Developing Factual Knowledge from Medical Data by Composing Ontology Structures

Marin Prcela[1], Dragan Gamberger[1], Nikola Bogunović[2]
[1] Ruđer Bosković Institute, Laboratory for Information Systems, Bijenička 54, 10000 Zagreb, Croatia
[2] Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia
Phone: (+385) 1-456 1159 E-mail: marin.prcela@irb.hr

Abstract - Decision support systems in medical applications essentially rely on knowledge acquired from existing patient databases. The core task in developing knowledge from such raw data is a mapping from the relational data model to the domain ontology structure. The paper will explicate the mapping process by introducing a software agent with extract, transform and load capabilities. The agent is able to read the data from the database and transform it to the ontology knowledge representation that can be effectively used by the embedded reasoning engine of the decision support system.

I. INTRODUCTION

Knowledge-based medical systems and applications can make clinical practices (semi)automated and in the same time improve their quality. The potential benefits are observable through structured and omnipresent standard formalized clinical practice guidelines that improve the quality of services they provide, enhance the quality of life for patients and reduce the costs both for the patients and the health service providers [4].

In modern knowledge engineering, building intelligent systems is not seen as the encoding of production rules or the creation of specific knowledge representations. Rather, the process is viewed as the design and assembly of domain ontologies, the knowledge bases that instantiate those ontologies, and domain-independent problem-solving methods. Given a task that must be automated, the challenge is to construct an appropriate problem-solving method, and to link that problem solver to an ontology that defines the relevant concepts in the application area. A lot of effort has already been made in developing such applications and systems [4].

The crucial part and the core of every such system is medical knowledge that has been in some way gathered, written and loaded into the system's knowledge base. Every decision and action that the platform performs is based on the knowledge from the knowledge base. The term factual knowledge represents the knowledge about patients (facts) that is extracted from the database and included in the reasoning process.

Nowadays every health institution keeps data about patients inside its local database. One must take this fact into consideration when building a large medical information and decision support platform. It is not possible to force the institution to adapt its complete infrastructure to the platform, rather the platform must adapt to the institution's infrastructure. Different institutions have different databases, which makes the process of that adaptation very difficult.

To ease that process of adaptation a novel KBDB-ETL (Knowledge Base from DataBase by Extraction, Transformation and Loading) component (agent) is introduced in this paper. That component should be able to read the data from the database and transform it to the form that the platform's reasoning agent can work with.

The organization of the paper is as follows. In the next section we describe the concept of the reasoning system using the knowledge represented in the ontological form. In the third section a brief description of related work is given. In the fourth section a detailed explication of the KBDB-ETL component at work is given. In the conclusion certain advantages and drawbacks of the implemented KBDB-ETL component are discussed.

II. ONTOLOGY BASED REASONING

Fig.1 illustrates the decision support system (DSS) experimentally realized for the knowledge based platform. In it the ontology has the central role of knowledge representation. It contains both domain concepts and factual knowledge about patient characteristics. Actionable knowledge is in the form of rules connecting domain concepts. The reasoner takes as its input ontological knowledge (both concept and factual) and rules. The results of reasoning are actionable suggestions in the ontological form, i.e. the patient with identification number *X* is an instance of the class for which therapy *Y* is suggested. In order for such reasoning to become possible, factual knowledge about real characteristics of the patient *X* has to be available in real time in the ontology (like patient *X* has property *Z*). It is the task of the component for developing factual knowledge to prepare this information based on data in the patient database.

The scenario of the DSS component at work is as follows: The DSS component is initially waiting in the idle state. The activity on the platform generates an event (1) that is served through the platforms middleware to the DSS interface. The event is forwarded to the DSS control unit (2). The control unit initiates (3) the KBDB-ETL unit to fetch (4, 5) the relevant data from the patient database (through the middleware interface). That data is transformed and loaded into ontology (6) and into the reasoner (7) as a set of facts. The reasoning process is performed and the knowledge generated in the reasoning process is loaded back into the ontology (8), and forwarded (9) to the ontology interpreter. Based on that knowledge the interpreter generates a platform action that is passed through the DSS interface (10) to the middleware (11). The procedure for the DSS component is finished and it returns to the idle state, waiting for the other possible event.

