Gathering Formalized Information Requirements of a Data Warehouse

Natalija Kozmina, Laila Niedrite and Janis Zemnickis
Faculty of Computing, University of Latvia, Raina blvd. 19, Riga, Latvia

Keywords: Data Warehouse, Information Requirements, Indicators, Formalization, Graphical User Interface.

Abstract: Information requirements of a data warehouse (DW) captured in natural language often have a common issue of being ambiguous, inaccurate, or repeating. We offer an approach to formalize DW information requirements based on our experience of using demand-driven methodology for DW conceptual design and distinction between quantifying and qualifying data. In this paper we demonstrate a working prototype of the iReq tool implemented for the purpose of collecting DW information requirements. Graphical user interface (GUI) of the iReq tool conforms to the requirement formalization metamodel acquired as a result of our previous research studies, is intuitive and user-friendly, and allows to define an unlimited number of requirement counterpart elements. The functionality of the iReq tool is wide; it allows deriving a conceptual model of a DW in a semi-automatic manner from gathered information requirements. Due to space limitations, in this paper we cover only such components as GUI for input of the information requirements illustrated with application examples, its underlying formal requirement repository, and a graph database (DB) to represent a glossary of terms for requirement definition.

1 INTRODUCTION

Companies use DW systems to evaluate their progress, they measure different aspects of performance and analyze performance indicators. "A *data warehouse* is a subject-oriented, integrated, non-volatile, and time-variant collection of data in support of management decisions" (Inmon, 2002). Data are stored according to a multidimensional data model of a DW. Data model must be implemented in alignment with the information requirements (Winter & Strauch, 2003) of a company.

Information requirements represent *quantifying data* needed for performance measurement and a lot of contextual attributes or so called *qualifying data* that allow to analyze numerical performance measurements in different perspectives and at various levels of abstraction. Information requirements must be appropriately transformed to the elements of multidimensional paradigm, e.g. dimensions, hierarchies, and cubes.

Methods for developing conceptual models of DWs are either supply-driven or demand-driven. Supply-driven methods use models of data sources to determine the existing information requirements and transform them into a DW model in a more or less automated way. However, these methods do not rely on actual analysis needs – instead they reflect only operational data. Information requirements in case of demand-driven methods (e.g. goal-oriented or requirement-oriented) are collected during the interviews. It is essential to elicit and analyze typically unstructured information in a guided manner during interviews (Prakash, 2016). Then, a conceptual model is constructed based on the knowledge about these requirements. To (partially) automate this process, information requirements must be represented formally. It is possible only when some unified pattern in the formulated information requirements can be discovered. A similar problem of lacking a unified method for eliciting and managing information requirements also exists in the field of goal-oriented requirement engineering for a DW system (Nasiri et al., 2015).

When the DW is used for performance measurement purposes, information requirements represent different performance indicators that are expressed as more or less complex sentences with similar structure. We could observe the similar structure of these sentences while analyzing a set of indicators from indicator database (Parmenter, 2010).

Kozmina, N., Niedrite, L. and Zemnickis, J.
Gathering Formalized Information Requirements of a Data Warehouse.
DOI: 10.5220/0006276802170224
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 1*, pages 217-224
ISBN: 978-989-758-247-9

We have proposed a formal specification of indicators in one of our previous works (Niedritis et al., 2011), and a method for transforming formally expressed information requirements or indicators into a conceptual model of a DW (Kozmina et al., 2013). The formal specification of requirements (Niedritis et al., 2011) is represented with an indicator metamodel that is based on analysis of the structure of sentences that express requirements. Results were used in a real DW project to extend the metamodel with elements discovered in terms of a case study (Kozmina & Niedrite, 2014).

These experiments were made without an appropriate tool support and the need for such tool was urgent, so a prototype tool was implemented. The goal of this paper is to discuss pertinence of the method for formal indicator definition and discover further improvements.

The rest of the paper is organized as follows. Background of the study is given in Section 2, Section 3 describes the working principles and examples of application of the iReq tool for input of formalized DW information requirements, related work is presented in Section 4, whereas Section 5 finalizes the paper with conclusions and future work.

