BIIIG : Enabling Business Intelligence with Integrated Instance Graphs

André Petermann [#*1], Martin Junghanns [#1], Robert Müller [*2], Erhard Rahm [#1]

[#]*Department of Computer Science, University of Leipzig*
Augustusplatz 10, 04109 Leipzig, Germany
[1]{petermann,junghanns,rahm}@informatik.uni-leipzig.de

[*]*Faculty of Media, University of Applied Sciences Leipzig*
Karl-Liebknecht-Str. 145, 04277 Leipzig, Germany
[2]mueller@fbm.htwk-leipzig.de

***Abstract*— We propose a new graph-based framework for business intelligence called BIIIG supporting the flexible evaluation of relationships between data instances. It builds on the broad availability of interconnected objects in existing business information systems. Our approach extracts such interconnected data from multiple sources and integrates them into an integrated instance graph. To support specific analytic goals, we extract subgraphs from this integrated instance graph representing executed business activities with all their data traces and involved master data. We provide an overview of the BIIIG approach and describe its main steps. We also present initial results from an evaluation with real ERP data.**

I. Introduction

Traditional business intelligence based on data warehouses is limited in its flexibility by the underlying data warehouse schema, e.g. star, snowflake or galaxy schema. Only predefined kinds of facts, dimensional objects and their relationships captured in the schema can be evaluated by the data analyst. While this is sufficient in many cases, it does not support a more elaborate analysis and mining of relationships between data objects to better understand the causes of certain results, e.g., in which way certain employees contribute to profit. Such relationships are frequently recorded in the underlying information systems but are not transferred to the data warehouse due to the typical focus on simple dimensional relationships. We therefore propose a new graph-based approach to business analysis supporting a more comprehensive analysis of relationships in addition to standard analysis techniques. For this purpose, we extract instances and their relationships from the relevant data sources and integrate them within a comprehensive instance graph that preserves existing relationships.

Enterprises typically use different heterogeneous but interrelated information systems for their daily business such as ERP (enterprise resource planning) or CIT (customer issue tracking) systems. The data objects within these systems represent either transactional or master data. *Transactional data* refer to business activities such as quotations, invoices, emails or calendar entries while *master data* (non-transactional data) refer to more static reference data such as customers, products or employees. Transactional data instances are typically related to other transactional data instances as well as to relevant master objects. We will capture such relationships referring to the same business activity within so-called *business transaction graphs* (BTG) in order to allow a fine-grained analysis of business activities. There are also frequent references between information systems, e.g. to link an activity to a remotely controlled transactional object or master object. For example, tickets in the CIT system may be related to sales orders in the ERP system or orders in the ERP system may refer to customer master data in other systems. Such links between systems are typically implemented by cross-system identifiers for customers, sales order etc. and can be maintained by dedicated middleware like enterprise service bus [16] software. We will utilize such existing links for integrating instances from different systems.

A. Motivating Example

For illustration, we consider a trade company with business activities supported by an ERP system and a separate CIT system, both keeping transactional and master data. The ERP system is used to process purchase orders and sales of products while the CIT system manages customer complaints. A non-trivial analysis task is to evaluate employees considering not only the frequency but also the way they contribute to the profit of the company. Our intended solution is to aggregate BTGs to determine profit and analyze occuring patterns of employee involvement.

Figure 4 shows a simplified sample for a sales process containing transactional (white nodes) and involved `Employee` objects (gray nodes) from both, the ERP and CIT system. Master data about `Employee` Alice is available in both sources and thus shown as a cross-system node. The calculation of profit has to consider all revenue-related (`SalesInvoice`; revenue 5,000) and expense-related objects (`PurchaseInvoice` and `Ticket`; total expense 4,000) resulting in a profit of 1,000. Information about employee involvement is available from relationship paths between transactional and `Employee` objects. In particular the combination of master data (e.g. employee name) and transactional metadata (relationship types, transactional classes) provides meaningful patterns about *who* was *how* involved in *which kind of business activity*. For

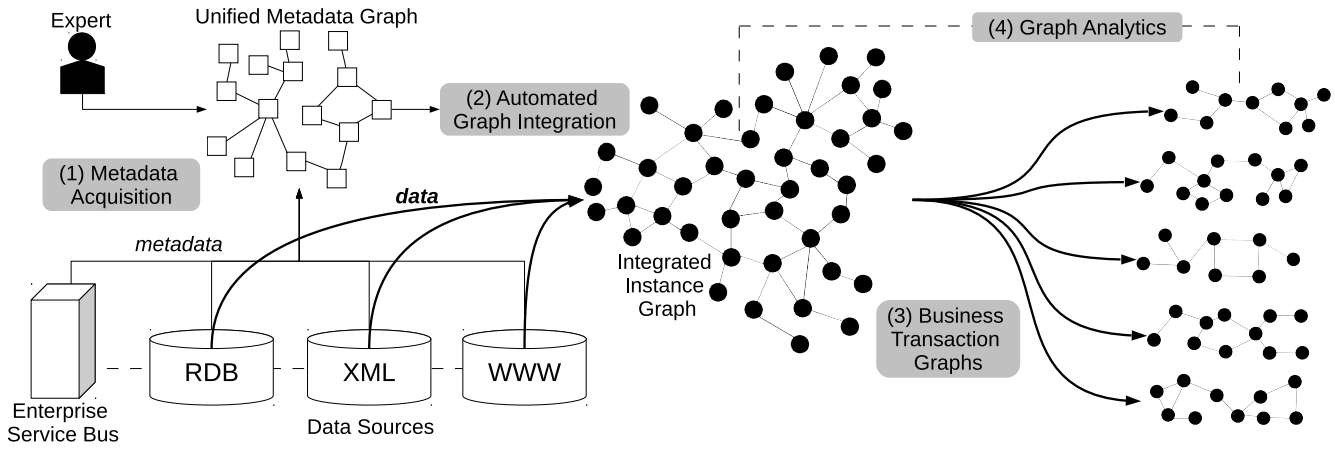

Fig. 1: BIIIG Overview

comprehensive analytics we have to aggregate profit-related measures and consider the involvement of all employees across relationship paths in not only one but all relevant BTGs.

