A Conceptual Justification for Model Transformations

Ove Sørensen\textsuperscript{1} \hspace{1cm} Bernhard Thalheim\textsuperscript{1} \hspace{1cm} Qing Wang\textsuperscript{2}

\textsuperscript{1} Department of Computer Science, University of Kiel, Christian-Albrechts-Platz 4, DE-24118 Kiel, Germany, Email: \{ove, thalheim\}@is.informatik.uni-kiel.de \\
\textsuperscript{2} Department of Information Science, University of Otago, Dunedin, New Zealand, Email: qing.wang@otago.ac.nz

Abstract

We analyze inter-model transformations of database schemas from a conceptual point of view. A central question for us is not just whether the information capacity of the transformed schema is sufficient, but rather its suitability for a given task. For this, we require criteria beyond the resulting degree of normalization which we subsume under the term “conceptual justification”. To illustrate our point, we take a closer look at a class of conceptual requirements that frequently cause practitioners to manually denormalize the logical schema: layered schemas, where the natural layering of the data clashes with the dominant access patterns and negatively impacts performance. We show how such requirements can directly influence the transformation process and give rise to conceptually justified logical schemas. We include an example which is based on the translation from the higher-order entity-relationship model to the relational model.

Keywords: model transformation, conceptual schema, logical schema, denormalisation, normal form, higher-order entity-relationship model, relational model, information capacity

1 Pragmatics of Conceptual Modeling

In the process of engineering information systems, a conceptual schema is typically the first artifact created after the requirements engineering phase. Further schemas are subsequently derived from this, usually formulated using a model that is more suited for implementation. A popular instantiation of this pattern makes use of an entity-relationship schema for the conceptual phase, and uses this as an input to generate a relational schema which is then installed in a DBMS.

A well known result of computer science is that there exists an algorithm to automate this translation from the entity-relationship model to the relational model. The relational schema that is the result of this transformation is guaranteed to be in the 3rd normal form, or even in the Boyce-Codd normal form, depending on the underlying assumptions. This means, that the schema that will be used in practice has certain desirable properties, such as freedom from various anomalies and data duplication which makes for a space-efficient encoding, easy querying, updates and possibly constraint definitions.

In practice however, logical schemas are frequently not founded on a conceptual schema at all. In the case where there exists a conceptual foundation, the logical schema is often not derived in a straightforward way – i.e. by the aforementioned algorithm – despite a plethora of modeling tools that offer such functionality. Although the normalized schemas generated by such tools possess the traits mentioned above, practitioners seem to feel that they are in some way inadequate for the task at hand. It can be observed that, instead of striving for higher normal forms than produced by the transformation algorithms and the even stronger integrity guarantees they entail, schemas are relaxed in a way that trades increased complexity of integrity maintenance for other characteristics such as increased performance. This process is known as denormalisation (see Figure 1).

Figure 1: Usual Information System Engineering Process

This yields a situation, where there is a disconnect between the logical and conceptual view of the system. There is no provision in the conceptual model to feed back the changes that have been applied to the logical schema, so the logical point of view dominates the later phases of the project, with no motivation to keep the conceptual schema up to date. A lot of crucial information and manageability is lost during the desynchronization of these two layers.

Our goal here is to investigate possibilities of expressing the adjustments typically applied in the logical layer in the conceptual layer. This would provide a conceptual justification for a logical schema beyond the typical characteristics of normal-form and preservation of information capacity. A strong link is thus maintained between the conceptual schema and the one that is actually implemented. This means, that the conceptual schema can serve as the interface for database operations in practice. We do not strive to introduce yet another normal form, nor to achieve any of the known ones after our transformation. We instead opt to consider application requirements formulated on the conceptual level and to provide an in-
The treatment of relationship types is somewhat more varied. The simplest approach is just to create a relation type for each relationship type. The participating entity types are referenced via foreign keys. These also make up the primary key of the created relation type. If the same entity fills more than one role, an attribute renaming scheme must be employed. Simple attributes of the relationship type can directly map to non-key attributes of the relation type. Higher-order relationship types can be supported in a straightforward manner when their order is well defined. In this case, the relationship types that they reference have a known relational representation if the process proceeds from lower to higher order.

