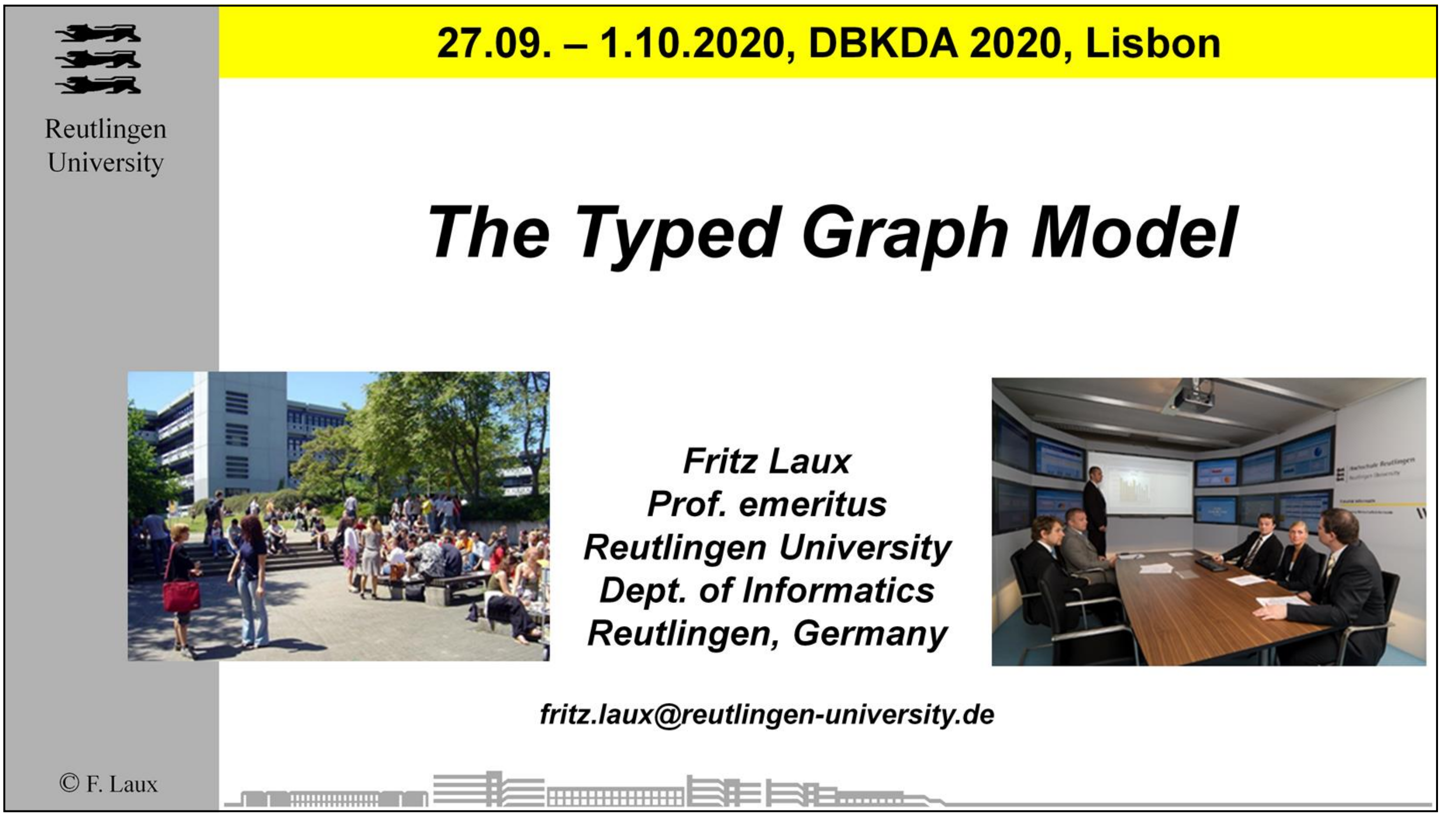


My name is Fritz Laux. I’m a retired professor from Reutlingen University where I was responsible for the database teaching and research since the start of our department.

I am a cofounder of DBTechNet, a European initiative of academics and industry to improve and promote database education.

Over the years I was part of three EU-funded projects and numerous research and development projects in cooperation with industry.

With 40 years of experience in database modelling and design I learned about the importance of database modelling. Data modelling is, and always will remain a crucial part of SW development.

As fashions and practices come and go, I tried to combine the best ideas.

This is why my recent work focusses on Graph Data Models.

Reutlingen University

Contents
Challenges
TGM
Examples
Comparison
Results
Conclusion
References

2 /14

Contents and Aim of the Talk

Contents

- Show that the Graph Model (GM) needs enhancements to make it ready for data modelling
- Present the Typed Graph Model (TGM) as extension of GM
  - ⇨ Formally compact, yet sufficient for the target aim
- Apply and compare the TGM to prevailing data models
  - ⇨ Show and discuss the results (benefits and pitfalls)

Answer the following questions:

- Which enhancements are needed?
- What is the semantic expressiveness of the TGM compared to competing models?
- Is it better matching the way we communicate reality?
- Is there support for multiple abstraction levels?

What can you expect from this presentation?

First I will point out some weaknesses of the original GM and suggest enhancements to make it ready for data modelling.

It will quickly become clear, that we need to capture more semantics.

Second, this leads to the Typed Graph Model (TGM) which provides more semantics and ensures data integrity .

The talk is built around examples for modelling typical relational-, and object-oriented structures by using the TGM.

During the presentation we will answer the following questions:

- Which enhancements are needed?
- What is the semantic expressiveness of our model compared to competing models?
- Is it better matching the way we communicate reality?
- Is there support for multiple abstraction levels?

