



A graph-based meta-model for heterogeneous data management

Ernesto Damiani^{1,2} · Barbara Oliboni³  · Elisa Quintarelli⁴ · Letizia Tanca⁴

Received: 8 March 2017 / Revised: 28 September 2018 / Accepted: 24 November 2018
© Springer-Verlag London Ltd., part of Springer Nature 2018

Abstract

The wave of interest in data-centric applications has spawned a high variety of data models, making it extremely difficult to evaluate, integrate or access them in a uniform way. Moreover, many recent models are too specific to allow immediate comparison with the others and do not easily support incremental model design. In this paper, we introduce GSMM, a meta-model based on the use of a generic graph that can be instantiated to a concrete data model by simply providing values for a restricted set of parameters and some high-level constraints, themselves represented as graphs. In GSMM, the concept of data schema is replaced by that of constraint, which allows the designer to impose structural restrictions on data in a very flexible way. GSMM includes GSL, a graph-based language for expressing queries and constraints that besides being applicable to data represented in GSMM, in principle, can be specialised and used for existing models where no language was defined. We show some sample applications of GSMM for deriving and comparing classical data models like the relational model, plain XML data, XML Schema, and time-varying semistructured data. We also show how GSMM can represent more recent modelling proposals: the triple stores, the BigTable model and Neo4j, a graph-based model for NoSQL data. A prototype showing the potential of the approach is also described.

Keywords Meta-modelling · Heterogeneous data · Graph-based data model · Graph-based constraints

✉ Barbara Oliboni
barbara.oliboni@univr.it

¹ Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

² Etisalat British, Telecom Innovation Center, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates

³ Dipartimento di Informatica, Università degli Studi di Verona, Verona, Italy

⁴ Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy

1 Introduction

The mass of digital data made available to applications has exploded in the last few years, and a rich panoply of flexible data modelling techniques, both structured and semistructured¹, have been proposed. Even more than volume, data diversity makes it difficult to access data in a uniform way and to integrate the heterogeneous results obtained from queries. Nevertheless, simultaneous use of differently prescriptive data representations has become the norm, and structured data models are often used side by side with semi-structured ones. The result is a plethora of data sources with no unique schema and, consequently, a possibly irregular, incomplete or even totally absent structure.

The current wave of interest in flexible data modelling has been largely driven by applications; in particular, the XML data model [33] has become widespread since the late Nineties because its hierarchical nature made it suitable to catalog-style applications, including large bioinformatics repertoires of proteins, genomes and DNA [16]. Later, attention has shifted to analytics applications (e.g. Web clickstream analysis) where data need to be translated from the data model used for their collection to others more suitable for analysis; this model shifting, often called *model management*, was initially envisaged by Bernstein et al. [7] and has been later investigated for service interoperability [35].

Today, flexible data modelling is playing a major role in the design of NoSQL databases for Big Data applications [12]. All NoSQL databases claim to be schema-less, which means there is no schema enforced by the database management systems. However, during data integration or data exchange, schema-less databases still need a supervised migration, due to the schema implicitly assumed when accessing the data. For example, migrating the schema of the data from a document datasource to a target relational database requires that a domain expert determines an appropriate schema that accurately describes the data, avoiding duplication and sparsity. Although NoSQL databases have been investigated from a number of viewpoints, above all scalability and performance, not much has been done in the way of effective comparison between two different NoSQL systems from the data modelling point of view, and even less to foster cross-system reuse of semi-structured modelling choices. Individual comparisons have been attempted in some vertical domains [31], but a general methodology and formalism for comparing and translating data models, though important during data integration processes, is still lacking.

We believe that being able to assess and compare data models using precise criteria, like run-time model revision, has acquired even more importance in the face of the large-scale storage systems needed for Big Data management. Unfortunately, structured and semistructured data models are often too specific to allow immediate comparison with each other, and do not easily support incremental model design; as a consequence, a unified framework to represent them is mandatory.

In this paper we describe the general semistructured meta-model (GSMM) [19]. A simple meta-model which accommodates both structured and semistructured information, GSMM leverages the use of *constraints* to accommodate all kinds of structures in a truly flexible way. Thanks to this distinctive feature, GSMM can be applied for the translation of any data model proposed in the literature into a common formalism, and is useful for easy *a priori* comparison and discussion of the features of concrete models, such as allowed sets of values, handling of object identifiers, relationship representation, and support for run-time model

¹ We say that data are *semi-structured* when, although some structure is present, it is not as strict, regular, or complete as the one required by the traditional database management systems [1].

revisions, e.g. to adapt to new query and access patterns; moreover, it supports effective inter-model translation and design.

GSMM includes a graph-based language, named *General Semistructured Language* (GSL), used to express the queries and the constraints in a concise and unambiguous way, as suggested by Bekiropoulos et al. [5], and more recently by Fan and Lu [21]. Rather than being a formal representation of schema-based semistructured data using tree grammars as formal framework [26], in the wake of the proposals of Atzeni et al. [3] and Bernstein et al. [7] our highly expressive meta-model and language accommodate semi- or fully structured data, allowing the representation of intensional information where a rigid schema is not possible.

Indeed, to deal with data that are “schema-less” and “self-describing”, we allow the modeller to impose restrictions on the structure of data by means of *constraints* graphically expressed in GSL. In the line of widely accepted standards like XML Schematron [6], our constraints are not expressed as a part of the schema, but stand by themselves and are directly applied to the data. In this way, our meta-model provides the data designer with a powerful tool for enforcing the desired degree of precision of the structure, supporting flexibility at the data representation level.

Differently from XML Schematron and in line with the most recent Data Modelling trends, we choose the graph paradigm because it is readily understood and widely accepted by data modellers. Indeed, graphs are a natural formalism to express relationships between concepts and are enjoying huge popularity among non-specialists, for instance, as a way to represent social network information (e.g. Twitter, Facebook, and LinkedIn). Also, graph-theoretical algorithms, such as procedures to compute sub-graph matching, are well understood and studied in the literature.

GSMM is based on a *generic* graph that can be instantiated into a number of *concrete* models by providing a) values for a restricted set of parameters (labels) and b) some high-level constraints, themselves represented as graphs. Although our meta-model is entirely implementation-agnostic, we discuss in detail its application to a number of practical data models, including the relational model and the graph-based model used by NoSQL databases like Neo4j [23]. Of course, we cannot show how to apply our meta-model to all possible data models, but our worked-out examples aim to provide designers with the necessary intuition of carry out their own meta-model-based comparisons between any two of the many available NoSQL models, including *column-family* models like BigTable.²

The structure of the paper is as follows: in Sect. 2 we introduce the GSMM meta-model, and in Sect. 2.2 we describe GSL, the graph-based formalism to express queries and constraints on GSMM data. In Sect. 2.3 we describe different types of constraints, while in Sect. 3 we apply them in order to represent and compare some well-known semistructured data models with our unified formalism. In Sect. 3.8 we classify the set of parameters for inter-model comparison, and in Sect. 4 we report an example of inter-model translation. In Sect. 5 we describe related work, and in Sect. 6 we sketch some conclusions and possible lines for future work.

2 The meta-model and the graph-based constraints

Our self-contained, graph-based meta-model can represent various aspects of (semi)-structured data, such as static or dynamic information, crisp or fuzzy data; furthermore, it is general enough that most data models proposed in the literature can be derived from it,

² Big Table is the model shared by popular NoSQL databases like Apache HBase and Cassandra [13].

including the relational model, OEM [29], DOEM [14,15], XML [33], WG-Log [20], and PSTDM [17], just to name a few.

We will apply GSMM also to represent recent data models inspired by OEM, like Neo4j (Sect. 3.7), which is graph-based and has proven exceptionally suitable to express (and explore) local relationships between nodes. Also, we will handle *BigTable* models, also called *soft schemata* [13], which can be seen as a semistructured version of standard relational schemata. BigTable defines a variable set of columns to be chosen at instantiation time within a *column family* and, consequently, allows choosing among multiple structures for each table entry.

Definition 1 A *GSMM graph* is a directed labelled graph $\langle N, E \rangle$, where $N = \{n_1, \dots, n_k\}$ is a (finite) set of nodes n_i , each associated to a tuple of labels L_{n_i} , with $|L_{n_i}| \geq 0$ and $|L_{n_i}| = |L_{n_j}| \forall i, j \in \{1, \dots, k\}$ and $E = \{e_1, \dots, e_p\}$ is a set of edges $e_j = \langle (n_h, n_k), R_{e_j} \rangle$, with n_h and n_k in N , and R_{e_j} a tuple of labels such that $|R_{e_j}| \geq 0$ and $|R_{e_i}| = |R_{e_j}| \forall i, j \in \{1, \dots, p\}$.

In order to represent the data, we must associate graph nodes with *content* (i.e., a *value*) by means of node labels. *Simple nodes* are nodes whose content label is a value, such as an integer or a string. *Complex nodes* have a \perp (undefined) value for the content label, showing that they represent abstract objects. The content of a complex node n is actually the sub-graph rooted in n .

2.1 Instantiation GSMM parameters

In order to obtain a specific *concrete model* suitable to represent data in a given context, all one has to provide is a set of *instantiation parameters*, which are the *node and edge label cardinalities*, and *the domains of node and edge labels*. In other words, the cardinalities and domains of the sets of node and edge labels are model-dependent, and fixed: once a concrete model has been instantiated, all its nodes have the same number of labels, and the same happens for all its edges. Among the meta-model instantiation parameters are *the sets of base types*, used as domains of the content labels of simple nodes. By delegating all the specific model features to the choice of the instantiation parameters, the comparison between different concrete models becomes straightforward, since they exactly express the concrete models. The comparison criteria are listed below:

1. *Cardinality of the tuple of node labels*. The higher is cardinality, the wider is the set of properties that can be associated with each node in the concrete model.
2. *Cardinality of the tuple of edge labels*. The tuple of labels that can be associated to edges shows the granularity of the concrete model's representation of semantic relationships between objects. For example, the OEM model represents only the containment relationship because a single edge label is actually used to represent the name of the edge endpoint, whereas the WG-Log model includes an edge label to specify the semantics of a relationship between two objects, which in turn have their own labels. So the cardinality of WG-Log edge labels is higher than the one of OEM.
3. *Domains of node and edge labels*. The sets of node and edge labels, together with their domains, allow to compare the application contexts of concrete models. In particular, labels may range over:
 - *time intervals*, allowing the representation of time by associating a temporal label to the attached item (node or edge);

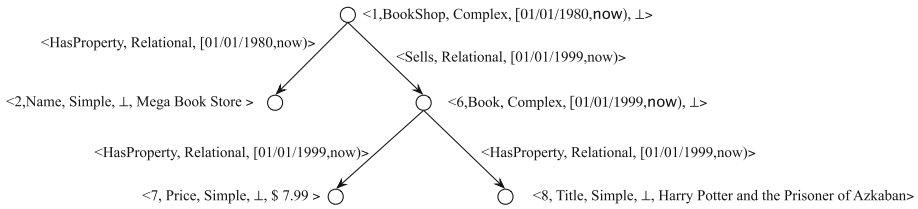


Fig. 1 A simple example of GSMM graph for a temporal database

- *the singleton set {isa}*, for the representation of specialisation/generalisation relationships;
- *object identifiers*, hence the *OID label* associated to nodes allows explicit *OID representation*;
- *simple values* (i.e., *base types*) admitted in the concrete model, e.g. natural numbers, to be used to represent ordering between the set of children of a chosen node. Thus, we can compare models with respect to the set of allowed base types;
- *simple values/OID pairs*, i.e., simple values that are replicas of objects represented elsewhere. Checking whether this type is supported, we can assess the concrete model’s degree of capability for de-normalisation, an important flexibility feature.

