An ontology-driven toolset for fast prototyping of medical data processing systems

Sergey Lebedev, Nataly Zhukova, Alexander Vodyaho, Dmitry Kurapeev and Mikhail Lushnov

***Abstract*—**Nowadays data processing systems are widely used by researchers for solving different problems in diverse subject domains including medicine. Usage of these systems encounters three problems. The first problem is a necessity to rebuild the system as often as the subject domain changes. The second problem is a necessity to integrate obtained results into the information system that is used by regular specialists. The third problem is that software development engineers usually do not appear to be specialists in the subject domain. All mentioned problems are highly significant to the medical domain. In the paper, an ontology-driven toolset for fast prototyping of medical data processing system is proposed. The toolset is based on the approach that defines a transition from the domain model to the software system prototype. The proposed approach is based on ontology-driven design and development, automation, and component reuse. The toolset is evaluated on a case study from medical domain.

***Keywords*—** Medical data processing, ontology-driven toolset, situation assessment.

I. Introduction

NOWADAYS one can observe the following situation: a number of projects devoted to medical data analysis have been implemented and interesting results have been received, e.g. [1], [2]. The results gained and evaluated by the researchers should be integrated into medical data processing systems to be used by practitioners. In other words, there is a necessity to somehow integrate new domain knowledge into existing software systems. But the integration process is obstructed by a conceptual gap between medicine experts and software specialists.

There are also two complicating factors. The first one is that the researchers periodically renew a medical knowledge. The ideas of continuing medical knowledge renovation are presented in a concept of evidential medicine [3]. So, to support this activity it is necessary to have a software system that can be easily adapted to changes in the medical domain.

The second one is that there are a lot of different medical realms and corresponding institutes. So, it seems reasonable to have a software system that can be "tuned" on different domains.

One can conclude that the medical software highly depends on the diversified dynamically evolving medical knowledge. This justifies a need for a solution that will simplify the transition from the medical domain description to the software system and so will help to close the said gap.

In the paper, a model-driven approach for fast prototyping of medical data processing system is described. The approach allows the creation of the system based on the formal description of the domain. Besides models, the approach also exploits such widespread software engineering techniques as automation and component reuse. The approach was implemented as a Java-based toolset and was evaluated on a medicine scenario.

The paper is organized in the following. Section II represents known works in the related domain. Section III describes the proposed approach, the technology for fast prototyping and elements of the proposed toolset. In Section IV the proposed approach and the toolset are evaluated on the medical case study. Section V discusses possible future works.

II. Related Works

Wanted software system must allow manipulating medical domain objects and relations between them and provide means to integrate separate objects into an integrated model. The model will allow such high-level activities as forecasting and decision making.

This manipulation process is known as a situation assessment (SA). There are some models that include a description of this process: JDL data fusion model [4], Observe-Orient-Decide-Act Loop model [5], Situation Awareness model [6]. A detailed review of these models one can find, for example, in [7]. But listed models are very abstract so formalization and implementation are needed.

In [8] an application of JDL model for processing medical data is described. But the authors do not provide any means for adaptation of the proposed approach to the changes of the domain model.

So, one faced with the problem of design and development of SA system for the medical domain.

Model-driven development [9] can be considered as an effective approach for minimizing the gap between a subject domain and a software development process. Traditional approaches such as UML-based approach are not very effective because UML is too tightly coupled with the code. More effective approach for building applications is an ontology based development [10].

The main advantage of the ontology-driven approach is that, on the one hand, the ontology can be used for the formal

The research was supported by Russian Science Foundation (project No. 17-15-01177).

representation of the subject domain in terms that are familiar to an expert. On the other hand, the formal ontology can be processed with computers and translated to a program code.

There are a number of works that describe the usage of ontologies for SA construction [11]-[13]. In these works, ontologies are used for representing of a subject domain and to conduct logical reasoning to find new objects and relations. So the approaches presented in these works are limited at most to logical means.

So there is a task to build a toolset that will simplify a transition from domain ontology to SA software system and that will utilize general programming language flexibility.