The presentation will conclude by some modelling guidelines for the TGM.

Reutlingen University

Challenges

- *Advocates of the GM like Robinson et al.[1)] recommend to use specification by example, which uses real objects like the following (p. 42):*

  - The problem with this is that we cannot exemplify all situations
  - The object “Review” depends on the existence of a “User” and a “Performance”
  - We cannot know if Billy is allowed to have multiple reviews (on the same performance?)

- *In order to express this semantics it is necessary to abstract and specify integrity constraints*
  - This means we have to deal with abstract things (like a generic Person) and not only with real objects (like Billy)

The popularity of the Graph Model (GM) stems primarily from its application to social networks. Even if the flexibility of the GM is tempting, its schema-less application is prone to data quality problems.

- Robinson and his colleagues from Neo4J recommend in their book to use "Specification by Example", which uses example objects.

This rises the question if such an instance level model can be used as a schema model as well.

- Let us use an example taken from Robinson's book.

It shows a User named Billy with its 5-star Review on a Performance dated 2012 July 9th .

- The problem with this is that we cannot exemplify all situations.

  For a good data quality, a review should depend on the existence of a user and a performance. But this cannot be derived from an instance model, that is to say, from only one example.

From this example we cannot know if Billy is allowed to have multiple reviews.

- This means that we have to deal with class things (like a generic Person) and not only with real objects (like Billy) and specify if a relationship is mandatory or optional.

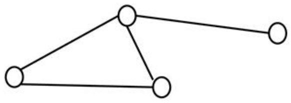
Reutlingen University

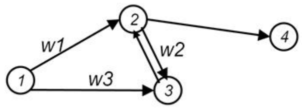
Contents
Challenges
TGM
Examples
Comparison
Results
Conclusion
References

4 /14

Original Graph Definition

- *A mathematical (directed) Graph G = (V,E) is defined as*
  - a set of Vertices V and
  - a set of Edges E connecting 2 (ordered) vertices (u,v), with u,v ϵ V.





  - The vertices can be numbered for identification and the edges may have „weight“ for calculating the cost of a path.

- *Shortcomings for data modelling:*
  (1) Two modelling elements are not sufficient to express data structures
  - e.g. even the relational model has 3 modelling elements
  - We want to distinguish different association types, e.g. inheritance, aggregation

  (2) The Graph Model is originally instance based
  - If we apply the GM on the Schema level, how can we ensure integrity constraints, e.g. capture the multiplicity of an association?

Let us start with a short recap of the original graph definition.

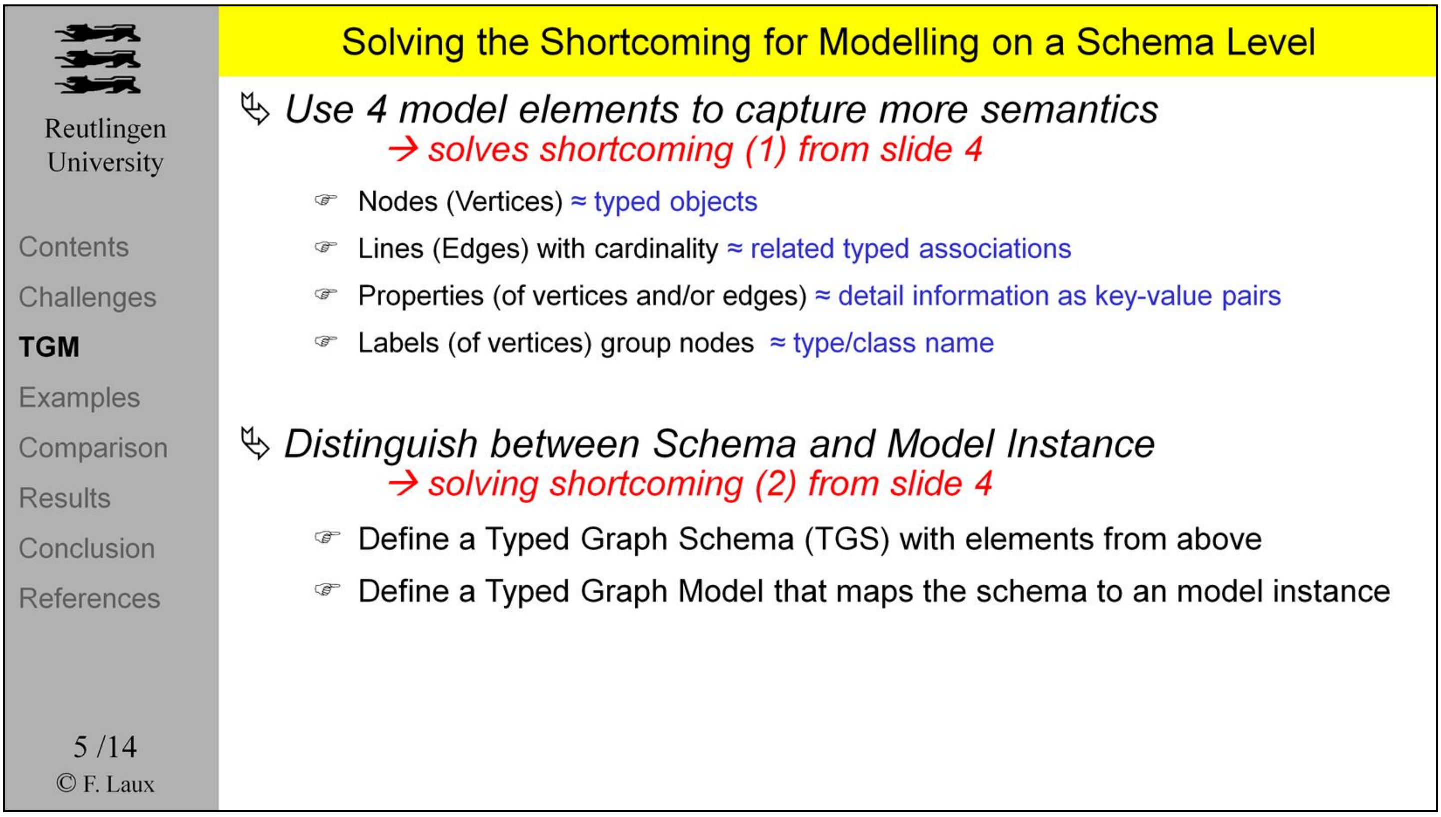
A mathematical Graph G is a tuple consisting of a set of Vertices V and a set of Edges E.

