Appendix 2

National Research Program

"Cyber-physical systems, ontologies and biophotonics for safe&smart city and society" (SOPHIS)

Project No.2
"Ontology-based knowledge engineering technologies suitable for web environment"

Scientific report

Period 1 & 2

2016

CONTENT

Introduction

The goal of the project No.2 is to develop the scientific expertise of the next generation IT systems by researching and further developing novel competitive model-based information technologies and their applications in modern web environment and to transfer the created expertise and technologies to concrete domains of Latvia's economics, as well as introducing them into the higher education study process.

The main tasks of stages 1 and 2 are:

- Development of the theoretical background for the ontology- and web technology-based graphical ad-hoc query language.
- Development of ontology-, web technology- and controlled natural language-based fast ad-hoc query language that will be directly usable by domain experts (without involvement of the programmer)
- Approbation in medical domain (CCUH) of ontology-, web technology- and controlled natural language-based fast ad-hoc query language that will be directly usable by domain experts (without involvement of the programmer).
- Development of the theoretical background for the implementation of distributed ontology- and web technology-based controlled natural ad-hoc query language parallel execution.
- Development of the web technology-based tool building technologies and methods for modeling of complex, hard-to-formalize systems.
- Developing new methods for natural language parsing into the form of semantic graphs based on n-ary predicate approach (such as AMR, FrameNet, BabelNet) for information extraction from natural language texts (natural language understanding tasks)
- Further development of C6.0 classification algorithm and joining the international scientific research initiatives.
- Research of competitive technologies for semantic graph parsing based on SemEval-2015 competition, and integrating them in semantic analysis toolchain for Latvian language, as used in LETA and elsewhere.
- Development of methods, algorithms and their support software for analysis of knowledge structure models.
- Study of related works in the domain of structural compatibility between processes, enterprise architectures and knowledge, as well as development of initial draft for ideal linkage model
- Development of demonstration prototype for integration of semantic network services into e-logistics portal.
- The extension of I4S functionality to evaluate the functional state of complex industrial control systems on the basis of their automated analysis.
- Development of approaches and methods for identification of knowledge structure and process compatibility.
- Analysis of the related studies and researches that are necessary to define the basic steps of the Semantic Web service development methodology.
- Develop on data models and ontologies based methods for big data accessibility; propose new suitable for internet methods for data query and visualization.

- Development of technologies for large scale NoSQL data base exploration and visualization.
- Develop business process model usage in program runtime event analysis to increase information systems security level.
- Business process runtime verification.

The goal of this report is to describe the main scientific and practical results during the reporting period. The results will presented according to the above mentioned tasks, in some cases the tasks from stages 1 and 2 will be merged into one section.

The parts of scientific results which are adequately presented in the corresponding publications will not be described in detail in this report, direct links to the corresponding publications will be provided instead.

2.1. Ontology based tools for knowledge analysis and mining semantics of natural language

2.1.1. Development of the theoretical background for the ontology- and web technology-based graphical ad-hoc query language (1-st period)

1) How to depict a data ontology for it to be easily understandable by a domain expert;
2) How to use such ontology as a base, on which one can build easy-to-use query language that can be exploited by the domain expert directly (without involving a programmer);
3) How to implement such a language efficiently so that one can get answers to typical queries in time less than a second (on data volume of several GBs, e.g., Children's Clinical University Hospital's (CCUH) one year data).
For this purpose we have further developed the concept of granular data ontology that gives us a possibility to divide large volume data into separate pellets (which we call slices). We have demonstrated a way of slicing traditional data ontologies (ER models), offered an experimental graphical query language that is based on the concept of granularity and explored the performance of query execution that has been achieved thanks to granularity. We have also researched the topology of granular data ontologies – it turns out that in natural conditions an ontology is granular if and only if it is *a star ontology*. This topology gives us a possibility to create essentially more convenient query language for domain experts. The results have been described in J. Barzdins, E. Rencis and A. Sostaks, Fast Ad Hoc Queries Based on Data Ontologies. In: H.M. Haav, A. Kalja, T. Robal (Eds.), Frontiers of AI and Applications, Vol. 270, Databases and Information Systems VIII, IOS Press, pp. 43-56, 2014.

2.1.2. Development of the ontology- and web technology-based controlled natural ad-hoc query language which can be used directly by end-user (without involvement of the programmer) (1-st period)

The first and most essential result: we have found six controlled natural language query templates (supplemented with a formal concept of scalar expression) that cover practically all ad-hoc queries one can think of for needs of hospital management (we assume the managers have sufficient MS Excel skills). We have tested this hypothesis on real CCUH data (year 2014) and real queries that were needed to generate the review and analysis of year 2014 in one particular CCUH clinic (intensive care clinic). The experiment approved the hypothesis – 100% of necessary query coverage was achieved.

2.1.2.1. Introduction

The description of the query language proposed in this document is devoted to the main class of language and system users – domain experts that are not programmers. It therefore begins with a detailed explanation of underlying data ontology.

Analysis of current situation suggests that it is hopeless to try to develop an easily perceptible query language that can be used on arbitrary ontologies, because such language has not yet appeared after 30 years since the invention of relational databases. Therefore we introduce an important subset of data ontologies called the *Semistar ontologies.*

2.1.2.2. Semistar Data Ontologies

A typical example of a semistar ontology is depicted in Fig. 2.1.2.1. This is a simplified version of data ontology used in Riga Children's Clinical University Hospital (the actually used ontology consists of 25 classes and 142 attributes). Semistar ontologies are data ontologies whose graphical representation corresponds to a star-like structure (having no loops) with a restriction on multiplicities, such that all associations must have the multiplicity equal to 1 at the end of the association that is nearer to the star's center. Semistar ontologies have only one type of associations between basic classes – the "has" relation (e.g. Patient has HospitalEpisodes, HospitalEpisode has TreatmentWards, etc.). Besides basic classes, a semistar ontology can also contain other classes called the classifiers (depicted with white background in Fig. 2.1.2.1). Associations between basic classes and classifier classes are coded as attributes (e.g. familyDoctor: CPhysician).

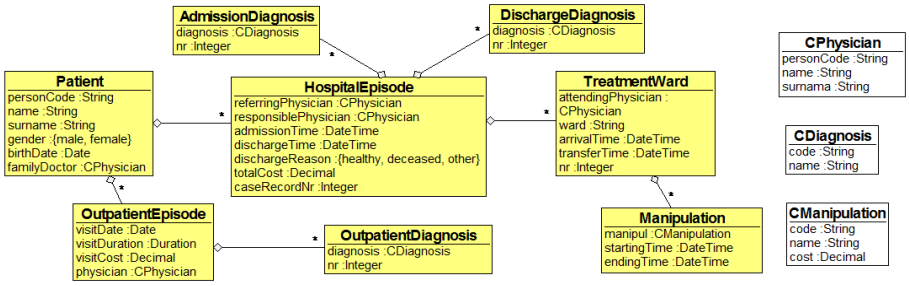

Fig. 2.1.2.1. Simplified semistar ontology used in Riga Children's Clinical University Hospital.

Semistar ontology is a practically important and expressive subset of all data ontologies, and practical use-cases often exploit exactly this type of ontologies. As can be seen in Fig. 2.1.2.1, hospital ontology viewed from patients' and physicians' point of view comes out to be a semistar ontology. Our experience shows that even in more general cases, when some ontology is not a semistar ontology, one can usually find an important subset of it to comply to principles of semistar ontology. We can always think of a semistar ontology as a subject-oriented ontology where the role of the subject can be performed by a patient (in case of medical domain), a customer (in case of some service domain), etc.

2.1.2.3. Attribute Expressions and Attribute Conditions

We allow attributes of basic classes to have two kinds of data types – the primitive types and the classifiers. We use the following predefined data types and operations:

- Integer (e.g. 75, -75), Real (e.g. 0.75, -75.0), operations: +, -, *, /;
- Boolean (true, false), operations: and, or, not;
- String (e.g. "abc"), operations: substring (e.g. "abcde".substring(2,3)="bc");
- Date (e.g. 2015.06.17), unary operations: year(), month(), day(), dayOfWeek(), binary operation: - (e.g. 2015.06.17-2015.05.12 = 1M5D);
- DateTime (e.g. 2015.06.17T10:45), unary operations: year(), month(), day(), hour(), minute(), second(), date(), binary operation: - (subtraction);
- Duration (e.g. 3Y4M5DT6H7M30.25S), unary operations: years(), months(), days(), hours(), minutes(), seconds().

If some attribute *a* has a classifier class as data type and this classifier class has some attribute *k*, then also *a.k* denotes a valid *attribute expression* and its data type will be that of attribute *k*. If *x* is an instance of some class, for which attribute *a* is defined, then also *x.a* (or *x.a.k*, if type of *a* is a classifier class) denotes a valid attribute expression. We can build more complex attribute expressions from simpler ones using the abovementioned operations allowed for the given data types. Some examples of attribute expressions: *personCode*, *x.personCode*, *x.familyDoctor.surname*, *x.admissionTime.month()*, *(dischargeTime-admissionTime).days()*, etc.

We can now compare two attribute expressions (or constants) using the comparison operators =, <>, <, <=, > and >= to obtain *attribute conditions*.

In attribute conditions we can use the equality (=) and non-equality (<>) comparison operators for any attribute expressions or constants, but the other four operators can be used only for expression or constants of data types Integer, Real, Date, DateTime, Duration and String.

In case of String expressions and constants there is one more comparison operator for matching substrings: ~. The attribute condition t1~t2 is true if and only if t2 is a substring of t1. For example, "abcdef"~"cd" is true, but "abcdef"~"cdd" is false. We can use a special symbol % to denote an arbitrary prefix of suffix in cases when we want to say particularly that the given string must start of end with the given substring. For example, "abcdef"~%"def" is true, because the first string ends with the other string (similarly, "abcdef"~"ab"% ir true), but "abcdef"~%"cd" is false.

Let us now inspect some examples of attribute conditions for the class "Patient": *personCode=250285-10507, personCode.substring(1,4)=2502, surname=Lapiņš, gender=female, familyDoctor.surname=Vanags, familyDoctor=nil* (meaning – family doctor does not exist), *familyDoctor<>nil, birthdate=1985.02.25, birthdate.year()=1985, surname~pi, surname~La%.*

Some more examples for the class "HospitalEpisode":
referringPhysician=familyDoctor, admissionTime=2015.10.25 16:30, referringPhysician.personCode=familyDoctor.personCode, dischargeTime-admissionTime>25d (meaning – 25 days), *admissionTime.date()-birthDate<=50d.*

Let us now assume that *AClass* is an arbitrary class of our ontology. We will from now on use a term "*AClass attribute condition*" to denote an attribute condition that can contain attributes of both *AClass* itself and its parents (we assume here that attribute names for parent and children classes are different). We cannot use attributes of *AClass* children classes in *AClass* attribute conditions, because every instance *x* of *AClass* can contain more that one instance of some children class, so we cannot use comparison operators that requires exactly one instance. We will be able to access the children of *AClass* by exploiting the concepts of EXISTS and NOTEXISTS.

2.1.2.4. Query Language: Basic Ideas

The query language we propose here is to be used for formulation of ad-hoc queries, meaning queries that can be formulated in one sentence (perhaps together with some subordinate clauses) in natural language, so that the end-user can still understand it very well. The language will work on semistar ontologies. The simplicity of the "has" relation is the main factor, which allows query language to be simple and similar to a natural language. It is therefore convenient to build queries in a controlled natural language. This feature allows the language to be easily perceptible by non-IT specialists.