For example, in temporal applications [28] the node labels in L_n are the object identifier, the node name (i.e., a string), the node type (Complex or Simple), the content (an elementary value, e.g. a number, a string), and the temporal element ranging on union of time intervals, thus $|L_n| = 5$. The edge labels in R_e are the edge name, the edge type (Temporal or Relational) and the temporal element, so $|R_e| = 3$. In Fig. 1, we report a simple GSMM graph representing temporal information related to *Mega Book Store* about the book *Harry Potter and the Prisoner of Azkaban*.

Some or all the instances of a particular concrete model will share some additional properties that depend on the real-world objects they represent or on the semantics of objects and relations taken into account by the model. In our approach, these properties are represented by means of *GSL constraints*, added to the concrete model.

2.2 The GSL language

Classical database constraints are used to impose (semantic) restrictions on data; typically, they are expressed with reference to a schema of which themselves become a part. In the case of flexible data models, however, often there is no a priori, and thus a different notion of constraint is required. *GSL* is capable of expressing constraints in ways that schema languages cannot. For example, by means of *GLS* we can require that the content of an element be controlled by one of its siblings, or impose that the root element of a tree, regardless of what type it belongs to, must have specific attributes. More importantly, besides being used to constrain specific values, *GSL* constraints can be stated for the entire data model we want to represent, becoming part of the description of what that model can or cannot express. We start by introducing the general notion of *rule*, which is either a query or a constraint, to be applied to instances of *GSMM* data graphs. In general, our rules are composed by (i) a *graph*, which is used to identify the sub-graphs (i.e., the portions of a database where the rule is to be applied), and (ii) a set of *formulae*, which dictate the restrictions imposed on those subgraphs.

For the graph part of a rule we use a variant of G-Log [30], a Turing complete complex object query language. GSL queries are composed of *coloured patterns*, i.e., graphs whose nodes and edges are coloured. A GSL rule has three colours: *red solid* (RS) and *red dashed* (RD) indicate, respectively, information that must and must not be present in the instance where the rule is applied, while *green* (G) indicates a desired situation in the resulting instance.

Unlike G-Log [30], in GSL we express forbidden situations by means of negated formulae. This does not increase the expressive power of GLS, yet it makes rules and rule sets much more readable; it is easy to prove that the two formalisms are equivalent.

In GSL, the specification of logical formulae associated with rules allows to predicate on the variable labels that appear in the graph part. In particular, we introduce two sets of variables $\mathcal{V}_{\mathcal{L}}$ and $\mathcal{V}_{\mathcal{R}}$, used as node labels and edge labels in GSL rules. In general, variables in $\mathcal{V}_{\mathcal{L}}$ and $\mathcal{V}_{\mathcal{R}}$ may range over domains of node and edge labels of the considered concrete model, or may assume an undefined value (i.e., \perp) when the label itself does not have a specific value.

Definition 2 A *coloured directed labelled graph* is an ordered pair $\langle G, Col \rangle$, where $G = \langle N, E \rangle$ is a GSMM data graph and $Col : N \cup E \rightarrow \{RS, RD, G\}$ is a total function. For each subset C of $\{RS, RD, G\}$, CG_C denotes the part of the coloured graph CG containing only the colours of C .

For example, $CG_{\{RS\}}$ represents the red solid part of CG . Now we introduce the construct we use to specify rules:

Definition 3 A *GSL graph* \mathcal{G} is a pair $\langle G, \mathcal{F} \rangle$ where G is a *coloured directed labelled graph* $\langle \langle N, E \rangle, Col \rangle$ and \mathcal{F} is a set of formulae. Moreover, the following properties hold:

- node labels can be either constants or variables in $\mathcal{V}_{\mathcal{L}}$;
- edge labels can be either constants or variables in $\mathcal{V}_{\mathcal{R}}$;
- \mathcal{F} is a set of Conjunctive Normal Form formulae on the constants and variables of \mathcal{G} .

We remark that GSL graphs are not necessarily connected. Examples of GSL graphs are shown in Figs. 2, 3c, 4 and 5,³ because the constraint applies to any pair of nodes connected by an edge, independently of their actual labels; the difference between GSL graphs used to express queries and those representing constraints is in the semantics of their application on a given instance (see Definition 9).

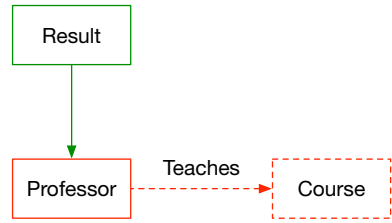
Indeed, when we use GSL graphs to express queries, the graph itself is applied to an instance (that in general does not satisfy it), and its application consists in modifying the instance, so that the obtained graph is satisfied by the result. In other words, query semantics is given as a set of pairs of instances (I, I') , where $I' \supseteq I$ is the result of applying the query to I .

For example, the query in Fig. 2 requires to find all professors who do not teach any course (note the use of a RD sub-graph for expressing the negation). Its application adds to the original instance a node labelled “Result” with some outgoing edges pointing to all the Professor nodes satisfying the requirements specified in the query.

When a GSL graph is used to express a constraint, again, *an instance satisfies the rule iff, whenever the red part is satisfied, also the green part is satisfied*. However, in this case we do not require the input instance I to be *modified* to satisfy the rule, but only check whether the rule is satisfied by I itself.

³ In the remainder of the paper we denote constants by means of lowercase words, whereas words denoting variables start with a capital letter.

Fig. 2 A GSL graph representing a query for finding all professors who do not teach any course



Applying queries or constraints is a morphism between graphs representing rules and graphs representing database instances. We formalise this morphism as *embedding* [30]:

Definition 4 An *embedding* i of a labelled graph $G_0 = \langle N_0, E_0 \rangle$ into another labelled graph $G_1 = \langle N_1, E_1 \rangle$, is a total mapping $i : N_0 \rightarrow N_1$ such that:

1. $\forall n \in N_0, L_{i(n)} \doteq L_n$ (where \doteq means that if both labels in the same position are constants, they must be equal, or if in a given position, one of the labels is a variable then it is mapped on the corresponding constant), and
2. $\forall \langle (n_1, n_2), L \rangle \in E_0 : \langle (i(n_1), i(n_2)), L \rangle \in E_1$.

An embedding i is also extended to edges by defining the mapping $i(\langle (n_1, n_2), R \rangle)$ as $\langle (i(n_1), i(n_2)), R \rangle$.

Thus, a graph is embeddable into another one if they share the same paths and if the relation between the first and the second graph is a function.

The following two definitions specify the concept of graph matching with respect to the positive or negative requests represented in coloured graphs.

Definition 5 Let G be a graph, $C = \langle \langle N, E \rangle, Col \rangle$ a coloured graph, and $C' = \langle \langle N', E' \rangle, Col' \rangle$ a subgraph of C . Let b_1 be an embedding between C' and G and b_2 be an embedding between C and G . The embedding b_2 is an *extension* of b_1 if $b_1 = b_2 \upharpoonright N'^4$.

Definition 6 Let G be a graph, $C = \langle \langle N, E \rangle, Col \rangle$ a coloured graph, and $C' = \langle \langle N', E' \rangle, Col' \rangle$ a subgraph of C . An embedding b between C' and G is *constrained* by C if either $C = C'$, or there is no possible extension of b to an embedding between C and G .

In other words, we may informally say that in GSL (like in G-log) a semistructured instance *satisfies* the graph part of a rule, if every embedding of the red solid part of the rule in the instance that is constrained by the red dashed part, can be extended to an embedding of the whole solid part (red and green).

Definition 7 Let G be a graph and $C = \langle C, \mathcal{F} \rangle$ a rule. C is *applicable* in G if there is an embedding of $C_{\{RS\}}$ in G .

Definition 8 Let G be a graph and $C = \langle C, \mathcal{F} \rangle$ a rule. G *satisfies* C ($G \models C$) if either C is not applicable in G , or, for all embedding b of $C_{\{RS\}}$ in G that are constrained by $C_{\{RS, RD\}}$, b can be extended to an embedding b' of $C_{\{RS, G\}}$ in such a way that the set of formulae \mathcal{F} is true w.r.t. the variable substitution obtained from b' .

Consider, for example, the graphs G_T and G_F and the constraint R in Fig. 3, requiring that, whenever a b node has a “child”, that child is labelled c . G_T satisfies the constraint R , whereas G_F does not satisfy the same constraint because of the subgraph in the dashed region.

⁴ The notation $b_2 \upharpoonright N'$ stands for the restriction of mapping b_2 to the nodes in N' .

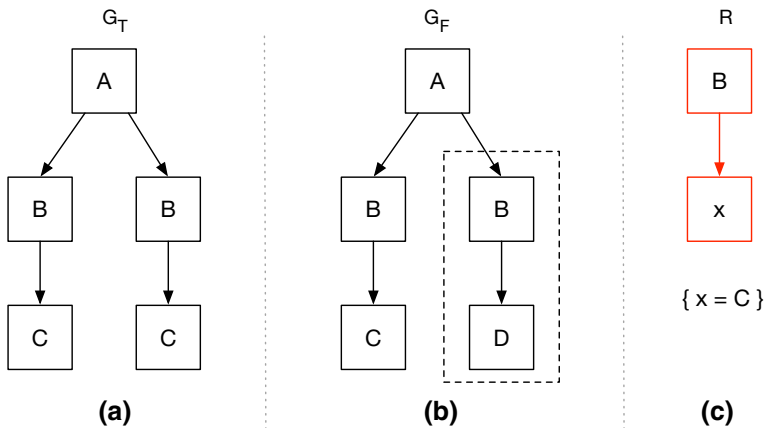


Fig. 3 Constraint satisfaction: only the graph G_T satisfies the constraint R

Definition 9 Let $\mathcal{G} = \langle G, \mathcal{F} \rangle$ be a GSL graph. $Sem(\mathcal{G})$ is a set of pairs $\{\langle I, v \rangle\}$, where:

- I is an instance, and
- each v is a labelled graph $I' \supseteq I$, such that $I' \models G$ and I' is minimal, **if \mathcal{G} is a query**;
- $v \in \{0, 1\}$, and $v = 1$ if $I \models G$, $v = 0$ otherwise, **if \mathcal{G} is a constraint**.