An edge is defined by the pair of vertices that connect these vertices.

Vertices are alternatively called Nodes; they can be numbered for identification and the edges may have a weight for calculating path costs.

- The main shortcomings for data modelling are first of all, that real world objects have structure and properties, which cannot be distinguished in the original GM.

  This weakness was tried to overcome with graph enhancements, like labels and properties attached to the vertices.
- Second, the GM is instance based and therefore captures only a particular situation as we have already seen in the previous slide. The GM cannot express structural constraints.



There have been some research on using the GM on a schema level. The nodes were considered as classes and edges are not instances any more but edge types. The proposed models usually fail when edge cardinality is necessary or models make no sufficient distinction between nodes, properties and their respective data types.

In order to overcome the restricted modelling capabilities we need mainly two extensions:

(1) More modelling elements, and

(2) Clear distinction between the abstraction levels: instance and schema.

To solve the overloading of nodes we add properties and types to the nodes which serve as classes on the schema level.

Nodes have a type. Types are stronger than labels because a type represents the allowed structure and value range of an object whereas a label is only a name.

Edges have a type too which defines the kind of association, for instance, aggregation or generalization.

Reutlingen University

Definition (simplified): The Typed Graph Model (TGM)

- *A typed graph data model is a tuple TGM = (N, E, TGS, ϕ) where:*
  - *N* is the set of named (labeled) hyper-nodes *n* with data types from schema TGS.
  - *E* is the set of named (labeled) hyper-edges *e* with properties of types from schema *TGS*.
  - ϕ is a homomorphism that maps each node *n* and edge *e* to the corresponding type element of TGS.
  - In addition *TGS* offers min-max multiplicity for each end of an edge plus integrity constraints C
- *Using the UML notation for visualizing TGM (and TGS)*

TGM	UML
$n \in N$	class
$e \in E$	association
min-max	(min .. max)
C	Constraints […] or { … }