B. Framework Overview

The motivating example illustrates only one possible kind of analytical questions answerable by the evaluation of business transaction graphs. Generally, the identification of frequent relationship patterns in BTGs of interest is promising high analytical value. Relationship-driven evaluations are not only valuable for business applications but also for other domains and different kinds of information systems. For example, staff-related treatment success in hospitals could be evaluated with data from clinical information systems. This can be supported by connecting all clinical records associated to the treatment of a certain patient within a BTG and the analysis of relationship patterns across all such BTGs.

To support this kind of evaluations we propose a new graph-based framework for business analytics called BIIIG (**B**usiness **I**ntelligence with **I**ntegrated **I**nstance **G**raphs). BIIIG uses a native graph database (currently Neo4j [3]) to efficiently manage even large graphs and to utilize the advantages of graph databases for the integration of heterogeneous data and the flexible evaluation of relationships. While data integration happens mainly on the instance level, BIIIG also captures the metadata from the data sources to uniformly and semantically describe instances and relationships. As shown in Fig. 1, the BIIIG approach entails four main steps:

(1) Metadata Acquisition and Unification: We first extract the schema for every data source and translate it into a generic graph format. The schema will be used not only to semantically describe the data but also enables the automated extraction of instance objects and relationships in step 2.

(2) Automated Graph Integration: In our main data store, called *integrated instance graph* (IIG), each data object (transactional or master data) is a node and each relationship is an edge. It can be very large and is stored in a graph database. The integration process is fully automated based on the unified metadata and requires no source-specific ETL design.

(3) Business Transaction Graphs: To support specific analytical tasks, we extract *business transaction graphs* (BTG) from the IIG. A BTG contains interrelated transactional data as well as involved master data. BTGs typically represent business activities and are often the granularity of interest.

(4) Graph Analytics: Both the IIG and the set of BTGs can be the basis for graph-based business analytics. With an appropriate user interface, analysts can intuitively navigate in the graphs to access any piece of recorded data with its relationships. In addition, declarative query languages enable new kinds of OLAP queries involving relationship pattern. Furthermore, graph mining techniques can be applied to both the IIG or the set of BTGs, e.g., to determine frequent patterns or to predict the outcome of future business activities.

The main contribution of this paper is outlining the BIIIG framework for graph-based business intelligence with its dedicated support of business transaction graphs. The approach has already been implemented in an initial version and could be applied to a real use case. In the next section, we briefly discuss related work about graph databases and graph analytics. After the introduction of our metadata and instance models (section III), we present the main steps of the framework in sections IV, V and VI. We then show the results of a first use case with data from a real ERP system (section VII). Finally, we summarize and conclude with an outlook to future work.

II. Related Work

Graph databases [21] and graph data models [5] have found increasing interest and adoption in recent years. Even business intelligence platform providers aim at providing graph data analysis, e.g., for SAP HANA [19] or Teradata [4]. However, these efforts are still under development so that the final capabilities are yet unknown. The property graph model [18] is applied in several systems and will also be used in BIIIG. It is simple but flexible due to a uniform representation of objects and relationships with different properties. State-of-the art graph query languages such as Neo4j Cypher [1] provide already useful analytical features for graph pattern matching and aggregation.

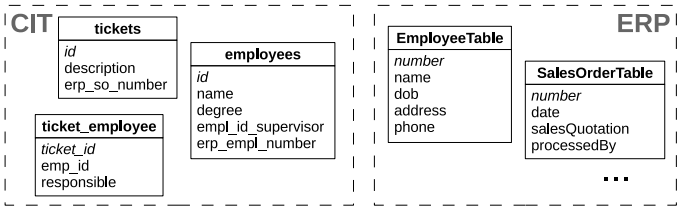

Fig. 2: Sample database tables of two interlinked systems

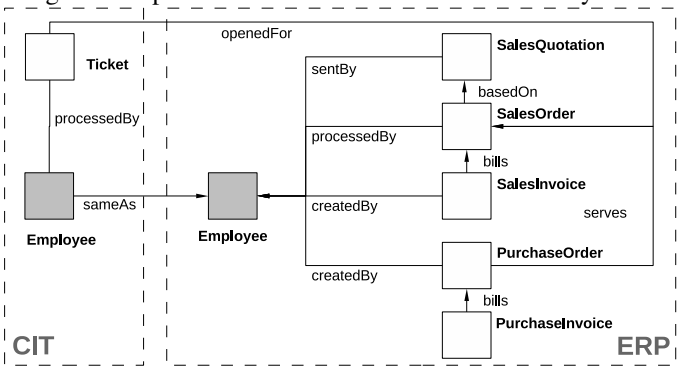

Fig. 3: Sample Unified Metadata Graph (simplified); master data classes are gray and transactional classes are white