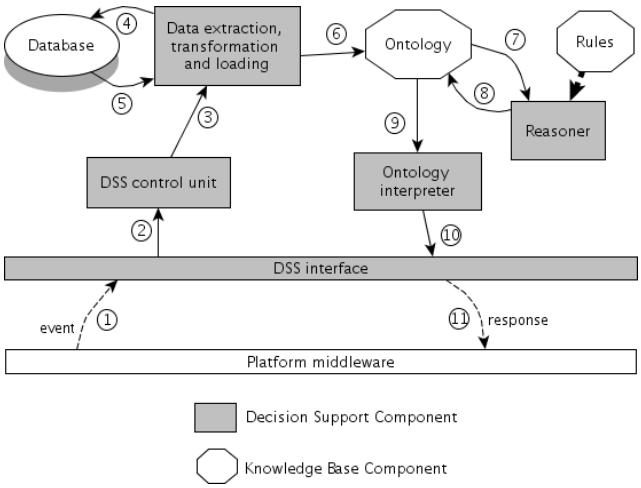

Fig. 1. The flow of data and a sequence of actions in a single DSS event handling session.

The experimentally realized KBDB-ETL system is developed in Java programming language, and supports two ontology types: the semantic web standard OWL ontology format (Jena API) and the Protégé Frames ontology language (Protégé API). The experimentally realized decision support system uses JESS as the reasoner (JESS API).

III. RELATED WORK

The described problem has been mostly addressed by the tools and methods from the area of the Semantic Web. The need for emerging the data stored in the database to the surface in the form of active and dynamic web pages is referenced by the term "deep web". The activity in that area has produced a number of various mapping tools.

The mappings in the intelligent information integration area are commonly built with ad-hoc software implementations (Observer [7], Pixel [5]) that are integrated within systems and built upon the specific system requirements.

Some effort has been made in developing heuristic methods that can make the mapping process (semi)automated (KAON-Reverse [8], MAPONTO [1]). The mappings are inferred on the basis of the previously made ones (semi-automatic), or on the basis of the syntactic/structural similarities (automatic).

Three approaches have been recognized in the process of database/ontology mapping [3]:

- Generation of a completely new ontology (limitations free) based on a given DB model.
- Description of a procedure how to fetch the data for each ontology item from the database.
- Mapping the complete existing database to the existing ontology (highest complexity).

Methods for mapping data from the database to the ontology may be compared by the following characteristics or capabilities:

- Expressiveness. Tools may vary in ability to define ontology classes, instances and relations;
- Parametrization of SQL and/or ontology statements. Allow data retrieval and ontology creation process to be dependent on the other data sources but the database itself;
- Handling missing database data (or ontology type). This issue must be handled in highly reliable applications like medical platforms;
- Glossary replacements. Allow translation of the database terminology to the ontology terminology by usage of the glossary;
- Ontology types supported (OWL, RDF(S), Protégé frames...);
- RDBMS supported (Oracle, MySQL, Informix...). The vast majority of the tools are RDBMS independent;
- Other characteristics: price, simplicity, human readability, maintenance, speed, ...

The most well known database/ontology mapping tools are D2R MAP with it's extended version eD2R [2], R2O [3], KAON [8] and MAPONTO [1].

eD2R [2] is a powerful XML-based declarative language. It describes mappings from database to RDFS/OWL ontologies (the only supported format). XML-tags in eD2R are describing the connection to the database, SQL queries and data mapping with very high expressiveness. However, parametrized ontology and SQL statements are not implemented; they are assumed to be implemented externally. eD2R is the representative example of the second above mentioned approach.

R2O [3] is a mapping language that maps the database data to the RDF(S) or OWL schema. It is conceived to cope with complex mappings regardless of the ontology/database structure similarities and differences. The syntax of the R2O language is very expressive, but also very complex and not intended to be read or created manually by humans (GUI tools are under development). R2O provides a series of other tasks like self verification, DB integrity verification etc. R2O fits the third above mentioned approach.

KAON [8] provides a visual mapping tool to define relations between database and ontology objects. Also, it provides KAON-Reverse [8], a heuristic method that makes the mapping process semi-automated by analyzing database structure, including relations, attributes, attribute types, primary keys, foreign keys/inclusion dependencies. It fits the third above mentioned approach.

Somewhat different approach is to describe the database structure using the Relational.OWL schema [6], and then to extract the data by using the SPARQL ontology query language [6].