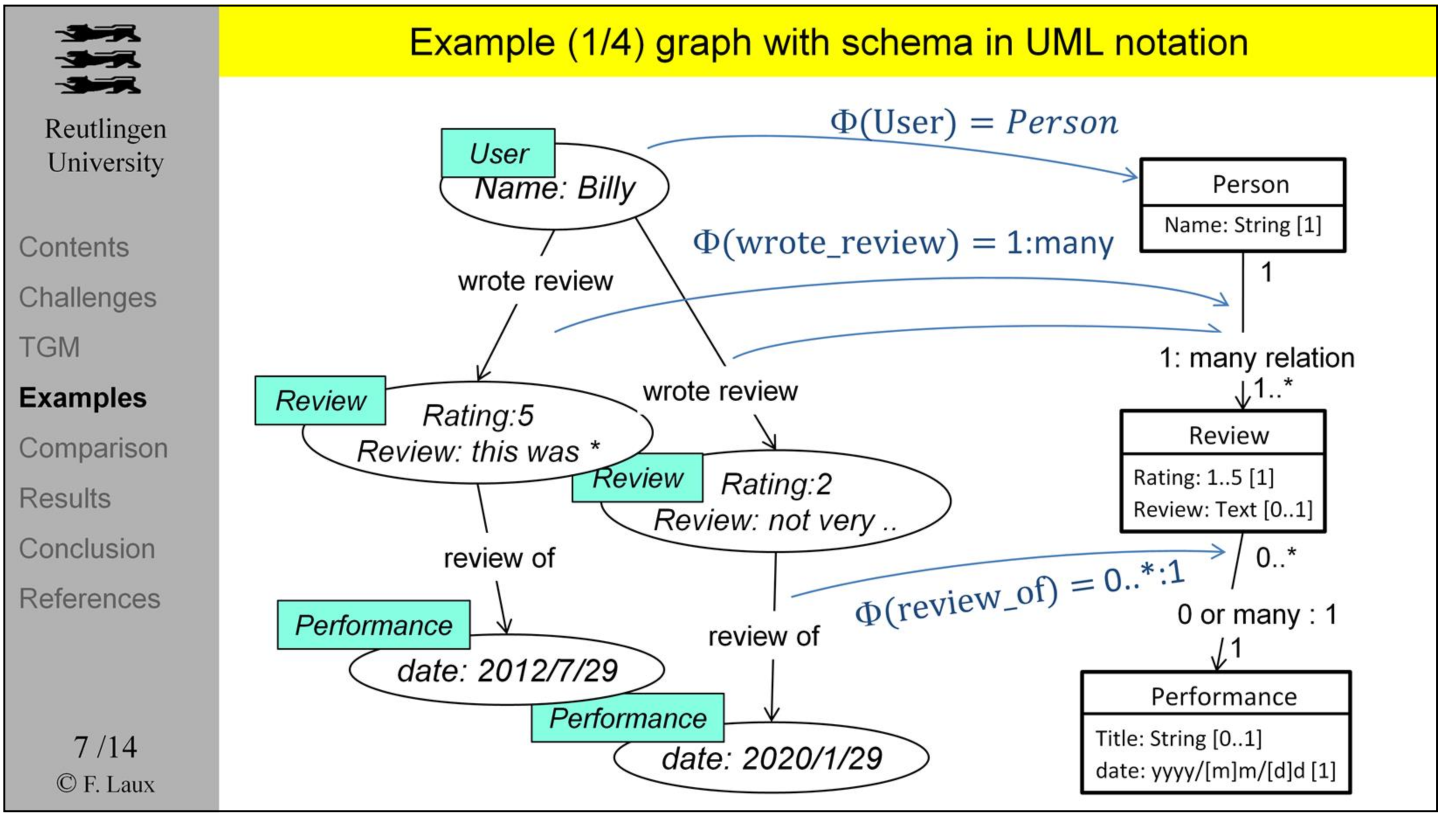
6 /14

- Our Typed Graph Model informally constitutes a property hyper-graph that conforms to a schema.
- It consists of typed hyper-nodes *N* from a Typed Graph Schema *TGS*
- and typed hyper-edges *E* from Schema *TGS*
- Essential for the integrity of a data graph instance is the homomorphism ϕ that maps each instance element to a schema element that defines its type and ensures the integrity of the instance.
- The Typed Graph Schema *TGS* offers min-max cardinality for each edge endpoint and supports additional integrity constraints.

- We use the UML class notation for visualizing nodes and UML associations for edges as it provides a compact rendering and extensions using constraints.

The fact that hyper-nodes and any data types are supported, including user-defined complex data types gives the TGM the potential to build schemata on different abstraction levels as we will see in the second example. This is very important to keep large data models manageable.

Next we turn back to our initial example from Robinson's book and show how it will be modeled and improved.



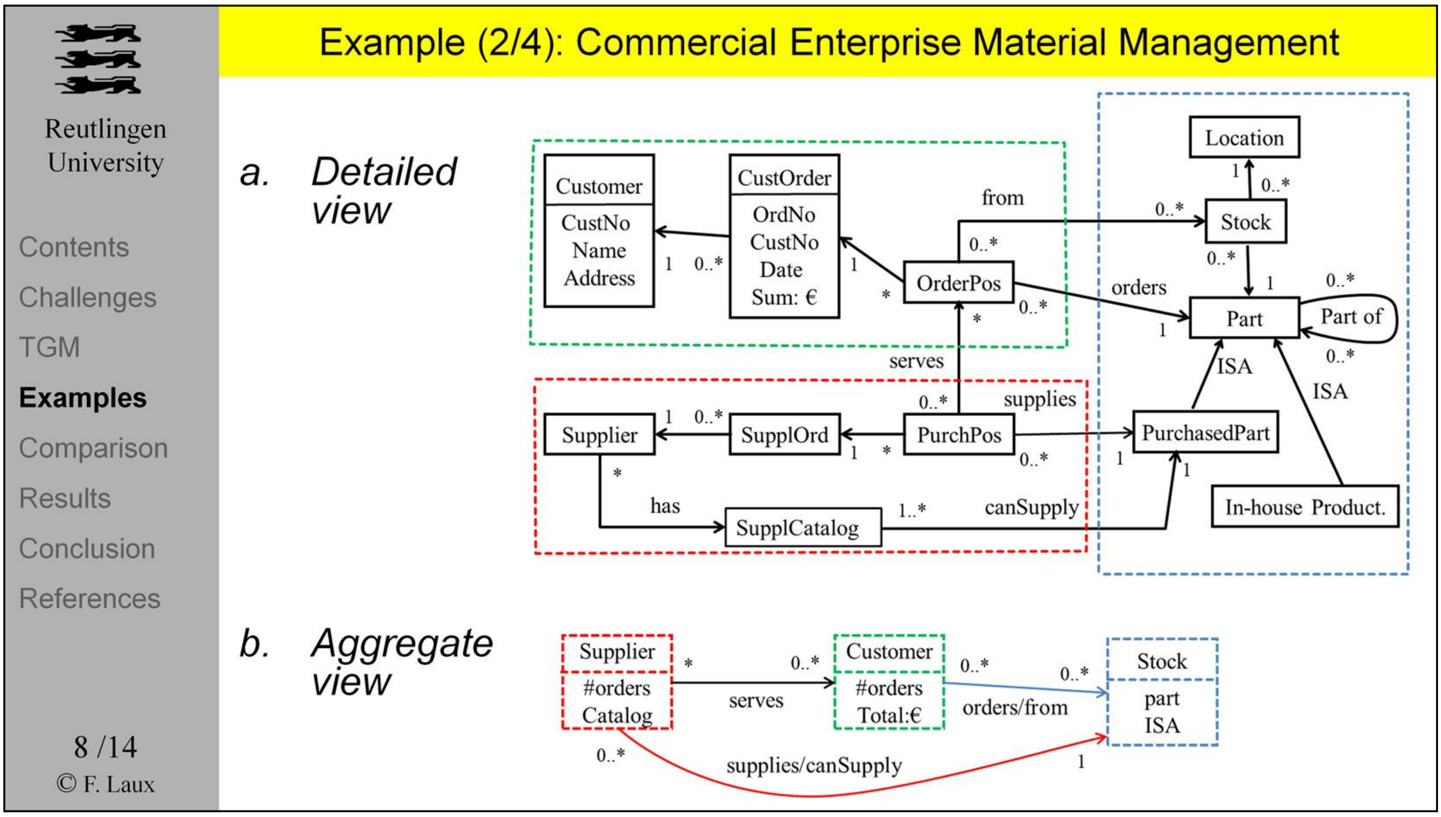
- On the left we have Robinson's example amended to show that Billy is allowed to write more than one Review. We use the same visualization as in their book.
- The schema on the right uses UML for better visual clarity of both levels.