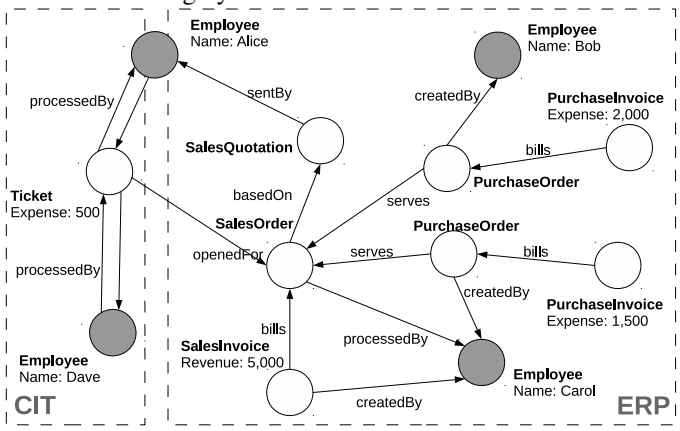

Fig. 4: Sample Business Transaction Graph with node classes, edge types and selected properties; master data is shown as gray and transactional data as white nodes

Graph databases are also popular in the semantic web community where RDF graphs [12] are stored in dedicated triple stores [10] and analyzed with the standardized query language SPARQL. However, RDF and SPARQL have not been widely used for business intelligence so far. RDF triples lead to an extremely fine-grained and voluminous data representation. Furthermore, SPARQL has limitations for graph analysis compared to languages such as Cypher, e.g. to find and return variable paths and graph patterns of interest [22]. While BIIIG aims at a general and comprehensive end-to-end solution for graph-based data integration and business intelligence, previous related studies mostly focussed on specific aspects. Several graph-based frameworks focus on the transformation of source data into problem-specific [14][7] or more general [15] graph-based data warehouse models but do not aim at preserving all relationships from the source systems that might become of interest in unexpected ways. A notable exception is DB2SNA [23] that extracts social networks from relational databases and analyzes them, albeit it is not concerned with business analytics. The automated extraction of interrelated data objects from ERP systems is discussed in [17], but without using a graph model and for the single analytical goal of process mining.

The majority of analytical graph algorithms, e.g. for pattern-mining [9], are based on graph models with homogenous data objects and relationships while we aim at analyzing graphs with heterogeneous data. Still, there are some approaches to find similar patterns in more complex graph models. PathSim [24] is a similarity measure for graph patterns in heterogeneous metadata. In [13] graph summarization is used for the visualization of similar patterns in multiple labelled graphs and [20] discusses the aggregation of property graphs by rule-based summarization. Further on, there are different approaches of graph-based OLAP frameworks. GraphCube [26] enables the graph-based aggregation of graphs with nodes providing a set of predefined dimensions. Graph OLAP [8] aims at aggregating multiple snapshots of homogenous networks and HMGraph OLAP [25] extends this approach by the support of heterogenous nodes and edges. While these approaches show the opportunities for OLAP on graph data, they either do not support heterogenous data [8], [26] or rely on a static data warehouse schema [25].

III. Data Model

BIIIG is based on simple but generic and flexible models to uniformly represent and integrate the metadata and instance data of heterogeneous data sources. In contrast to data warehouses we are not trying to define a specific global schema to which subsets of the data sources need to be mapped. We follow a bottom-up approach for data integration that preserves the sources metadata as well as instance data and relationships but integrate them into uniform graph models. At the metadata level, we describe sources in terms of classes and associations to which instance objects and their relationships belong. In the following, we first introduce the so-called unified metadata graph (UMG) representing the metadata from all data sources. We then specify the instance graph model.

A. Unified Metadata Graph

The UMG is used to describe the classes and associations for every source as well as the associations between sources. We can semi-automatically derive the UMG representation for diverse data sources, in particular for relational databases. Furthermore, we can derive mappings from the data sources to the UMG that support an automatic extraction and integration of instance data. The UMG also provides additional value for the data analyst, as it reveals available classes and associations in a model of a, compared to the instance graph, compact size.

The UMG is defined as a pair $G^M = \langle V, E \rangle$ with the set of classes v as nodes and the set of associations E as edges. Each class $v \in V(G^M)$ is defined as $v = \langle d, c, t, a_i, A, \mu \rangle$ with data

source name d, class name c, class type t (transactional or master data), identity attribute a_i, the optional set of class attributes A and a schema translation (mapping) μ. An association $e \in E(G^M)$ is defined as $e = \langle r, v_s, v_e, a_s, a_e, i, A, \mu \rangle$ with relationship type r, start class v_s, end class v_e, start reference attribute a_s, end reference attribute a_e, the direction indicator i, the optional set of association attributes A and a mapping μ. If the direction indicator is true-valued, associations will be directed from v_s to v_e. Based on the definition, each class of the UMG is associated to a single source. By contrast, associations can link either classes from the same source or from different sources and thus support data integration.

To simplify data integration for heterogeneous sources, we limit the mandatory attributes for classes and associations to the minimal information required for identity and reference. Attributes for classes and associations in A thus are optional and document what attributes have been defined within a source. For schema-less sources (e.g., XML documents), we can document the attributes found in the instance properties. This information is intended to provide the analyst with the attributes available for analysis. The schema and instance translation between data sources and the UMG is described by class and association-specific mappings μ. The implementation of these mappings depends on the kind of data source. For relational sources it can be based on SQL statements (see section IV). To help generate nodes and edges of the integrated instance graphs, the mappings should specify the derivation of class and association instances from a source to a generic representation as pair-sets $\{\langle a_1, b_1 \rangle, .., \langle a_n, b_n \rangle\}$ with attributes a (identity, reference or property attribute) and values b.