If cardinality constraints are added to the model, there are opportunities to reduce the number of relation types required to represent relationship types. Relationships with exclusively \( (1, 1) \) cardinalities offer the possibility to combine the participating types into a single relation type. If a role has maximum cardinality 1, the required attributes for the relationship type may be merged into this entity type’s relation type. Cardinalities other than 0,1 and \( * \) can not be translated in a straightforward way.

is-a relationship types are often considered a special case. One possible translation of these hierarchies involves flattening them into a single relation type, where every tuple possesses every attribute of the hierarchy. This mandates the use of structural nulls. Sometimes, a special attribute is added that represents the type of each tuple in this relation. The other way of translating is-a hierarchies makes use of a special relation type for each type in the hierarchy that is linked to the table of its super type via a foreign key. For hierarchies that are both total and exclusive, super type relation types might not exist and instead be merged into the subtype relation types instead.

The unifying theme for all transformations based on this approach is that they are deterministic, do not require user interaction, operate locally and derive the structure of the relational schema solely from the structure of the entity relationship model. Rarely is the functionality of the conceptual model considered, and then by considering selected queries only. There are few notable exceptions (Casanova et al. 1993) (Markowitz & Shoshani 1992) (Kolp & Zimányi 2000).

1.1 Canonical Relational Transformation

Before we proceed to the main part of this paper, we sketch what we call the canonical transformation of entity-relationship schemas (Chen 1976) to relational schemas. This approach can be considered the baseline upon which a large number of transformation algorithms are based (Fahrner & Vossen 1995).

Strong entity types directly map into distinct relation types. They retain all their simple attributes and also their primary keys. Complex attributes are either flattened, or can generate derived relationship types which can reference the entities that they belong to via the primary key of the corresponding table.

Weak entity types also induce a distinct relation type with its attributed copied from the entity type. Unlike strong entity types, their primary keys are not just comprised of the key attributes of the weak entity type, but also of the identifying strong entity type(s). This is amended with a foreign key constrained referencing the relation types that were created for those strong entity types. Special consideration can be given if the identifying relationship types have attributes of their own.

The treatment of relationship types is somewhat varied. The simplest approach is just to create a relation type for each relationship type. The participating entity types are referenced via foreign keys. These also make up the primary key of the created relation type. If the same entity fills more than one role, an attribute renaming scheme must be employed. Simple attributes of the relationship type can directly map to non-key attributes of the relation type. Higher-order relationship types can be supported in a straightforward manner when their order is well defined. In this case, the relationship types that they reference have a known relational representation if the process proceeds from lower to higher order.

1.2 HERM

We use the extended entity-relationship model (HERM) (Thalheim 2000) for the representation of schemas. It generalizes the classical entity-relationship model by adding constructs for richer structures such as complex nested attributes, relationship types of higher-order, cluster types that allow disjoint union of types (displayed by \( \oplus \)), by an algebra of operations, by rich sets of integrity constraints, by transactions, by workflows, interaction stories, and by views. An IsA-association and the subtype can be compacted to a unary relationship type which can be extended by specific identification. Cardinality constraints are supported with participation semantics.

Techniques for translation of a HERM specification to relational and object-relational specification are discussed in detail in (Thalheim 2000). For surveys etc. see: http://www.is.informatik.uni-kiel.de/~thalheim/slides.htm.

1.3 An Example

Figure 3 depicts a fragment of a HER-Schema that might be used to keep track of the billing as viewed
from the perspective of a small company selling services. Employees are assigned to fixed teams that perform services. A work log is generated in the process. Any number of these jobs can be bundled and a single bill generated for them. A job might not appear in a bill, because bookkeeping is a delayed process.