Developing factual knowledge is the process that has to be repeated very often, practically for every reasoning circle. It means that the process must be fast and appropriate for on-line execution. Ad-hoc software implementations are not appropriate solutions because different implementations of the knowledge based platform may have to work with differently implemented patient database systems. The developing factual knowledge component is the only part of the DSS that has to be re-implemented for every possible database system and their upgrades. It means that some interpreter language that enables flexible but fast fetching, transforming and loading the data into ontology form is necessary. In the practical implementations, the second of three above described approaches best fits the needs of the

repetitive mapping. D2R is a typical representative of this approach. However, some adjustments and improvements can be made in order to make the process of integration easier. The KBDB-ETL component introduces the improvements that are described in the following chapter.

IV. KBDB-ETL COMPONENT AT WORK

The KBDB-ETL component has a built in interpreter language that enables fetching, transforming and loading the data. The program written in KBDB-ETL language consists of set of KBDB-ETL statements that are executed sequentially. Each statement consists of two parts:

- SQL statement
- Ontology statement

In the Fig.2. the role of the KBDB-ETL statement is depicted. The SQL statement is updated according to the data from the external sources and the glossary. Then it is executed and the data from the database is fetched. The ontology statement is updated according to the data from the database, the data from the external sources and the glossary data. Ontology statement is "executed" and the results are stored in the ontology.

A. Fetching data from the database

Each SQL statement fetches the data in a form of a table, where each row represents a single result set.

TABLE I
DATA FETCHED FROM THE DATABASE

Result Set	PatientID	FirstName	LastName	Sex	DOB	...
1	2045	Jonh	Dillinger	M	25.12.1939	
2	3046	Lorelai	Gilmoure	F	31.1.1914	
3	10002	Joel	Fleischman	M	12.1.1946	
...						

The SQL syntax alone is very expressive; it provides a powerful tool to extract very specific data items from the database with a little effort. Prior to the execution, the SQL statement is being updated with the glossary replacements and the event parameters as described later in the text.

B. Storing the data in the ontology

The ontology statement is executed once for each result set fetched from the database. The ontology statement has a following syntax:

```
[(Class_1 Individual_1
   [(Property  Class_2 Individual_2)]*
)]*
```

The square brackets paired with "*" sign denote that the expression enclosed may be repeated zero or more times. Here are the examples of valid ontology statements:

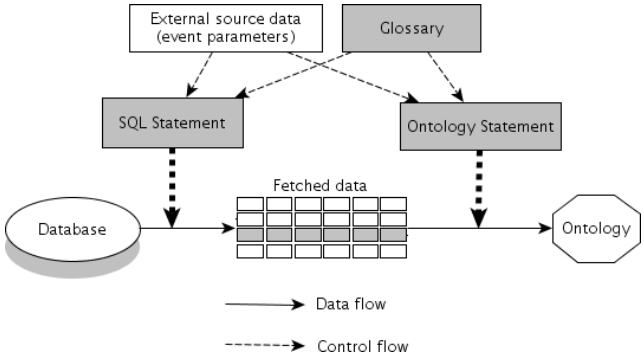

Fig. 2. Process of transformation from the database data to the ontology factual knowledge.

```
(Drug Diuretic)
```

Ontology will be updated with new individual named *Diuretic* of a class named *Drug*. If the ontology already contains an individual with the same class and individual name it will be reused. Since in the ontology statement there are no denoted individual properties none will be added.

```
(Test Echo_Results
  (LVEF_value Integer 43)
)
```

Individual named *Echo_Results* of a class *Test* is found/created in the ontology and integer value "43" is added as a property value to its property named *LVEF_value*.

```
(Patient Lorelai
  (has_Symptom Symptom Pulmonary_Oedema)
  (has_Symptom Symptom Peripheral_Oedema)
)
```

Individuals *Lorelai*, *Pulmonary_Oedema* and *Peripheral_Oedema* of the corresponding classes are found/created. Symptoms *Pulmonary_Oedema* and *Peripheral_Oedema* are added as a property values in patients property slot *has_Symptom* for the individual *Lorelai* (property *has_Symptom* must be configured to allow multiple values). If slot allows only a single value every time the new value is added, the old one will be erased.

If individual with the same name already exists in the ontology it will be reused. That way each individual can be created in any statement and edited from one or more of the following statements.