Fig. 3 illustrates an example UMG for the two interlinked relational sources shown in Fig. 2; it also covers the classes and relationship types in the instance example of Fig. 4. The three tables of the CIT source are represented by two similarly named classes and an undirected association representing the n:m relationship between tickets and employees. In the UMG model, we have an `Employee` class for each source since both sources have an employee table (with different attributes and ids). However, the UMG contains a `sameAs` association between the `Employee` classes as well as another cross-system association between the `Ticket` and `SalesOrder` classes.

B. Integrated Instance Graph

Data objects and relationships from all sources are uniformly represented and integrated within the *integrated instance graph* (IIG). With this central data structure, BIIIG utilizes the versatility of graphs to represent heterogeneous data and to support the flexible analysis of data and relationships. For this purpose, the IIG should meet the following specific requirements:

- uniform representation of transactional and master data instances of different classes from different sources
- relationships of different semantic kinds
- directed relationships between any two objects
- multiple relationships between any two objects
- unrestricted number of named properties for objects and relationships.

The IIG as well as business transaction graphs (section V) are instance graphs that are defined as follows. An *instance graph* $G^I = \langle V, E \rangle$ consists of a set of nodes V and a set of edges E. A node representing a data object $v \in V(G^I)$ is defined as $v = \langle u, S, c, t, P \rangle$ with the unique identifier u, the set of source identifiers S, the class name c, the node type t (master or transactional data) and the set of named properties P. Class names refer to classes of the UMG. Corresponding objects for $sameAs$-connected classes from different sources are combined within one node. In this case we have multiple source identifiers in S referring to the original data objects to provide provenance information. An edge $e \in E(G^I)$ is defined as $e = \langle f, r, v_s, v_e, P \rangle$, with edge identifier f, relationship type r, the connected nodes v_s, v_e and the set of properties P. Relationship types refer to associations of the UMG. The identifier f differentiates multiple relationships between the same two nodes and the sequence of v_s and v_e expresses the relationship direction. For undirected associations we maintain edges in both directions. For both, nodes and edges, the set of properties may contain arbitrarily many pairs $P = \{\langle a_1, b_1 \rangle, .., \langle a_n, b_n \rangle\}$ of attribute a and value b.

For integrated objects we can merge the properties from the sources. For the example in Fig. 2, we can combine employees objects with `CIT.employees.erp_empl_number = ERP.EmplyeeTable.number` and merge their properties from both sources (`name, degree, dob, address, phone`).

It is easy to see that instance graphs meet all mentioned requirements. Nodes and edges can be heterogeneous as they may belong to different classes and relationship types and consist of different sets of properties. The support of node classes and edge types also ensures a semantic expressiveness of the graph structure for business-oriented analysis tasks. This graph model can be implemented with graph database systems supporting the property graph model.

IV. Metadata and Instance Integration

The generation of the UMG can partially be automated but needs support by an expert knowing the data sources. By contrast, the integration of data objects and relationships into the IIG is fully automated based on the UMG and its schema translations.

A. Metadata Acquisition

Metadata acquisition requires that each data source is expressed by classes and associations as defined in the UMG model. Furthermore, we need to link data sources at the metadata level either by utilizing existing references across sources or by specifying additional ones.

The source-specific metadata acquisition is inherently dependent on the underlying data model and chosen modeling decisions. While our approach can accommodate different kinds of data sources, we have so far focussed on the integration of relational databases due to their dominant role in

enterprise information systems. For UMG generation we need information about all tables including their primary key and foreign key constraints. Ideally we can query tables T, primary keys P^{T}, foreign keys $\mathsf{F}_{\mathsf{n}}^{\mathsf{T1},\mathsf{T2}}$ and all further columns $\mathsf{C}_{\mathsf{n}}^{\mathsf{T}}$ from the standardized SQL Information Schema of a database D.

We then try to automatically derive classes and associations assuming that tables either represent classes or complex (n:m) associations while n:1 or 1:1 relationships are expressed by foreign key constraints within class-like tables. Deviations from this common modeling approach need to be dealt with manually after the automatic translation steps.

For a class-like table T with primary key P^{T} we derive a class description $v = \langle d, c, t, a_i, A, \mu \rangle$ where we set the source name $d = \mathsf{D}$, the class name $c = \mathsf{T}$ and the identity attribute $a_i = \mathsf{P}^{\mathsf{T}}$. All further column names $\mathsf{C}_{\mathsf{n}}^{\mathsf{T}}$ except the foreign keys are included in A. The class type t has to be specified manually. Mapping μ is expressed by an SQL statement selecting all class instances with their primary key a_i and values for all further attributes in A. For the example class `Ticket` the mapping is thus simply :

```
SELECT id, description
FROM CIT.tickets.
```