A canonical relational transformation of this HER-Schema looks as follows:

```
Team(TeamID, Size)
Service(ServiceName, TeamID, ServiceID)
Job(LogID, LogText, TeamID, ServiceName)
Position(LogID, BillID, Price)
Bill(BillID, BillDate)
```

With the additional foreign keys:

```
Job(TeamID) \rightarrow Team(TeamID)
Job(ServiceName) \rightarrow Service(ServiceName)
Position(LogID) \rightarrow Job(LogID)
Position(BillID) \rightarrow Bill(BillID)
```

Further constraints may be added to implement the remaining minimum cardinalities. We can observe, that there is no duplication of information. The generated schema should thus be easily maintainable.

2 Model Transformations

When we speak about *model transformations*, we mean the transformation of a schema into another schema formulated in a different data model. The traditional point of view mandates for this transformation to be as neutral as possible. The relationship between the original and transformed schema should be characterised by some notion of equivalence, which should – as much as possible – be derived by properties of the two models involved; hence the focus on the achieved normal form after transforming from the entity relationship model to the relational model. This could also be considered the *lowest common denominator* approach. A central question in this approach is concerned with the exact nature of the term *equivalence*.

2.1 Information Capacity

Relative information capacity (Hull 1984) is a term that was originally coined to investigate the equivalence of two relational schemas $P$ and $Q$. In the context of the cited paper, it is derived from one of the proposed notions of *schema domination*. Schema domination may be based on one of the following criteria:

1. an arbitrary mapping from instances of schema $P$ to schema $Q$, plus an inverse map that yields the original instance
2. as above, with added restriction that the mappings must be expressible in some kind of calculus (e.g. using selection, projections, joins and unions only)
3. the ability to express any query on schema $P$ also on schema $Q$
4. a mapping from instances of schema $P$ to schema $Q$ and its inverse, where both of them are generic (i.e. independent of the actual data)

2.2 Suitability

While this approach yielded some interesting results, it was soon found to be too strict for some of the challenges faced in practice. Some implications were also perceived as counterintuitive (Qian 1996). If the intended functionality of the transformed schema is taken into account, certain concessions regarding the notion of equivalence can be made, that seem to do a better job at capturing how schemas are actually translated (Miller et al. 1993). The authors of the latter paper postulate a hierarchy of operational goals that influence the suitability of a given transformation of a schema $P$ into a schema $Q$:

1. querying the data contained in $Q$ via $P$
2. as above, plus additionally guaranteeing that the *entire* information stored in $Q$ can be retrieved
3. as above, but also providing the ability to alter the information stored in $Q$ via $P$
4. querying the data contained in $Q$ via $P$ and vice versa (implying all three goals mentioned above)

This hierarchy is primarily motivated by the challenges faced during an information integration process. In the context of the higher-order entity-relationship model, such integration processes have already been described by their property of maintaining a sufficient information capacity (Lehmann & Schewe 2000). When we compare this point of view to the notion of information capacity, we find that the goal of equivalence was replaced by suitability for a given purpose – i.e. the fitness for acting as a surrogate for the conceptual schema. Because we want to focus on the case where an entirely new logical model is derived from a conceptual model for implementation purposes, we require transformations to fulfill the 3rd criterion if the conceptual schema is to be the interface to the data. The question of the “proper” transformation for a given conceptual schema remains open.

This line of thinking leads us to the problems at hand: can the suitability of a logical model that implements some conceptual model be characterised? If so, can we find a way to escape the *lowest common denominator* trap and identify properties that currently necessitate manual adjustments to the logical model? Can we justify the choices that we make from a conceptual point of view? Can we find an algorithm to engineer logical schemas that we deem suitable?

3 Conceptual Requirements

The class of conceptual requirements that we would like to address as an example is concerned with the handling of layered schemas. In Figure 3, we can see that the Job entity type represents an actual activity that was performed in the real world. After the activity has been completed and stored in the database, it *can be used as a Position* in a Bill. The structure of the schema and the data stored therein reflects the history of the real world accurately. This is a desired quality of a conceptual model.