Let us introduce an example query that will be exploited further in this section – *count Patients, who have at least one HospitalEpisode, which has Manipulation with manipul.code=02078.* This natural language sentence is understandable by the domain expert. Let us now inspect a bit more complicated query: *count Patients, who have at least one HospitalEpisode, which has at least one TreatmentWard, which has at least one Manipulation with manipul.code=02078.* This sentence may cause a certain ambiguity as it is not clear whether the asked *Manipulation* refers to *HospitalEpisode* or to *TreatmentWard*. It could be used in both meanings. In other words, relative pronouns such as "who" and "which" not always give us accurate understanding of what we relate to. To cope with such situations we introduce a concept of so called short name in our controlled natural language. Formally, the short name is a variable over instances of the given class – *count Patients p, where exists p.HospitalEpisode e, where exists e.TreatmentWard t, where exists t.Manipulation m, where m.manipul.code=02078.* Now we are able to specify also the second of abovementioned meanings – *count Patients p, where exists p.HospitalEpisode e, where exists e.TreatmentWard t, where exists e.Manipulation m, where m.manipul.code=02078.* We have also unified other components of the natural language, e.g. we use the keyword "where" instead of "who", "which" and "with",

and the keyword “exists” instead of “have/has at least one”. The dot notation after the short name must be perceived as the “of” relation – *count Patient p, where exists HospitalEpisode e of Patient p, where exists TreatmentWard t of HospitalEpisode e, where exists Manipulation m of TreatmentWard t, where manipul.code of Manipulation m equals 02078*.

Formally speaking, the short name must be used before every attribute name to get rid of ambiguities. However, in cases when it is clear to which class the particular attribute refers the short name can be omitted. We also allow omitting other features of the language that are not critical for understanding of queries (e.g. one can omit the empty parentheses after the unary Date and DateTime operations like *year()* or *minute()*).

Let us now introduce some basic notations that we will use describe the query language. We will use the terms *parent class* and *child class* to refer to classes that are higher or lower in the “have” hierarchy. For example, the class “TreatmentWard” has two parent classes – “HospitalEpisode” (direct parent) and “Patient” (further ancestor) and one child class “Manipulation”. If *x* is an instance of the class “TreatmentWard”, then its parent instances will be denoted as *x.HospitalEpisode* and *x.Patient*. In both cases they denote exactly one instance, i.e. that of the class “HospitalEpisode” and of the class “Patient”, respectively. We use the same dot notation also for accessing instances of child classes, but in this case we obtain a set of instances. For example, *x.Manipulation* would be a set of manipulations reachable from the given treatment ward *x*.

In more complicated cases another concept of *brother class* is important. If *x* is an instance of the class “HospitalEpisode”, then by *x.HospitalEpisode* we understand *y.HospitalEpisode*, where *y=x.Patient* (i.e. *y* is the closest parent of *x*, which is also parent class of the given class “HospitalEpisode”). Similarly, if *x* is an instance of the class “TreatmentWard”, then *x.OutpatientEpisode=y.OutpatientEpisode*, where *y=x.Patient*.

If *AClass* is an arbitrary class of the ontology, we will use the term *AClass attribute expression* to denote attribute expressions of both *AClass* itself and all of its parent classes (we assume here that parents and children share no common attribute names). We cannot use attribute expressions of child classes here, because there can be many children instances for the given *AClass* instance. We will be able to access these instances by introducing quantors *exists*/*notexists* later.

We can also perceive our query language as an analogue to some many-sorted predicate language with a difference that it is written in such syntax that is more user-friendly. There has been an attempt to create such a language *(Yang, J.S.H., Chin, Y.H., Chung, C.G.: Many-sorted First-Order Logic Database Language. In: The Computer Journal, Vol. 35, No. 2, pp. 129-137 (1992))* though it has not led to a practical implementation.

2.1.2.5. Query Patterns

Queries are to be written in a controlled natural language and are based on seven sentence templates. The main part of the templates is the so called *selection condition*,

which is a selection condition over instances of the given class. We assume that selection conditions are to be written in a natural language. We describe the used language constructs more formally at the end of this section. However, the sentence templates described in this section can be understood without knowing the precise syntax of selection conditions.

Pattern T1.

COUNT AClass [x] WHERE <selection condition>

Semantics: counts instances of *AClass*, which satisfy the selection condition. Examples:

- *COUNT Patients, WHERE EXISTS HospitalEpisode, WHERE referringPhysician=familyDoctor* (count of patients who have been referred to hospital by their family doctors);
- *COUNT HospitalEpisodes, WHERE dischargeTime-admissionTime>15d* (how many episodes have lasted longer than 15 days);
- *COUNT HospitalEpisodes e1, WHERE EXISTS HospitalEpisode e2, WHERE e1<>e2 AND e2.admissionTime>e1.dischargeTime AND e2.admissionTime-e1.dischargeTime<30d* (how many there have been such episodes, after which the patient has returned to hospital in less than 30 days).

Pattern T2.

{SUM/MAX/MIN/AVG/MOST} <attribute expression> FROM AClass [x] WHERE <selection condition>

Semantics: selects instances of *AClass*, which satisfy the selection condition, calculates the attribute expression for each of these instances obtaining a list to which the specified aggregate function is then applied. Examples:

- *SUM totalCost FROM HospitalEpisodes, WHERE dischargeReason=healthy AND birthDate.year()=2012* (how much successful treatments of patients born in 2012 have cost);
- *MOST diagnosis.code FROM DischargeDiagnoses, WHERE nr=1 AND dischargeReason=deceased* (get the most frequent main (nr=1) death diagnosis).

Pattern T3.

SELECT FROM AClass [x] WHERE <selection condition> ATTRIBUTE <attribute expression> ALL DISTINCT VALUES

Semantics is obvious. Examples:

- *SELECT FROM HospitalEpisodes, WHERE dischargeReason=deceased, ATTRIBUTE responsiblePhysician.surname ALL DISTINCT VALUES*;

- *SELECT FROM DischargeDiagnoses, WHERE nr=1 AND dischargeReason=deceased, ATTRIBUTE diagnosis.code ALL DISTINCT VALUES.*

Pattern T4.

SHOW [n/ALL] AClass WHERE <selection condition>

Semantics: shows n or all instances of *AClass* which satisfy the selection condition.

Pattern T5.

FULLSHOW [n/all] AClass WHERE <selection condition>

Semantics: the same as "show", but shows also the child class instances attached to the selected *AClass* instances.

Pattern T6.

SELECT AClass x WHERE <selection condition>, DEFINE TABLE <x-expr'1> [(COLUMN C_1], …, <x-expr'n> [(COLUMN C_n)] [, KEEP ROWS WHERE <C_i selection condition>] [, SORT [ASCENDING/DESCENDING] BY COLUMN C_i] [, LEAVE [FIRST/LAST] n ROWS]

Semantics: selects all instances of *AClass*, which satisfy the selection condition, then makes a table with columns C_1 to C_n, which for every selected *AClass* instance *x* contains an individual row, which in column C_1 contains the value of the <x-expr'1>, …, in column C_n contains the value of the <x-expr'n>. Then it is possible to perform some basic operations with the table like filtering out unnecessary rows, sorting the rows by values of some column and then taking just the first or the last n rows from the table. Examples:

- *SELECT HospitalEpisodes x, WHERE dischargeReason=deceased, DEFINE TABLE x.surname (COLUMN Surname), x.dischargeTime.date() (COLUMN Dying_date), (COUNT x.Manipulation, WHERE manipul.code=02078) (COLLUMN Count_02078), (SUM manipul.cost FROM x.Manipulation, WHERE manipul.code=02078) (COLUMN cost_02078)*;
- *SELECT CPhysicians k, WHERE name=Gatis AND EXISTS HospitalEpisode, WHERE responsiblePhysician=k, DEFINE TABLE surname (COLUMN Physician_surname), (COUNT HospitalEpisodes, WHERE responsiblePhysician=k) (COLUMN Episode_count), (MOST diagnosis.code FROM AdmissionDiagnoses, WHERE nr=1 AND responsiblePhysician=k) (COLUMN Most_frequent_main_diagnosis), KEEP ROWS WHERE Episode_count>5, SORT DESCCENDING BY COLUMN Episode_count, LEAVE FIRST 10 ROWS.*

Let us now talk a bit more precisely about the means for defining columns. The expression <*x-expr'i*> defines the value of column C_i in the row that corresponds to the *AClass* instance *x*. This expression can be defined in one of four ways:

```
<x-expr'i> ::= <x-dependent attribute expression> | <x-
dependent count expression> | <x-dependent
```

{SUM/MAX/MIN/MOST] expression> | <x-dependent child attribute selector expression>

<x-dependent attribute expression> examples: *x.surname, x.dischargeTime.date().* Prefix "*x.*" can be used before attributes of both *x* and its parents. Semantics is obvious.

<x-dependent count expression> examples: *(COUNT x.Manipulations, WHERE manipul.code=02078), (COUNT HospitalEpisodes, WHERE responsiblePhysician=x).* In the first example we use the prefix *x* in "x.Manipulations" to denote that we do not select from the whole set of manipulations, but only from those that are reachable from *x*.

<x-dependent {SUM/MAX/MIN/MOST} expression> examples*: (SUM manipul.cost FROM x.Manipulations, WHERE manipul.code=02078), (MOST diagnosis.code FROM AdmissionDiagnoses, WHERE nr=1 AND responsiblePhysician=x).*

<x-dependent child attribute selector expression> examples: *(x.DischargeDiagnosis, WHERE nr=1).diagnosis.code, (x.TreatmentWard, WHERE nr=*).ward* (by * we denote the number of the last instance of TreatmentWard connected to the given HospitalEpisode *x*). This is a new kind of construction whose general form is as follows: *(x.<name of x children class A>, WHERE <selection condition>).<name of attribute a of class A>*. Its value is defined in the following way – we start by taking all instances of class A that are reachable from *x*, then select of them those instances that satisfy the selection condition and then create a list of values of the attribute *a* of the selected instances. The most important case here is the one where this list contains only one instance, e.g. in the following table definition example:

SELECT HospitalEpisodes x, WHERE dischargeReason=deceased, DEFINE TABLE x.surname (COLUMN Surname), x.dischargeTime.date() (COLUMN Dying_date), (x.DischargeDiagnosis, WHERE nr=1).diagnosis.code (COLUMN main_diagnosis), (x.TreatmentWard, WHERE nr=).ward (COLUMN last_ward).*

Pattern T7.

There are two more cases in the definition of the table, where table rows come from other source, not being instances of some class. Being very similar these two cases form two subtemplates of the last template:

a) **SELECT FROM AClass [a] WHERE <selection condition> ATTRIBUTE <attribute expression> ALL DISTINCT VALUES x, DEFINE TABLE…**
b) **SELECT FROM INTERVAL (start-end) ALL VALUES x, DEFINE TABLE…**

Semantics of both cases is obvious. Examples:

- *SELECT FROM TreatmentWards ATTRIBUTE ward ALL DISTINCT VALUES x, DEFINE TABLE x (COLLUMN Ward), (SUM manipul.cost FROM Manipulations, WHERE ward=x) (COLUMN Cost);*
- *SELECT FROM INTERVAL (1-12) ALL DISTINCT VALUES x, DEFINE TABLE x (COLUMN Month), (COUNT HospitalEpisodes, WHERE*

admissionTime.month()=x) (COLUMN Episode_count) (MOST diagnosis.code FROM AdmissionDiagnoses, WHERE nr=1 AND admissionTime.month()=x) (COLUMN Most_frequent_main_diagnosis).