For a foreign key $\mathsf{F}_{\mathsf{n}}^{\mathsf{T1},\mathsf{T2}}$ in a class-like table T1 referencing a table T2, we derive a directed n:1 (or 1:1) association $e = \langle r, v_s, v_e, a_s, a_e, i, A, \mu \rangle$ where v_s and v_e are the classes mapped to T1 and T2, the start reference attribute is the primary key $a_s = \mathsf{P}^{\mathsf{T1}}$ and relationship type $r = \mathsf{F}$ as well as the reference attribute $a_e = \mathsf{F}$ are the foreign key column. We further assume no association-specific attributes ($A = \emptyset$) and directed associations (direction indicator $i =$`true`) from the referencing to the referenced class. Mapping μ is expressed by a SQL statement to select all association instances with the id values of the inter-related objects. For the example association `processedBy` between `SalesOrder` and `Employee` in the ERP system this is achieved by the following SQL statement :

```
SELECT number, processedBy
FROM ERP.SalesOrderTable
WHERE processedBy IS NOT NULL.
```

For an association table T1 representing a binary n:m relationship via foreign keys $\mathsf{F}_1^{\mathsf{T1},\mathsf{T2}}$ and $\mathsf{F}_2^{\mathsf{T1},\mathsf{T3}}$ without a further primary key we derive an undirected association (i=`false`) and use the table name as the relationship type ($r = \mathsf{T1}$), the foreign keys as reference attributes ($a_s = \mathsf{F}_1$, $a_e = \mathsf{F}_2$), the classes corresponding to the two referenced tables T2,T3 as c_s and c_e, and include all further column names in A. The SQL statement for mapping μ thus selects the foreign keys to specify the two inter-related objects and all further association attributes. For association `processedBy` between `Ticket` and `Employee` in the CIT system the SQL mapping is :

```
SELECT ticket_id, empl_id, responsible
FROM CIT.ticket_employee.
```

The automatically generated classes and associations need to be enhanced by an expert, in particular to categorize classes as master or transactional data and to deal with cases not covered by the sketched translation approach. Furthermore, the expert can rename classes, associations and attributes to improve understandability for business analysts. An expert can also specify associations between different sources. Existing cross-system links need to be specified by dedicated associations in the UMG, such as the `sameAs` and `openedFor` associations in Fig. 3. In the example, these associations can be derived from foreign keys in the CIT tables (`erp_empl_number`, `erp_so_number`) referring to the ERP system. An example mapping for the latter is an SQL statement equivalent to other n:1 associations :

```
SELECT id, erp_so_number
FROM CIT.tickets
WHERE erp_so_number IS NOT NULL.
```

Associations between sources can be also determined with the help of a link discovery or schema matching tool [6].

B. Instance Graph Integration

The generation of the integrated intance graph is an automatic process with three main steps. First, class mappings are used to transform each source data object into a node in the IIG. Second, we use assocation mappings to generate edges representing relationships between objects. Finally, we combine corresponding data objects that are interconnected by a `sameAs` edge (relationship).

Algorithm 1 describes these steps in more detail. The input of the algorithm is the UMG including the class and association mappings, the result is the IIG. First, all source data objects are added as nodes to the integrated instance graph within a loop over all classes (lines 2-11). The class specific mapping μ (e.g. SQL statement) is used to query or extract the instances from a data source in order to represent them uniformly as data objects consisting of sets of attribute-value pairs (3). Every new node (4) obtains the class name and type from the currently processed class (5,6). The set of source identifiers S is implemented as an array. Initially, it contains only one identifier represented as the concatenation of the data source and class names as well as the object identity (7). The attribute name for the object identifier `class.id_attr` corresponds to a_i of the class definition. The chosen identifier format allows tracing back any node to its originating data source (provenance). In particular, it enables querying nodes by source identifiers as implemented in our edge integration. We further include all nonempty properties of the data object into the node, except the one already included in identity (8) and add the constructed node to the IIG (9).

Second, all relationships are added as edges to the IIG within a loop over all associations (12-29). Analogous to nodes, the association-specific mapping μ is used to query relationships, represented as sets of attribute-value pairs (13). The connected nodes will be queried from the IIG by their source identifier, after those identifiers have been concatenated with their respective source and class names (15,16). We will only continue with the current edge, if we find a node for both identifiers (17). The edge obtains its relationship type from the currently processed association (18) and receives all nonempty property values of the relationship except the ones already included in reference attributes (19). At the end the

Algorithm 1 Automated Instance Graph Integration

```
Input: unified metadata graph (umg)
Output: integrated instance graph (iig)
 1: iig = new Graph
 2: for all cls in umg.classes() do
 3:   for all obj in cls.instances() /* μ */ do
 4:     node = new Node
 5:     node.type = cls.type
 6:     node.class = cls.name
 7:     node.sids = [concat(cls.source,cls.name,obj[cls.id_attr])]
 8:     node.properties = obj.where(value != NULL and attr != cls.id_attr)
 9:     iig.add(node)
10:   end for
11: end for
12: for all asn in umg.associations() do
13:   for all rel in asn.instances() /* μ */ do
14:     edge = new Edge
15:     start_sid = concat(asn.start_class.source,start_class.name,rel.start_attr)
16:     end_sid = concat(asn.end_class.source,asn.end_class.name,rel.end_attr)
17:     if edge.start_node = iig.get_node(start_sid)
        and edge.end_node = iig.get_node(end_sid) then
18:       edge.type = asn.type
19:       edge.properties = rel.where(value != NULL
              and attr != asn.start_attr and attr != asn.end_attr)
20:       iig.add(edge)
21:       if not asn.directed then
22:         edge2 = edge.clone()
23:         edge2.start_node = edge.end_node
24:         edge2.end_node = edge.start_node
25:         iig.add(edge2)
26:       end if
27:     end if
28:   end for
29: end for
30: while edge = iig.first_edge_of_type(sameAs) do
31:   edge.end_node.sids += edge.start_node.sids
32:   edge.end_node.properties += edge.start_node.properties
33:   edge.start_node.edges.redirect_to(edge.end_node)
34:   iig.delete(edge)
35:   iig.delete(edge.start_node)
36: end while
37: return iig
```

