
30. Teorey, T.J.: Database Modeling & Design. Morgan Kaufmann Publishers Inc, USA (1999)
31. Turetken, O., Schuff, D., Sharda, R., Ow, T.T.: Supporting Systems Analysis and Design through Fisheye Views. Communications of the ACM 47(9) (2004) 72–77
32. Vernadat, F.B.: Enterprise Modeling and Integration Principles and Applications. Chapman & Hall, London (1996)