We remark that, in order to reduce constraint checking time, one can check violation instead of constraint satisfaction. Intuitively, there is a constraint violation if there is at least a subgraph G_1 of G matching (with respect to embedding notion) the graph part of the constraint that does not satisfy the formulae in \mathcal{F} . The set \mathcal{F} is the conjunction of its formulae, and thus, there is a violation if at least one is false.

2.3 Constraints

In the next section, we describe the use of GSL for the representation of the constraints that express restrictions on the structure and types of the data entities supported by a concrete data model. By comparing the constraints of two different concrete models, we obtain a qualitative assessment, or even a quantification, of their flexibility.

To start with, however, we shall familiarise with the notation by observing some simple constraints that hold for a specific database instance, representing information about Professors, Students and Courses, rather than for an entire data model [10,11].

We consider a database whose model is expressed according to GSMM and represent some simple constraints in Fig. 4. The constraint of Fig. 4b contains a formula that imposes restrictions on the possible values of a label. In Fig. 5 we show some **cardinality constraints**. The application of both Fig. 5a, b expresses the constraint stating that every Professor Teaches *exactly one* Course.

3 Constraints and concrete models

In this section we describe the two types of constraints supported by our meta-model.

- **High-level, or “concretisation”, constraints.**

Concretisation constraints hold *for all instances of a given concrete data model*. They

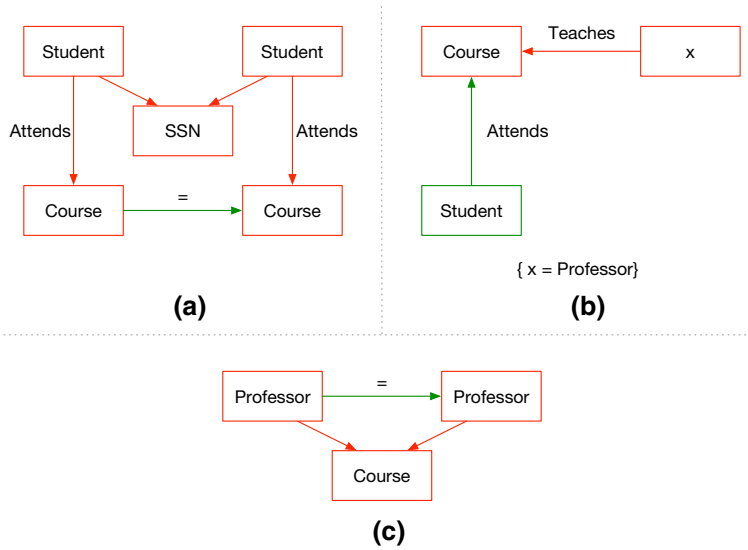


Fig. 4 **a** A constraint asserting that the SSN of a student functionally determines the courses the student is taking; **b** a constraint stating that courses are only taught by professors and requiring (green part) that for each course there is at least one student attending it; **c** the constraint forbids two different professors to teach the same course

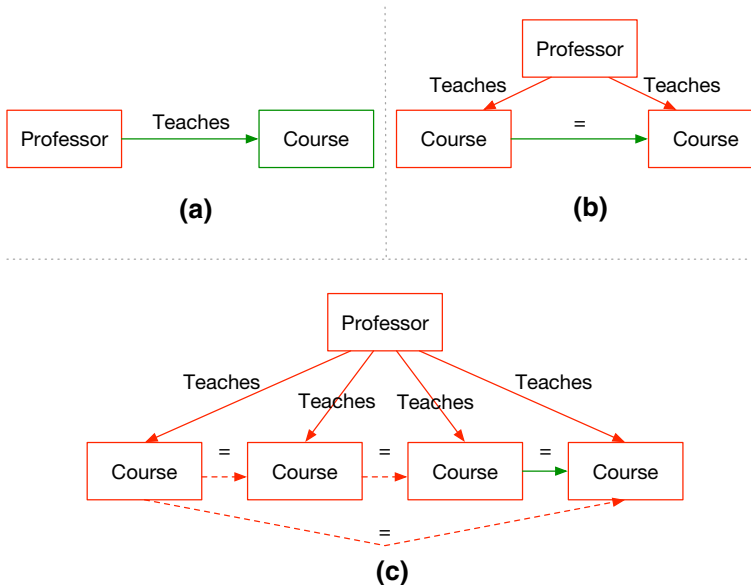
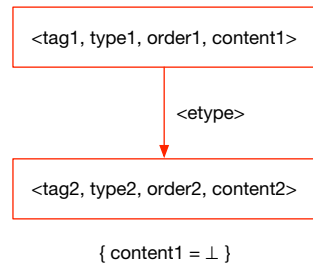


Fig. 5 **a** Every Professor Teaches *at least* one Course; **b** every Professor Teaches *at most* one Course; **c** each Professor must teach exactly three courses

provide a concise representation of the data model’s expressive power.

For example, to characterise the XML data model we can use a concretisation constraint stating that each attribute must be connected to its parent element by means of

Fig. 6 Content label of non-terminal nodes must be undefined



an “*attribute-of*” edge. For all concrete models supporting a “content” label, we should also specify the constraint that abstract (i.e., non-terminal) nodes content is undefined (as anticipated in natural language after Definition 1).

Figure 6 shows this constraint in the XML context. We remark that in this case labels are composed by variables.

– **Low-level, or “domain related”, constraints.**

These constraints are defined on instances of concrete models. The introductory examples shown in Sect. 2.3 belong to this category. While all the instances of a particular concrete model must satisfy all the high-level constraints specified for that model, only some of the instances satisfy a particular low-level constraint.

For example, the low-level constraint of Fig. 11g (on the GSMM temporal instance of Fig. 11b) dictates that the time interval of a Book must start after the time interval of its Author.

This constraint makes sense in all the instances representing bookshops information and actually contributes to the semantics of the “Author_of” relationship. Depending on the particular concrete model, we may need to represent also *dynamic* low-level constraints, defined on temporal semistructured data to impose restrictions on data evolution. For example, the constraint of Fig. 11g is a dynamic low-level constraint.

We remark that given two models M_1 and M_2 the way to translate M_1 into M_2 may not be unique. For instance, the data designer might be called to make choices about how to translate a model where order is supported into an unordered model (e.g. XML versus relational).

We do not provide guidelines to the designers except for the recommendation to be coherent in the choices made within the same system.

Next, we show the use of our meta-model to specify some classic flexible data models. This exercise will help us to:

1. Show how the features of GSMM allow the expression of many different constraints;
2. Show how to perform an inter-model comparison w.r.t. the modelling constructs provided by the different models (Sect. 3.8 and Table 1);
3. Show our meta-model capability of supporting inter-model translation (Sect. 4).

3.1 The relational data model

Mappings between graph-based and relational data models have been deeply investigated [32]. No wonder that we can represent both a relational schema and its instance by using a *semistructured data graph* $\langle N, E \rangle$, where:

- the cardinality $|L_n|$ of the sets of node labels is 3. Each node $n_i = \langle oid_i, name_i, type_i \rangle$ has an object identifier $oid_i \in UID$, and the node type $type_i \in \{ DBname, RelationName, Att, KeyAtt, SetOfAtt, AttValue \}$.
- the cardinality $|R_e|$ of the tuple of edge labels is 0, and each edge $e_j = \langle (n_h, n_k) \rangle$ with n_h and n_k in N .

An example of a relational database schema, a possible graph-based representation for the schema and an instance (similar to the one proposed by Virgilio et al. [32]), is reported in Fig. 7. We also represent the related GSMM graphs.

In Fig. 7e, f we represent a primary key and a foreign key constraint.

Representing instances of a relational schema by using our graph-based data model is a straightforward application of the same technique, as shown in Fig. 7d.

3.2 The object exchange model

The object exchange model (OEM) [29] has been introduced in the context of the seminal TSIMMIS project carried out at Stanford University, one of the first attempts to support fast integration of heterogeneous information sources.

OEM is a graph-based data model where the basic idea is that each object has a label that describes its meaning. The label is used to extract information about objects that represent the underlying data.

The information that can be extracted is limited to the inclusion/containment relationship; indeed OEM does not actually represent the semantics of relationships between objects.

An OEM graph is a GSMM rooted graph $\langle N, E, r \rangle$, where:

- the cardinality $|L_n|$ of the sets of node labels is 3. Each node $n_i = \langle oid_i, type_i, content_i \rangle$ has an object identifier $oid_i \in UID$, and the node type $type_i \in \{ Root, Object \}$.
- the cardinality $|R_e|$ of the tuple of edge labels is 1, and each edge $e_j = \langle (n_h, n_k), R_{e_j} \rangle$ with n_h and n_k in N , has a label $R_{e_j} = \langle ename_j \rangle$, where $ename_j$ is actually the name of the pointed node n_k .

Again, $r \in N$ is the root of the graph, and the root node has type “Root”. Consequently, an OEM graph must satisfy the high-level constraints in Fig. 8a, b.

3.3 XML and XML schema

In this section we apply our meta-model to derive a simple concrete model supporting XML information and XML Schema. XML datasets are often called *documents* because they can be serialised as plain text. However, unlike generic text documents, XML documents are not completely unstructured.

A XML document is a sequence of nested elements, each delimited by a pair of start and end tags (e.g., $\langle tag \rangle$ and $\langle /tag \rangle$). The sequence is itself enclosed by a *root element*. Figure 9 shows a well-formed XML document.