- The function Φ maps the User to the data type Person which ensures that the User must have exactly one name. This is indicated by the number 1 in brackets. The Review itself is tied to the complex type Review with a mandatory Rating and an optional Review text. The Performance can have 2 properties, an optional title and a mandatory date. Even the date format is clearly prescribed by a format template.

The association cardinalities between Person and Review signify that a Person has at least one Review. The mapping Φ ensures that there are no Users without Review.

The homomorphism Φ preserves structure between both graph levels. This means, that wrote review instances are tied to the 1 to many relation and therefore no second author is allowed to link to Billy's reviews.

A Review always refers to exactly one Performance, but, a Performance may have any number of Reviews, including none.



In this example we demonstrate the modelling capabilities and its semantic expressiveness.

The graph schema represents a commercial enterprise that sells products and parts **to** customers. The enterprise assembles products from parts and if the stock level is not sufficient, it purchases parts from different suppliers.

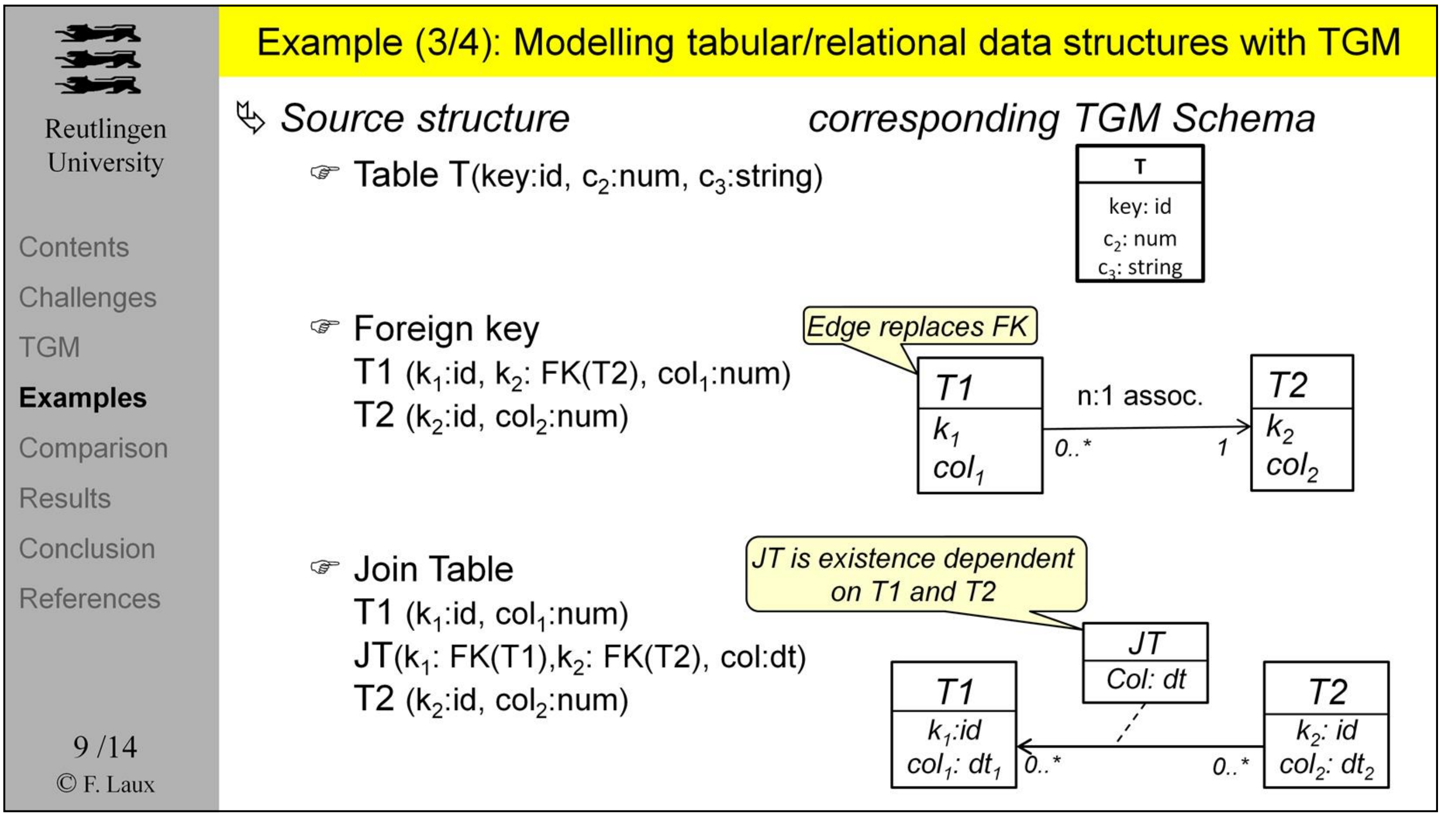
The figure models this situation using TGM in UML rendering.

It demonstrates the abstraction power of the TGM showing two schema abstraction levels.
The upper part (a) shows the model on a detailed level. The properties are suppressed in the diagram for simplicity, except for Customer and CustOrder.

The schema is grouped into 3 disjoint sub-graphs depicted with dashed lines.

In the lower part (b) these sub-graphs are shown as hyper-nodes. This allows a simplified and more abstracted view of the graph model.