III. Approach and Toolset for SA Fast Prototyping

A. Ontology Technologies and Instruments

Ontology can be seen as a formal description of some domain. Ontologies are dynamically evolved in the realm of Semantic Web. In this area, ontologies are represented with OWL [14] languages. OWL language allows describing domain in the form of triples: subject-predicate-object. Also, OWL language is based on the description logic that can be used for inferring some additional information. OWL ontologies can be joined with the help of an import. Ontology can be split into TBox (terminological component) and ABox (assertion component). TBox includes classes and properties (binary relations) to represent some common information about the domain. ABox includes instances of classes that describe some concrete part of the domain.

To query data from OWL ontology SPARQL [14] language is used. This language is to some extent similar to SQL language. As SQL languages SPARQL also allows data construction queries. Typical SPARQL query includes two parts: "action part" that describes data to be extracted or constructed, and conditional ("where") part that describes templates for triple matching.

Also, there is SPIN [15] language that allows representing SPARQL queries in the form of OWL triples. With the help of this language, it is possible to link SPARQL queries to ontology and use them while inference process.

The Sematic Web domain provides a plethora of different instruments and libraries to work with ontologies. For the proposed toolset the following instruments and libraries are used. As an ontology editor, TopBraid Composer Free Edition [16] is used. The editor is implemented as an Eclipse plugin. It allows linking of SPARQL queries to ontology classes. SPIN language is used under the hood. The editor also includes inference engine that can be used to run these queries in a cyclic manner.

On a program level, Jena [17] and TopBraid APIs [18] are used to work with ontologies. The APIs provide methods for loading ontologies, making different SPARQL queries, modifying ontologies and exporting them to a file.

For ontology visualization Ontodia [19] web application is used. To use this instrument it is necessary to set up SPARQL-endpoint. The endpoint can be viewed as a web service that receives SPARQL-request and answers with data from the ontology. As SPARQL-endpoint Blazegrah graph database is used. Blazegraph [20] provides two types of interfaces to work with the underlying database: the user-oriented web interface and program API. To simplify usage of these services they are encapsulated in Docker [21] containers.

Wide support of ontologies with different instruments substantiates its usage for the proposed model-driven approach.

B. SA Calculation Process Model

SA can be defined as the following calculation process – see Fig. 1, where $Node_{i,j}$ denotes the j-th calculation node of i-th calculation level and $e_{i,j}$ denotes element calculated by j-th node on the i-th level. As SA deals with calculation of objects and relations:

$$e \in E,\ E = O \bigcup R,$$

where O is a set of objects, and R is a set of relations.

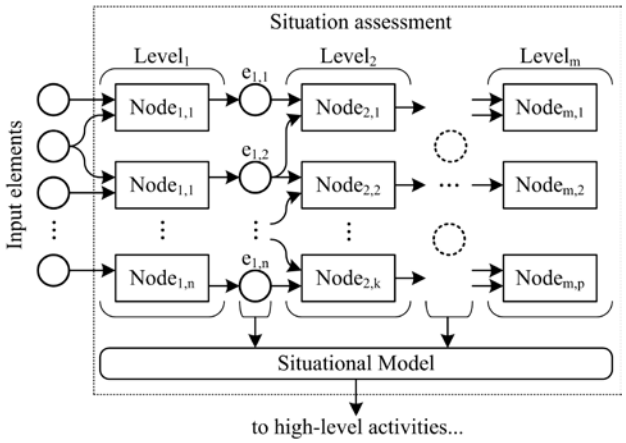

Fig. 1 SA calculation process model

SA calculation process has a hierarchical structure. The whole process is split into a set of calculation nodes scattered over a set of calculation levels. Each node implements a function for calculation of an element. The input of the function is a set of previously calculated or input elements. The output of the function is used by functions of higher node: the higher nodes depend on the lower ones. As it can be seen the unknown elements are calculated on the base of the known one – previously calculated or input elements. Input elements come from the external environment (e.g. from a user of the system). A situational model represents integrated high-level fragment of the domain and is built upon the calculated elements. This model can be used for high-level activities, such as decision making, forecasting and so on.

The represented calculation process allows clarifying the declared task: to implement the transition from the domain model to SA software system it is necessary to construct an intermediate model of SA calculation process that fixes calculation dependencies among domain elements.