Let us conclude this section by defining more formally the constructs of a controlled natural language allowed in the selection conditions. They can, of course, be guessed from the examples given above.

```
<AClass selection expression> ::= AClass [<short name>]
[WHERE <selection condition>]

<selection condition> ::= <attribute condition> |
<quantor condition> | (<selection condition> {AND|OR}
<attribute condition>) | (<selection condition> {AND|OR}
<quantor condition>)

<quantor condition> ::= {[NOT]EXISTS | FORALL} [short
name.] AClass [short name] [WHERE <selection condition>]
```

Short name provides a name for the given object and can be any string different from the class and attribute names. The abovementioned grammar provides a formal language (for formulating selection expressions) that is close to a natural language and therefore easily perceptible. From a natural language's point of view selection expressions are sentences in a controlled natural language that exploit both words of a natural language (like *AND, OR, WHERE, EXISTS, NOTEXISTS*) and "foreign" words – attribute expressions whose syntax and semantics were described above. The grammar is only needed as a guide how to build the selection expressions. We do not use it in teaching the language to domain experts. We, instead, use the same principle exploited when a child learns to speak a natural language, i.e. learning from examples. It was therefore necessary to first see the sentence templates together with some examples and only afterwards see the formal grammar underlying parts of these templates.

2.1.2.6. Another Example

Let us consider the following example from real life. A two-year old child of our friend got seriously ill. Family doctor recommended our friend to consult a leading specialist in Riga Children's Clinical University Hospital. The only thing our friend remembered about the hypothetical diagnosis stated by his family doctor was two keywords – "bronchitis" and "allergy". The question now arose – how to find the leading specialist in this case? It would perhaps be the physician of the hospital who has supervised the most patients as their responsible physician, such that name of their admission diagnosis contain strings "bronchitis" and "allergy". For this to be found our friend wrote the following query:

SELECT CPhysicians x WHERE EXISTS HospitalEpisode, WHERE responsiblePhysician=x, DEFINE TABLE x.surname (COLUMN doctorSurname), (COUNT HospitalEpisodes WHOSE responsiblePhysician=x AND EXISTS AdmissionDiagnosis, WHERE diagnosis.name~bronhīts) (COLUMN bronchitisCount), COUNT HospitalEpisodes WHOSE responsiblePhysician=x AND EXISTS AdmissionDiagnosis, WHERE

diagnosis.name~allergy) (COLUMN allergyCount), SORT DESCENDING BY COLUMN bronchitisCount, KEEP ROWS WHERE bronchitisCount>5 AND allergyCount>3

Answer to this query is seen in Fig. 2.1.2.2. So the conclusion here is that perhaps Mr. Mēness would be the most appropriate physician who could consult the child of our friend.

doctorSurname	bronchitisCount	allergyCount
Laimītis	111	6
Dīķis	97	7
Mēness	87	13

Fig. 2.1.2.2. Answer to the abovementioned query.

This example also shows the need for a strict policy related to the practical use-cases of our query language – do we have to allow any patient accessing any such information about physicians of the hospital or not?

2.1.2.7. Formal Definition of the Query Language

Definite Clause Grammar

To define the language formally, we will use the Definite Clause Grammar (DCG) and its concept of parametric non-terminals, which will be depicted in **bold** (the terminal symbols will be underlined). We will use the following set of parameters:

- **AClass** – parameter, whose values can be names of basic classes;
- **KClass** - parameter, whose values can be names of classifier classes;
- **x** – parameter, whose values can be instances of **AClass**;
- **a** – parameter, whose values can be names of **AClass** attributes with primitive data types;
- **b** – parameter, whose values can be names of **AClass** attributes with classifier data types;
- **k** – parameter, whose values can be names of **KClass** attributes (they can only be of primitive data types).

We will use the traditional grammar syntax for writing productions of definite clause grammar with one special feature for separating three cases of optional parts of the language:

- `[a|b]` – a or b, or none of them;
- `{a|b}` – a or b;
- `⌊text⌋` - optional part of the sentence without any semantic meaning that can be provided as an illustration for alleviating the understanding the natural language sentence.

For example, we can now better understand the abovementioned instance query expression: `show [all] [N] ⌊instances of⌋ AClass`. This notation allows us writing the following sentences:

- `show Patient;`
- `show all Patient;`
- `show all 3 Patient;`

- `show 5 instances of Patient;`
- `…`

Parts included in half-brackets can also be written incompletely or slightly differently according to the language rules. For example, we can also write: `show 1 instance of Patient`.

Scalar Elements and Class Elements

Let us start with the first production described in DCG syntax:

```
<AClass x scalEl1>::=x.a
```

This production describes the first type of so called scalar element with respect to some basic class `AClass` and its particular instance `x`. For example, if we take the class `Episode` as `AClass` and some instance denomination `e` as `x`, then we come out with the following regular grammar production, where we list the whole set of attributes of class `Episode`:

```
<Episode e scalEl1>::=e.duration | e.historyNum | …
```

The semantics of this production is obvious – it gives us the value of the mentioned attribute on the given instance `e`.

To be able to get values of classifier attributes, we need another DCG production:

```
<AClass x scalEl2>::=x.b.k;
```

Now, let `AClass` and `BClass` be two arbitrary basic classes. We will call `BClass` the *parent* of `AClass` (and `AClass` the *child* of `BClass`), iff there is an association between those two classes with cardinality 1 at the end of `BClass`. It is easy to see that each class of a Star ontology can only have one parent (except for the Master class that does not have a parent). We will call `CClass` the ancestor of `AClass`, iff either `CClass` is a parent of `AClass` or `CClass` is ancestor of the parent of `AClass`.

If CClass is some ancestor of AClass and c and d are attributes of CClass with primitive and classifier data types respectively, then we can come out with two more productions:

```
<AClass x scalEl3>::=x.CClass.c;
<AClass x scalEl4>::=x.CClass.d.k;
```

Again, the semantics of all four expressions is to get the value of the given scalar attribute in the given instance `x`. Here and further – if this value is null, we can still perform standard operations with it, but the result will always be null.

Of course, we also need to access the other kind of attributes – the ones with a classifier data type. Therefore we introduce the next two productions:

```
<AClass x classEl1>::=x.b;
<AClass x classEl2>::=x.CClass.d;
```

Now we can summarize four scalar element types and two class element types into one non-terminal each:

```
<AClass x scalEl>::=<AClass x scalEl1> | <AClass x scalEl2> |
<AClass x scalEl3> | <AClass x scalEl4>

<AClass x classEl>::=<AClass x classEl1> | <AClass x classEl2>
```

We have used a notation in form `x.CClass` to describe scalar and class elements above. This notation will be used further in more general meaning, so it must be precisely defined here. We use the notion of `x.CClass` (where `CClass` is some basic class) to introduce some means for defining sets of instances of basic classes. If `x∈AClass`, we define the notion of `x.CClass` to be one of the following:

- if `CClass` is ancestor of `AClass`, then value of `x.CClass` is the **instance** of `CClass` connected with `x`;
- if `AClass` is the parent of `CClass`, then value of `x.CClass` is the **set of instances** connected with `x`;
- if `AClass` is ancestor of `CClass`, then value of `x.CClass` is the **set of instances** reachable from `x`;
- if `AClass` = `CClass` = `Master class of the ontology`, then value of `x.CClass` is **not defined**;
- if AClass is a classifier class, then value of `x.CClass` is the **set of instances** of `CClass`, which has some attribute, whose value is `x`;
- in all other cases we can always find the nearest common ancestor of `AClass` and `CClass`, and then value of `x.CClass` is the **set of instances** reachable from the common ancestor.

The concept of this instance set definition is not artificial. It is very intuitive and we can understand it (at least the typical cases) even without delving into the formal definitions. We believe that this is exactly how we think about our domain ontology being domain experts. For example, a doctor can ask someone to find those patients, who have some episodes, in which there has been a movement within a specific ward. This question (logical expression) can be written very intuitively in our syntax (it will be seen later).

Scalar Expressions and Logical Expressions

Being familiar with the concept of scalar element, we can now develop a concept of scalar expression:

```
<AClass x scalExpr>::=/* any expression that can be formed
using instances of <AClass x scalEl>, constants and allowed
operations mentioned above */
```

Likewise, we introduce concepts of logical element and logical expression:

```
<AClass x logEl1>::=/* any expression that can be formed using
instances of <AClass x scalExpr>, constants and allowed
comparison operations mentioned below */

<AClass x logEl2>::=/* any expression that can be formed using
instances of <AClass x classEl> and comparison operations =
```

```
and <> */

<AClass x logEl>::=<AClass x logEl1> | <AClass x logEl2>

<AClass x logExpr>::=/* any expression that can be formed
using instances of <AClass x logEl> and operations and, or and
not */
```

When forming <AClass x logEl> from instances of <AClass x scalExpr>, data types must be compatible. The allowed comparison operators depend on data types t1 and t2 of involved scalar expressions, and they are:

- =, <>, if t1=t2;
- <, <=, >, >=, if t1 and t2 are Integer or Real (in any combination), or t1=t2=String;
- ~, if t1=t2=String (meaning the containment operation).

Some examples of scalar expressions from the hospital ontology (here *x* is an instance of class "Manipulation"):

- x.count * 5 (its data type is Integer);
- x.performer.personalCode.substring(1,4) (its data type is String);
- x.Patient.birthDate.month() (its data type is Integer).

Some examples of logical expressions from the hospital ontology (here *x* is an instance of class „Manipulation"):

- x.count>5;
- x.performer.personalCode.substring(1,2)="20" and x.performer.personalCode.substring(3,4)="10" (meaning – has the performer born on October 20th?);
- x.Patient.birthDate.month()=4 or x.Patient.birthDate.month()=6 or x.Patient.birthDate.month()=9 or x.Patient.birthDate.month()=11;
- k1.ward="12-69" and k2.ward="12-69" and k1<>k2 (where k1 and k2 are instances of class „Movement").

Scalar and logical expressions can also be constructed from more than one instance of basic classes. For example, if x, y, z, etc. are instances of some basic classes AClass, BClass, CClass, etc. respectively, we can define more complicated expressions:

```
<AClass x BClass y scalExpr>::=/* any expression that can be
formed using instances of <AClass x scalExpr> and <BClass y
scalExpr> and allowed operations */

<AClass x BClass y CClass z scalExpr>::=/* any expression that
can be formed using instances of <AClass x scalExpr>, <BClass
y scalExpr> and <CClass z scalExpr> and allowed operations */

…

<AClass x BClass y logExpr>::=/* any expression that can be
formed using instances of <AClass x logExpr> and <BClass y
logExpr> and operations and, or and not */
```

```
<AClass x BClass y CClass z logExpr>::=/* any expression that
can be formed using instances of <AClass x logExpr>, <BClass y
logExpr> and <CClass z logExpr> and operations and, or and not
*/

…
```

Selection Expression

We have now defined all the means necessary for building the query language. Let us now define the selection expression, where we want to select instances x of some basic class AClass (terminals are underlined here):

```
<AClass x selectExpr>::=AClass [x
[where <AClass x logExpr> [and [not] exists x.BClass [y
 [where <AClass x BClass y logExpr> [and [not] exists
[x|y].CClass [z
  [where <AClass x BClass y CClass z logExpr> … ]]]]]]]]
```

The semantics of this expression is as follows – it defines a subclass of AClass that contains all the instances of AClass satisfying the logical expression stated in the *where* part of the expression. For example, to find patients, who has some episodes, in which there has been a movement within the ward "12-69"), we would write:

```
Patient p, where exists p.Episode e, where exists e.Movement
m, where m.ward="12-69"
```

Alternatively, we can use shorter syntax here, which is also allowed for rationalizing reasons:

```
Patient, where exists Episode, where exists Movement, where
ward="12-69"
```

We can continue the selection expression as deep as we want to, but our experience shows that end uses very rarely delves deeper that to the second level of existence quantifiers.