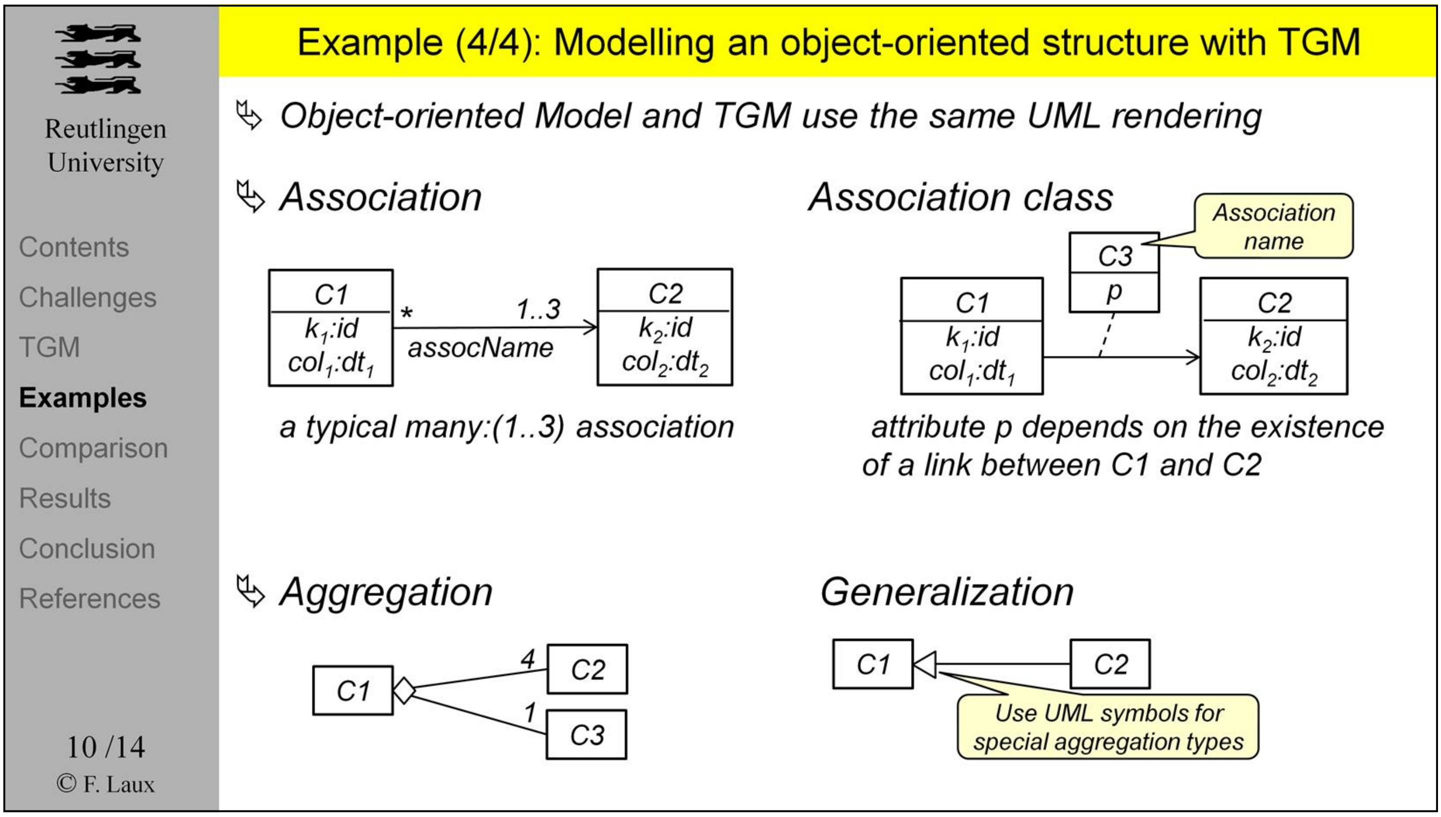
- Also, some aggregate properties (for example, #orders and the total of customer orders) are shown to illustrate the modelling capabilities. The hyper-edges connecting these abstracted nodes must use the most general multiplicity of the edges it combines.
- In our example the edge *orders/from* combines two edge types, the *orders* edge - with an optional 1 - 1 multiplicity and the *from* edge - with unlimited multiplicity, which leads to the most general multiplicity.



The TGM models a relational data structure in a straight forward manner:

- The table name is used as node label and the attributes are used as node properties including their data types.
- If we have a foreign key relationship, it is mapped to an edge with many to one cardinality. The foreign key attribute k_2 can be omitted because the edge carries already the necessary information.
- A tuple of a join table depends on the existence of the foreign keys. Therefore we need to attach the non-key attribute Col in our example here

➢ as property of the edge-type linking the tables. This is rendered in UML as an association class.