C. Proposed Approach

To solve the clarified task the authors proposed the following approach – see Fig. 2, where solid arrows denote transitions between approach steps (numbered rounded rectangles), dotted arrows denote some information artifacts (rectangles) usage, dotted lines with a solid circle denote the fact that an artifact produced in the previous step is passed to the next step, circled R denotes reused components, circled A – automated components, circled M – model-based components, DISA stands for Domain-Independent Situation Assessment, DSSA – Domain-Specific Situation Assessment, DI – Domain-Independent.

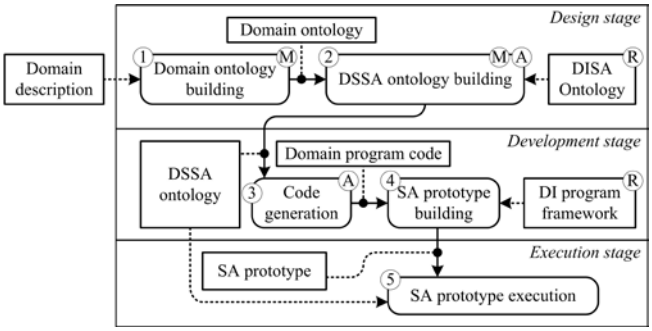

Fig. 2 The approach

At the first step, the domain ontology is built based on the domain description.

On the second step, DSSA ontology is built. This ontology formalizes the calculation process represented in Fig. 1 for the given domain. To support this step the authors created DISA ontology (see paragraph III.D.). This ontology includes classes and properties that are used to construct SA calculation process for the given domain. Also, it includes a number of rules that allows automation of building of DSSA ontology. At this step one makes a transition from the domain model to the model of SA calculation process.

On the third step, the domain-specific code is automatically generated based on the DSSA ontology. The generated code represents the same calculation process (see Fig. 1) with the help of programming language classes. To support this step the authors implemented an algorithm that generates Java code from DSSA ontology (see paragraph III.F.).

On the fourth step, the generated code is integrated into the DI programming framework which implements a common logic of SA calculation process. After the integration, a programmer should implement all functions of calculation nodes. To support this step the authors created a Java-based framework – see paragraph III.G. As a result of this step, the prototype of SA software system is created.

On the last step, the created prototype is used to process some data. On this step, DSSA ontology is used to dynamically change the program structure of the calculation process (see paragraph III.G.).

It can be seen, that the presented approach uses such techniques as model-driven design and development, component reuse, and automation. The approach is implemented by the authors as a toolset. This toolset includes: a) a number of components that can be reused for any domain: DISA ontology and Java-based DI program framework; b) the implementation of the algorithm for ontology-based code generation; c) a method that describes how to apply the approach to different domains (see paragraph III.E.).

D. DISA Ontology

As it was said the proposed DISA ontology is used to automate DSSA ontology creation. In other words, DISA ontology helps to make a transition from the domain model to SA calculation process model.

The transition includes two steps. The first one is a fixing of computation dependencies among domain ontology elements. The second one is a building of the SA calculation process model out of these computation dependencies.

To support the first step DISA ontology includes two types of *computability properties* (as a reminder, ontology property represents a binary relation) – *generative* and *associative* one. A generative property says that the property itself and its range computationally depend on its domain. Whereas an associative property says that the property itself computationally depends on its domain and range. In other words, mapping a domain property on the computability one makes it possible to interpret the former as a function. To map properties a user should use OWL property inheritance mechanism.

To fix computation dependencies in an explicit form DISA ontology includes *Computation Unit* class. An instance of this class fixes computation dependency between base elements (function's arguments) and derived elements (function's value). DISA ontology also includes a set of rules that automatically generate instances of this class for each computability property. Fig. 3 illustrates the described step with an abstract example. Within this example "has" property can be interpreted as the following function: *f*(*Patient*) = <*has*, *Disease*>.