C. References to the result set (operator "!")

To insert data fetched from the database we use the prefix operator "!" paired with the result set column name. If any word in the ontology statement starts with this prefix it is being replaced with the corresponding value found in the current result set. Here is an example:

```
(Patient !PatientID
  (first_name String !FirstName)
  (date_of_birth Date !DOB)
)
```

For the data shown in the Table I. this ontology statement is evaluated to:

```
(Patient 2045
  (first_name String John)
  (date_of_birth Date 25.12.1939)
)
(Patient 3046
  (first_name String Lorelai)
  (date_of_birth Date 31.1.1914)
)
(Patient 10002
  (first_name String Joel)
  (date_of_birth Date 12.1.1946)
)
```

In this example each result set produces single ontology statement. For the first result set *!PatientId*, *!FirstName* and *!DOB* are evaluated to *2045, John* and *25.12.1939* respectively.

D. Glossary replacements (operator "@")

In general case the terminology used in the database does not match the one used in the ontology. This problem is handled by the use of glossary replacements. Each term that occurs in the ontology statement with the prefix "@" is replaced with its matched pair in the glossary. Also, it is possible to define more than one glossary in the system; but in every moment only one glossary is marked as active. This way each KBDB-ETL statement can have it's own defined glossary replacements.

Here is an example:

```
glossary symptoms.txt

// symptoms.txt listing
//  LowLVEF LVEF_value_under_45
//  HighTemperature Fever

(Patient 2045
  (hasSymptom Symptom @LowLVEF)
)
```

Keyword *glossary* activates the glossary file "symptoms.txt". The ontology statement is evaluated to:

```
(Patient 2045
  (hasSymptom Symptom LVEF_value_under_45)
)
```

If there is no match for the word in the active glossary it is left intact.

E. Parameters (operator "$")

The DSS component has to be able to handle a series of various tasks. From the DSS point of view, the tasks are described by the type of the event that occurred and the data in the database. Also, event can carry some attached information that describe the details of the situation that caused the event. To handle this kind of problem KBDB-ETL statements are parametrized. Every word in KBDB-ETL statement that has a prefix "$" is replaced by its parameter value. Here is an example:

```
// Parameters extracted from the event:
//  type      test_performed
//  TestID    30
//  PatientID  2045

(Patient $PatientID
  (performedTest Test $TestID)
)
```

This statement is evaluated to:

```
(Patient 2045
  (performedTest Test 30)
)
```

The other mapping tools are not handling the parameters within the mapping component, they assume that this is implemented externally. Introducing the parameters into the mapping component allows the mapping process to be guided by the data from the external sources. This largely eases the implementation of the system that is using the mapping tool, with minor changes in the mapping language syntax. It also supports the implementation of the ontology updating process within the mapping component. The maintenance of the mapping process is eased, since it can be performed without interfering into the system's source code.

F. Handling missing values

If some values are missing in the result set they can be disregarded or replaced with a default value. Here is an example:

```
(Patient !PatientID
  (first_name String !first_name)
  (height Float !height,0)
)
```

If in the result set there is no value for column *first_name* the corresponding property will be disregarded. If a missing value is in the *height* result set column, it will be replaced with its substitute default value which is in this case *0* (separated by comma). However, if value for the column *PatientId* is missing, the complete ontology statement must be disregarded, since an individual with no name cannot be found/created and no default value is offered.

G. Combining operators

The precedence of parameters in statement execution is from the right to left. Here is an example of combining the operators to extract, transform and load the data from the database to the ontology:

```
// Glossary:
//  LowLVEF LVEF_value_under_45
//  HighTemperature Fever
//  EchoTest Echocardiogram
```

```
// Parameters extracted from the event:
//   type      test_performed
//   TestID    30
//   PatientID  2045

// SQL statement:
SELECT TestName FROM Test WHERE testId = $TestID

// Ontology statement:
(Patient $PatientID
  (performedTest Test @!TestName)
)
```

In the SQL statement *$TestID* is evaluated to *30* based on the parameters extracted from the event. For example, let *EchoTest* be the only value fetched by this statement.