2 BACKGROUND

In the course of our research, we found out that there is a need of: (i) models for representing requirements in a more formal way, and (ii) (semi-automatic) methods to handle the process of mapping requirements to design.

A schematically captured semi-automated method for transforming information requirements to the conceptual model of a DW could be seen in Figure 1. The first step of the method is creating a repository of formal requirements. The method uses a set of requirements that complies with the requirement formalization metamodel (Kozmina & Niedrite, 2014) and is stored in the formal requirement repository. The GUI of iReq tool presented in this paper ensures requirement input using a glossary of terms (and its synonyms) derived from data elements of the DW source systems and stored in a graph database (see Section 3). Then, (a) simplified DW schema(s) is (are) acquired as an output of the Pre-schema Generation Algorithm (PGA) that analyses the structure of collected requirements. Later, a developer processes pre-schemas, for instance removes duplicate attributes or builds hierarchies. One of the improved schemas that meets the requirements best is chosen during an interview with a client to instantiate a conceptual model of the DW. Finally, requirement priorities (Kozmina & Niedrite, 2014) are analyzed to learn, for example, which of the planned reports should be developed prior to others, which schema elements to incorporate into dashboards, etc.

A Requirement mapping component can adjust DW schema in accordance with evolving business requirements in a semi-automatic fashion (Solodovnikova et al., 2015). The algorithm processes new and obsolete requirements and deploys procedures that create new DW schema versions. The algorithm takes advantage of the DW logical and physical level metadata, requirement formalization metamodel, and information about data elements of the DW source systems to propagate changes in requirements in the DW schema.

This paper covers such components as GUI for formal requirements input, its underlying repository, and a graph DB to represent a glossary of terms. A detailed explanation of other components is a subject of a separate paper.

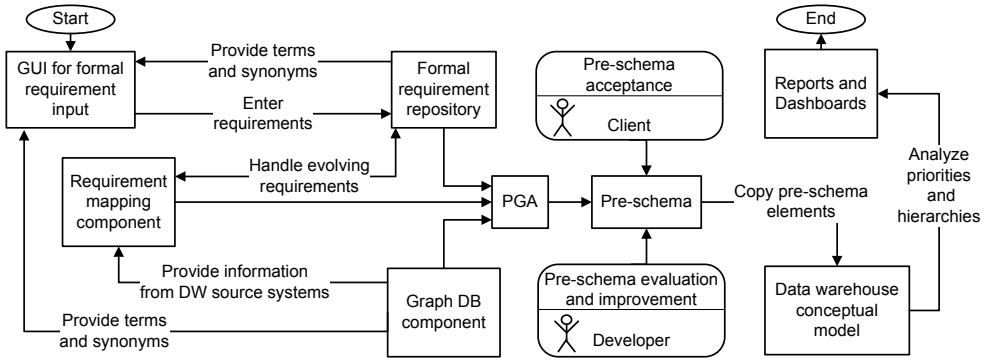


Figure 1: Method for transforming information requirements to the conceptual model of a DW.

3 iREQ TOOL FOR FORMAL REQUIREMENTS MANAGEMENT

iReq is a web-based tool with a back-end written in PHP using Laravel framework and a front-end (GUI) developed using JavaScript, CSS, jQuery framework, and Bootstrap. DW indicators are stored in the database management system (DBMS) MariaDB, and information about DW source systems and a glossary – in a graph database Neo4j.

Main requirements for the iReq GUI are: (i) conformance to the requirement formalization metamodel, (ii) ability to define and save requirements, (iii) intuitive and user-friendly GUI, and (iv) ability to define an unlimited number of child elements. Current implementation of the iReq tool meets the requirements, however, as the number of requirement counterpart elements grows, the demonstrativeness of the requirement gets lower. It is particularly noticeable when iReq tool is used on a device with a small screen.