Algorithm 2 Business Transaction Graph Isolation

```
Input: integrated instance graph (iig)
Output: set of business transaction graphs (btgs)
 1: btgs = new GraphSet
 2: trans_nodes = iig.get_nodes_by_type(Transactional)
 3: while trans_nodes.size() > 0 do
 4:   btg = new Graph // business transaction graph
 5:   seed_node = trans_nodes.random()
 6:   btg.add(seed_node)
 7:   cc_nodes = new NodeSet // causally connected nodes
 8:   cc_nodes.add(seed_node)
 9:   while cc_nodes.size() > 0 do
10:     node = cc_nodes.random()
11:     for all edge in node.edges() do
12:       if not btg.contains(edge) then
13:         if node == edge.start_node then
14:           next_node = edge.end_node
15:         else
16:           next_node = edge.start_node
17:         end if
18:         if not btg.contains(next_node) then
19:           btg.add(next_node)
20:           if next_node.type == Transactional then
21:             cc_nodes.add(next_node)
22:           end if
23:         end if
24:         btg.add(edge)
25:       end if
26:       cc_nodes.remove(node) // all edges traversed
27:       trans_nodes.remove(node) // allocated to single btg
28:     end for
29:   end while
30:   btgs.add(btg)
31: end while
32: return btgs
```

constructed edge is added to the IIG (20). In the case of an undirected relationship we construct a second edge with same relationship type and properties (22) but switched start and end nodes (23,24) and add it to the IIG (25).

In the final step, we fuse nodes representing the same logical object (30-37). For that, all `sameAs` edges are processed (30) and the start node is merged into the end node. First, the identifier array and the set of properties of the end node are merged with those of the start node (31,32) and second, all in- and outgoing edges of the start node are redirected to the end node (33). Finally, the `sameAs` edge as well as the initial start node are deleted (34,35).

V. Business Transaction Graphs

A main feature of BIIIG is the analysis of business activities represented by subgraphs of the IIG called business transaction graphs (BTG). The notion of BTGs is based on the observation that *transactional data objects* represent steps within business activities, e.g. the provision of a product quotation is represented by a corresponding quotation object. Such activity steps cause further actions, for example a sales order represented by its own transactional data object and so on. Assuming a data object has to be existent before it can be linked, edge directions provide information about the (inverse) sequence of corresponding business activities without the need for timestamps. Thus, we consider such relationships between transactional objects as *causal connections* that are represented by edges or paths between *transactional nodes* in the IIG. Transactional data objects also have relationships with *master data objects* (e.g., the employee providing the quotation), with corresponding *master data nodes* and edges in the IIG. However, a path between transactional nodes via master data nodes is in general no hint for a causal connection. For example, if two quotations involve the same product, these quotations can be completely independent. Hence, we define that two transactional nodes will be considered as *causally connected*, if they have at least one connection through an edge or a path involving only transactional data objects.

We therefore define a *business transaction graph* as a subgraph of the integrated instance graph, where all transactional nodes are causally connected. Because of their fundamental analytical value, a business transaction graph also contains all master data nodes connected to the transactional nodes.

Based on this BTG definition, the IIG can be transferred to a set of BTGs, which may overlap in master data nodes but not in transactional nodes or edges. Algorithm 2 shows how BTGs can be derived from the IIG. It starts with a candidate set of all transactional nodes contained in the IIG (2). The generation of a single BTG starts with an arbitrary transactional seed

node from this set (5) and is followed by the traversal of all connecting paths, where all passed nodes and edges are stored in a BTG (9-29). The traversal is stopped as soon as a master data node is reached (20). Every traversed node is removed from the candidate set to ensure that any transactional node as well as connected edges are exactly processed once (27). This leads to a linear time complexity of $O(|V(G)| + |E(G)|)$.

VI. Instance Graph Analytics

BIIIG aims at supporting comprehensive graph-based business analytics including graph mining and the evaluation of relationship patterns (like our motivating example). The design of the analysis capabilities for BIIIG has just begun and will be described in future publications. To get a first impression of our approach we sketch a few basic operators for selection, projection and aggregation. Projection and aggregation generate tabular results supporting a flexible and powerful OLAP post-processing using standard relational technology. Since BIIIG currently uses the graph database Neo4j to manage the IIG as well as the set of BTGs we can also use its query functionality for analysis, in particular the query language Cypher. At the end of this section we briefly discuss how well our requirements are already met by Neo4j and Cypher.

A. Operators

As the basis for analytical operations, we consider the set of BTGs as an indexed set of n graphs $\mathcal{G} = \{G_i\}_{1 \leq i \leq n}$.

a) Selection: The selection operator $\sigma_{H,\theta}(G_i)$ returns a set of m subgraphs $\{G_i^j \mid 1 \leq j \leq m; G_i^j \subseteq G_i\}$ each matching a specified search pattern. A search pattern consists of a search predicate θ referring to node and edge variables in a variable definition $H = \langle V, E \rangle$. The simplest search pattern for employee involvement describes a direct edge from a transactional node to an employee node, expressed by $H_{ex} = \langle \{v_1, v_2\}, \{e_1\} \rangle$ and $\theta_{ex} : t(v_1) = \texttt{Transactional} \wedge c(v_2) = \texttt{Employee} \wedge v_s(e_1) = v_1 \wedge v_e(e_1) = v_2$. The result of $\sigma_{H_{ex},\theta_{ex}}(G_i)$ is a set of subgraphs each containing exactly two nodes and one edge or will be an empty set, if the pattern does not occur.