Aggregate Functions

When we have defined means for selecting subsets of classes, we can exploit this concept to define aggregate functions over these subsets:

```
<AClass count>:= count <AClass x selectExpr>

<AClass max>:= max [N] <AClass x scalExpr> from <AClass x
selectExpr>

<AClass min>:= min [N] <AClass x scalExpr> from <AClass x
selectExpr>

<AClass avg>:= avg <AClass x scalExpr> from <AClass x
selectExpr>

<AClass most>:= most [N] <AClass x scalExpr> from <AClass x
selectExpr>

<AClass sum>:= sum <AClass x scalExpr> from <AClass x
selectExpr>
```

```
<AClass aggregExpr>::=<AClass count> | <AClass max> | <AClass
min> | <AClass avg> | <AClass most> | <AClass sum>
```

There are also some constraints regarding the data types that are allowed to use in scalar expressions of abovementioned query expressions. In `max`, `min`, `avg` and `sum` allowed data types of `<AClass x scalExpr>` are `Integer`, `Real` and `Duration`. In `most` any data type except `Real` is allowed, but it is mostly used for scalar and date data type. The return data type in `max`, `min`, `avg` and `sum` is the same as its expression's data type with one exception – if expression in `avg` is of type `Integer`, the result will be of type `Real`. The return type for `count` is `Integer`.

`max`, `min` and `most` sentences can be supplemented by an integer N specifying the count of values to be returned. If this feature is used, the return value will be a list of N values, each of the type specified by the scalar expression.

Relative Selection Expressions

This selection expression is used to define a global set of instances of some class. However, we must also be able to define such instance set locally (meaning that we have some starting instance x of basic class AClass, from which we want to find all connected instances satisfying some condition). Therefore we also have to introduce another form of selection expression called the relative selection expression:

```
<AClass x relative BClass selectExpr>::=x.BClass [y
[where <AClass x BClass y logExpr> [and [not] exists
[x|y].CClass [z
 [where <AClass x BClass y CClass z logExpr> … ]]]]]
```

In fact, we define here a function f(w), whose values are particular subclasses of AClass. Now we can define relative aggregate functions with respect to some starting instance x of basic class AClass:

```
<AClass relative BClass count>::=count <AClass x relative
BClass selectExpr>

<ACLass relative BClass max>::=max <AClass x relative BClass
selectExpr>

<AClass relative BClass min>::=min <AClass x relative BClass
selectExpr>

<Aclass relative BClass avg>::=avg <AClass x relative BClass
selectExpr>

<AClass relative BClass most>::=most <AClass x relative BClass
selectExpr>

<AClass relative BClass sum>::=sum <AClass x relative BClass
selectExpr>

<AClass relative BClass aggregExpr>::=<AClass relative BClass
count> | <AClass relative BClass max> | <AClass relative
BClass min> | <AClass relative BClass avg> | <AClass relative
BClass most> | <AClass relative BClass sum>
```

More Advanced Expressions

Another possible context is, when x is free variable of some scalar expression (this will be used, when we will talk about table query expressions). Then we can define advanced scalar and logical expressions (here, the parameter Scalar can assume names of primitive data types as values):

```
<Scalar x BClass y scalExpr>::=/* any expression that can be
formed using instances of <BClass y scalEl>, variable x and
constants and allowed operations */

<Scalar x BClass y CClass z scalExpr>::=/* any expression that
can be formed using instances of <BClass y scalEl>, <CClass z
scalEl>, variable x and constants and allowed operations*/

...

<Scalar x BClass y logExpr>::=/* any expression that can be
formed using instances of <BClass y logEl>, logical variable x
and operations and, or and not */

<Scalar x BClass y CClass z logExpr>::=/* any expression that
can be formed using instances of <BClass y logEl>, <CClass z
logEl>, logical variable x and operations and, or and not */

...
```

Now we can define relative selection expressions with respect to the scalar value x as well as relative aggregate functions:

```
<Scalar x relative BClass selectExpr>::= BClass [y
[where <Scalar x BClass y logExpr> [and [not] exists y.CClass
[z
 [where <Scalar x BClass y CClass z logExpr> … ]]]]]

<Scalar relative BClass count>::=count <Scalar x relative
BClass selectExpr>

<Scalar relative BClass max>::=max <Scalar x relative BClass
selectExpr>

<Scalar relative BClass min>::=min <Scalar x relative BClass
selectExpr>

<Scalar relative BClass avg>::=avg <Scalar x relative BClass
selectExpr>

<Scalar relative BClass most>::=most <Scalar x relative BClass
selectExpr>

<Scalar relative BClass sum>::=sum <Scalar x relative BClass
selectExpr>

<Scalar relative BClass aggregExpr>::=<Scalar relative BClass
count> | <Scalar relative BClass max> | <Scalar relative
BClass min> | <Scalar relative BClass avg> | <Scalar relative
BClass most> | <Scalar relative BClass sum>
```

Query Expressions

Let us know define three types of query expressions available in our query language more formally.

Instance query expressions can be defined as follows:

```
<AClass instanceQueryExpr> ::= show [all] [<N>] ⌊instances of⌋ AClass

<N> ::= 1 | 2 | 3 | ...
```

Scalar query expressions was already defined, when we talked about aggregate expressions, which happens to be the same thing. However, we also want to use scalar expressions in building more complex logical expressions, so we have to define also simple scalar expressions:

```
<AClass scalarQueryExpr>::=<AClass relative BClass aggregExpr>

<AClass simpleScalarQueryExpr> ::= /* any <AClass scalarQueryExpr>, whose result is no more than one value (in other words – which does not use the parameter N) */
```

Simple scalar query expressions can now be used to extend the definition of scalar element so we can use it in building more complex logical expressions:

```
<AClass x scalEl>::=<AClass x scalEl1> | <AClass x scalEl2> | <AClass x scalEl3> | <AClass x scalEl4> | (<AClass simpleScalarQueryExpr>)
```

For example, we can now write such a query for finding the count of patients, which have arrived at hospital for more than five times: `count Patient p, where (count p.Episode)>5`.

Finally, table query expressions can be defined as follows:

```
<tableQueryExpr> ::= <AClass tableQueryExpr> | <Scalar tableQueryExpr>

<AClass tableQueryExpr> ::= select <AClass x selectExpr> define table ⌊where rows correspond to x and columns are⌋ <AClass x scalExpr> [(column <text>)], <AClass x scalExpr> [(column <text>)], ... [, keep rows where <logExpr>] [, take [first|last] <N> rows]

<Scalar tableQueryExpr> ::= [<scalarSelect x> | <intervalSelect x>] define table ⌊where rows correspond to x and columns are⌋ <Scalar x BClass y CClass z … scalExpr> [(column <text>)], <Scalar x BClass y CClass z … scalExpr> [(column <text>)], ... [, keep rows where <logExpr>] [, take [first|last] <N> rows]

<text> ::= /* Arbitrary text string (column caption) */

<logExpr> ::= /* Any logical expression that can be formed using column captions and logical operations*/

<scalarSelect x> ::= select [all|<N>] [distinct] values x of <BClass y scalExpr> from <BClass y selectExpr>

<intervalSelect x> ::= select ⌊all⌋ values x from interval (<N>-<N>)
```

Here we not only select instances from basic class *AClass*, but also introduce a variable x, whose scope is the selected set of instances, so we can later relate to this variable, when we need to get some information about particular rows of the table. We can therefore continue the select sentence by specifying table columns after keywords define table and using x as variable over all rows of the table.

It is also allowed to perform select operation on some classifier class C, but in this case the where clause can only contain attributes of class C

2.1.3. Development of the web technology-based tool building technologies and methods for modeling of complex, hard-to-formalize systems. (1-st period)

We have formalized the requirements of modeling tools for such systems. According to these requirements we have developed a metamodel for defining DSLs and tools. According to this metamodel we have developed experimental graphical tool-building platform that can be used to modelling in web such systems that are difficult to formalize. The platform provides interactivity, collaboration, different machines support (computers, tablets, and smartphones), reactivity and live HTML. Exploited technologies: Meteor (web application development platform), MongoDB (database), KonvaJS (depicting diagrams), and Bootstrap (HTML, CSS, and JavaScript). Results are described in the paper "A.Sprogis. DSML Tool Building platform in WEB" which is submitted to the DB&IS 2016.

2.1.4. Research of the ontology-based linked data technologies for applications of e-government and e-health. (1-st period)

Conclusion: Use-cases of ontology-based linked data grow fairly rapidly in the world (around 30% in a year). These use-cases are generally based on open data in form of RDF, they mainly refer to providing different (not known a priori) research, statistics, reviews.

Linked data technologies are not widely used in fields of e-management and e-medicine. The main problem: data of e-management and e-medicine are strictly regulated by the law and have restricted access that contradicts the technologies of ontology-based linked data. However, by widening the volume of open date in state level the technologies of linked data could be used at least partially also in fields of management and medicine – mainly in aspects of statistics and research.

2.1.5. Developing new methods for natural language parsing into the form of semantic graphs based on n-ary predicate approach (such as AMR, FrameNet, BabelNet) for information extraction from natural language texts (natural language understanding tasks). (1-st period)

By integrating FrameNet n-ary relation extraction and BabelNet inspired Named Entity Linking approaches we have developed a unified linguistic ontology suitable for extracting Curriculum Vitae like semantic information (a semantic graph) about persons and organizations mentioned in unstructured newswire texts. We have also developed a new classification algorithm nicknamed C6.0 and used for implementing a semantic parser for Latvian, English, Czech and Chinese, with which we participated in SemEval-2015 competition where it performed on par with other state of-the-art parsers and was among the three winning parsers in various testing categories. These scientific results are described in "Barzdins G., Paikens P., Gosko D. Riga: from FrameNet to Semantic Frames with C6.0 Rules. SemEval 2015 Task 18: Semantic Dependency Parsing. Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Association for Computational Linguistics, pp. 960–964." (http://www.aclweb.org/anthology/S15-2160).

2.1.6. Development of ontology-, web technology- and controlled natural language-based fast ad-hoc query language that will be directly usable by domain experts (without involvement of the programmer) (2-nd period)

The ad-hoc querying process is slow and error prone due to inability of business experts of accessing data directly without involving IT experts. The problem lies in complexity of means used to query data. We propose a new natural language- and *semistar* ontology-based ad-hoc querying approach which lowers the steep learning curve required to be able to query data. The proposed approach would significantly shorten the time needed to master the ad-hoc querying and to gain the direct access to data by business experts, thus facilitating the decision making process in enterprises, government institutions and other organizations.

These results are described in more detail in the paper "J.Barzdins et.al. Self-service Ad-hoc Querying Using Controlled Natural Language" which is submitted to the DB&IS 2016.

2.1.7. Approbation in medical domain (CCUH) of ontology-, web technology- and controlled natural language-based fast ad-hoc query language that will be directly usable by domain experts (without involvement of the programmer) (2-nd period)

Approbation was performed on data of year 2015 of Riga Children's Clinical University Hospital for the needs of intensive care ward. Answers to all ad-hoc queries that were formulated for the needs of analysis of operation of the intensive care ward in year 2015 were obtained using the proposed query language in online mode. When the necessary question was formulated in natural language it took couple of minutes to reformulate it in the proposed language. All the queries were executed in less than 0.3 seconds on average. The approbation also showed the further research directions which are development of subclass and attribute definition features, as well as necessity for very advanced access rights mechanism, because the real hospital data are highly sensitive.