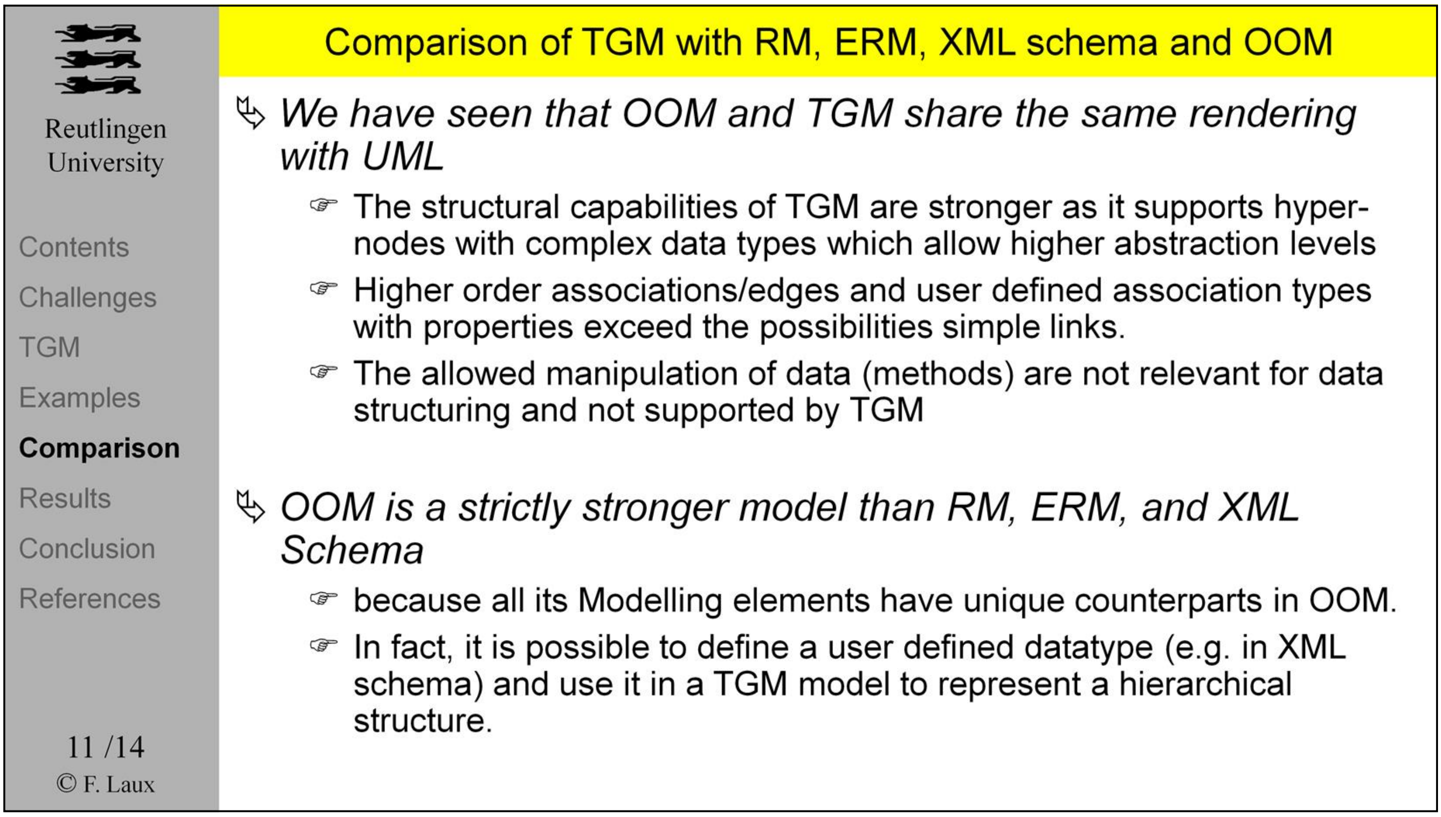


Because we already use UML for rendering the TGM, it is easy to see that classes correspond one-to-one with typed hyper-nodes. Any methods are simply ignored as we only deal with the network structure of the Object-Oriented Model.

Any complex internal class structure can be directly modeled by appropriate data types.

- The UML provides a rich set of association types, which need to be mapped to the types of the edges. Our TGM provides types not only for nodes but also for edges
- With this information it is also possible to model different association types like aggregation, generalization, etc. Even user defined associations are possible, for instance, an aggregate could be further qualified as un-detachable or detachable composition.

The arrow of the edge only indicates the reading direction of the association name but does not limit the navigation of the TGM.