In the ontology statement *$PatientID* is evaluated to *2045* based on the parameters extracted from the event. *@!TestName* is first evaluated to *@EchoTest* based on the fetched value from the SQL statement. *@EchoTest* is evaluated to *Echocardiogram* based on the glossary match. Therefore, the final statement is:

```
(Patient 2045
  (performedTest Test Echocardiogram)
)
```

H. Ontology updating

To denote which KBDB-ETL statement should be executed at which event one can use the *update_on* keyword. Here is an example:

```
update_on ((type==test_performed)&&(TestID==30))

SELECT TestName FROM Test WHERE testId = $TestID
(Patient $PatientID
  (performedTest Test !TestName)
)
```

All KBDB-ETL statements after the *update_on* keyword will be executed if the corresponding boolean expression is evaluated to be true. The variables in boolean expression (*type* and *TestID*) are the ones delivered with the event (external sources). A good practice in system integration would be triggering event on every database change and adding to the event the information about which database table/record has been changed/added/deleted.

If the variable used in boolean expression is not denoted in the event, the complete expression is evaluated to be false. The syntax in boolean expression is similar to Java programming language.

V. CONCLUSION

Developing factual knowledge is one of the principal tasks of the reasoning process based on ontological knowledge representation. We have demonstrated that the KBDB-ETL approach with its simple interpreter language is able to resolve some of the drawbacks of the similar existing mapping tools. The distinguishing feature is that the component allows the mapping process to be guided and controlled by the external data sources. That is accomplished by parameterized SQL queries and ontology statements. The syntax of the KBDB-ETL language is simple due to the assumed transformation from the strictly defined database structure to the strictly defined ontology structure.

Although the simplicity of the KBDB-ETL approach might suggest that also some more complex relations among data contained in the database could be detected and manipulated by the KBDB-ETL component, this is assumed to be a bad design practice. It would be better to perform numerically complex computations only once in preprocessing part. However, practice indicates that the need for complex calculations is rather rare in this kind of systems. Low and medium complex computations can be expressed by using the integrated SQL statement syntax, but it is recommended to encode as much as possible of this kind of knowledge into the knowledge base system component. Knowledge base should contain as much domain knowledge as possible.

The KBDB-ETL component is conceived on the need for manipulating data within the medical platform environment but it can be easily integrated in any application that has the need for database/ontology mapping.

The problems and prospecting work concerning this configuration are:

- Setting up the KBDB-ETL component and the ontology interpreter component demands some expert knowledge (“knowledge integration problem”);
- If it is known what data has changed in the database allow that information to make the reasoning process faster.

ACKNOWLEDGEMENT

This research work is supported by the European Community, under the Sixth Framework Programme, Information Society Technology – ICT for Health, within the STREP project “HEARTFAID: a Knowledge based Platform of Services for supporting Medical-Clinical Management of the Heart Failure within the Elderly Population”, and Croatian Ministry of Science, Education and Sport project "Machine Learning Algorithms and their Application".

REFERENCES

[1] An Y, Borgida A, Mylopoulos J: *Refining the Semantic Mappings from Relational Tables to Ontologies*. In Proceedings of 2nd International workshop on Semantic Web and Databases (SWDB'04) in conjunction with Very Large Databases (VLDB'04), p. 84-90.

[2] Barrasa J, Corcho O, Gomez-Perez A: *Fund Finder Wrapper: A case study of database-to-ontology mapping*. Semantic Integration Workshop, SI-2003, p. 9-15.

[3] Barrasa J, Gomez-Perez A: *Upgrading Relational Legacy Data to the Semantic Web.* Proceedings of the 15th International Conference on World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06, p. 1069-1070.

[4] de Clercq A., Blom J.A., Korsten H.M., Hasman A.: *Approaches for creating computer-interpretable guidelines*

that facilitate decision support. Artificial Intelligence in Medicine 31(1):1-27.

[5] Goasdoue F, Lattes V, Rousset M: *The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project*. International Journal of Cooperative Information Systems, 7:127-140.

[6] Laborda C, Conrad S: Relational.OWL - *A Data and Schema Representation Format Based on OWL*. Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), 43:89-96.

[7] Mena E, Kashyap V, Sheth A, Illarramendi A: *OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies*. International journal on Distributed And Parallel Databases (DAPD), 8(2):223-272.

[8] Stojanovic N, Stojanovic Lj, Volz R: *A reverse engineering approach for migrating data-intensive web sites to the Semantic Web*. Intelligent Information Processing 2002 (IIP-2002, Part of the IFIP World Computer Congress WCC2002), p. 141-154.