3.3.1 Plain XML

Plain XML documents like the one in Fig. 9 can be represented quite straightforwardly in our framework: a *Plain XML graph* is a GSMM rooted graph $\langle N, E, r \rangle$, where:

- the cardinality $|L_n|$ of the sets of node labels is 4. Each node n_i has as tuple of labels $L_{n_i} = \langle tag_i, type_i, order_i, content_i \rangle$; the type label $type_i$ indicates whether the node

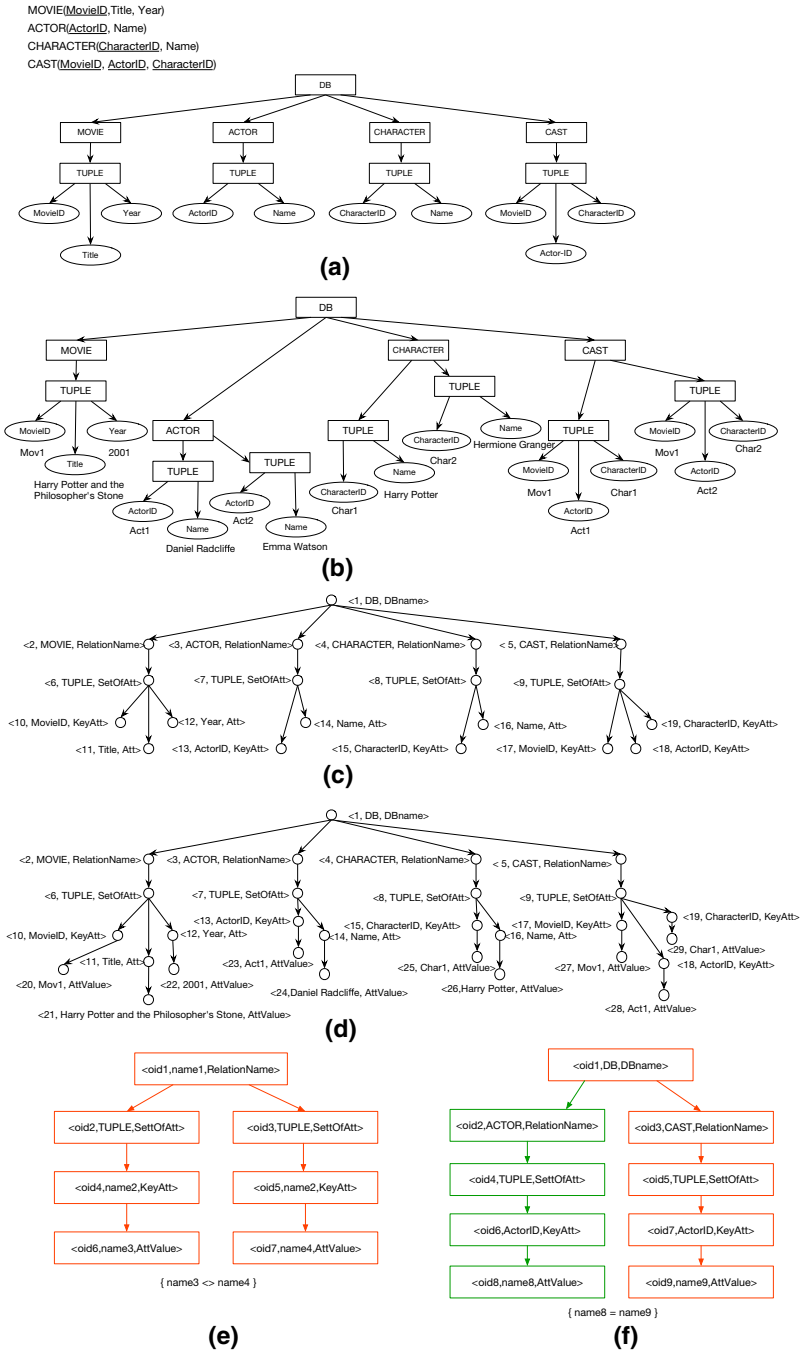


Fig. 7 **a** A relational database schema and its graph-based representation; **b** graph-based representation of a relational instance; **c** the GSSM graph for the relational schema and **d** corresponding instance; **e** a primary key constraint (with the key composed by a unique attribute); **f** one of the foreign key constraint on the MOVIE relation

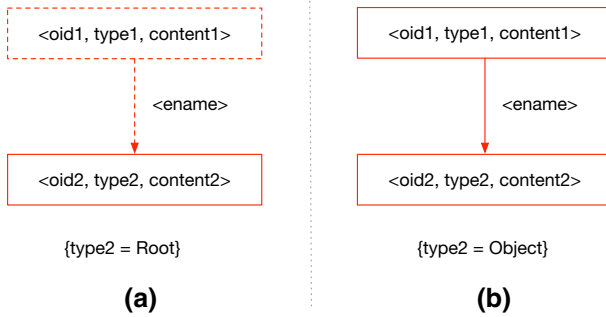


Fig. 8 **a** A node without incoming edges must have type *Root*; **b** each node with an incoming edge has as type *Object*

```
<?xml version="1.0" encoding="UTF-8"?>
<computer>
  <maker> Toshiba </maker>
  <model >
    <modelname serialcode = "12303B" > Satellite Pro 4200 </modelname>
    <year> 2001 </year>
    <description>
      A versatile laptop computer product.
    </description>
  </model>
  <plant>
    <address> Osaka, Japan</address>
  </plant>
</computer>
```

Fig. 9 A well-formed XML document

is the *Root*, an *Element*, *Text*, *Attribute*, *Processing Instruction* or *Comment*,⁵ whereas the label $order_i$ assumes as value a natural number representing the relative order of the node w.r.t. other children of its parent node, or \perp for root, text and attribute nodes. Moreover, the label $content_i$ can assume *PCDATA* or \perp (undefined) as value.

- The cardinality $|R_e|$ of the tuple of edge labels is fixed to 1, where the unique label represents the edge type. Each edge $e_j = \langle (n_h, n_k), R_j \rangle$, with n_h and n_k in N , has a label $R_j = \langle etype_j \rangle$, where the label $etype_j \in \{AttributeOf, SubElementOf\}$. Note that edges simply represent the “containment” relationship between different items of an XML document and do not have names.

For Plain XML the following high-level constraints hold: (i) in an XML document the root node has type label “*Root*”, (ii) the *content* label of element nodes is undefined (as shown in Fig. 6), and (iii) the *tag* label for text nodes is not explicitly specified.

3.3.2 XML schema

Although XML information can be treated as schema-less data, the notion of *XML Schema* has been introduced to represent sets of instances sharing the same structure. An XML Schema

⁵ Plain XML documents may also contain *ENTITY* nodes, not unlike macro calls that must be expanded before parsing. We do not consider *ENTITY* expansion in this paper.

is an XML document complying to a standard structure, itself expressed as a schema; for example, a schema's root node has always the label "schema" and may have a child of type *namespace* [34].

Our representation of XML schemata is twofold:

- An XML schema is a *low-level constraint* that identifies a set of instances.
- An XML schema is itself an XML document; as such, it is represented as in Sect. 3.3, and must satisfy a suitable set of low-level constraints.

An *XML Schema graph* is a *GSMM rooted graph* $\langle N, E, r \rangle$, obtained as an extension of a *Plain XML graph*. In particular:

- the cardinality $|L_n|$ of the tuples of node labels is 6. Each node n_i has as tuple of labels L_{n_i} the corresponding labels of the Plain XML representation plus the two labels uri_i , representing the resource identifier attached to that node, and $namespace_i$, representing the node namespace.
- the cardinality $|R_e|$ of the tuples of edge labels is 1, where the unique label represents the edge type as in Plain XML.

This approach is a simple and effective way to characterise XML schemata and all their instances a priori. Specifically, an XML graph representing a schema must satisfy, among others, the low-level constraints shown in Fig. 10.

3.4 Time

In this section we apply our high-level data model to the context of temporal applications (see, for example, TGM [28]) for representing semistructured data dynamics. In this case we use a time interval to represent when an object exists in the outside world or in the database.

A *semistructured temporal graph* is a *GSMM rooted graph* $\langle N, E, r \rangle$, where:

- the cardinality $|L_n|$ of the sets of node labels is 5, where each node $n_i = \langle oid_i, name_i, type_i, content_i, time_i \rangle$ has an object identifier $oid_i \in \mathcal{UID}$, the node name, the node type in $\{Complex, Simple\}$, a time interval $time_i \in V \cup \{\perp\}$, where V is a set of time intervals, and the node content.
- The cardinality $|R_e|$ of the sets of edge labels is fixed to 3, where each edge $e_j = \langle (n_h, n_k), R_j \rangle$, with n_h and n_k in N , has three labels $R_j = \langle ename_j, etype_j, Et_j \rangle$, where $etype_j \in \{Relational, Temporal\}$ is the type of the edge, and the last one $Et_j \in V$ is the time interval representing the valid time. Edges of the *Relational* type are used to represent classical relationships between two nodes; *Temporal* edges are used to store and represent the update of the content of a given node (it allows the representation of historical values).

Among others, instances of semistructured temporal graphs must satisfy the high-level constraints in Fig. 11c–f.

In Fig. 11b we show a portion of a *semistructured temporal graph* containing information about books and authors. Note that this labelled graph satisfies the high-level constraints described above. Once an instance of a semistructured temporal graph has been constructed, low-level constraints may be applied to enforce static or dynamic properties. For example, with the constraint of Fig. 11g on the instance of Fig. 11b we could enforce that the time interval of a Book starts after the time interval of its Author.

Note that, in general, the time interval of a relationship is not connected to the time interval of the related objects. If we represent *valid time* in the real world, the constraint above must

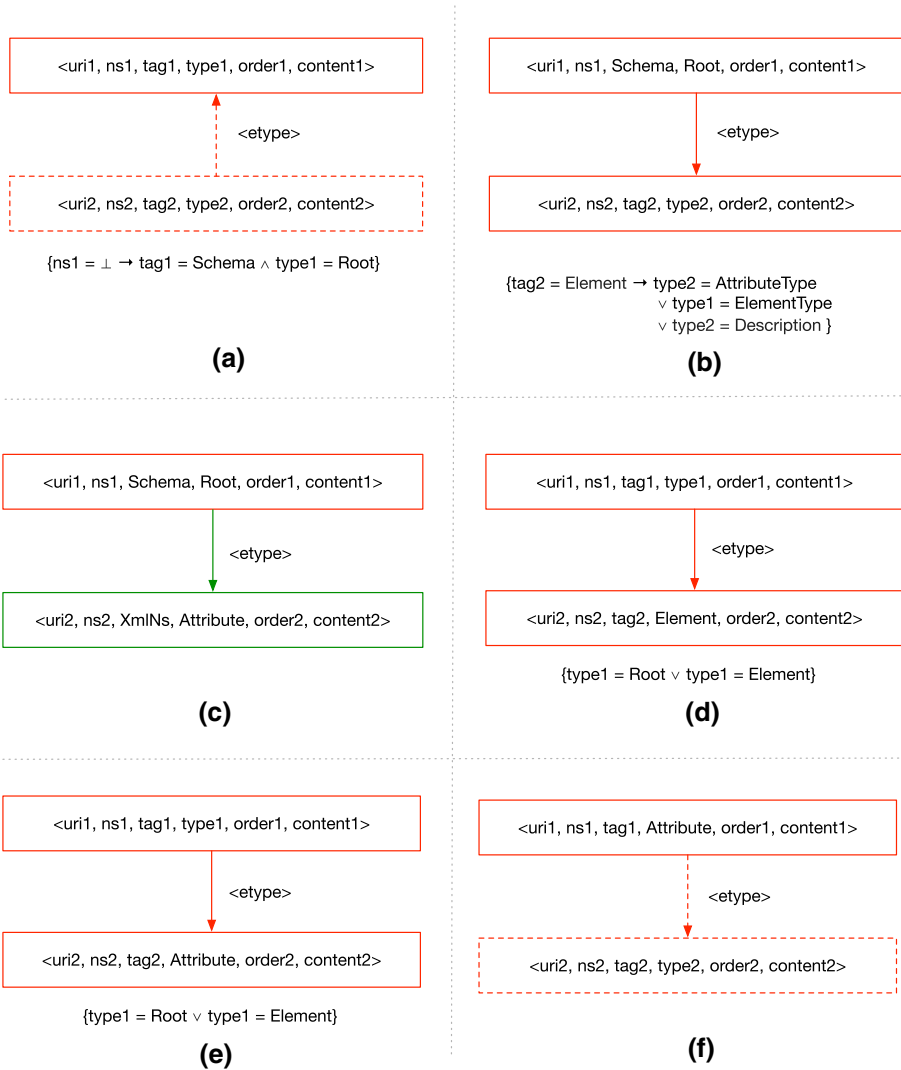


Fig. 10 **a** The root of the graph has as type *root* and as tag *Schema*; **b** each element node must be a root child and its label must be *ElementType*, *attribute Type*, or *Description*; **c** root must have an attribute with tag *XmlNs* as child; **d** each *Element* node is a child of either the root, or another *Element* node; **e** each *Attribute* node is a child of either the root, or an *Element* node; **f** each *attribute Type* node is a leaf

be added because it gives semantics to the “Writes” relation: a book can become valid only after its author was born!