3.1 An Example of Requirement Formalization

We refer to the version of our proposed requirement formalization metamodel published in (Kozmina & Niedrite, 2014). However, in the course of iReq tool development, we updated the metamodel with two more classes – i.e. *Business Process* and *Stakeholder*. Both classes have an added value, e.g. data on Business Process improves the traceability of requirements, while data on *Stakeholder* makes it possible to process conflicting requirements stated by different stakeholders. Data on both *Business Process* and *Stakeholder* derived from the formalized requirements and associated with corresponding DW schema elements would help manage user rights on the stage of analytical reporting, and keep track of DW schema changes in terms of certain business processes.

In this section we describe the requirement formalization metamodel with iReq running examples of iReq application depicted on Figure 2 and Figure 3. Informal requirements are as follows: "*Show information on student to academic staff ratio in each faculty*" (see formal version on Figure 2) and "*Show information on students from Riga that attend lectures held in Latvian*" (see formal version on Figure 3). The bottom row elements represent parts of the formal requirement statement, whereas the remaining elements contain the model class names.

The key element of the metamodel is a *Requirement*. Typically, a *Requirement* emerges in terms of one or another *Business Process* (e.g. Study process) and is expressed by a *Stakeholder* (e.g. University Senate). Each requirement is characterized by one or multiple *Groups* (e.g. Strategic Plan 2010-2020), while a *Theme* as a coarser level of grouping (e.g. Studies) may form one or more groups. Each requirement is assigned a *Priority* value according to MoSCoW prioritization technique (e.g. should). Each *Requirement* classified either as *Simple* or *Complex*. A *Complex Requirement* is composed of two or more *Requirements* either joined with an *Arithmetical Operator* (e.g., '/') or a *Comparison*, or not joined with anything.

As results of the case study in (Kozmina & Niedrite, 2014) show, a *Simple Requirement* may consist of an *Expression* only – in this case, it allows to compare a part of the requirement with some expression or a pre-defined constant value.

A *Complex Expression* contains two or more *Expressions* with an *Arithmetical Operator* in between, whereas a *Simple Expression* can be *Qualifying* data (e.g. employee_group, language, city) or a *Constant* (e.g. "academic staff", "LV", "Riga"). A *Simple Requirement* may consist of an *Operation* that denotes a command applied to an *Object*, and an optional *Typified Condition*. A *Complex Operation* consists of two or more *Actions*, which are either *Aggregation* ("roll-up"; e.g. count) or *Refinement* ("drill-down"; e.g. show). In its turn, an *Object* is either an instance of *Quantifying* data (measurements, e.g. student, employee) or *Qualifying* data (attributes or properties of measurements, e.g. faculty, employee_group). Slicing of information about *Objects* is ensured by adding a constraint, i.e. a *Typified Condition* as seen on Figure 3: "where language = 'LV' and city = 'Riga' ". A *Complex Condition* joins two or more conditions (e.g. "language = LV", "city = Riga") with a *Logical Operator* from the set of values: {"and", "or", "not"}. A *Simple Condition* consists of a *Comparison* (values: {>, <, <=, >=, =, "is"}) of two *Expressions*.