If we now compare this to the result of the canonical relational translation as presented in Section 1.3, we can see that the conceptual layering is reflected in the relational schema, manifested in the foreign key *Position(LogID) → Job(LogID)*. In the logical implementation, two joins are required to access the names of the services that belong to some bill. Even worse, this is only due to the fact, that *ServiceName*
4 Transforming Requirements

When performing a model transformation of a conceptual schema, we aim for a conceptually justified result. This means, the conceptual requirements as well as the targeted model must be taken into consideration. The requirements constrain the degrees of freedom that arise when the result does not have to conform to some normal form, and the targeted model supplies the means to implement (some of) the requirements. One could argue, that there is also an optimization problem hidden in this process, but this rests on the assumption, that some accepted quality metric exists and that the stated requirements are contradiction-free. These are issues that we will not take into account this time. For now, we focus on achieving a translated schema that is conceptually justifiable. Research on the other aspects can be based on these results.

As stated above, the concrete realisation of the translation heavily depends on the targeted logical model. In this section, we shall concentrate on the relational model. In section 1.3, it was shown how the intrinsic layering of the entity relationship schema carries over to the relational schema. Splitting relations and duplicating attributes are employed to break this layering in the relational model. We will now show how to justify these operations by an annotated conceptual schema.

4.1 Cross-Cutting

The canonical translation is a local process, in the sense that each entity/relationship type can be translated individually as long as it's known how the conceptual types that it directly references are represented in the logical model. For our conceptually justified translation, we consider macro-structures as well. Observe, that a single relationship type can be considered a nested relation. For example, the Position relationship type corresponds to the relation Position(Price, Bill(BillID, BillDate), Job(Team(...), WorkLog(...), Service(ServiceName, ServiceDescription))).

This does of course not hold for an entire schema, but it also applies to a cross-cutting tree, the only differences being that some attributes/roles may be left out, and the nested relation might be truncated (i.e. there are relations without further nesting that were derived from relationship types).

Let us consider how to translate a single cross-cutting tree in isolation. Our example in Figure 4 corresponds to the nested relation Position(Price, Bill∗(BillID, BillDate), Job∗(Service∗(ServiceName, ServiceDescription))). It is helpful for our purposes, to annotate the derived nested relations with the cardinality constraints and key information.

It turns out, that not every cross-cutting tree can be translated directly. This is due to the HERM, which allows for relationship types to have attributes. They can only be represented, if those types retain their identity during the transformation. Because roles supply the relationship types' identities, they can not be arbitrarily projected out. In our example, we projected Job onto the Service role. As a side effect, there will be information loss in the Position type (remember that we assume set semantics) – positions belonging to the same bill will be collapsed into one if they reference the same service, so that the individual prices can not be distinguished.

The first step in the transformation is to make
sure that this does not happen. For every entity type in the nested relation, we add enough attributes so that it is identified. For every relationship type, we add the type belonging to each missing role. For types that were added only to maintain identity, it is sufficient to add only those roles and attributes that play a role in identifying its instances. This process must be carried out recursively until identity is grounded in entity types. There is an exception to this rule for relationship types that have a role with a maximum cardinality of 1. In those cases, it is sufficient to make sure that one of those roles is represented in the nested relation. The relationship type can derive its identity from this role. In our example, we can take this shortcut and add the role referencing Work Log to the Job type. The nested relation for our tree is now Position(Price, Bill¹⁺(BillID, BillDate), Job⁰⁺(Service⁰⁺(ServiceName, ServiceDescription), WorkLog¹⁺(LogID))).

This opens up the possibility to introduce surrogate entity types and add them to a relationship type with a newly introduced 1.1 role, further simplifying the translated schema. Whether this kind of simplification is beneficial in a given situation, must be evaluated in the context of the overall schema translation. This method enables the designer to leave out lower parts of the type hierarchy that are deemed unimportant for the context of the tree.