3.5 The triplestore database

A Triplestore dataset allows the storage and retrieval of triples. A triple is a data entity in the form subject–predicate–object. Example of triples are “Peter is 40” or “The t-shirt is white”. Triples can be easily managed by using the Resource Description Framework (RDF). The

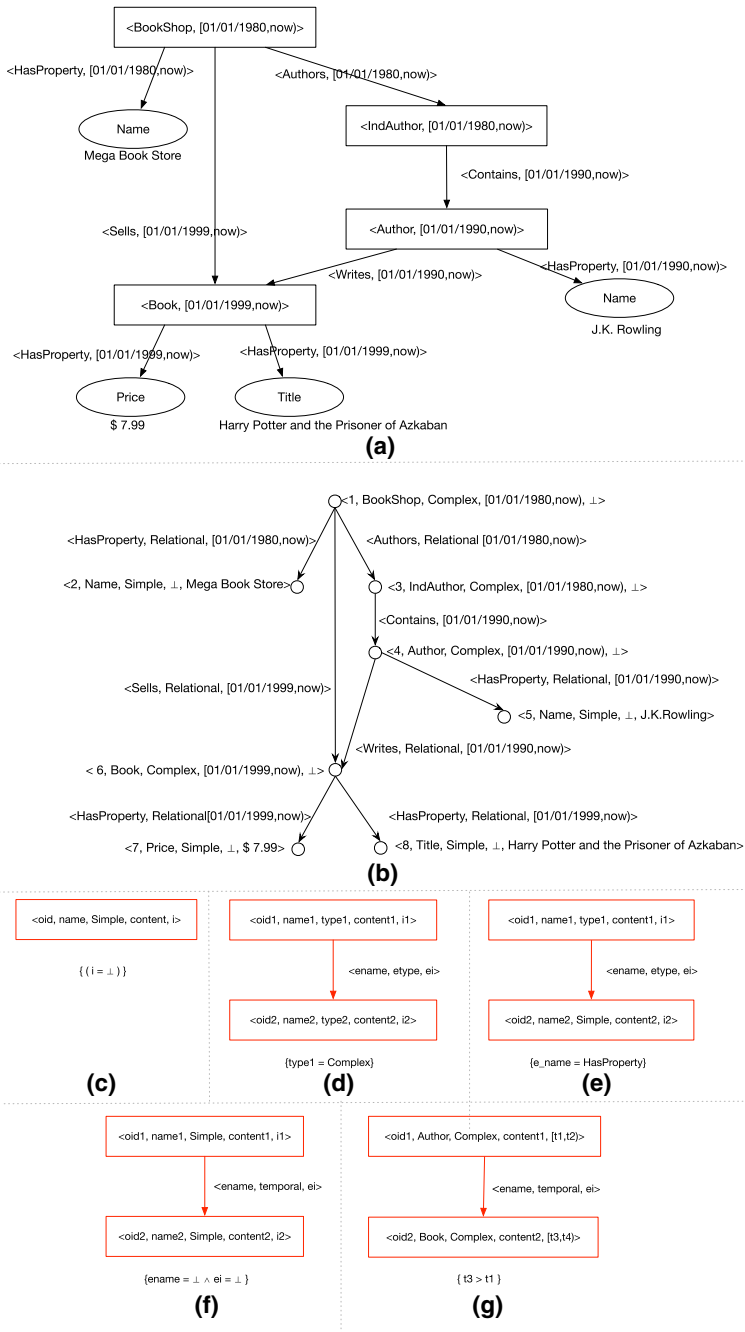


Fig. 11 **a** A simple TGM instance representing Bookshop temporal database; and **b** the corresponding GSMM graph. Some examples of high-level constraints: **c** simple nodes do not have a time interval; **d** simple nodes are leaves; **e** edges pointing to simple nodes are named “HasProperty”; **f** temporal edges do not have a name, neither they have a time interval. A low-level constraint on time: **g** the time interval of a Book starts after the time interval of its Author

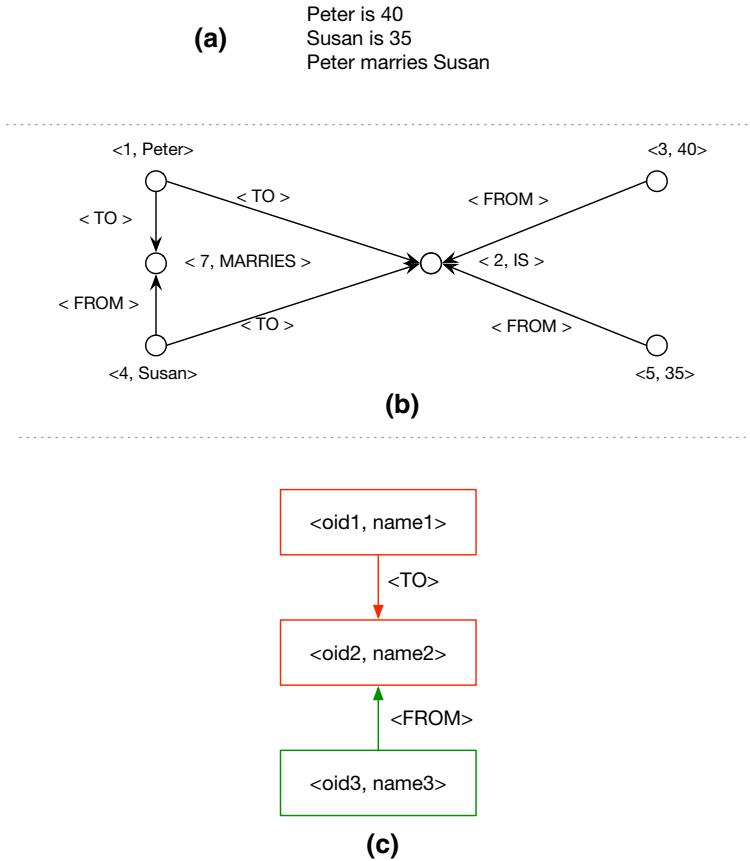


Fig. 12 **a** A triplestore database and **b** its GSMM graph. The high-level constraint: **c** each node representing a predicate, i.e., having an incoming edge labelled $\langle TO \rangle$, must have also an incoming edge labelled $\langle FROM \rangle$

RDF data model uses triples for expressing statements about resources (in particular web resources) and supports reification, i.e., the possibility to add properties (e.g. provenance properties) of a relation/predicate.

We can represent a triple-based database by using a *GSMM graph* $\langle N, E \rangle$, where:

- the cardinality $|L_n|$ of the sets of node labels is 2. Each node $n_i = \langle oid_i, name_i \rangle$ has an object identifier $oid_i \in UID$, which can be an URI for the RDF model, and a string representing the node label $name_i$. In case we need to represent the RDF Blank Node, we suppose to have \perp as value of $name_i$.
- the cardinality $|R_e|$ of the set of edge labels is 1, and each edge $e_j = \langle (n_h, n_k), R_j \rangle$ with n_h and n_k in N , has a label $R_j = \langle epredicate_j \rangle$, where $epredicate_j \in \{TO, FROM\}$.

A triplestore dataset is translated into GSMM by introducing nodes for subjects, predicates and objects (see Fig. 12b for an example) and must satisfy the high-level constraint in Fig. 12c each node having an incoming edge labelled $\langle TO \rangle$ (i.e., representing a predicate) must also have a not dangling incoming edge labelled $\langle FROM \rangle$. Predicates are modelled as nodes to support the RDF reification.

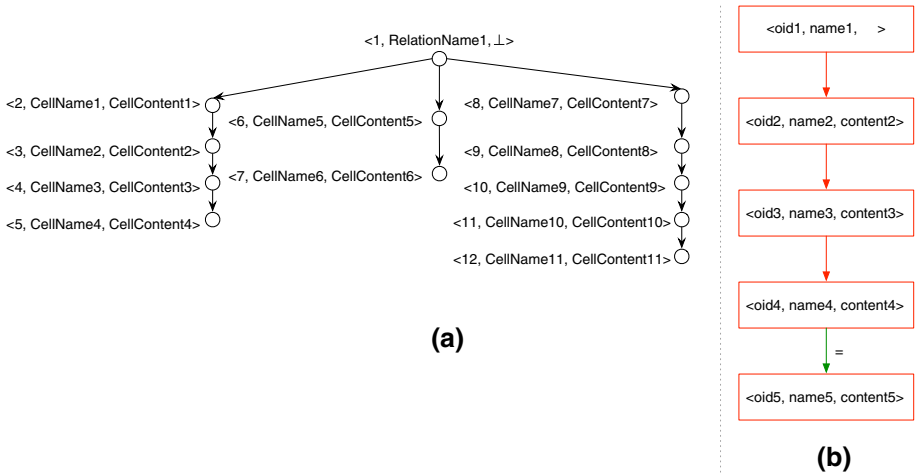


Fig. 13 **a** The GSSM graph of a table in the BigTable data model; **b** the maximum cells per row constraint for $n = 3$

3.6 BigTable

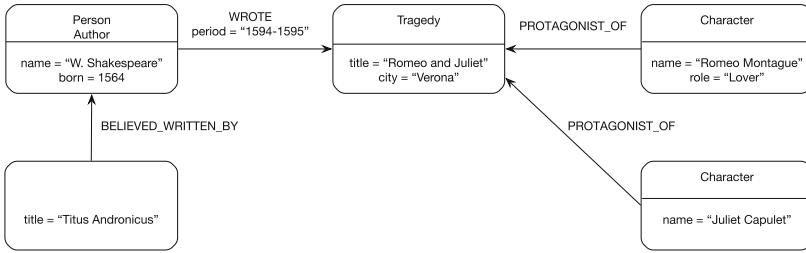
Under the BigTable data model, each table is a collection of rows composed of an arbitrary number of cells, and uniquely identified by a key. BigTable rows are often called *wide rows*, because the columns cells belong to are not pre-defined as in relational databases.