On the second step, it is necessary to define an appropriate place for each computability dependency represented by an instance of Computation Unit class within the overall SA calculation process. To support this step DISA ontology includes two classes *Node*, *Level*, and a number of properties that define relations between these classes. E.g. there is a "dependOn" property that allows fixing computational dependence between two nodes. Also, DISA ontology includes rules that automatically generate instances of Node and Level classes out of Computation Unit class instances to form SA calculation process model (see Fig. 3)

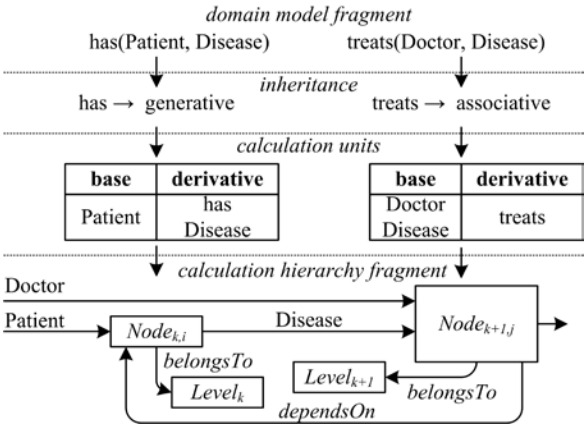

Fig. 3 Mappings

Rules are implemented as SPARQL CONSTRUCT-queries that are linked to DISA ontology with the help of SPIN language. Queries are run by the inference engine embedded into the TopBraid Editor. The engine executes queries in a cyclic mode. Inference continues while there are some CONSTRUCT-queries that can be run. An example query is represented in Fig. 4. The query implements the generation of Computation Unit class instances for domain properties inherited from the generative computability property.

```
CONSTRUCT {
   ?cu a :GenerativePlainCU .
   ?cu :hasBase ?domain .
   ?cu :hasDerivative ?r .
   ?cu :hasDerivative ?range .
}
WHERE {
   ?r (rdfs:subPropertyOf)+ :generative .
   ?r rdfs:domain ?domain .
   ?r rdfs:range ?range .
   BIND (afn:namespace(?this) AS ?ns) .
   BIND (IRI(fn:concat(?ns, "cu_", afn:localname(?r))) AS ?cu) .
}
```

Fig. 4 An example of SPARQL rule

E. DSSA Ontology Construction Method

The authors suggest a method that defines how to build DSSA ontology for some subject domain on the base of DISA Ontology. The method includes the following steps:

1) Build a domain ontology;
2) Import DISA Ontology into the domain ontology;
3) Inherit domain properties from one of the computability property of DISA ontology;
4) Run rules to automatically construct instances of Computation Unit class;
5) Choose classes which instances will be received from the external environment;
6) Run rules to automatically construct SA calculation process model from instances of Node and Level classes.

As a result, the SA calculation process model (ontology) is received. It is worth of mentioning that the model is constructed taking into account what can be constructed in principle.

F. Code Generation

The code is automatically generated on the base of the DSSA Ontology received in the previous step. As a result, one gets program structure of SA calculation process with stubs for implementing functions of separate calculation nodes.

The algorithm of code generation is implemented as a cyclic process – see Fig. 5.

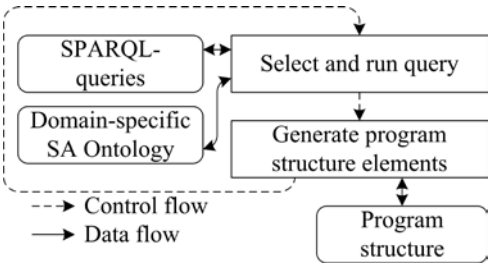

Fig. 5 Code generation algorithm

An iteration of the algorithm comprises the following steps:

1) Receive a fragment of DSSA ontology with the help SPARQL SELECT-query;
2) Generate program structure fragment for the received ontological elements. To generate Java structures CodeModel library [22] is used. While program elements generation previously constructed program structure can be used.

The order of iterations is defined by logical dependencies between program components. For example, before generating program classes for nodes it is necessary to generate classes for the domain objects and properties.

It is necessary to mention that the code generation algorithm does not depend on the domain model because SPARQL-queries are composed of DISA ontology elements.

In order to build a prototype from the generated code, it is necessary to integrate the generated code into the domain-independent program framework of SA system. To make this integration possible generated elements of the SA program structure are inherited from abstract classes of the framework.

On the final step, a programmer overrides stubs of program classes of calculation nodes.

G. SA Framework

The architecture and basic cycle of the proposed domain-independent program framework are shown in Fig. 6. The framework is realized with Java version 1.8.