In the next step, the nested relation can be flattened. This is when the actual cross-cutting is performed. For each of the relations in the tree, we need to determine the key, and whether the cardinalities permit it to be maintained when merging two layers. Starting with the relations derived from entity types, we merge their attributes with their enclosing relation. If the maximum cardinality of the merged relation is 1, the merged key attributes are also key attributes of the outer relation. After all nested relations in the currently processed layer have been merged (possibly employing some attribute renaming scheme for duplicate attributes), we need to take care of the case where no key attributes remain. In this case, the key of the newly created relation consists of all keys attributes that were just merged. This process is iterated, starting from the leaves until we obtain a single flat relation. Here is the transformation of the tree in our example:

1. Position(Price, Bill¹⁺(BillID, BillDate), Job⁰⁺(Service⁰⁺(ServiceName, ServiceDescription), WorkLog¹⁺(LogID))).
2. Position(Price, BillID, BillDate, Job⁰⁺₁(ServiceName, ServiceDescription, LogID)).
3. Position(Price, BillID, BillDate, ServiceName, ServiceDescription, LogID).

We can see that even the isolated translation of a single tree can yield a denormalised relation. The result merges the components of the tree (Bill, Position and Job) in a single relation. The cardinality constraints of the participating relations imply that data might possibly be duplicated. BillDate is an example for this, because it will be repeated for every Position of a Bill. The relation derives its key from Work Log, although the entity type was not a part of the tree. This simplifies the translated relation. The layering was obviously flattened as we intended – we can now cut across the layers without using joins.

The conceptual requirements encoded in the tree are mirrored in this structure. For any given Bill, we have read access to the Service and Price for each of its Positions. While it is too early to make an assessment about the write performance without taking the entire translated schema into account, we can already see the trade-offs that were made (e.g. for updating BillDate). When this result is viewed in the context of the entire schema, the duplicated data is not the only noteworthy feature. Some types have also been decomposed horizontally. For example, we will find some information for every Job that appears as a Position in our relation, but Jobs for which this is not the case are entirely absent. This is something we have to keep in mind when we switch our focus to entire schemas.

### 4.2 Tree Interactions

Now that we have seen how to transform an individual cross-cut from the conceptual to the logical model, we need to expand this method to entire schemas. Remember that we introduced these trees to override the natural layering of the data. It is likely, that the desired organisation of the logical schema reflects this organisation most of the time. We can not demand that the designer decomposes the entire conceptual schema into trees; instead, our transformed trees need to be able to coexist not only with other transformed trees, but also with canonically transformed parts of the schema.

![Figure 5: Two Cross-Cutting Trees](image)

We extend our example schema with another tree as shown in Figure 5. For every Job, a record of the used Consumables should be kept. For a reporting task, we calculate an overview for each Team, Service and the amount of Consumables that were used. In section 4.1, we already derived the translation of the first tree:

Position(Price, BillID, BillDate, ServiceName, ServiceDescription, LogID)

In addition to this relation, we can derive another one for the second tree:

1. used(Amount, Consumable⁰⁺(ConsName), Job⁰⁺(Team⁰⁺(TeamID, Size), Service⁰⁺(ServiceName, ServiceDescription), WorkLog¹⁺(LogID)))
2. \text{used}(\text{Amount, Consumable}^{0,*}(\text{ConsName}), \text{Job}^{0,*}(\text{TeamID, Size, ServiceName, ServiceDescription, LogID}))

3. \text{used}(\text{Amount, ConsName, TeamID, Size, ServiceName, ServiceDescription, LogID})

This translation is quite similar to the result of the canonical transformation. The major difference is that the Team and Service types make an appearance although Work Log would be sufficient to provide identity to Job and the Size of a Team as well as the ServiceDescription are duplicated for each Consumable and Service. This is of course due to the structure of the tree itself.

Each tree has its own individual version of each relevant attribute. This is another kind of duplication that can be introduced by cross-cutting. If one were to update the ServiceDescription of a Service. Such a change would have to be mirrored at least two times. It would be desirable to combine the relational representation of different cuts to minimise such overhead. Unfortunately, there are strict requirements that must be met in order to not lose any information in the process. The semantic meaning of a tree depends heavily on the relationship type in which it is rooted. In the logical layer, this is also reflected in its relational structure and the information that it contains. Two trees with different roots can only possibly share a relational representation, if one of the trees is a subtree of the other, and every instance of the subtree has a corresponding instance of the containing tree. In the conceptual layer, this means that each role linking the root relationship type of the subtree to the root relationship type of the containing tree has a minimum cardinality that is greater than zero.