GSSM can represent a BigTable database by defining a graph node per cell. The property ID and the property value of each cell are stored in the value of the corresponding GSSM node. Each GSSM node gets its set of outgoing edges via the BigTable row containing the corresponding cell's adjacency list (often called *adjacency row*). We remark that according to this construction each outgoing edge is represented individually, expressing the fact that in BigTable each element of the adjacency list has its own cell in the adjacency row. The GSSM representation (see Fig. 13a) shows us that, compared to other concrete data models, BigTable supports efficient insertions and deletions. The maximum number n of cells allowed per row in a concrete BigTable model can be represented by specifying a GSSL concretisation constraint over the maximum degree of nodes in the GSSM graph that represents it. As an example, in Fig. 13b, we report the constraint specifying that the maximum cells per row must be $n = 3$.

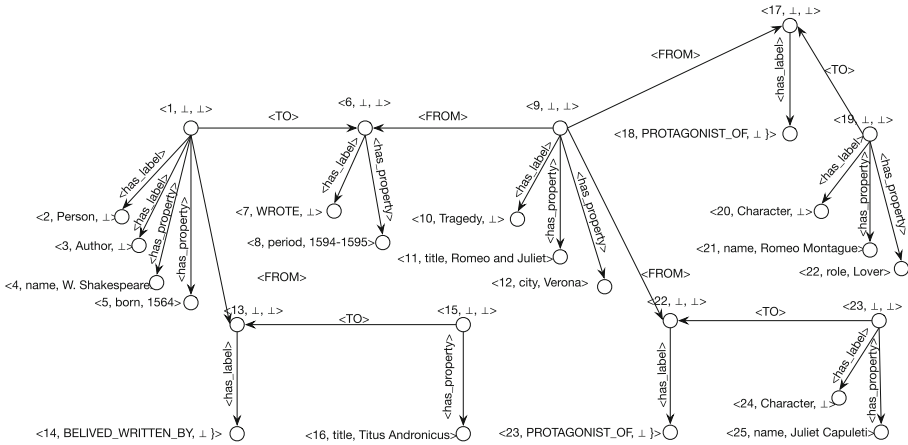
We can represent a triple-based data model by using a *GSSM rooted graph* (N, E, r) , where:

- the cardinality $|L_n|$ of the sets of node labels is 3. Each node $n_i = \langle oid_i, name_i, content_i \rangle$ has an object identifier $oid_i \in UID$, a string as $name_i$ and a content $content_i$ that could be \perp .
- the cardinality $|R_e|$ of the tuple of edge labels is 0.

If a given BigTable model backend supports key order, the outgoing edges will be ordered by the ID of their endpoint. Again, the GSSM representation allows one to assess the concrete model's runtime flexibility: ease of updating node IDs means that nodes which are frequently co-accessed can easily be assigned IDs with small absolute difference at run-time.



(a)



(b)

Fig. 14 a A Neo4j graph and b its corresponding GSMM graph

3.7 The Neo4j graph database

Neo4j is an open-source and graph-based database that stores data structured in graphs rather than in tables [23].

Graph-based data models allow the representation of connections and make available information by using navigation operations. This issue is becoming very important in the information management context, since nowadays there are no isolated pieces of information: as an example we can consider the Internet of Things (IoT) where every data source is connected and huge amounts of data correspond to even larger amounts of links expressing relations among them. For this reason, graph-based models are often used as a general description of Big Data.

In Neo4j data are stored in form of nodes and relationships (edges). Nodes and edges can have zero or multiple properties, each associated with a value. Moreover, a given node can also be labelled with multiple labels: each node label indicates a name of the node itself; each edge can be labelled with a type label (see Fig. 14a for an example). Differently from the other data models we have dealt with, Neo4j allows one to use multiple (and not predefined) labels, thus, in our translation into GSMM each label or property will be considered a node.

A Neo4j graph is a GSMM graph $\langle N, E \rangle$, where:

- the cardinality $|L_n|$ of the sets of node labels is 3. Each node $n_i = \langle oid_i, label/property_i, propertyvalue_i \rangle$ has an object identifier $oid_i \in UID$, the node name $label/property_i$ which assumes values in the string domain and represents the name (label) or the property of the node, and the $propertyvalue_i$, which indicates either the value of the property or can assume as value \perp in case the $label/property_i$ represents a property.
- the cardinality $|R_e|$ of the tuple of edge labels is 1. Each edge $e_j = \langle (n_h, n_k), R_{e_j} \rangle$ with n_h and n_k in N , has a label $R_{e_j} = \langle etype_j \rangle$ with $etype_j \in \{TO, FROM, has_property, has_label\}$.

Similarly to the high-level constraint specified for the Triplestore model (see Fig. 12c), also in the translation of Neo4j, each node having an incoming edge labelled $\langle TO \rangle$ (i.e., representing an edge in the original Neo4j graph) must also have a not dangling incoming edge labelled $\langle FROM \rangle$.

3.8 Parameters

Instantiation parameters shown in Table 1 are the *cardinality of node labels* (named $|L_n|$), the *cardinality of edge labels* (named $|R_e|$), the *domain of node labels* (named Node Label Domain), and the *domain of edge labels* (named Edge Label Domain), for the six concrete models described above.⁶ By inspecting Table 1, we can carry out fast “a priori” comparison of the models. For example, OEM only distinguishes two kinds of nodes, while OEM edges are labelled with a single label (actually, this label corresponds to the name of the pointed node, and edges represent the containment relation). The XML Infoset is quite similar to OEM, though it has a wider repertoire of node types and uses an enumeration type rather than a generic string for the edge label value. Intuitively, a model that uses enumeration values as edge labels can support *design rules*, saying when attribute rather than element containment should be used. Note that all the XML-based models represent order between node children, while OEM and TGM do not. Plain XML is the one model that does not provide object identifiers.

4 Inter-model translation

Our meta-model can be used for inter-model comparisons and translation as well. In particular, given two or more concrete models expressed by means of the GSMM formalism, we would like to be able to translate instances from one model to another one.

The translation task is mainly based on the following steps:

- for each node and edge label of the source model, try to find a corresponding label in the destination model. Whenever this basic translation is not possible, try to express each node or edge label of the source model, which does not have a corresponding label into the destination model, with a construct (e.g. a label, an additional node or edge) available in the destination model.
- to each label of the destination model that is not useful to express components of the instances of the source model assign an undefined value.

Next, we show how our technique can facilitate the inter-model translation process by considering TGM and plain XML as examples of concrete models.

⁶ For the sake of conciseness Table 1 does not explicitly consider Base Types, because they may be very large.

Table 1 Instantiation of meta-model parameters for some concrete models

Concrete model	$ L_n $	$ R_e $	Node label domain	Edge label domain
Relational	3	0	$OIDS \times Node\ Name\ Set$ $\times \{DName, Relation\ Name, Att,$ $Key\ Att, Set\ Of\ Att, Attr\ Value\}$	
OEM	3	1	$OIDS \times \{Root, object\} \times$ $(OEM\ Base\ Types \cup \{\perp\})$	$Node\ Name\ Set$
Plain XML	4	1	$TagSet \times \{Root, Attribute \dots\} \times$ $\mathbb{N} \cup \{\perp\} \times (PCDATA \cup \{\perp\})$	$\{Subelement\ Of, Attribute\ Of\}$
XML Schema	6	1	$URI \times NameSpaceSet \times TagSet \times$ $\{Root, Element, Text, Attribute \dots\} \times$ $\mathbb{N} \cup \{\perp\} \times (XMLBaseTypes \cup \{\perp\})$	$\{Subelement\ Of, Attribute\ Of\}$
TGM	5	3	$OIDS \times Node\ Name\ Set \times$ $\{Complex, Simple\} \times$ $(TGMBaseTypes \cup \{\perp\}) \times V$	$Edge\ Name\ Set \times \{Relational, Temporal\} \times$ V
Triplestore	2	1	$OIDS \times (Node\ NameSet \cup \{\perp\})$	$\{TO, FROM\}$
BigTable	3	0	$OIDS \times Node\ NameSet \times$ $(BigTableBaseTypes \cup \{\perp\})$	
Neo4j	3	1	$OIDS \times Node\ NameSet \times$ $(Neo4j\ BaseTypes \cup \{\perp\})$	$\{TO, FROM, has_property, has_label\}$

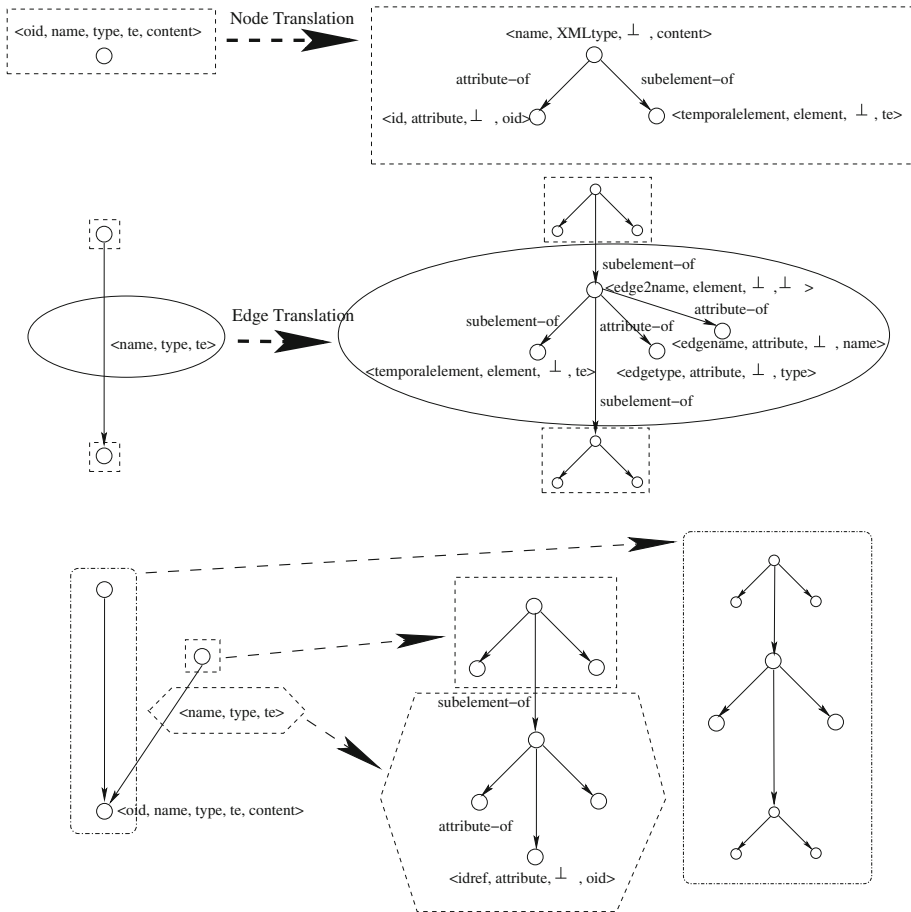


Fig. 15 Translating TGM into plain XML

4.1 Translating TGM into XML

We start from a TGM instance \mathcal{G} translated into the corresponding GSMM graph $G = \langle N, E, r \rangle$.