2.1.7.1. Introduction

A concise description of abovementioned fast ad-hoc query language is given in Section 2.1.2. Together with the language we also developed a web-based system (using HTML, CSS and Javascript technologies) in which the language is incorporated. In this section we will review the approbation of the language in the intensive care ward of Riga Children's Clinical University Hospital. The intensive care ward was chosen for the approbation based on the fact that the ad-hoc analysis of its operation requires various ad-hoc queries that are not known in advance and are thus hardly implementable within the existing information system of the hospital.

The query system takes hospital data of one year (year 2015) and allows formulating queries based on them. These data include about 35'000 hospital episodes and about 70'000 outpatient episodes. The total volume of these data is about 2 GB. Approbation of the system started with development of hospital data ontology (seen in Fig. 2.1.7.1). This was done by taking the ER model of the relational database of the hospital as a basis. The ER model was slightly transformed to obtain the ontology seen in Fig. 2.1.7.1 which happened to come out naturally as a semistar data ontology. As already mentioned in description of Section 2.1.1, the proposed query language is to be exploited on exactly this type of data ontologies. ER models can always be transformed to semistar ontologies as long as they are subject-oriented as in case of hospital ER model.

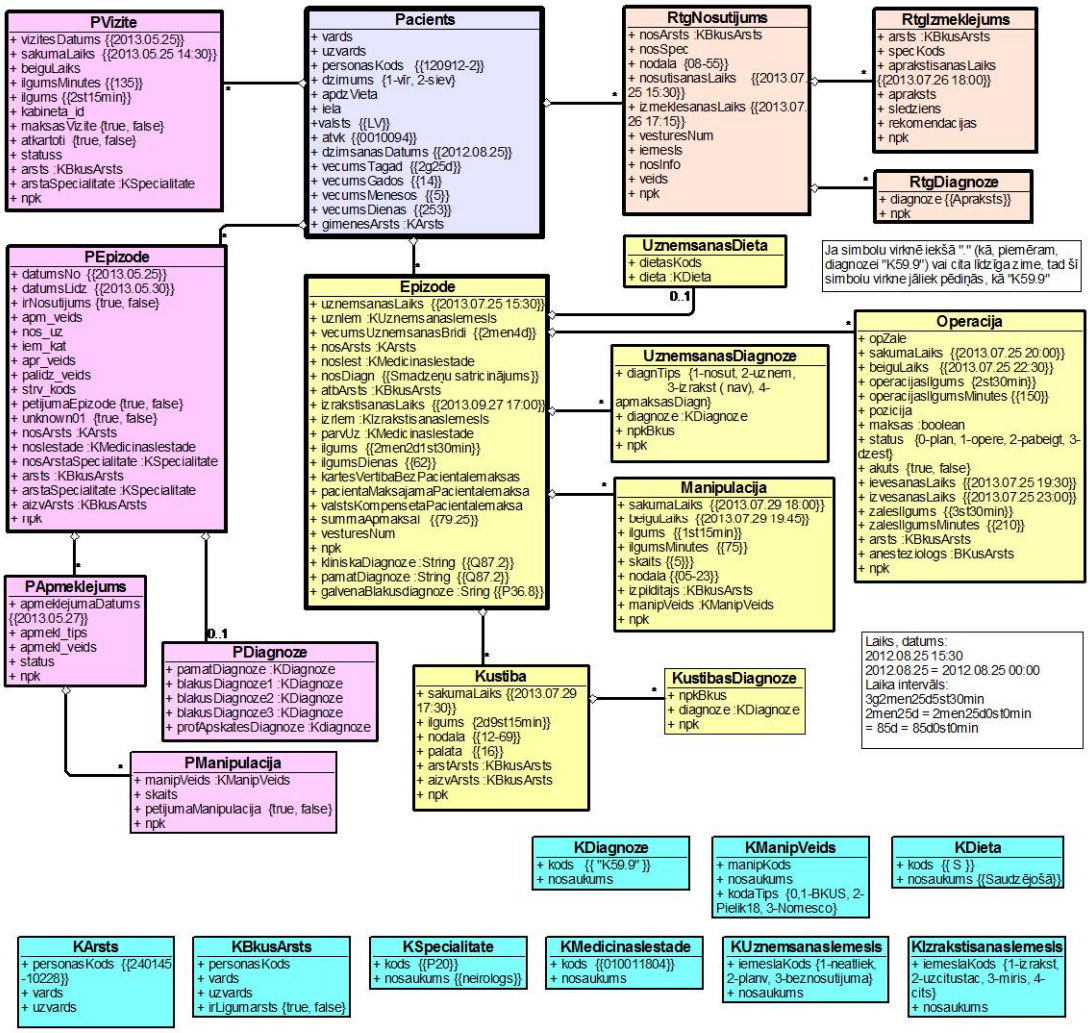

Fig. 2.1.7.1. Hospital data ontology.

For one to be able to use our proposed ad-hoc query system, data must first be exported from the real relational database of hospital to the data ontology seen in Fig. 2.1.7.1. Since real ER model is very similar to the given data ontology, the export is not very sophisticated. A simple program was written for this task that exports one-year hospital data to the given format in about half a minute. This export must be done each time when the query system is started. Since the export is quite fast, the system could also be used in operative mode in future, i.e. new hospital data could be exported every night, not only once a year. However, such an operative co-operation with a database requires more research and experimental testing that is planned for the next stages of this project.

2.1.7.2. Approbation of the query language and its serving system in the context of real use-cases

To analyze the operation of intensive care ward in 2015, manager of the ward together with some colleagues construed queries in natural language (of course, imprecisely) that are needed for analyzing the most essential aspects of the operation. These queries were made more precise by the help of the manager of the ward and a

representative of the given system. Then the queries were rewritten in the proposed language without big difficulties. Let us now see some examples of these queries.

Count of treated patients (epizodes) in ward 12-69 (the intensive care ward):

skaits Epizodes kurām eksistē Kustība, kurai nodaļa=12-69

(count Epizodes where exists Kustība, where nodaļa=12-69)

Count of deceased patients in ward 12-69:

skaits Epizodes kurām izrIem.iemeslaKods=3 un eksistē kustība kurai npk= un nodaļa=12-69*

(count Epizodes where izrIem.iemeslaKods=3 and exists kustība where npk= and nodaļa=12-69)*

Count of deceased patients by various CCUH wards (in table format):

atlasit no Kustibam atributa nodala visas atskirigas vertibas x, izveidot tabulu x (kol Nod), (skaits Epizodes kuram izriem.iemeslakods=3 un eksiste Kustiba kurai npk= un nodala=x) (kol Mir_pac_skaits), (vid /ilgums/ Epizodes kuram izriem.iemeslakods=3 un eksiste Kustiba kurai npk=* un nodala=x) (kol Vid_ilg_ mir_pac), atlasit rindas kuram Mir_pac_skaits<>0*

(select from Kustibam attribute's nodala all distinct values x, define table x (col Nod), (count Epizodes where izriem.iemeslakods=3 and exists Kustiba where npk= and nodala=x) (col Mir_pac_skaits), (avg /ilgums/ Epizodes where izriem.iemeslakods=3 and exists Kustiba where npk=* and nodala=x) (col Vid_ilg_ mir_pac), leave rows where Mir_pac_skaits<>0)*

Generate a table of deceased patients showing their gender, birth date, death date, main diagnosis and death ward:

atlasīt visas Epizodes kuram izriem.iemeslakods=3, izveidot tabulu pacients.dzimums (kol Dzimum), pacients.dzimsanasdatums (kol Dzim_dat), izrakstisanaslaiks.Date (kol Mir_atums), pamatdiagnoze (kol Pam_diagn), (Kustiba kurai npk=).nodala (kol Mir_nod)*

(select all Epizodes where izriem.iemeslakods=3, define table pacients.dzimums (col Dzimum), pacients.dzimsanasdatums (col Dzim_dat), izrakstisanaslaiks.Date (col Mir_atums), pamatdiagnoze (col Pam_diagn), (Kustiba where npk=).nodala (col Mir_nod))*

Average time of patients staying in ward 12-69:

vid ilgums no Kustības kurām nodaļa=12-69

(avg ilgums from Kustības where nodaļa=12-69)

Average time of patients staying in ward 12-69 that have deceased:

vid ilgums no Kustības kurām nodaļa=12-69 un Epizode.izriem.iemeslakods=3

(avg ilgums from Kustības where nodaļa=12-69 and Epizode.izriem.iemeslakods=3)

Total time of patients staying in ward 12-69:

sum ilgums no Kustības kurām nodaļa=12-69

(sum ilgums from Kustības where nodaļa=12-69)

More detailed description of all patients that have visited ward 12-69 (in table format so that they can later be analyzed using MS Excel):

atlasīt visas epizodes x, kurām eksitē kustiba, kurai nodala=12-69, izveidot tabulu (skaits kustibas, kurām nodala=12-69) (kol skaits), vesturesNum (kol Num), pacients.dzimums (kol Dzim), pacients.dzimsanasDatums (kol DzimDat), uznemsanasLaiks.Date (kol UzLaik), IzrakstisanasLaiks.Date (kol IzrLaik), pamatDiagnoze (kol Pamatdg), kliniskaDiagnoze (kol KlinDg), galvenaBlakusDiagnoze (kol BlakDg), (UznemsanasDiagnoze, kurai diagnTips=1).diagnoze.kods (kol Tip1Dg),(UznemsanasDiagnoze, kurai diagnTips=1).diagnoze.nosaukums (kol Tip1Nos), (UznemsanasDiagnoze, kurai diagnTips=2).diagnoze.kods (kol Tip2Dg),(UznemsanasDiagnoze, kurai diagnTips=2).diagnoze.nosaukums (kol Tip2Nos), (UznemsanasDiagnoze, kurai diagnTips=4).diagnoze.kods (kol Tip4Dg), (UznemsanasDiagnoze, kurai diagnTips=4).diagnoze.nosaukums (kol Tip4Nos), summaApmaksai (kol Samaks), Ilgums (kol IlgEpiz), (kustiba, kurai nodala=12-69).ilgums (kol IlgKust), izrIem.Iemeslakods (kol IzrIem), (skaits operacijas) (kol OpSkait)

(select all epizodes x, where exists kustiba, where nodala=12-69, define table (count kustibas, where nodala=12-69) (col skaits), vesturesNum (col Num), pacients.dzimums (col Dzim), pacients.dzimsanasDatums (col DzimDat), uznemsanasLaiks.Date (col UzLaik), IzrakstisanasLaiks.Date (col IzrLaik), pamatDiagnoze (col Pamatdg), kliniskaDiagnoze (col KlinDg), galvenaBlakusDiagnoze (col BlakDg), (UznemsanasDiagnoze, where diagnTips=1).diagnoze.kods (col Tip1Dg),(UznemsanasDiagnoze, where diagnTips=1).diagnoze.nosaukums (col Tip1Nos), (UznemsanasDiagnoze, where diagnTips=2).diagnoze.kods (col Tip2Dg),(UznemsanasDiagnoze, where diagnTips=2).diagnoze.nosaukums (col Tip2Nos), (UznemsanasDiagnoze, where diagnTips=4).diagnoze.kods (col Tip4Dg), (UznemsanasDiagnoze, where diagnTips=4).diagnoze.nosaukums (col Tip4Nos), summaApmaksai (col Samaks), Ilgums (col IlgEpiz), (kustiba, where nodala=12-69).ilgums (col IlgKust), izrIem.Iemeslakods (col IzrIem), (count operacijas) (col OpSkait))

Queries about patients with infectious diseases:

skaits kustibas, kuram nodala=12-69 un (epizode.pamatdiagnoze.substring(1,1)=A vai B)