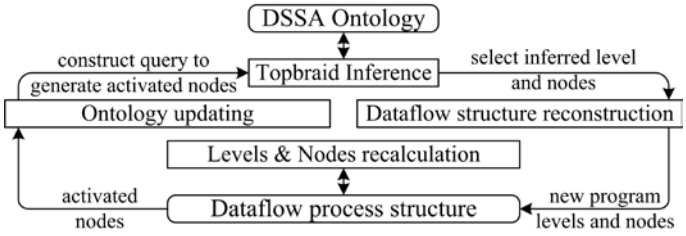

Fig. 6 Domain-independent framework architecture

The Fig. 6 shows usage of DSSA ontology in the process of

framework operation. Ontology is used for the building of the actual program structure of SA calculation process. In contrast to the development stage here the program structure is defined by the list of domain elements which have been already calculated (and not by the set of elements that can be generated in principle). For example, the node of the second level can be generated only when the object data have been received from the external system. It is worth mentioning that for access to DSSA ontology only DISA ontology concepts are used and that is why the framework itself does not depend on the domain.

The module "Dataflow structure reconstruction" updates program structure of SA process based on newly inferred ontological levels and nodes. The module "Levels & Nodes recalculation" updates situational model using current SA calculation process program structure. For this purpose, control is passed from level to level and from node to node (see Fig. 1). On each node, the user-defined function is recalculated.

The nodes, which have calculated a value for the first time, are called activated. A set of activated nodes is used by the "Ontology updating" module. The module comprises SPARQL CONSTRUCT-query that add activated node into the ontological counterpart of the program structure of SA calculation process. Then TopBraid Inference engine is used to construct a set of nodes that depend on the nodes that have been just activated and possibly on those that were activated on previous steps. So that the usage of the DSSA ontology allows not implementing the described mechanism and also allows adapting SA calculation process structure to the current environment state.

IV. Evaluation of the Proposed Approach and the Toolset

A. Case Study Description

The suggested approach was evaluated on a case study from medical domain. The case study can be formulated in the following way.

The SA system receives a stream of urine analyses results. Each analysis is described with a set of parameters which include both information about patient and parameter values (data). For each patient, a subset of parameter values (a batch) is chosen. On the base of the batch with the help of linear regression model, one finds if the given categorical parameter (independent variable) can be considered as a predictor for the chosen numerical parameters (dependent variable). As a result, one should receive a set of group-specific predictors for each patient. These dependencies are fixed in a graph like structure and used as a situation model.

The case study was evaluated on the real data set of urine analyses that includes more than13000 records for more than 4000 patients. The set was provided by Federal Almazov North-West Medical Research Center (or simply Almazov center) [23] and is not freely available.

To model SA calculation process the whole set of records is divided into small portions with 10 records in each. The input of each portion is followed by SA operation cycle.

B. Medical Data Processing System Prototype

1) DSSA Ontology Construction

In accordance with the suggested approach on the first step, the domain ontology was built for the described case study – the case study ontology. The fragment of such ontology is presented in Fig. 7 where rectangles with **C** letter designate ontology classes and **OP** combination designates ontology properties. To visualize ontology fragments Ontodia web service was used.

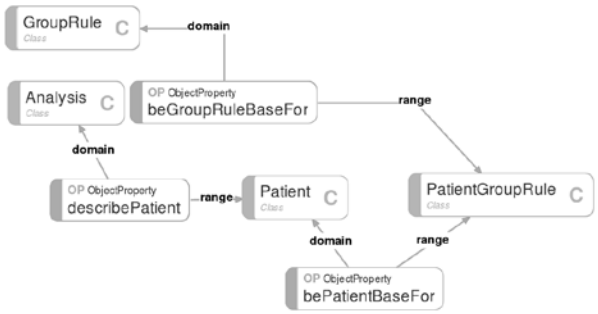

Fig. 7 Case study ontology

This ontology fragment is used for describing next steps.

Then DISA Ontology was imported into the case study ontology. A number of case study ontology properties were inherited from the computability one (generative or associative).

On the next step, SPARQL-queries were executed to automatically generate Commutation Unit instances which fixed computation dependencies among domain elements. These instances for the fragment shown in Fig. 7 are presented in Fig. 8. Calculation dependencies are represented with rectangles that include strings that start with "cu_": "cu_GroupRule", "cu_PatientGroupRule" and "cu_descibePatient".