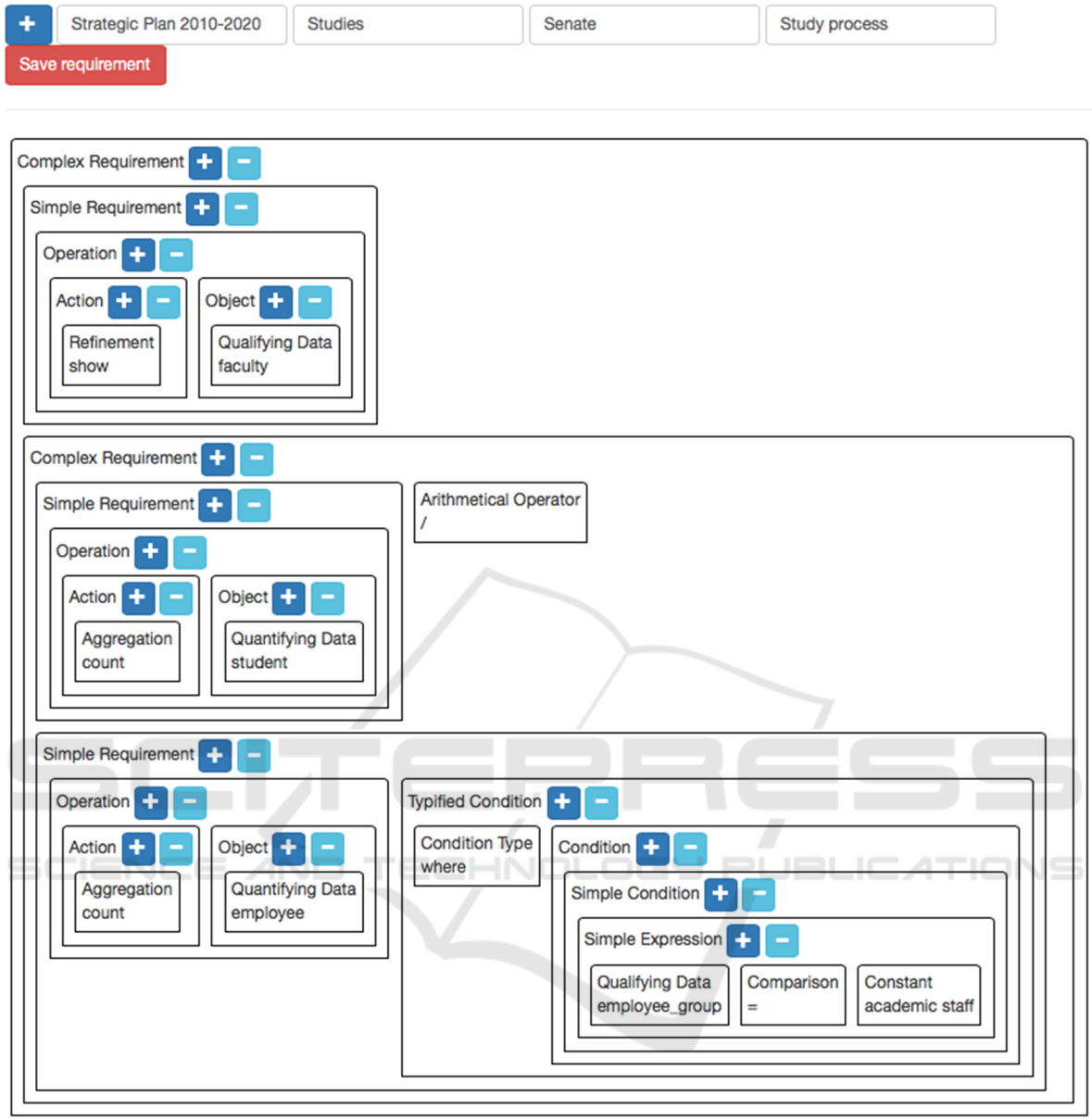


Figure 2: An example of the information requirement *"show information on student to academic staff ratio in each faculty"* represented as a complex formalized requirement *show (faculty) count (student) / count (employee) where employee_group = "academic staff"* in iReq tool.

Principles of reformulation of information requirements (indicators) applied in examples above:

- A component to be measured is treated as an aggregated number of all occurrences of this component. For example, "students" is reformulated to "count (student)" (also acceptable: "count (student occurrence)"), where count is the most suitable aggregate function;
- If a requirement contains such keywords as "%", "percent", "percentage", or "ratio", then % is treated as division of partial quantity by its total quantity, and ratio – a comparison of two different quantities by division. For instance, "student-to-staff ratio" is reformulated to "count (student) / count (employee)".