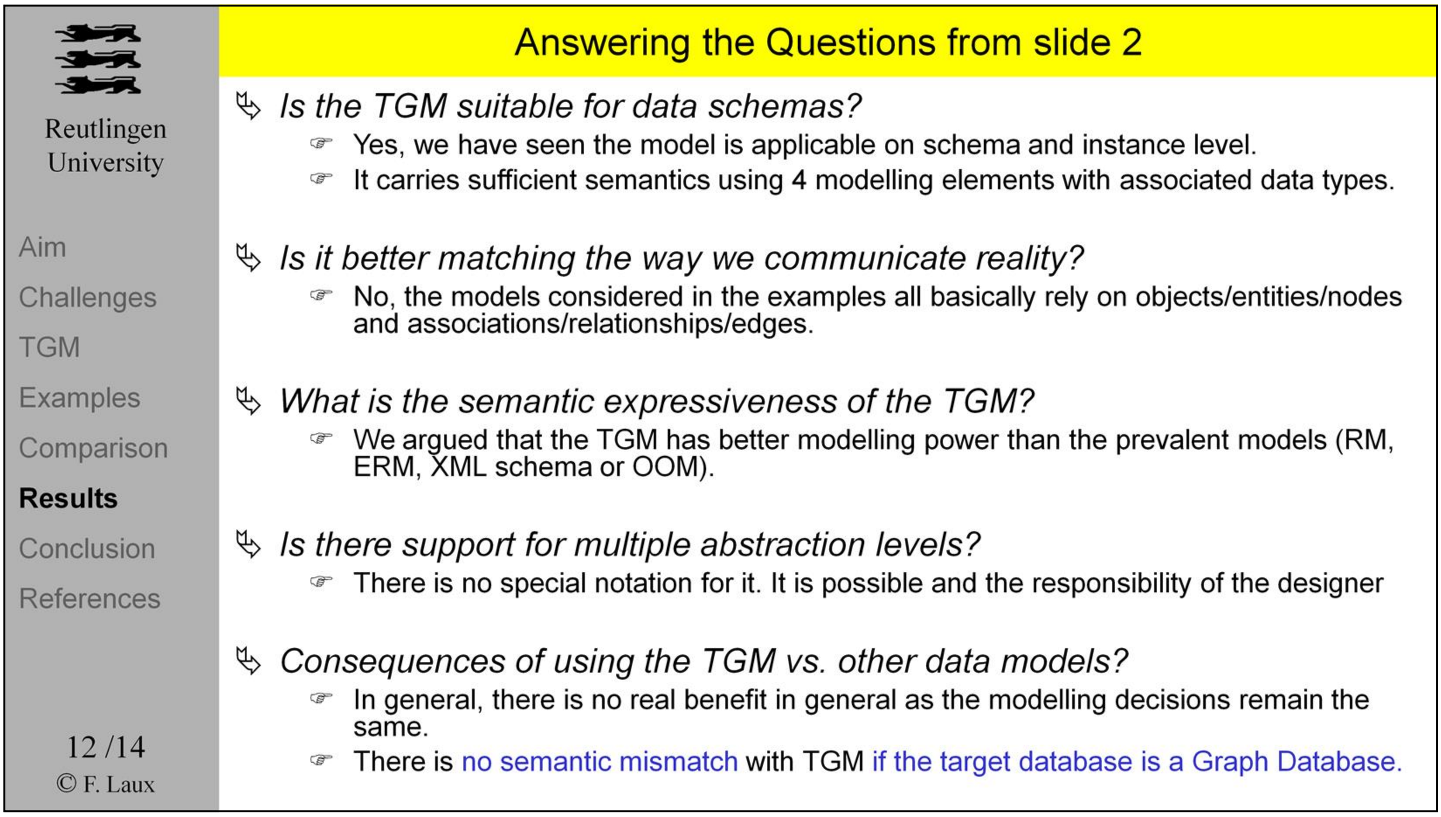


The Object-Oriented Model (OOM for short) and the TGM share the same UML notation. Any modelling element of OOM has a unique UML rendering and corresponds to a distinct TGM modelling element. This means, any object-oriented structure can be represented as a TGM.

This indicates that TGM is stronger than the OOM if only data structuring is concerned.

- The OOM is a strictly stronger model than the RM, ERM, and XML Schema because all its modelling elements have unique counterparts in OOM if we use user defined data types.

In fact, it is possible to define a user defined datatype, for example, in XML schema and use it in a TGM model to represent a hierarchical structure.



Let me now answer the questions from the beginning of my talk:

Is the TGM suitable for data schemas?

> Yes, we have seen the model is applicable on schema and instance level.
>
> It provides sufficient semantics by using 4 modelling elements, namely nodes, properties, labels and edges. Each element has a data type to ensure data integrity.

Is it better matching the way we communicate reality?

> No, the models considered in the examples all rely basically on objects and associations even if they use different names.

What is the semantic expressiveness of the TGM?

> The TGM has better modelling power than the prevalent models. This was only argued and shown by examples. The problem with a formal proof is that there are many variants of the Object-Oriented Model.

Is there support for multiple abstraction levels?

> There is no special notation for it, but with hyper-nodes and hyper-edges. It is however possible and the responsibility of the designer

What are Consequences of using the TGM vs. other data models?

> In general, there is no real benefit as the modelling decisions still remain the same. There is however no semantic mismatch with TGM if the target database is a Graph Database or if link analysis is important.

Reutlingen University

13 /14

Lessons learned

- *Use the TGM on the meta-level*
  - Model entities sets/classes as nodes
  - Model detail information (attributes) as properties
  - Use UML notation for a compact representation
  - It is a modelling decision whether to model a data element as property or as node (compactness vs. precision)
  - Model associations as edges and add properties if needed
  - Add cardinalities to the association type.
- *In real world scenarios the TGM tends to become large*
  - Suppress properties in the diagram
  - Use higher abstraction level aggregates like category, stereotype, component, etc. to provide an overview model
  - Model partial structures separately

Our conclusion is:

The TGM is designed to be used on the meta-level.

This means that nodes represent entity sets or classes with properties that define the details.

The use of UML is preferred for a compact visual representation.

The model is not orthogonal which gives the freedom but also the burden for modelling decisions. The consequence is that it is hard to establish quality criteria for modelling.

When using ternary or higher order edge types it is not always easy to decide on the correct cardinality.

In real world scenarios the TGM like other model tends to become large.

It may help to suppress properties in the diagram and provide separate lists for properties.

Overview diagrams which use higher abstraction level aggregates can provide a view that is easier to comprehend.

To model partial structures separately may help to reduce complexity.

Reutlingen University

References

1) *I. Robinson et al.: Graph Databases, 2nd ed., O'Reilly Media, 2015*

2) *N. Spyratos and T. Sugibuchi: PROPER - A Graph Data Model Based on Property Graphs,* ISIP – 10th International Workshop, Communications in Computer and Information Science, vol.622, Springer, 2015, pp. 23-35

3) *J. Hidders: "Typing Graph-Manipulation Operations", Proc. 9th International Conference on Database Theory (ICDT), 2003, pp. 391-406*

4) *M. Gyssens et al.: "A graph-oriented object database model", IEEE Transactions on Knowledge and Data Engineering, Vol. 6, Num. 4, 1994, pp. 572–586*

5) *R. Angles: The Property Graph Database Model,* Proc. 12th Alberto Mendelzon International Workshop on Foundations of Data Management, CEUR WS Proc., 2018, URL: http://ceur-ws.org/Vol-2100/paper26.pdf

Our work started out from Robinson's book and involved more than 20 sources listed in the paper, out of which we present here the 4 most important ones.