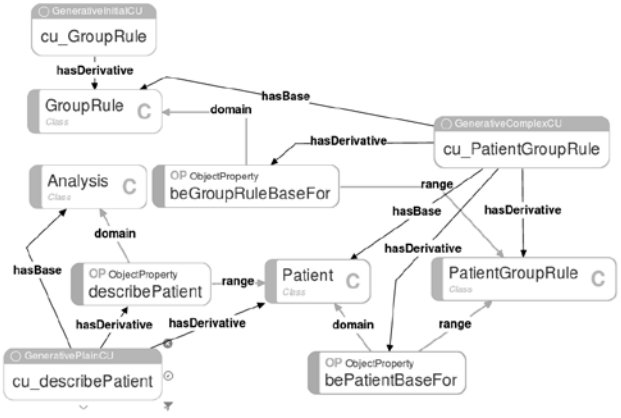

Fig. 8 Calculation dependencies

On the next step, a set of classes was chosen to be inherited from "Percepted" class. In the previous figures, it includes "Analysis" and "GroupRule" classes.

The previous step allowed running of SPARQL-queries to

construct SA calculation process ontology. The fragment of this ontology is shown in Fig. 9. On the figure nodes and levels are represented with rectangles with the corresponding titles. It can be seen from the figure that "node_cu_PatienGroupRule" depends on both "node_cu_GroupRule" and on "node_cu_describePatient", and is linked to the third level.

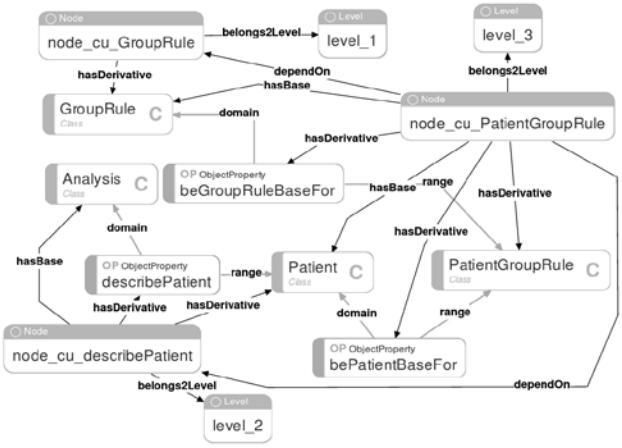

Fig. 9 SA calculation process ontology fragment

The generated calculation process model is presented in Table I.

Table I Dataflow ontology description

Level #	Node #	Depends on Node	Function Description
1	1	-	Get analyses from the external system
1	2	-	Get batch forming rule
1	3	-	Get graph forming rule
2	1	1.1	Extract patient data from analyses
2	2	1.1	Extract separate parameters from analyses
3	1	1.2, 2.1	Specify group rule with a patient data and get a patient group rule
4	1	2.2, 3.1	Form a batch from separate parameters based on the patient group rule and links it with the patient and analyses
5	1	1.2, 4.1	Construct a graph fragment from the batch based on the graph rule using linear regression and links it with the patient and the batch

The batch forming rule is used to limit a batch relatively to some group of patients. The rule represents a string containing a list of parameter names describing a patient. For the evaluated case study, gender and diagnosis parameters were used. The rule comprises system input data.

The batch forming rule is transformed into the patient group rule at the node 3.1. This transformation substitute parameter names with the corresponding values for a given patient. So, one can form a batch for a group of patients with the same gender and diagnosis: e.g., a group for male patients with I20.0 diagnosis (according to ICD-10 classification [24]).

The graph forming rule defines a dependent-independent variable pair. This rule defines for which analysis parameters a linear regression model should be used. For the case study, the urine color was used as an independent variable and such parameters as ketone bodies, red blood cells, white blood cells and etc. are used as dependent parameters.

2) Prototype Construction

On the base of generated SA calculation process ontology, the executable code was generated with the help of the algorithm implemented by the authors. Then the code was built in the proposed framework. On the final step, all node functions were implemented. As a result, SA prototype presented in Fig. 10 was received.