b) Projection: The projection operator returns specified metadata and property values of graphs as a table. Projection is typically applied to the results of a selection operation. The projection $\pi_{a_1(x_1),..,a_k(x_l),m_1(x_1),..,m_n(x_l)}(\sigma_{H,\theta}(G_i))$ extracts values of k properties a_i and n metadata elements m_j of node or edge variables $x \in V(H) \cup E(H)$. The result is a table $T_i = \{\langle b_1, .., b_{k+n} \rangle^j\}$ with one row of k+n values per subgraph $G_i^j \subseteq G_i$ of the selection result. Reviewing our example the projection $\pi_{c(v_1),r(e_1),\texttt{name}(v_2)}(\sigma_{H_{ex},\theta_{ex}}(G_i))$ determines for every employee involvement its kind of business activity (transactional class $c(v_1)$), relationship type $r(e_1)$ and employee name. An example result row would be $\langle \texttt{SalesQuotation}, \texttt{sentBy}, \texttt{Alice} \rangle$.

c) Aggregation: In business analytics we frequently want to derive aggregated values such as the profit per BTG. Similar to projection, aggregation is typically applied to the result of a selection operation, in particular with a search pattern locating the base values to aggregate. The aggregation $\gamma_{a(x)}(\sigma_{H,\theta}(G_i))$ applies an aggregation function γ (e.g. SUM or AVG) on values of a specified property a for a node or edge variable $x \in V(H) \cup E(H)$. Aggregation extracts the a values from all subgraphs of the selection result and aggregates them into a single measure value. We can use separate search patterns for revenue-related nodes $H_r = \langle \{v_1\}, \emptyset \rangle$ with $\theta_r : c(v_1) = \texttt{SalesInvoice}$ and expense-related nodes $H_e = \langle \{v_2\}, \emptyset \rangle$ with $\theta_e : c(v_2) = \texttt{PurchaseInvoice} \vee c(v_2) = \texttt{Ticket}$ to calculate the profit for BTG G_i by $\texttt{profit}_i = \texttt{SUM}_{\texttt{Revenue}(v_1)}(\sigma_{H_r,\theta_r}(G_i)) - \texttt{SUM}_{\texttt{Expense}(v_2)}(\sigma_{H_e,\theta_e}(G_i))$.

B. Relational Postprocessing

Since projections determine relational tables and aggregations determine scalar values we can use these results for a relational postprocessing to support a comprehensive analytics across all BTGs $\mathcal{G} = \{G_i\}$. For this purpose, we extend projection result tables $\mathcal{T} = \{T_i\}$ with an BTG index column and unite all records into a single table $T_p = \{\langle i, b_1, .., b_{k+n} \rangle\}$. Similarly, we can maintain all aggregation results in a combined table $T_m = \{\langle i, m_1, m_2, ... \rangle\}$ with a column i for the BTG index and one additional column per BTG measure m such as profit or number of customer complaints. To relate measures and projection results we can apply a natural join $T_p \bowtie T_m$ via the BTG index i. Since projections often express relationship patterns of interest, e.g. on employee involvement, the resulting join table supports comprehensive, multidimensional grouping and aggregations across BTGs. For our example, we can thus determine the employees, business activities or relationship types that are most frequently involved in high profit BTGs or those with customer complaints.

C. Capabilities of Neo4j and Cypher

A fundamental lack of Neo4j is the missing support for managing and thus also querying sets of graphs. Furthermore, the projection to non-graph structures (e.g. tables) in the RETURN clause is mandatory. While this satisfies our current approach, it might be limiting for analytics involving chained graph operations. It is also not possible to apply further relational operations on multiple selections, which are projected to tables. As the major positive point, our selection and projection operators on single graphs are well covered. Cypher supports the specification of search patterns, where the MATCH and WHERE clauses correspond to H and θ. The following example query implements our example projection for employee involvements:

```
MATCH (v1)-[e1]->(v2)
WHERE v1.type="Transactional" AND v2.class="Employee"
RETURN v1.class,type(e1),v2.Name;
```

Cypher also supports the aggregation of property values. The following example query corresponds to the revenue part of our profit example :

```
MATCH (v1)
WHERE v1.class="SalesInvoice"
RETURN SUM(v1.Revenue);
```

VII. Experimental Evaluation

For an initial evaluation of the BIIIG framework, we used data of a real installation of the open source ERP system ERPNext [2]. The used data is kept in a MySQL database and covers fictive business activities for application development and testing. While the database size is relatively small we found the data realistic and useful for an initial proof-of-concept. We could automatically determine the tables, columns and primary keys following the approach of IV-A. However, the metadata description of the ERP database was missing foreign keys and we had to support the generation of the UMG by a custom script. In particular, foreign key candidates were chosen based on names and datatypes and validated by table joins. In the result, we derived 102 classes and 583 associations. Initially, class names were chosen corresponding to table names and association names corresponding to foreign key column names or (for n:m relationships) the name of the reference table. These names turned out to be meaningful for a person familar to ERP systems but we translated class and association names to more common business terms and also typified classes as transactional or master data. We implemented algorithm 1 in Java and used it together with the generated UMG to create the IIG. The IIG for the used ERP data has 8,358 nodes, 38,892 edges and 87,746 nonempty properties. The integration process took about 10 seconds on commodity hardware and the IIG size for the graph database Neo4j is about 30 MB on disk. Algorithm 2 was implemented using the native API of the graph database. In our current implementation we use a second database instance with isolated subgraphs (including redundant master data), where every node provides a dedicated property `btg_id` representing its BTG index. We generated 1,983 BTGs in about 2 seconds. The BTG size ranges from 2 to 221 nodes and 2 to 883 edges. Spot tests of BTGs yielded only reasonable results in the context of business activities, for example interrelated transactional data objects from first sales activities over product purchase and invoicing up to general ledger accounting as well as the involved master data such as employees, customers and products. We expect real-world BTGs containing such data to have at least the size of the biggest ones resulting from the experiment dataset. Finally, we could successfully execute queries corresponding to the examples in section VI-C.