4.3 Cross-Cut Induced Decompositions

There are two differing views on how the information in a cross-cut should be integrated with the rest of the schema. One possibility is to regard the relations that express the cuts purely as an extra to those that were created by the canonical relational transformation. But breaking the natural layering of the data on the conceptual level also induces a horizontal decomposition of the entity- and relationship-types that are involved in such a cross-cut, which opens up another possibility.

For each cut, we can partition the class of each optionally participating type (i.e. types whose role in the tree has a minimum cardinality of zero) into two sets: those instances that are referenced by a higher layer, and those that are not. In some cases, a useful conceptual distinction arises from this partition. It can be computed by reverse-engineering the logical translation of the tree. The resulting decomposition justifies a relational translation, where the cut is not just an extra to the canonical translation, but can actually serve as a primary store for the decomposed types.

Figure 6 expands on our well known example. It shows the horizontal decomposition that is induced by the first cross-cut at the relationship type Job. Remember that the relational transformation of the cut was

\text{Position}(\text{Price, BillID, BillDate, ServiceName, ServiceDescription, LogID})

\text{BilledJobTeam}(\text{LogID, TeamID})

\text{Team}(\text{TeamID, Size})

\text{WorkLog}(\text{LogId, LogText})

\text{UnbilledService}(\text{ServiceName, ServiceDescription})

\text{UnbilledJob}(\text{TeamID, ServiceName, LogID, LogText})

\text{Consumable}(\text{ConsName, ...})

\text{used}(\text{TeamID, ServiceName, LogID, ConsName, Amount})

Additional foreign keys can be generated:

\text{BilledJobTeam}(\text{TeamID}) \rightarrow \text{Team}(\text{TeamID})

\text{BilledJobTeam}(\text{LogID}) \rightarrow \text{Position}(\text{LogID})

\text{Position}(\text{LogID}) \rightarrow \text{WorkLog}(\text{LogId})

\text{UnbilledJob}(\text{TeamID}) \rightarrow \text{Team}(\text{TeamID})

\text{UnbilledJob}(\text{ServiceName}) \rightarrow \text{UnbilledService}(\text{ServiceName})

\text{used}(\text{ConsName}) \rightarrow \text{Consumable}(\text{ConsName})

The cluster type of \text{used} can not easily be translated to the relational model. The attributes \text{used}(\text{TeamID, ServiceName, LogID}) either reference their corresponding attributes in \text{UnbilledJob}, or \text{BilledJobTeam}(\text{TeamID}) and \text{Position}(\text{ServiceName, LogID}) respectively.

5 Conclusion

The traditional notion of inter-model transformation is based on information capacity. We showed, how shifting the focus to the suitability of a transformation for a given task is required for developing a novel type for those relationships of the first class, we will call \text{Billed Job}, the the type for the latter class we will call \text{Unbilled Job}. Note that this is a partition on the whole class of Jobs. This distinction also applies to each of the lower-order types below Job in the cut, because they were also duplicated. We create the corresponding classes in the entity-relationship schema for Service and Work Log (remember that the latter was added to preserve the identity of the relationship type). Because Team was not part of the cut, it does not have to be duplicated. The inverse types are created for those instances, that were not included in a cut, unless this is not necessary due to cardinality constraints.

Next, the cardinality constraints are adjusted to represent the knowledge that the newly created types were part of a cut, or not. For the \text{Billed} types, this means increasing the minimum cardinality to 1 where it was zero (e.g. Billed Service). For types that were referenced by a relationship type that was split, that means decreasing the minimum cardinality to zero where it was higher (e.g. Team). Finally, the split types are replaced by cluster types for roles which referenced them before the transformation (e.g. \text{used}).

Such a conceptual schema justifies a translation, where a cross cut serves as the single authoritative store for decomposed instances, as manifested in the nonzero minimum cardinalities of the involved types. For the transformation of the entire schema, it is then sufficient to perform a canonical transformation of the part that is not involved in the cut. Finally, relations for unmarked roles and attributes of types in the cross-cut are added.