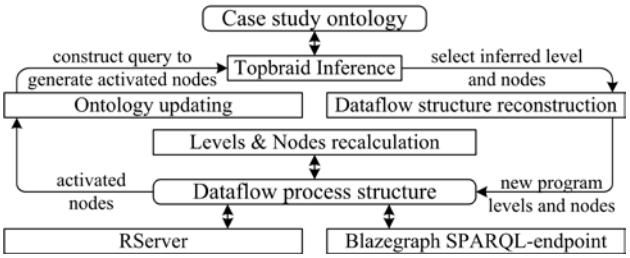

Fig. 10 Case study SA prototype

For the prototype implementation, two external services were used. The first one is an R language interpreter which is realized on the base of RSever. The service was used for the calculation of linear regression model.

The second one is a service which is used for storage and fetching data presented as RDF graph. For this purpose, Blazegraph graph database was used. Such a solution allows replacing data passing between nodes with notification passing. For data saving and fetching SPARQL-queries are used.

Interaction with described services was implemented within calculation nodes so that the framework itself was not modified.

C. Prototype Operation Results

The fragment graph constructed by SA prototype is presented in Fig. 11, where "ly" stays for light yellow color.

Fig. 11 Parameter dependency graph fragment

Received dependencies can be used for the construction of decision rules that may facilitate the practicing doctor making decisions process while treatment of a patient. For example, from Fig. 11 it can be seen that there is a dependence between the straw-yellow color and the parameter Bacteria for men diagnosed with I120.8. This indicates that if a patient has a straw-yellow color, then a doctor should pay attention to the fact that the patient may have an infectious pathology of the urinary tract.

Thus, with the application of the described prototype, the

evidence-based reasoning of the medical decisions is improved. A doctor has the ability to make a decision based on the intellectual processing of large data sets of tens of thousands of observations, which he cannot do manually.

D. Evaluation and Discussions

The number of automatically generated and reused components is represented in Table II.

The main goal of the proposed toolset is to speed up the process of SA creation for medical domain. One way to roughly estimate the speed up is to estimate time costs of automatically generated code as if it was created manually. To do this the Constructive Cost Model II model implemented as an online service was used[1]. The service accepts the total number of lines of codes as an input data. The total number of lines of automatically generated code amounted to 1677. As a result, the following estimations were obtained (with all drivers to be set on normal): effort = 5.2 person-months; schedule = 6.3 months.

Table II The number of automatically generated or reused components

Stage	Type	Volume (classes)
Framework	reused	40
Domain-specific SA ontology	Auto generated	32
Domain-specific code	Auto generated	92

The speed up was also gained thanks to the lifting of the abstraction level of the design stage up to the domain model engineering.

The constructed prototype can be tuned with the help of the batch and graph forming rules without ontology modification. Any more complex modification implies the case study ontology reconstruction. To mention just a few:

- new data source, e.g., blood analyses;
- new batch types, e.g., a filtered batch, a batch with interpolated data;
- parameter pairs interpretations and so on.

It is necessary to note that it seems obvious that usage of some ad hoc approaches may and will result in a more effective SA system from the performance point of view than in the case of proposed toolset usage. But authors suppose that on the stage of research and evaluation of some medical hypothesis it is more important to quickly build or rebuild (if necessary) a working prototype. After a thorough evaluation, the prototype can be integrated into the hospital medical information system with necessary revisions.

The main distinction of proposed approach and the toolset based on the approach from known systems is that it uses ontologies for design and development of SA system based on some general programming language and is not restricted to logical means.

V. Conclusion and Future Work

The proposed toolset can be considered as an effective instrument for building information systems particularly sensitive to the domain changes. It is based on the following rationales: 1) to close the gap between domain experts and software developers it is necessary to start from the domain description; 2) to be able to use this description programmatically it must be formalized and ontology is a good candidate for this task; 3) the toolset should preserve flexibility provided by a general programming language; 4) the main value will be gained if the toolset allows automation and domain-independent elements reuse rather than simply provides components to be manually programmed.

The suggested approach opens good perspectives for reuse of architectural knowledge. It can be used not only for medicine oriented systems development but for many other subject domains.