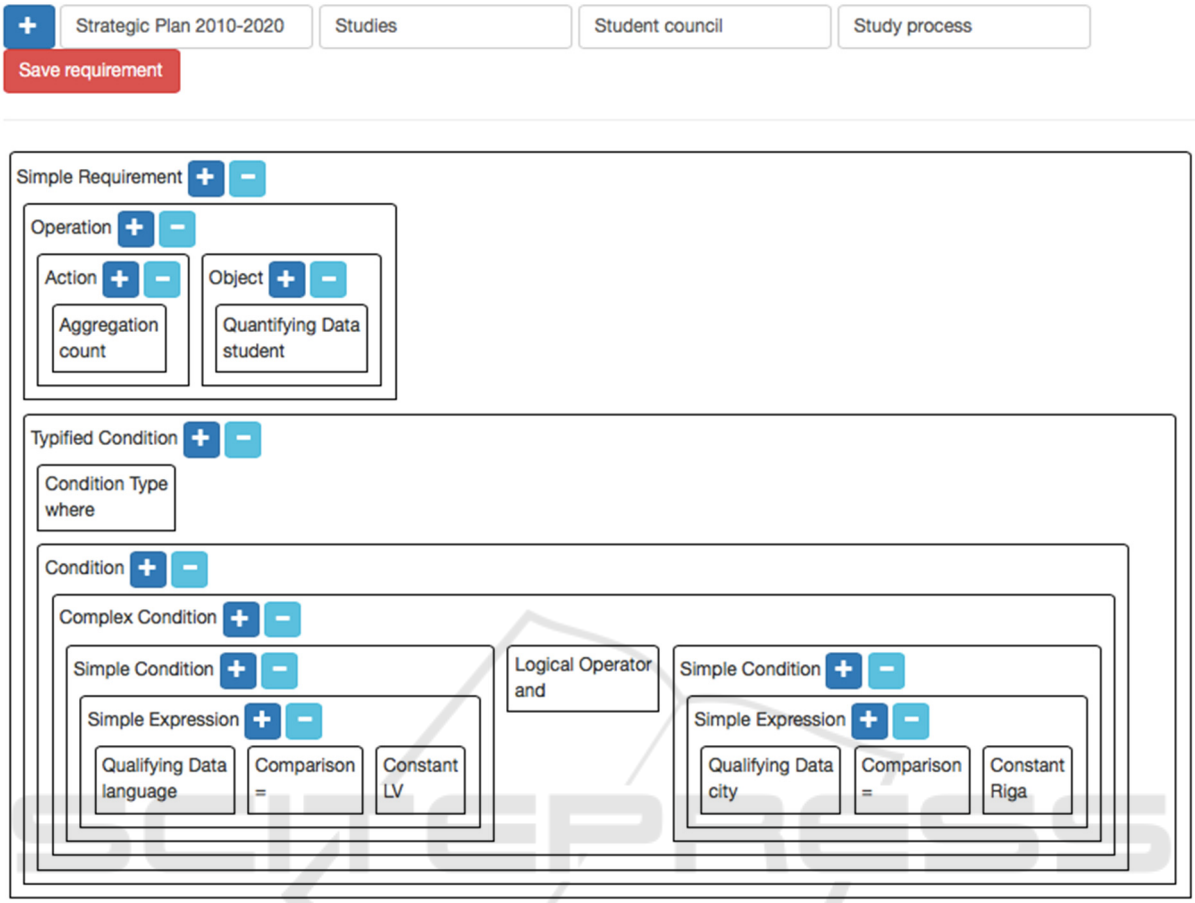


Figure 3: An example of the information requirement *"show information on students from Riga that attend lectures held in Latvian"* represented as a simple formalized requirement with complex condition *count (student) where language = "LV" and city = "Riga"* in iReq tool.

The full list of principles of information requirements (indicators) reformulation is available in (Niedritis et al., 2011). These principles appeared empirically and serve to translate requirements from natural language to a state that is compatible with the requirement formalization metamodel.

3.2 Configuration File

Main purpose of the iReq tool is to provide input of formalized indicators in accordance with the requirement formalization metamodel (Kozmina & Niedrite, 2014). Classes and relationships defined in the metamodel are translated into GUI component of the iReq tool by means of the configuration file in JSON format that describes the metamodel.

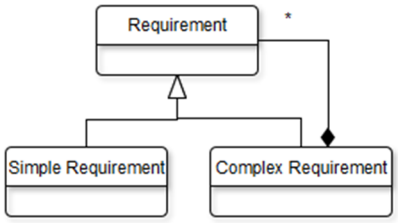
Figure 4(a) depicts a fragment of the metamodel illustrating a Requirement element (which is either Simple or Complex), and a Complex requirement that contains one or many Requirement elements. Figure 4(b) demonstrates a corresponding part of the GUI with Requirement root element and two child elements (i.e. Simple and Complex Requirements) as a result of processing a configuration file.