In order to obtain another GSMM graph $G' = \langle N', E', r' \rangle$ related to an XML document, which represents the information originally contained in \mathcal{G} , we have to apply the primitives represented in Fig. 15:

1. for each $n \in N$, with $n = \langle oid, name, type, te, content \rangle$, transform it in a subtree of G' composed by three nodes n_1, n_2 , and n_3 .

The first node n_1 has as tag label $name$, as type label $root$ if $n = r$, $element$ otherwise, an undefined order label, and $content$ as content label (note that $content$ is a defined value only for simple nodes of the TGM graph).

The two children of n_1 , named n_2 and n_3 have the labels $\langle id, attribute, \perp, oid \rangle$ and $\langle temporalelement, element, \perp, te \rangle$, respectively.

Note that the edge connecting n_1 to n_2 is labelled $attribute - of$, whereas the edge from

n_1 to n_3 is labelled *subelement – of*. Moreover, we do not use the order label of Plain XML because TGM does not consider an order relation between the children of a given node.

- For each edge $e = \langle (m_1, m_2), R_e \rangle \in E$, with $R_e = \langle name, type, te \rangle$, we transform it into a subtree of four nodes n_1, n_2, n_3 , and n_4 .

The node n_1 is an element with tag *edge2m2*⁷ (and undefined order and content labels) which has three children: n_2 is an element which has as tag *temporalelement* and as content te , n_3 is an attribute with tag *edgetype* and value $type$, n_4 is an attribute with tag *edgename* and value $name$.

- TGM instances are modelled as DAGs, whereas XML documents can be represented in a graphical way by means of trees.

The third primitive of Fig. 15 is introduced in order to solve this distinction between the two concrete models we are considering.

If the original graph G contains a node n with more than one incoming edge, we translate one path to n as explained in the previous two steps, and we consider all the other edges to n as elements with an *idref* attribute whose value is the object identifier of the pointed node.

4.1.1 An algorithm for translating TGM into XML

The above considerations allow us to formalise a depth-first algorithm for translating a TGM graph represented with the GSMM formalism into plain XML code.

TGM2XML(set of nodes N, set of edges E, node r)

```
{
  for all nodes n in N
    paint n white
  TGM2XMLCODE(N, E, r)
}
```

TGM2XMLCODE(set of nodes N, set of edges E, node n)

```
{
  paint n grey
  if (n = <oid,nodename,type,contentvalue,t> is a complex node)
  {
    write:``<nodename id=``oid``>
      <temporalelement> t </temporalelement>``
    for all outgoing edges e_i=((n,x_i),<Ename_i,relational,Et_i>)
      pointing to a white node x_i
    {
      write:``<edge2x_i edgename=``Ename_i`` type=``relational``>
        <temporalelement> Et_i </temporalelement>``
      TGM2XMLCODE(N, E, x_i)
      write:``</edge2x_i>``
    }
    for all outgoing edges e_i=((n,x_i),<Ename_i,relational,Et_i>)
      pointing to a black node x_i
    {
      write:``<edge2x_i edgename=``Ename_i``
        type=``relational`` idref=``objx_i``>

```

⁷ An edge pointing to m_2 .

```

        <temporalelement> Et_i </temporalelement>''
    write:''</edge2x_i>''
  }
  paint n black
  write:''</nodename>''
}
else
{
  write:''<nodename id=''oid''> contentvalue ''
  if (n has a temporal outgoing edge to x_j)
    write:''<edge2x_j edgename=''Temporal'' type=''temporal''>
      </edge2x_j>''
  write:''</nodename>''
  paint n black
}
}
}

```

Consider the TGM instance reported in Fig. 11a and its translation into GSMM of Fig. 11b. The XML code produced by TGM2XML is the following:

```

<BookShop id = 1>
  <TimeInterval> [01/01/1980,now) </TimeInterval>
  <Edge2Name edgename = HasProperty type = relational>
    <TimeInterval> [01/01/1980,now) </TimeInterval>
    <Name id = 2> Mega Book Store </Name>
  </Edge2Name>
  <Edge2IndAuthors edgename = Authors type = relational>
    <TimeInterval> [01/01/1980,now) </TimeInterval>
  <IndAuthors id = 3>
    <TimeInterval> [01/01/1980,now) </TimeInterval>
  <Edge2Author edgename = Contains type = relational>
    <TimeInterval> [01/01/1990,now) </TimeInterval>
  <Author id = 4>
    <TimeInterval> [01/01/1990,now) </TimeInterval>
    <Edge2Name edgename = HasProperty type = relational>
      <TimeInterval> [01/01/1990,now) </TimeInterval>
      <Name id = 5> J.K. Rowling </Name>
    </Edge2Name>
  <Edge2Book edgename = Writes type = relational>
    <TimeInterval> [01/01/1999,now) </TimeInterval>
  <Book id = 6>
    <Edge2Price edgename = HasProperty type = relational>
      <TimeInterval> [01/01/1999,now) </TimeInterval>
      <Price id = 7> 7.99 </Price>
    </Edge2Price>
    <Edge2Title edgename = HasProperty type = relational>
      <TimeInterval> [01/01/1999,now) </TimeInterval>
      <Title id = 8>
        Harry Potter and the Prisoner of Azkaban
      </Title>
    </Edge2Price>
  </Book>
</Edge2Book>
</Author>

```

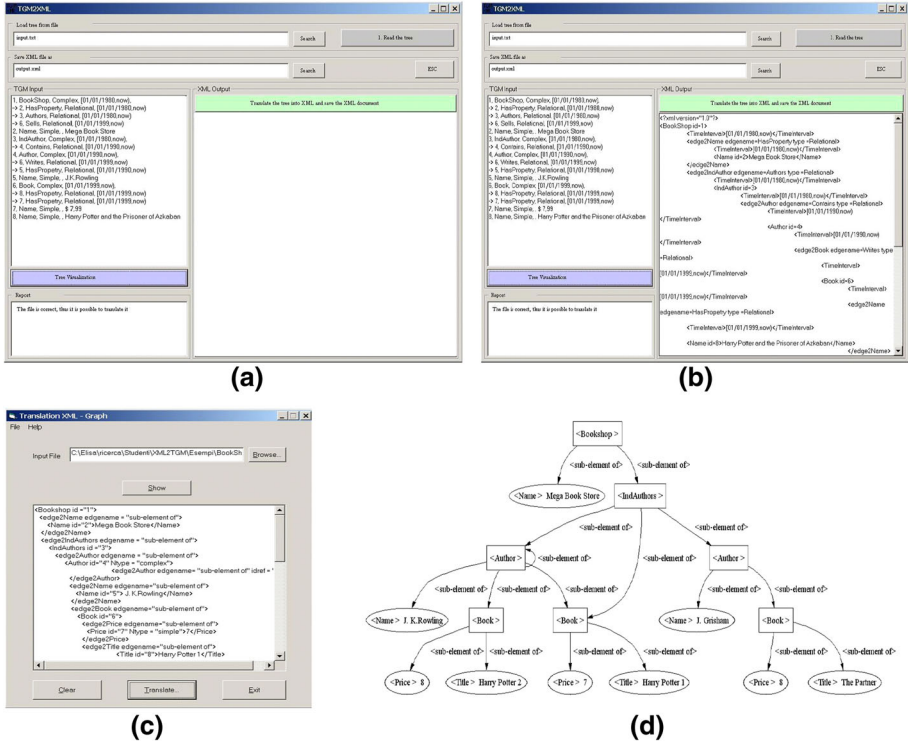



Fig. 16 **a** A tool for translating TGM graphs into XML document; **b** the XML translation of a TGM graph; **c** a tool for translating an XML document into a TGM graph; **d** the visualisation of the resulting TGM graph

```

</Edge2Author>
</IndAuthors>
</Edge2IndAuthors>
<Edge2Book edgename = Sells type = relational idref = 6>
  <TimeInterval> [01/01/1999,now) </TimeInterval>
</Edge2Book>
</BookShop>

```

4.2 A software translator

Note that the element Edge2Book has an idref attribute, because the node labelled Book in the graph of Fig. 11b has two ingoing edges.

We have developed a software tool for translating TGM graphs into XML documents and viceversa. Figure 16a, b shows how the textual representation of a TGM graph reporting information about Books is translated into an XML document. In the bottom part of Fig. 16, we show the reverse step: the XML document about Books which is produced by the previous translation (Fig. 16c) is coded into the TGM graph depicted in Fig. 16d.

5 Related work

Our meta-model's main goal is the uniform representation of flexible data models; it can be applied to inter-model comparison and translation, aimed at mediation between heterogeneous data sources. An early approach to this problem was a unified framework for the management and the exchange of semistructured data [4], described according to a variety of formats and models. In particular they consider various schema definition languages for XML, OEM and a model to store Web data, and show that the primitives adopted by all of them can be classified into a rather limited set of basic types. On these basic types, they define a notion of "meta-formalism" that can be used to describe, in a uniform way, heterogeneous representations of data, and give the definition of an effective method for the translation between heterogeneous data representations. The main difference between this early proposal and our meta-model is related to schemata: Atzeni et al. [4] assumes that a schema is available for each data source, whereas we do not require to have the schema in advance, but rather consider schemata as constraints that can be specified if needed. Another early effort is the hypergraph data model (HDM) [27], a simple low-level modelling language based on a hypergraph data structure together with a set of associated constraints. Here the constraint specification language is not formalised; the authors define a small set of transformations as schemata expressed in HDM, which are used for inter-model transformation. Again, the main difference between this proposal and our meta-model is related to schemata translation. Our work focuses on flexible models which may be schemaless thus, we do not target schemata translation, but provide a very general graph-based formalism allowing the representation of different aspects of data. Our inter-model translation, as shown by example in Sect. 4, relies on the generality of nodes and edge labels, which can be specialised to the labels allowed by the source and destination models. Moreover, while the meta-model by McBrien et al. [27] provides some basic primitives whereof the basic constructs of models can be *built*, ours provides a high level, general formalism whose *specialisations* are the models themselves. Other early proposals ([8,9], and [25]) dealing with inter-model translation do not propose a unifying meta-model but focus on the possibility to translate information from a model to another. Bernstein et al. provide a generic framework that can be used to merge models in different contexts [8]. Bowers et al. proposed an approach to represent, in a uniform way, information coming from different models [9]. They use RDF and provide a mapping formalism for inter-model translation. With the advent of NoSQL data models and systems, some researchers pointed out the need of analysing these new systems for a data modelling point of view [22], while others noticed that small differences in data modelling features may have a huge impact on performance of NoSQL systems [2] However, we are not aware of any attempt to provide a comprehensive meta-model and a comparison framework like our own. Rather, recent research has focused on one-on-one data model comparison in vertical domains. Lee et al. identify a set of informal expressive power criteria that lead to choosing the XML Infoset over the relational model as a concrete data representation for patient data [24]. Unfortunately, lack of formalisation of their criteria prevents them for providing a rigorous assessment methodology.