(count kustibas, where nodala=12-69 and (epizode.pamatdiagnoze.substring(1,1)=A or B))

skaits kustibas, kuram nodala=12-69 un npk= un epizode.izriem.iemeslakods=3 un (epizode.pamatdiagnoze.substring(1,1)=A vai B)*

(count kustibas, where nodala=12-69 and npk= and epizode.izriem.iemeslakods=3 and (epizode.pamatdiagnoze.substring(1,1)=A or B))*

Queries about patients with oncologic diseases:

skaits kustibas, kuram nodala=12-69 un (epizode.pamatdiagnoze.substring(1,1)=C vai epizode.pamatdiagnoze.substring(1,3)>=D00 un epizode.pamatdiagnoze.substring(1,3)<=D48)

(count kustibas, where nodala=12-69 and (epizode.pamatdiagnoze.substring(1,1)=C or epizode.pamatdiagnoze.substring(1,3)>=D00 and epizode.pamatdiagnoze.substring(1,3)<=D48))

skaits kustibas, kuram nodala=12-69 un npk= un epizode.izriem.iemeslakods=3 un (epizode.pamatdiagnoze.substring(1,1)=C vai epizode.pamatdiagnoze.substring(1,3)>=D00 un epizode.pamatdiagnoze.substring(1,3)<=D48)*

(scount kustibas, where nodala=12-69 and npk= and epizode.izriem.iemeslakods=3 and (epizode.pamatdiagnoze.substring(1,1)=C or epizode.pamatdiagnoze.substring(1,3)>=D00 and epizode.pamatdiagnoze.substring(1,3)<=D48))*

Query about anesthesia:

atlasit intervala (1-12) visas vertibas x, izveidot tabulu x (kol Menesis), (skaits Manipulacijas kuram sakumalaiks.month=x un manipveids.kodatips=2 un manipveids.manipkods=04170) (kol JET)

(select from interval (1-12) all values x, define table x (col Menesis), (count Manipulacijas where sakumalaiks.month=x and manipveids.kodatips=2 and manipveids.manipkods=04170) (col JET))

Query about a special kind of manipulation:

atlasit intervala (1-12) visas vertibas x, izveidot tabulu x (kol Menesis), (skaits Manipulacijas kuram sakumalaiks.month=x un manipveids.kodatips=2 un (manipveids.manipkods=04106 vai manipveids.manipkods=04108 vai manipveids.manipkods=04116 vai manipveids.manipkods=04119 vai manipveids.manipkods=04120 vai manipveids.manipkods=04121 vai manipveids.manipkods=04122 vai manipveids.manipkods=04123 vai manipveids.manipkods=04124 vai manipveids.manipkods=04129 vai manipveids.manipkods=04133 vai manipveids.manipkods=04135)) (Kol Region)

(select from interval (1-12) all values x, define table x (col Menesis), (count Manipulacijas where sakumalaiks.month=x and manipveids.kodatips=2 and (manipveids.manipkods=04106 or manipveids.manipkods=04108 or manipveids.manipkods=04116 or manipveids.manipkods=04119 or manipveids.manipkods=04120 or manipveids.manipkods=04121 or manipveids.manipkods=04122 or manipveids.manipkods=04123 or manipveids.manipkods=04124 or manipveids.manipkods=04129 or manipveids.manipkods=04133 or manipveids.manipkods=04135))(col Region))

A depiction of an example query together with its answer in the proposed system is seen in Fig. 2.1.7.2. The bottom box in Fig. 2.1.7.2 shows the query while the answer is seen in the top box. The middle box shows the result of a syntactical analysis of the query performed by the proposed system. Here the query is made more precise. This is a very important feature of the system for practical usability, because the end-user who is not a programmer can thus easily seen, in which parts of the query he/she has made mistakes.

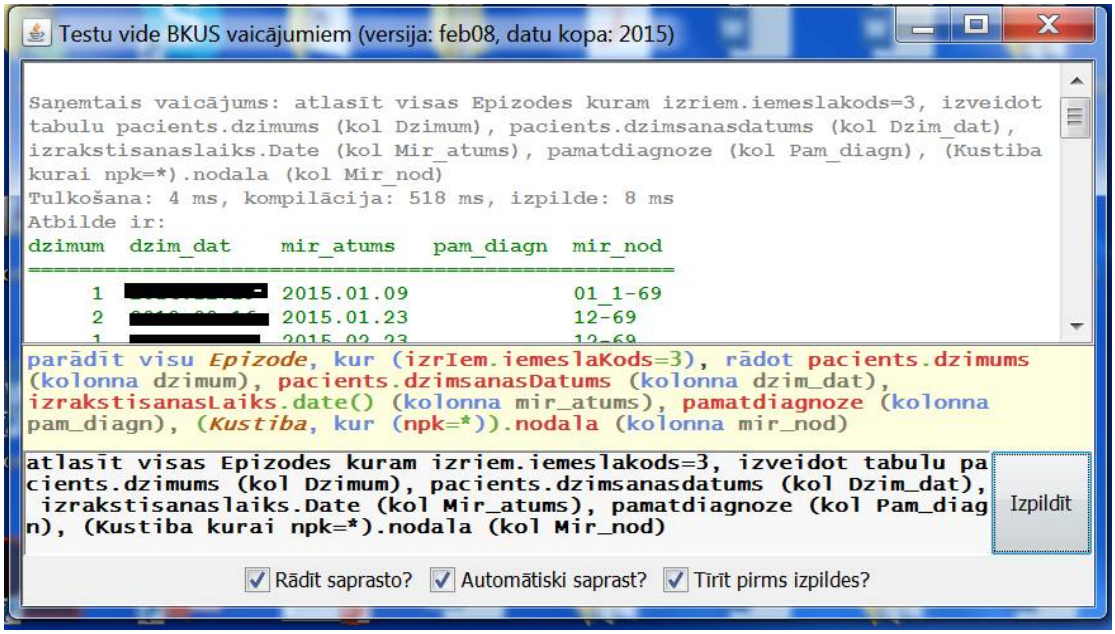

Fig. 2.1.7.2. Query system showing a query, its syntactical coloring and the answer.

This experiment showed that the offered ad-hoc query system is expressive enough, because we could get answers to all of the questions manager of the ward had stated. Answers to these questions were provided either directly from our system or from tables generated by our system that are further processed by some spreadsheet application (e.g. Microsoft Excel) and using only the most common features of the application. We have incorporated only some minor features of table processing in our system, because we assume that domain experts who will use our system are already familiar with MS Excel and will be able to use its potential if our tool gives them the needed data ordered in tables.

2.1.7.3. Performance of the system

Let us now talk a bit about the performance of our system. We have tested the performance on a set of about 100 typical queries. The result of the test is seen in a histogram in Fig. 2.1.7.3. The tool gave us answers to practically all queries (on one-year data) in less than 0.3 seconds, i.e. we did not experience the annoying waiting at any moment. According to statistics there are about 350’000 hospital episodes together in all hospitals in Latvia per year (about ten times more than in Riga Children’s Clinical University Hospital). It means that all these data would take up less than 20 GB of RAM. Currently a quad-core computer with 32 GB of RAM costs about 1’000 euro. Since the semistar data ontology is granular, the query execution can be done in parallel on all four cores thus improving the execution time four times

(our experiment seen in Fig. 2.1.7.3 was performed on only one core). We can conclude that the performance of the query execution over data of all the hospitals in Latvia would only be 2.5-3 times slower than it is now providing the ability to answer a vast majority of queries in less than one second.

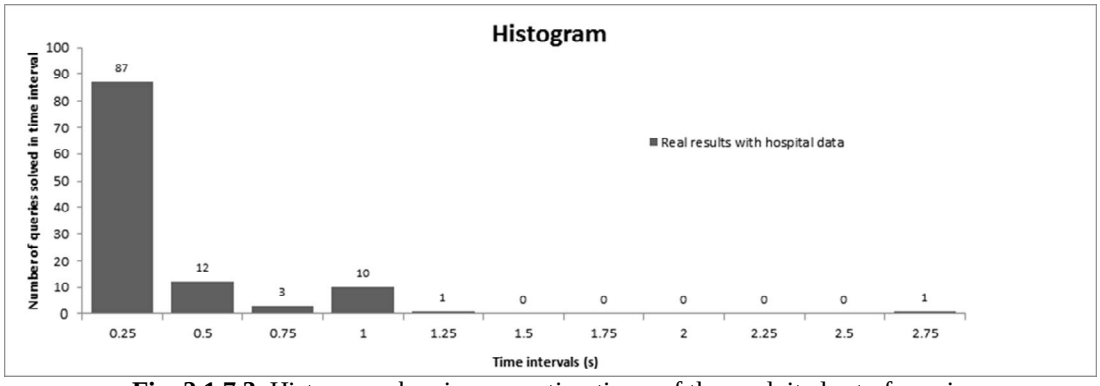

Fig. 2.1.7.3. Histogram showing execution times of the exploited set of queries.

We are, of course, not limited by only one computer with four cores. We can also use several computers connected via high throughput Ethernet thus reducing the waiting time even more (e.g. one second on ten-year data of all Latvian hospitals). Of course, sufficient performance on very large data volumes is another research topic that requires more studies.

2.1.7.4. Usability of the system

One of the most critical aspects of the given query language is how easy it is to learn the language by domain experts that are not programmers. During the approbation of the language we also contemplated, how much time was needed for the manager of the ward to understand the language. To test this aspect more formally we also performed group tests. General situation from the language teaching point of view was best demonstrated in an experiment with experienced nurses who study to obtain Master Degree at the Medical Faculty, University of Latvia. We presented a two hour long lecture about the language and the tool for querying the data. One third of that time was devoted to explanation of the underlying data ontology (to explain to non-programmers what is a class, an attribute, etc.). Afterwards the language was explained on examples, and homework was given to test the level of understanding. The homework consisted of two parts. Firstly, students had to understand sentences written in our controlled natural language and to write them in a good really natural language. Secondly, they had to work in the opposite direction turning natural language sentences into our formal language. Results obtained from this experiment can be seen in Table 1. The main conclusion here is that another two hours long lecture after the completion of homework would be beneficial for a better understanding of the proposed query language.

Table 2.1.7.1. The results of the experiment.

	Number of students succeeded (n=15)				
Task execution level (%)	≥90	≥75<90	≥50<75	≥25<50	<25
Understanding of queries	9	2	2	1	1
Writing of queries	3	5	2	3	2

In this experiment students exploited the query system available in web (http://85.254.199.40), where the data ontology was simplified (see Fig. 2.1.7.4, though the Latvian version of the ontology and the tool was used in the experiment). The data volume was still about that of one year of hospital data, but in this case data were deeply anonymised because of their high degree of sensitivity. The treatment processes used in these teaching data corresponded to real cases, but data related to patients and physicians, as well as other data that are not related to the treatment process were artificially generated. Our experience shows that students being domain experts of this ontology rapidly got very interested in the querying process and started to perceive this as a game. This fact had a beneficial impact on the learning process.

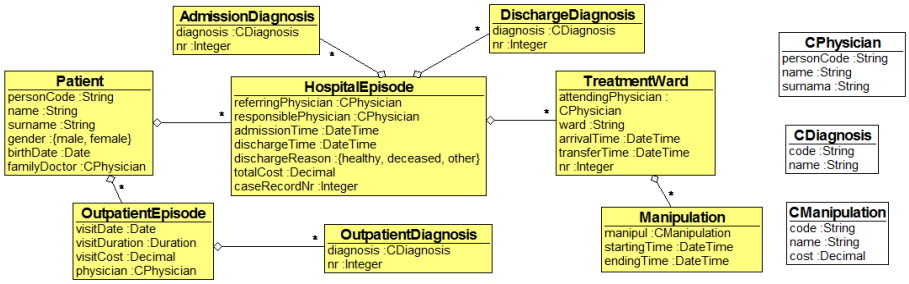

Fig. 2.1.7.4. Simplified version of hospital data ontology used in the teaching process.