The configuration file consists of a set of objects and their attributes, where each object defines some class of the metamodel. Attribute examples are *id* (unique ID, e.g. "arithOper"), *name* (e.g. "Arithmetical Operator"), *parent*, *action*, and *values*. An example of the *action* attribute value is "dropdown", which enables a dropdown menu with a set of values. Examples of *values* attribute are: "+", "-", "*", "/" of the *Arithmetical Operator* object. Attribute *parent* contains a set of values that refer to parent objects of a particular element.

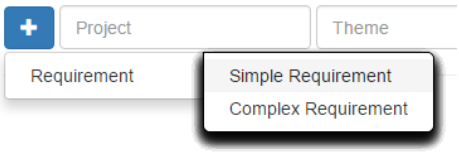
A fragment of the configuration file is shown on Figure 4(c), and it contains 3 objects ("req", "simpleReq", and "complexReq"), and its parent-child relations. Although the configuration file ensures indicator input that conform to the

metamodel, currently it doesn't prevent from all error cases, e.g. a user can add more than one "Arithmetical Operator" elements in a row.

(a)



(b)



(c)

```
{
    id: 'req',
    name: 'Requirement',
    parent: null,
    values: null,
    action: null
},
{
    id: 'simpleReq',
    name: 'Simple Requirement',
    parent: ['req', 'complexReq'],
    values: null,
    action: null
},
{
    id: 'complexReq',
    name: 'Complex Requirement',
    parent: ['req', 'complexReq'],
    action: null,
    values: null
},
```

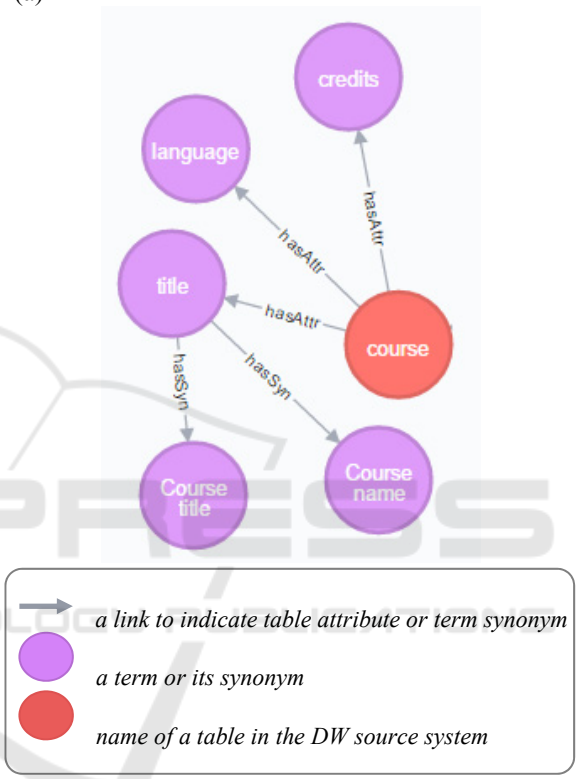
Figure 4: Parent-child relations of the elements presented as a fragment of the (a) metamodel, (b) GUI of the iReq tool, (c) configuration file of the iReq tool.

3.3 Glossary of Terms & Term Suggestions

In iReq tool, a user describes quantifying and qualifying data with words and phrases, which are later employed in pre-schema generation algorithm. To decrease the rate of typos and semantic mistakes, iReq uses both (i) previously entered words and phrases stored in formal requirement repository, and (ii) a glossary of terms.

A glossary of terms in iReq tool is derived from data elements of the DW source systems (either manually or exported from the DW source system as a .CSV file) and implemented as a Neo4j graph DB. Terms are connected by links of different kind. It can be easily modified and supplemented with new terms and its synonyms.