VIII. Conclusion and Future Work

We proposed the BIIIG framework for graph-based data integration and business intelligence. It utilizes simple but flexible graph models to uniformly represent metadata and instance data of diverse sources such as ERP and related systems. Metadata acquisition and instance integration are largely automatic and retain valuable relationships represented in the source data for later business analysis. BIIIG also includes the automatic generation of so-called business transaction graphs to enable the focused analysis of business activities with all their steps and involved people as well as other resources. BIIIG will provide comprehensive query and data mining facilities based on graph patterns although the analysis component is still under development. An initial version of BIIIG for relational data sources based on an existing graph database has already been implemented and was successfully applied to a real ERP use case. In future work, we will complete the specification and implementation of the BIIIG analysis capabilities. We will also investigate the need and implications for more general BTGs where transactional objects can be used in more than one BTG to represent cross-BTG dependencies as already addressed in [11]. We will further apply and evaluate BIIIG for more and larger use cases.

Acknowledgments

This project is partly funded within the EU program *Europa fördert Sachsen* of the European Social Fund. We are grateful to Web Notes Technologies for providing us with the data for the ERPNext use case.

References

[1] "Cypher query language," http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html.
[2] "ERPNext, ERP system," www.erpnext.com.
[3] "Neo4j, graph database," www.neo4j.org.
[4] "Teradata Aster SQL-GR," http://www.asterdata.com/product/sql-gr.php.
[5] R. Angles, "A comparison of current graph database models," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th Int. Conf. on*, 2012.
[6] Z. Bellahsene, A. Bonifati, and E. Rahm, Eds., *Schema Matching and Mapping*. Springer, 2011.
[7] D. Bleco and Y. Kotidis, "Business intelligence on complex graph data," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, 2012.
[8] C. Chen et al., "Graph OLAP: Towards online analytical processing on graphs," in *Data Mining. ICDM'08. Eighth IEEE Int. Conf. on*, 2008.
[9] H. Cheng, X. Yan, and J. Han, "Mining graph patterns," in *Managing and Mining Graph Data*. Springer, 2010, pp. 365–392.
[10] C. David, C. Olivier, and B. Guillaume, "A survey of RDF storage approaches," *ARIMA Journal*, vol. 15, pp. 11–35, 2012.
[11] D. Fahland et al., "Many-to-many: Some observations on interactions in artifact choreographies." in *ZEUS*, 2011, pp. 9–15.
[12] G. Klyne, J. J. Carroll, and B. McBride, "Resource description framework (RDF): Concepts and abstract syntax," *W3C rec.*, vol. 10, 2004.
[13] D. Koop, J. Freire, and C. T. Silva, "Visual summaries for graph collections," 2013.
[14] Y. Kotidis, "Extending the data warehouse for service provisioning data," *Data & Knowledge Engineering*, vol. 59, no. 3, pp. 700–724, 2006.
[15] D. Lam et al., "Graph-based data warehousing using the core-facets model," in *Advances in Data Mining. Appl. and Theor. Aspects*, 2011.
[16] F. Menge, "Enterprise service bus," in *Free and open s. softw. c.*, 2007.
[17] E. Nooijen et al., "Automatic discovery of data-centric and artifact-centric processes," in *Business Process Management Workshops*, 2013.
[18] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," *Bulletin of the Amer. Society for Inf. Sci. and Tec.*, vol. 36, no. 6, 2010.
[19] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner, "The graph story of the SAP HANA database." in *BTW*, 2013, pp. 403–420.
[20] M. Rudolf et al., "SynopSys: Large graph analytics in the SAP HANA database trough summarization," in *GRADES*, 2013.
[21] S. Sakr and E. Pardede, *Graph Data Management: Techniques and Applications*. Information Science Reference, 2011.
[22] A. Seaborne, "SPARQL 1.1 property paths," http://www.w3.org/TR/sparql11-property-paths/.
[23] R. Soussi et al., "DB2SNA: an all-in-one tool for extraction and aggregation of underlying social networks from relational databases," in *The inf. of tec. on social network analysis and Mining*, 2013.
[24] Y. Sun, J. Han, C. C. Aggarwal, and N. V. Chawla, "When will it happen?: relationship prediction in heterogeneous information networks," in *Proc. of the 5th ACM int. conf. on Web search and data mining*, 2012.
[25] M. Yin, B. Wu, and Z. Zeng, "HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis," in *Proceedings of the 15th int. workshop on Data warehousing and OLAP*, 2012.
[26] P. Zhao, X. Li, D. Xin, and J. Han, "Graph cube: on warehousing and olap multidimensional networks," in *SIGMOD Conference*, 2011.