2.1.7.5. Conclusion

The approbation of the language also showed several additional features that could alleviate formulation of queries like subset definition feature and attribute definition feature. Such features would allow one to write queries in more concise way than it was seen in examples above. These and similar features are currently under development and require some technical work to be implemented. Another useful feature would be to obtain event distribution in time, which could further be analyzed in MS Excel using its time axis component.

2.1.8. Development of the theoretical background for the implementation of distributed ontology- and web technology-based controlled natural ad-hoc query language parallel execution. (2-nd period)

Small businesses with sensitive data (like hospitals) require simple and cheap means for data analysis and ad-hoc queries. Traditional BI solutions are too expensive and heavyweight for them. We propose a solution based on the granular ontologies. We call a data ontology granular if its corresponding in-stances (data) can be divided into separate parts called slices. We propose an efficient implementation architecture for the parallel execution of ad-hoc queries based on distributed granular ontologies.

2.1.8.1 Introduction

Amount of data gathered by enterprises grows significantly every year. Even small and medium businesses are collecting data for reaching their business goals. Data alone do not guarantee a success – data should be transformed into information and it should be used accordingly in order to succeed. This process has been often referred as Business Intelligence (BI). Typically BI tools offer wide possibilities of data analysis however they require significant amount of investment and IT expertise. Typically a data warehouse should be built but it is a heavyweight and robust process. It is also an expensive process which is not al-ways affordable by small enterprises. Needless to say that the BI processes involve IT experts to translate the business requirements and queries to the language, which is understandable by the computer. For example, the Children's Clinical University Hospital (Riga, Latvia) gathers sensitive data of clinical processes. Data are stored in the relational database and maintained by the team of local IT experts. The IT experts have been heavily involved also in the BI processes translating manager's, clinician's and researcher's questions to the SQL queries. Although there are predefined reports, the business requirements are changing very often and, in fact, IT experts are overloaded that makes the business decision processes very slow and error-prone, because of miscommunication and hurry. Giving a business expert the direct access to data would be the solution, however the problem is that business experts do not possess the required skills to formulate the queries by themselves, because of complexity of query languages used to retrieve answers from data stores.

There are several proposals how to provide the means for business expert direct data access. One of the most known approaches – Self-Service Business Intelligence (SSBI) – has been proposed by Microsoft [1]. It provides a rich set of tools (Power BI) that allows end-user to build sophisticated data visualizations and make data analysis mainly through spreadsheet applications. Another approach called Ontology-Based Data Access (OBDA) [2] has been used by the Optique project [3]. The main idea behind OBDA is to provide a business expert with access to the data via an ontology that is specific to the business domain. The ontology hides from the business expert technical details and exhibits a business specific vocabulary of classes and properties that he is familiar with. The business expert formulates queries in terms of the classes and properties the ontology provides.

In order to enhance the possibility of direct data access by business experts we have already introduced notion of granularity for ontologies [4], [5]. In order for this to be understood without reading our previous work, in Section 2.1.8.2 we will briefly explain the basics of granular data ontologies. In Section 2.1.8.3 we introduce the architecture of parallel execution of queries on granular data and sketch the possible implementation strategies of granular ontology-based data access SSBI solution. We argue that it is possible to build more lightweight BI system which is more suitable for small and medium businesses, but could be scaled also for larger enterprises.

2.1.8.2 Granular Ontologies

We use UML Class Diagrams to depict data ontologies, and we depict the concrete data of the ontology as *legal instances* of the corresponding class diagram. The specification of legality can be performed either only through multiplicities, which must always be satisfied, or additionally through OCL expressions or in any other way (even using the natural language). We will only consider multiplicities as means for specifying instance legality constraints. The most important aspect of legality constraints is the one that demands certain number of instances at the other end of some link. For example, if a class diagram D contains classes A and B connected by an association with a multiplicity 1..* at the B end, then there can be no legal instance of such a class diagram, where there are some object a:A with no corresponding object b:B connected to it.

Let us assume we have a data ontology in a form of UML Class Diagram $\mathbf{D}$ and a class A belonging to the ontology $\mathbf{D}$. Let us also assume we have some legal instance $\mathbf{G}$ of the diagram $\mathbf{D}$. $\mathbf{G}$ consists of two kinds of elements – class instances called *objects* and association instances called *links*. Since we only operate with non-oriented associations, also the links are non-oriented. Therefore we can perceive $\mathbf{G}$ also as a non-oriented graph. Let us now take an arbitrary instance $\mathbf{x}$ of class A such that $\mathbf{x} \in \mathbf{G}$ (in shorthand notation: $\mathbf{x} \in A \cap \mathbf{G}$). We can now introduce a concept of a *slice respective to the object* $\mathbf{x}$ within instance $\mathbf{G}$ being the maximal subgraph of G (let us denote it with $\mathbf{S(x,G)}$), such that $\mathbf{S(x,G)}$ consists of the object x and all those objects that are reachable from $\mathbf{x}$ via edges.

When we inspect some data ontology $\mathbf{D}$, it always comes together with the set of its legal instances $\mathbf{U_D}$. We will now call a class $A \in \mathbf{D}$ a *Master class*, iff the two following statements are satisfied:

1) $\forall \mathbf{G} \in \mathbf{U_D} \forall \mathbf{x} \in A \cap \mathbf{G} \forall \mathbf{y} \in A \cap \mathbf{G} (\mathbf{x} \neq \mathbf{y} \Rightarrow \mathbf{S(x,G)} \cap \mathbf{S(y,G)} = \emptyset)$
2) $\forall \mathbf{G} \in \mathbf{U_D} \bigcup_{\mathbf{x} \in A \cap \mathbf{G}} \mathbf{S(x,G)} = \mathbf{G}$

The first statement states that all the slices corresponding to instances of the Master class are distinct, that is, they do not have common objects. The second statement states that these slices cover the whole instance $\mathbf{G}$. Data ontologies (together with the legality constraints), for which there exists a Master class, are called *granular* ontologies. We depict the Master class with a bold frame in granular ontologies (in case there is more than one Master class in an ontology, we just choose one).

In such a way data conforming to the granular ontology can always be sliced into slices having the same structure with a single identifying object of that particular class. Let's examine an example of granular ontology (see Fig. 2.1.8.1). It is a simplified ontology of the Children's Clinical University Hospital that covers basic clinical data.

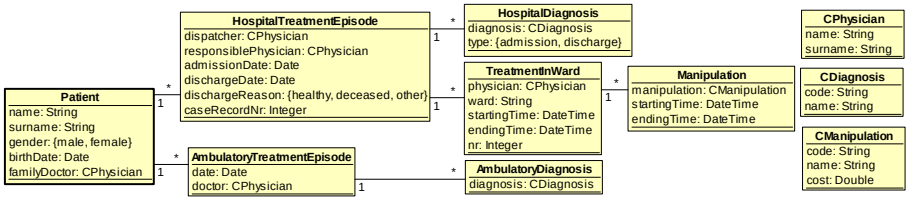

Fig. 2.1.8.1. Simplified ontology of the Children's Clinical University Hospital.

We can easily see that the class *Patient* is the master class of this granular ontology, because every other concept is tied to a patient. Any patient can be involved in several hospital treatment episodes and several ambulatory treatment episodes. Every episode can consist of several diagnoses. A hospital treatment episode consists of several treatments, each of which can have many manipulations performed on the patient. It must be noted that there are three so called classifier classes (*CPhysician*, *CDiagnosis* and *CManipulation*), to which we have not drawn associations from the basic classes, but instead used them as attribute types. This has allowed us making the ontology granular (the part of the ontology without including the classifier classes). These special classes are actually not logical classes belonging to the patient information. Instead, they are technical classes – classifications and registers – which are more or less persistent with regard to our logical ontology. In case of ER-type ontology, the logical and technical classes are concepts of the same level, and this does not help in understanding the ontology. In real world ontologies the two types of classes can usually be separated naturally making the logical part of the ontology granular.

The main gains of granular ontologies are: 1) Understandability of data schema; 2) Potential user-friendly query language; 3) Efficient query execution.

2.1.8.3 Querying granular data

In this section we discuss architecture for efficient query execution on the granular data. From the granularity definition follows that data in different slices are not connected. It means we can query each slice separately. Therefore it is possible to distribute data to multiple computational *nodes* (processing units, e.g., CPU-s or GPU-s, depending on implementation) and execute most of query logic in each *node* separately. Only small centralized overhead is required to process query results prepared by *nodes*. Each *node* may contain one or multiple data slices depending on the size of slices. We assume that each *node* contains multiple slices.

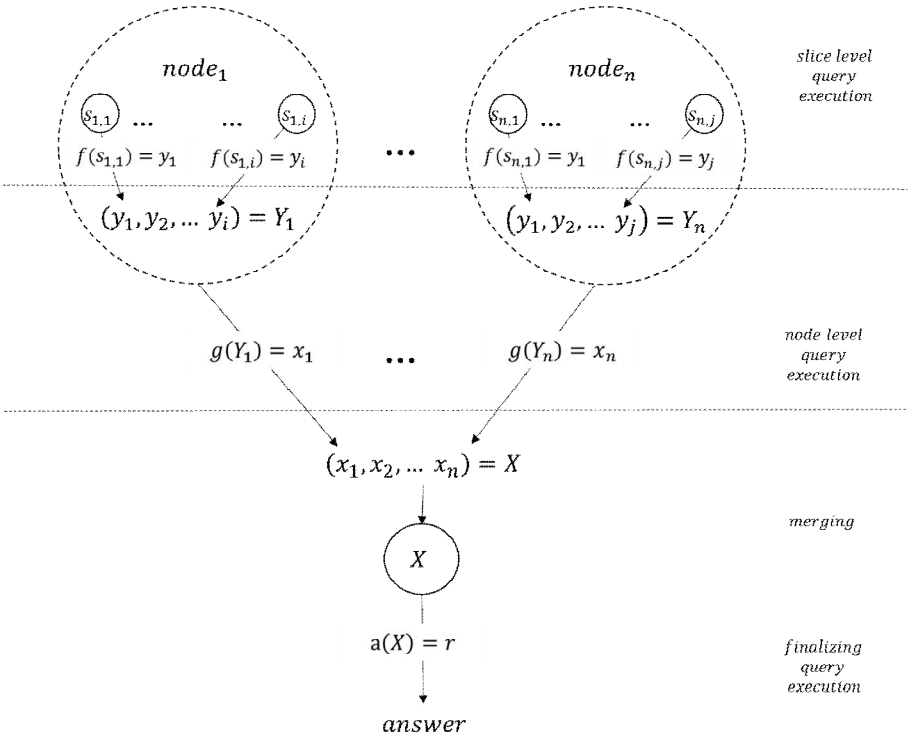

Fig. 2.1.8.2. Architecture of parallel execution of queries for distributed granular databases

Let's look at execution of query through three different levels of abstraction: slice, node, whole query (See Fig. 2.1.8.2). Query may be denoted by function. We introduce multiple functions (forming query together) to represent different levels of abstraction. Function $f(s_{i,j})$ is executed over single slice, function $g(Y_i)$ is executed over slices within single node, function $a(X)$ corresponds to whole query. Input for function f is one slice ($s_{i,j}$ in the Fig. 2.1.8.2). Input for function $g(Y_i)$ is a set (array) of results of function f executed over each slice in the node. Input for function a(X) is a set (array) of results of function g executed over each node. Output of function a(X) is an answer to the query. Type of the results of functions depends on the query kind. Result of the query is calculated in following way:

answer = $a(<g(<f(s_{1,1}),\ldots f(s_{1,i})>),\ldots\ g(<f(s_{n,1}),\ldots f(s_{n,i})>)>)$

Let's discuss in more details the function f. Function f has two parts. Function f_1 determines whether a slice is relevant to the query and function f_2 calculates an intermediate query result for this slice (used further to calculate whole query). Thus the input for f_1 and f_2 is a slice. f_1 returns *true* or *false*, f_2 is calculated only if f_1 returns *true*. In the input of function g results of f_2 are used. Only slices where f_1 returned *true* are used to compute the function g.