(a)



(b)

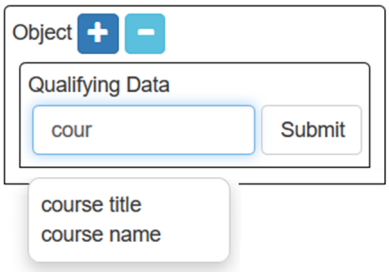


Figure 5: A fragment of the (a) glossary of terms and (b) data input with term suggestions.

An instance of the graph is seen on Figure 5, where a DW source system contains a table "Course" with columns "Title", "Language", and "Credits". Column and table names are connected with a "hasAttr" link. Terms "Course title" and

"Course name" are added to the glossary on Figure 5, and are connected to an attribute "title" with a "hasSyn" link that denoted that these two terms are synonyms of "title". When a user would start typing "title", he/she could see "Course title" and "Course name" as possible options.

Although it is recommended to select terms from the glossary, a user may decide not to choose a term, if the glossary seems incomplete. It is particularly useful, if there is a need to state a new information requirement on the data that doesn't exist in a DW source system and in the glossary respectively. Such requirements may indicate a necessity for adding new data to the DW source system.

3.4 Data Model of iReq Formal Requirement Repository

Each requirement indeed has a hierarchical tree structure where the complete formalized requirement itself is a root node linked to its counterpart elements by a parent-child relationship (e.g. a *Complex Requirement* contains *Simple Requirements*). Besides, one parent element may have an arbitrary number of children. When creating a new child node, iReq tool keeps track of its parents. All elements are parsed starting from the children at the lowest level, moving up the tree data structure from children to parents, and stored to the database.

The data model describes how requirement data entered via iReq tool are physically stored in the database. Table *classes* stores data on all the elements that each requirement consists of, including their type that corresponds to the metamodel class (*Quantifying Data, Action, Simple Condition,* etc.), while *class_rels* stores data on its relations according to parent-child structure. Data on stakeholders, business processes, and its relations (*stakeholders_rels*, *business_processes_rels*) are stored in a similar fashion. Each requirement is associated with a particular theme and group. Also, each successfully saved or updated requirement is interpreted as an event; its data is stored in table *reqs* and its counterpart elements are tied together with *reqs_id*.

We deliberately didn't divide requirement elements into simple and complex on physical level. The element type is captured in *classes.type*, while parent-child relations and cardinality restrictions stated in the requirement formalization metamodel are defined in the configuration file. Thus, if relations between classes change with time or new classes of requirement counterpart elements appear in the metamodel, it doesn't affect the physical data model.

4 RELATED WORK

The iReq tool presented in this paper is a working prototype based on demand-driven methodology for deriving a conceptual model of a DW semi-automatically. Below we list several papers that address the same issue.

In (Cravero Leal et al., 2013) a business-oriented approach for DW development is presented. VMOST (vision, mission, goals, strategies, objectives, and tactics) business strategy analysis is followed by the alignment of the elements according to BMM or Business Motivational Model (OMG, 2015). Then, i* strategic dependency (SR) and strategic rationale (SR) models are developed that later are manually transformed into a multidimensional UML class diagram. Cravero Leal et al. (2013) provide a set of guidelines for producing a conceptual model of a DW, however, a prototype tool is yet to come.

In its turn, a tool that generates a DW conceptual model in an automatic manner is described in (Thenmozhi & Vivekanandan, 2013). It employs a hybrid design approach (both demand-driven and supply-driven) and takes advantage of matching information requirements formally represented using ontology with the data source ontology. In terms of this approach, a designer has to state explicitly goals, context (to derive attributes), and measures, whereas in iReq the distinction of attributes and measures is done automatically.

Pardillo & Mazón (2011) discuss applications of ontologies in the context of DW design such as incompleteness of multidimensional model, specification of additivity constraints, reconciling requirements and data sources, etc. For instance, if the aim is to automatize the process of matching information requirements with data sources, it is advised to employ semantic knowledge from multidimensionally-annotated ontologies. Above-mentioned issues may be a subject for future work on improvement of the iReq tool.

5 CONCLUSIONS AND FUTURE WORK

In this paper we demonstrated the iReq tool implemented for the purpose of collecting DW

information requirements (i.e. indicators). GUI of the iReq tool conforms to the requirement formalization metamodel (Kozmina & Niedrite, 2014), is intuitive and user-friendly, and allows to define an unlimited number of requirement counterpart elements.