When choosing a NoSQL data model, the computer scientist is faced with the contrasting requirements of dealing with data whose structure is not easily captured by traditional approaches, and allowing for fast revisions of representational choices at run-time. The use of GSMM and GSL marks a step forward towards a rigorous treatment of the fundamental issues of flexible data modelling and querying. For example, GSMM provides a way for *a posteriori schema derivation*: intuitively, a schema *represents* an instance if it contains

its skeleton while disregarding multiplicities and values. Given a database instance, we can obtain a schema by drawing a constraint that defines the structure of the document [18]. In particular the constraint specifies, by means of first-order formulae, all the possible paths starting from the root node, and sets also the admitted labels of nodes and edges. We claim that such schemata may play an important role in assessing similarity and differences between individual data representations.

6 Conclusions and future work

We have presented a graph-based meta-model (GSMM) and a language (GSL) aimed at bringing flexible data model properties into a unified framework. In our future work, we will define flexibility metrics on concrete data models based on their GSMM representations. We expect to be able to establish benchmarks supporting a priori assessment of data models, guiding adoption decisions. Another line of investigation concerns the use of GSL as a language for those models where no language is defined. As an example, consider the Unified Modelling Language (UML), where schemata (models, in UML notation) may be specified by means of different notations, at different levels of detail (e.g. class diagrams). A specification based on GSMM could allow the expression of constraints to be associated to a UML class diagram or to any UML diagram. Moreover, constraints expressed in GSL may easily be transformed from more general to more specific representations of the same information by means of different UML notations.

References

1. Abiteboul S (1997) Querying semi-structured data. In: Proceedings of the international conference on database theory, vol 1186. Lecture notes in computer science, pp 262–275
2. Angles R (2012) A comparison of current graph database models. In: Proceedings of the 2012 IEEE 28th international conference on data engineering workshops, ICDEW '12. IEEE Computer Society, Washington, DC, pp 171–177
3. Atzeni P, Cappellari P, Torlone R, Bernstein PA, Gianforme G (2008) Model-independent schema translation. *VLDB J* 17(6):1347–1370
4. Atzeni P, Torlone R (2001) A unified framework for data translation over the web. In: Proceedings of the 2nd international conference on web information system engineering. IEEE Computer Society, pp 350–358
5. Bekiropoulos K, Keramopoulos E, Beza O, Mouratidis P (2010) A list of features that a graphical xml query language should support. *Comput Syst Sci Eng* 25(5):13–21
6. Benda S, Klímek J, Nečáský M (2013) Using schematron as schema language in conceptual modeling for xml. In: Proceedings of the ninth Asia-Pacific conference on conceptual modelling, vol 143, APCCM '13. Australian Computer Society, Inc., Darlinghurst, pp 31–40
7. Bernstein PA, Halevy AY, Pottinger RA (2000) A vision for management of complex models. *SIGMOD Rec* 29(4):55–63
8. Bernstein PA, Pottinger R (2003) Merging models based on given correspondences. Technical report UW-CSE-03-02-03. University of Washington
9. Bowers S, Delcambre L (2000) Representing and transforming model-based information. In: Proceedings of International workshop on the semantic web at the 4th European conference on research and advanced technology for digital libraries (SemWeb)
10. Bunemann P, Fan W, Siméon J, Weinstein S (2001) Constraints for semistructured data and XML. *SIGMOD Rec* 30:47–54
11. Bunemann P, Fan W, Weinstein S (1998) Path constraints on semistructured and structured data. In: Proceedings of 17th symposium on principles of database system. ACM Press, pp 129–138
12. Cattell R (2011) Scalable SQL and NoSQL data stores. *SIGMOD Rec* 39(4):12–27

13. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. *ACM Trans Comput Syst* 26(2):4:1–4:26
14. Chawathe SS, Abiteboul S, Widom J (1998) Representing and querying changes in semistructured data. In: *Proceedings of the fourteenth international conference on data engineering*. IEEE Computer Society, pp 4–13
15. Chawathe SS, Abiteboul S, Widom J (1999) Managing historical semistructured data. *Theory Pract Object Syst* 5(3):143–162
16. Chen L, Oughtred R, Berman HM, Westbrook J (2004) Targetdb: a target registration database for structural genomics projects. *Bioinform Appl Notes* 20(16):2860–2862
17. Combi C, Oliboni B, Quintarelli E (2012) Modeling temporal dimensions of semistructured data. *J Intell Inf Syst* 38(3):601–644
18. Cortesi A, Dovier A, Quintarelli E, Tanca L (2002) Operational and abstract semantics of a query language for semi-structured information. *Theor Comput Sci* 275(1–2):521–560
19. Damiani E, Oliboni B, Quintarelli E, Tanca L (2003) Modeling semistructured data by using graph-based constraints. In: *OTM workshops proceedings*. Lecture notes in computer science. Springer, Berlin, pp 20–21
20. Damiani E, Tanca L (1997) Semantic approaches to structuring and querying web sites. In: *Proceedings of 7th IFIP working conference on database semantics (DS-97)*
21. Fan W, Lu P (2017) Dependencies for graphs. In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems, PODS '17*. ACM, pp 403–416
22. Indrawan-Santiago M (2012) Database research: Are we at a crossroad? reflection on NoSQL. In: *Proceedings of the 2012 15th international conference on network-based information systems, NBIS '12*. IEEE Computer Society, Washington, DC, pp 45–51
23. Kaur K, Rani R (2013) Modeling and querying data in NoSQL databases. In: *Proceedings of the IEEE international conference on Big Data*, pp 1 – 7
24. Lee KK-Y, Tang W-C, Choi K-S (2013) Alternatives to relational database: comparison of NoSQL and XML approaches for clinical data storage. *Comput Methods Progr Biomed* 110(1):99–109
25. Levy AY, Rajaraman A, Ordille JJ (1996) Querying heterogeneous information sources using source descriptions. In: *Proceedings of the twenty-second international conference on very large databases*. VLDB Endowment, Saratoga, Calif., Bombay, India, pp 251–262
26. Makoto M, Lee D, Mani M, Kawaguchi K (2005) Taxonomy of XML schema languages using formal language theory. *ACM Trans Internet Technol* 5(4):660–704
27. McBrien P, Poulouvasilis A (1999) A uniform approach to inter-model transformations. In: *Conference on advanced information systems engineering*, pp 333–348
28. Oliboni B, Quintarelli E, Tanca L (2001) Temporal aspects of semistructured data. In: *Proceedings of the eighth international symposium on temporal representation and reasoning (TIME-01)*. IEEE Computer Society, pp 119–127
29. Papakonstantinou Y, Garcia-Molina H, Widom J (1995) Object exchange across heterogeneous information sources. In: *Proceedings of the eleventh international conference on data engineering*. IEEE Computer Society, pp 251–260
30. Paredaens J, Peelman P, Tanca L (1995) G-Log: a declarative graphical query language. *IEEE Trans Knowl Data Eng* 7(3):436–453
31. Vicknair C, Macias M, Zhao Z, Nan X, Chen Y, Wilkins D (2010) A comparison of a graph database and a relational database: a data provenance perspective. In: *Proceedings of the 48th annual southeast regional conference, ACM SE '10*. ACM, New York, NY, pp 42:1–42:6
32. Virgilio RD, Maccioni A, Torlone R (2014) Graph-driven exploration of relational databases for efficient keyword search. In: *Candan KS, Amer-Yahia S, Schweikardt N, Christophides V, Leroy V (eds) Proceedings of the workshops of the EDBT/ICDT 2014 joint conference (EDBT/ICDT 2014)*, Athens, Greece, March 28, 2014, Vol. 1133 of *CEUR workshop proceedings*, CEUR-WS.org, pp 208–215
33. W3C (1998) World wide web consortium. Extensible Markup Language (XML) 1.0. <http://www.w3c.org/TR/REC-xml/>
34. W3C (2001) World wide web consortium. XML schema. <http://www.w3c.org/TR/xmlschema-1/>
35. Zang T, Calinescu R, Kwiatkowska MZ (2011) Metamodel-driven SOA for collaborative e-science application. *Comput Syst Sci Eng* 26(3):215–226



Ernesto Damiani is a Professor of computer science at the University of Milan, where he leads the SEcure Service-oriented Architectures Research (SESAR) Lab. He is the Founding Director of the Center on Cyber-Physical Systems at Khalifa University, in the UAE. He received an honorary doctorate from Institut National des Sciences Appliquées de Lyon, France (2017), for his contributions to research and teaching on Big Data analytics. He is the Principal Investigator of the H2020 TOREADOR project on Big data as a service. His research spans Cyber-security, Big Data and cloud/edge.



Barbara Oliboni is assistant professor at the Department of Computer Science of the University of Verona. She received the Ph.D. degree in Computer Engineering by the Politecnico of Milan. Her main research interests are related to the database field, with an emphasis on semistructured data, temporal information, business processes management, and clinical information management. She is member of the AIME (Artificial Intelligence in MEDicine) board. She is part of the Program Committee of International Conferences, and reviewer for International Journals.



Elisa Quintarelli obtained her Ph.D. in Computer and Automation Engineering at Politecnico di Milano where is now an Associate Professor. She is the author of about 80 papers and her research interests are in databases, with a focus on semistructured data, intentional query answering, data mining and context-based personalisation. She is part of the Program Committee of International Conferences, and reviewer for International Journals.



Letizia Tanca received a PhD in Applied Mathematics and Computer Science in 1988 and now is a full professor at Politecnico di Milano. She is the author of about 150 papers on databases and database theory, deductive and active databases, graph-based languages, semantic-web information management, and more recently on context-aware knowledge management and Big Data analytics.