The suggested approach allows increasing the level of abstraction of code development stages up to the level of domain ontology engineering. Also, ontologies are used for automation of design and development stages of software engineering process. In combination, the gained advantages allow decreasing time and complexity of the design and development of SA system.

The main tasks for future development are the following: i) estimation of the possibility to work with big models, ii) evolving and evaluation of theoretical backgrounds, iii) testing approach on the problems from different subject domains.

[1] http://csse.usc.edu/tools/cocomoii.php

References

[1] P. Perner, R. Brause and H.-G. Holzhütter (Eds.), *Medical Data Analysis: 4th International Symposium, ISMDA 2003, Berlin, Germany, October 9-10, 2003, Proceedings*. Vol. 2868. Springer Science & Business Media, 2003.

[2] N. Maglaveras, I. Chouvarda, V. Koutkias and R. Brause (Eds.), *Biological and Medical Data Analysis: 7th International Symposium, ISBMDA 2006, Thessaloniki, Greece, December 7-8, 2006. Proceedings*, Vol. 4345, Springer, 2006.

[3] J.H. Howick, *The philosophy of evidence-based medicine*, John Wiley & Sons, 2011.

[4] E. Blasch, É. Bossé and D. A. Lambert, *High-level information fusion management and systems design*, Artech House, 2012.

[5] M. Nilsson, "Human decision making and information fusion: Extending the concept of decision support," 2007.

[6] M. R. Endsley and D. J. Garland, *Situation awareness analysis and measurement*, CRC Press, 2000.

[7] P.H. Foo and G. W. Ng, "High-level Information Fusion: An Overview," *J. Adv. Inf. Fusion* 8.1, 2013, pp. 33–72.

[8] M. Lushnov, V. Kudashov, A. Vodyaho, M. Lapaev, N. Zhukova and D. Korobov, "Medical Knowledge Representation for Evaluation of Patient's State Using Complex Indicators," *International Conference on Knowledge Engineering and the Semantic Web*, Springer International Publishing, 2016.

[9] O. Pastor and J. C. Molina, *Model-driven architecture in practice: a software production environment based on conceptual modeling*, Springer Science & Business Media, 2007.

[10] J. Z. Pan, S. Staab, U. Aßmann, J. Ebert and Y. Zhao (Eds.), *Ontology-driven software development*, Springer Science & Business Media, 2012.

[11] L. Li, W. G. Wu and N. Liu, "Ontology model for situation awareness of city tunnel traffic," *Applied Mechanics and Materials*, Vol. 347, Trans Tech Publications, 2013.

[12] N. Baumgartner, W. Gottesheim, S. Mitsch, W. Retschitzegger and W. Schwinger, "Improving situation awareness in traffic management," *Proc. Intl. Conf. on Very Large Data Bases*, 2010.

[13] N. Baumgartner, S. Mitsch, A. Müller, W. Retschitzegger, A. Salfinger and W. Schwinger, "A tour of BeAware–A situation awareness framework for control centers," *Information Fusion*, Vol. 20, 2014, pp. 155-173.
[14] D. Allemang and J. Hendler, *Semantic web for the working ontologist: effective modeling in RDFS and OWL*, Elsevier, 2011.
[15] SPARQL Inferencing Notation (SPIN). https://www.w3.org/Submission/spin-overview/. Accessed 28 November 2017
[16] TopQuadrant. http://www.topquadrant.com/. Accessed 28 November 2017
[17] Apache Jena. A free and open source Java framework for building Semantic Web and Linked Data applications. https://jena.apache.org/. Accessed 28 November 2017
[18] The TopBraid SPIN API. http://topbraid.org/spin/api/. Accessed 28 November 2017
[19] Ontodia. https://github.com/ontodia-org/ontodia. Accessed 28 November 2017
[20] Blazegraph graph database. https://www.blazegraph.com/. Accessed 28 November 2017
[21] Docker. https://www.docker.com/. Accessed 28 November 2017
[22] CodeModel project. https://javaee.github.io/jaxb-codemodel/. Accessed 28 November 2017
[23] Federal Almazov North-West Medical Research Centre. http://www.almazovcentre.ru/?lang=en. Accessed 28 November 2017
[24] ICD-10. http://apps.who.int/classifications/icd10/browse/2010/en. Accessed 28 November 2017