DW information requirements input by mean of the iReq tool may take up more time, if a requirement consists of a large set of counterpart elements, because each element has to be added separately using GUI. An experienced iReq user, who has no difficulties with defining formal requirements manually, might want to enter DW information requirements as an input expression that would be processed by iReq tool and saved into the formal requirement repository. This feature is not available in current version of the iReq tool, but is planned to be added to iReq GUI in the future.

The aim of this paper was not to discuss further use of the collected indicators with a purpose to generate a DW candidate schema (i.e. pre-schema) semi-automatically according to the process depicted on Figure 1. Module of the iReq tool, which generates pre-schemas, handles formal requirements in compliance with particular algorithms (Kozmina et al., 2013), optimization mechanisms, and produces graphical representation of the DW pre-schemas. It is planned to give user an opportunity to manually accept, reject, or unite pre-schemas to acquire an optimal DW conceptual model that is aligned to user requirements. The analysis of such functionality of the iReq tool, its implementation, and practical evaluation of its adequacy in terms of generation of the DW conceptual model are a subject of a separate paper.

REFERENCES

Cravero Leal, A., Mazón, J.-N., Trujillo, J., 2013. *A business-oriented approach to data warehouse development*. Ingenieria e Investigación, 33(1):59-65.

Inmon, W.H., 2002. *Building the Data Warehouse*, Wiley Computer Publishing, 3rd edition.

Kozmina, N., Niedrite, L., 2014. Extending a Metamodel for Formalization of Data Warehouse Requirements. In: *Johansson, B. et al. (eds.) Perspectives in Business Informatics Research (BIR'14)*, LNBIP 194, Springer, Berlin, pp. 362-374.

Kozmina, N., Niedrite, L., Golubs, M., 2013. Deriving the Conceptual Model of a Data Warehouse from Information Requirements. In: Proc. of the 15th Int. Conf. on Enterprise Information Systems (ICEIS'13), vol. 1, pp. 136-144.

Nasiri, A., Zimányi, E., Wrembel, R., 2015. *Requirements Engineering for Data Warehouses*. Revue des Nouvelles Technologies de l'Information (EDA), RNTI-B-11:49-64.

Niedritis, A., Niedrite, L., Kozmina, N., 2011. Performance Measurement Framework with Formal Indicator Definitions. In: *J. Grabis, M. Kirikova (eds.) Perspectives in Business Informatics Research (BIR'11)*, LNBIP 90, Springer, Berlin, pp. 44-58.

OMG, *The Business Motivational Model*, 2015. Available online: http://www.omg.org/spec/BMM/1.3.

Pardillo, J., Mazón, J.-N., 2011. *Using Ontologies for the Design of Data Warehouses*. Int. Journal of Database Management Systems (IJDMS), 3(2):73-87.

Parmenter, D., 2010. *Key Performance Indicators: Developing, Implementing, and Using Winning KPIs*, Jon Wiley & Sons, Inc., 2nd edition.

Prakash, D., 2016. Eliciting Information Requirements for DW Systems. In: *Proc. of 28th Int. Conf. on Advanced Information Systems Engineering (CAiSE'16, Doctoral Consortium)*. Available online: http://ceur-ws.org/vol-1603/10000044.pdf.

Solodovnikova, D., Niedrite, L., Kozmina, N., 2015. *Handling Evolving Data Warehouse Requirements*. In: Communications in Computer and Information Science, vol. 539, pp. 334-345.

Thenmozhi, M.; Vivekanandan, K. *A Tool for Data Warehouse Multidimensional Schema Design using Ontology*. International Journal of Computer Science Issues (IJCSI), 10(2):161-168, March 2013.

Winter, R., Strauch, B., 2003. A Method for Demand-driven Information Requirements Analysis in Data Warehousing Projects. In: *Proc. of 36th Annual Hawaii Int. Conf. on System Sciences (HICSS'03)*, USA, IEEE, pp. 1359-1365.