\text{Position}(\text{Price, BillID, BillDate, ServiceName, ServiceDescription, LogID})

\text{BilledJobTeam}(\text{LogID, TeamID})

\text{Team}(\text{TeamID, Size})

\text{WorkLog}(\text{LogId, LogText})

\text{UnbilledService}(\text{ServiceName, ServiceDescription})

\text{UnbilledJob}(\text{TeamID, ServiceName, LogID, LogText})

\text{Consumable}(\text{ConsName, ...})

\text{used}(\text{TeamID, ServiceName, LogID, ConsName, Amount})

Additional foreign keys can be generated:

\text{BilledJobTeam}(\text{TeamID}) \rightarrow \text{Team}(\text{TeamID})

\text{BilledJobTeam}(\text{LogID}) \rightarrow \text{Position}(\text{LogID})

\text{Position}(\text{LogID}) \rightarrow \text{WorkLog}(\text{LogId})

\text{UnbilledJob}(\text{TeamID}) \rightarrow \text{Team}(\text{TeamID})

\text{UnbilledJob}(\text{ServiceName}) \rightarrow \text{UnbilledService}(\text{ServiceName})

\text{used}(\text{ConsName}) \rightarrow \text{Consumable}(\text{ConsName})

The cluster type of \text{used} can not easily be translated to the relational model. The attributes \text{used}(\text{TeamID, ServiceName, LogID}) either reference their corresponding attributes in \text{UnbilledJob}, or \text{BilledJobTeam}(\text{TeamID}) and \text{Position}(\text{ServiceName, LogID}) respectively.
translation process. A more fine-grained view on what constitutes a suitable logical schema is needed.

We have seen how even simple structures such as trees cutting through the natural layering of an entity-relationship schema can open up new possibilities for inter-model transformations. Some of the typical adjustments that are usually performed in the logical layer can be explained by such structures on the conceptual level. In the case where the reasoning of the engineer for his alterations are aligned with the conceptual explanation, the resulting logical schema can be considered conceptually justified.

For cross-cutting trees, we showed how the translation to the relational model leads to a denormalised schema with duplicated information, yet a lower number of required joins for the relevant types. Such adjustments can now be managed from a conceptual point of view, overcoming the typical disconnect between the implemented schema and the conceptual view.

We expanded on this result, by showing how layered schemas can induce a horizontal decomposition of the involved types, which leads to a more efficient logical representation with fewer duplications compared to the naïve approach.

5.1 Open Problems

A number of open problems remain. The canonical transformation of entity-relationship schemas has the nice property of creating a relational representation that is easy to query and maintain. This is due to the normalisation of the result. Because we focus on suitability and justification, we lose those advantages where they conflict with more important requirements. It should not be the responsibility of the user, to figure out how to maintain, query and use denormalised data. This problem can be overcome by providing an interface at the conceptual level. It would internally use a translation facility for queries and updates in a conceptual language to the transformed schema, which could select the proper relations to take advantage of the duplication when queried and update all instances of a fact when updated. If a true coupling between the conceptual and logical level is desired, this interface should also support schema evolution.

This paper covered a very specific form of conceptual requirement that can shape the translation process. It remains to be seen, if other influences on the logical schema that play a role in practice can also be incorporated in this process. Possible examples include considerations on locking / concurrency, replication, rights management, and auditing. The question of consistency comes into play as soon as heterogeneous requirements are mixed.

Our translation was closely tied to the higher-order entity-relationship model and the relational model. It might be worthwhile to find a more general, unifying approach and research how it is influenced by different data models. There are already approaches that could be used for such an undertaking (Atzeni et al. 2008). Finally, underlying all this is the question what constitutes a “good” inter-model translation. The term suitability is crucial here. It is our opinion, that conceptual justification can be the foundation for its definition.

References


Chen, P. (1976), ‘The entity-relationship
model—toward a unified view of data’, *ACM Transactions on database systems* 1(1), 9–36.