For better understanding lets discuss several real life query examples from the Hospital's use case shown in the Section 2.1.8.2.

Query 1 (Natural language): "number of patients, on which the manipulation Z is performed"

f_1: is the manipulation "Z" performed in this slice (for this patient)?

f_2: return 1 (We should count this patient if f_1 returns *true*.)

g: number of patients with manipulation "Z" in this node – actually sum of f_2 results;

a: count of patients with manipulation "Z" in all nodes – sum of *g* results;

Query 2 (Natural language): "average cost of all patients born in 2014"

f_1: is a patient born in 2014?

f_2: return the patient's cost – sum of all manipulation costs.

g: sum of patients costs in this node and patients count (in different nodes there may be different number of patients born in 2014);

a: average patient (born in 2014) costs;

Query 3 (Natural language): "find 10 most expensive patients (showing their name, surname and cost)"

f_1: *true* (any patient may be among the most expensive).

f_2: return patient's name, surname and cost (sum of all manipulation costs).

g: find 10 the most expensive patients in this node;

a: find 10 the most expensive patients among most expensive patients in all nodes;

As it was already stated we are interested in efficient and user friendly query execution. Some ideas regarding user-friendly query language have been discussed previously [5]. For the creation of user friendly and efficient query language we need an infrastructure were queries can be executed. An option is to compile queries in a high level (user friendly) query language to basic query operations. In this research we identify operations required to define queries typical for our case study.

It appeared that typically implementation of functions *g* and *a* requires only few aggregate functions. Like the ones used in query examples:

Query 1: g, a: sum(array<int>) -> int

Query 2: g: SumAndCount(array<decimal>) -> {sum:decimal, count:int}

a: average(array<{sum:decimal, count:int}> -> decimal

Query 3: g: TopX(array<{number:decimal, data:object}>,10) ->
array<{number:decimal, data:object})

a: MultiSetToSet(array<array<T>) -> array<T>, TopX

We need also functions: min, max, groupByKeyAndSum, orderBy.

Situation with elementary operations for functions f_1 and f_2 is more complicated. These functions perform deeper data analysis. There is only one instance of the master class in a slice. Typical queries use this instance as a pointer to further navigate to

instances required in query from this slice. Typical operations include: navigate by association, get attribute value, filter by attribute value, set operations like: multi set to set, first, single, contains, for all. As it is easy to see the typical operation set corresponds to LINQ [6] operations. Therefore for functions f_1 and f_2 we use a LINQ subset. LINQ queries for previously mentioned examples are as follows (p denotes instance of Patient, the master class in this example):

Query1: f_1: result = p.hospitalTreatmentEpisode
.SelectMany(ep => ep.treatmentWard)
.SelectMany(k=>k.manipulation)
.Any(m=>m.manipulation.code=="Z");

f_2: result = 1;

Query2: f_1: result = p.birthDate.Year == 2014;

f_2: result = p.hospitalTreatmentEpisode
.SelectMany(ep=>ep.treatmenthWard)
.SelectMany(k=>k.manipulation)
.Sum(m=>m.manipulation.cost)
.Value;

Query3: f_1: result = True;

f_2: int cost = p.hospitalTreatmentEpisode
.SelectMany(ep=>ep.treatmenthWard)
.SelectMany(k=>k.manipulation)
.Sum(m=>m.manipulation.cost)
.Value;

result = new { p.name, p.surname, cost};

An efficient implementation of LINQ for distributed datasets have already been developed [7] and in the proposed architecture LINQ is used to process a small and connected dataset (slice). Thus problems with efficiency are not expected.

2.1.8.4 Implementation Options

In this Section we sketch the possible implementation strategies of query execution for granular ontology-based data access SSBI solution. We consider two main directions for the implementation: 1) Parallel - GPU-based and 2) Distributed - Spark-based.

GPU-accelerated Database Management Systems (GADBMS) are usually in-memory databases that use graphical cards with multiple graphical processing units (GPU-s) alongside to CPU-s to execute queries and sometimes also render and visualize answers. GADBMS are typically hardware specific. They can be used with graphical cards of the specific vendor (e.g. NVIDIA) or that supports a parallel computing platform, like, CUDA or OpenCL [8],[9],[10].

For example, GPUdb is a big data object store and calculation engine that is accelerated with NVIDIA GPU-s. It is particularly fast for queries that need to scan

all data, e.g., sum or count. GPUdb runs efficiently on the cluster of five nodes where each node costs just 1000$. It executes queries and renders visualizations on 2 billion tweet records in less than a second [11]. Thus, such implementation is suitable for small and medium businesses because of low cost of infrastructure and efficient performance.

One of the biggest challenges for GADBMS is data placement strategy. Since GPU's memory is limited, deciding which part of data should be offloaded to GPU's memory is a difficult problem [8]. Granularity makes easier the efficient data placement. Since data consist of unlinked slices it may be distributed between GPU-s efficiently by just preserving borders of slices. GPU-s are responsible for calculating the result of the slice and node level calculations of the query, but CPU is used to merge the results.

Apache Spark is an open-source cluster in-memory computing framework [12]. Apache Spark allows to load and process data in the RAM of multiple machines. It supports multiple programming models for distributed computing. Resilient Distributed Datasets (RDD) are the fundamental programming abstraction which can be manipulated using the Spark Programing Interface that provides a set of transformations and actions over RDD-s [13]. Apache Spark is a suitable implementation environment also for our query execution engine.

Although the best performance can be achieved using in-memory databases, it is possible to build efficient solution also on file-system based data storages. The Apache Hadoop software library [14] is a framework that is used for the distributed processing of large data sets across clusters of computers. Simple programming models like MapReduce [15] are used within the framework. Data are stored on hard drives using HDFS (Hadoop Distributing File System). HDFS can be used to implement a custom distributed data processing solution. For example, Apache Hive is data warehousing software built on top of Hadoop (HDFS) and it can be used for ad-hoc querying. Apache Hive uses SQL-like language to describe queries [16].

We have discussed efficient implementation options of proposed granular ontology-based architecture that can be used to develop the cost-effective scalable querying system. However an open question remains, how to exploit the granularity in the development of user-friendly query languages for the SSBI.

2.1.8.5 Conclusions and Future Work

There are two main problems one must solve in order to build a usable query language: 1) how to provide an efficient implementation for a sufficiently expressive language; 2) how to make the language user-friendly, so that it can be used by non-programmers.

Since it is a big problem to satisfy both abovementioned conditions equally well, it is no surprise there are no query languages around providing both features. Our research concerning granular ontologies is a step in this direction.

We have proposed an architecture for query execution based on granular ontologies. The architecture is divided into several levels – slice level, node level and whole query level –, which are then merged together efficiently. We have also sketched the

basic operations used in each level, to which a user-friendly query language can be compiled. Finally, we have described the basis for implementation of the architecture. The implementation can be either GPU-based or Spark-based. We have a clear understanding, how the architecture can be efficiently implemented in both cases.

Our future work includes the implementation of the proposed architecture and the exploration of the kind of higher-level language constructs, which can be efficiently translated to our low-level operations. We are also planning to research more deeply the possibilities provided by the granularity feature itself, e.g., how far can we go exploiting it and still managing to implement a higher-level query language efficiently.

Since our main purpose is to provide good solutions to both abovementioned problems at the same time, we have to work on both of them simultaneously – to be able to efficiently implement the query language and to ensure that the language is user-friendly and easy to use by business experts (non-programmers).

References

1. Aspin, A.: Self-Service Business Intelligence. In: High Impact Data Visualization with Power View, Power Map, and Power BI, pp. 1-18, Apress (2014)
2. Kogalovsky, M.R.: Ontology-Based Data Access Systems. In: Programming and Computer Software 38.4 (2012)
3. Optique | Scalable End-user Access to Big Data, http://optique-project.eu/
4. Barzdins, J, Rencis, E., Sostaks, A.: Granular Ontologies and Graphical In-Place Querying. In: J. Grabis, M. Kirikova, J. Zdravkovic, and J. Stirna (Eds.), Short Paper Proceedings of the PoEM 2013, CEUR-WS, vol 1023, pp. 136-145 (2013)
5. Barzdins, J., Rencis, E., Sostaks, A.: Fast Ad Hoc Queries Based on Data Ontologies. In: H.M. Haav, A. Kalja, T. Robal (Eds.), Frontiers of AI and Applications, Vol. 270, Databases and Information Systems VIII, pp. 43-56, IOS Press (2014)
6. Rattz, J.C., Freeman, A.: Pro LINQ: Language Integrated Query in C#, Apress, (2010)
7. Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, Ú., Gunda, P. K., Currey, J.: DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In: OSDI, vol. 8, pp. 1-14 (2008)
8. Bellatreche, L., Saake, G.: GPU-Accelerated Database Systems: Survey and Open Challenges. Transactions on Large-Scale Data-and Knowledge-Centered Systems XV: Selected Papers from ADBIS 2013 Satellite Events 8920 (2015)
9. Nvidia: C.U.D.A. Compute unified device architecture programming guide. (2007)
10. Stone, J.E., Gohara, D., Shi, G.: OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in S&E, 12(1-3), 66-73 (2010)
11. Glaser, E.: GPUdb: GPU-Accelerated Distributed Database, http://on-demand.gputechconf.com/gtc/2015/presentation/S5484-Eli-Glaser.pdf
12. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Vol. 10, p. 10 (2010)
13. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 2-2). USENIX Association. (2012)
14. White, T.: Hadoop: The definitive guide. O'Reilly Media, Inc. (2012)
15. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp. 107-113 (2008)
16. Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, 2(2), pp. 1626-1629 (2009)
17. Zviedris, M., Barzdins, G.: ViziQuer: A Tool to Explore and Query SPARQL Endpoints, The Semantic Web: R&A, LNCS, vol. 6644, pp. 441 – 445 (2011)

18. Kharlamov, E., Solomakhina, N., Özçep, Ö.L., Zheleznyakov, D., Hubauer, T., Lamparter, S., Watson, S.: How semantic technologies can enhance data access at siemens energy. In: The Semantic Web–ISWC 2014, pp. 601-619, Springer (2014)

2.1.9. Further development of C6.0 classification algorithm and joining the international scientific research initiatives. (2-nd period)

Based on the SemEval-2015 competition results and their practical application in the LETA media monitoring automation, we were able to join an international consortium submitting and winning a Horizon-2020 project "SUMMA" under H2020-ICT-16 BigData-research call. Building on the C6.0 classification algorithm expertise we have developed a character-level neural translation methodology “G.Barzdins, S. Renals, D. Gosko. (2016). Character-level Neural Translation for NextGen Media Monitoring in the SUMMA Project. LREC-2016, 23-28 May 2016, Portorož (Slovenia).” and automatic thesaurus corpus-sample selection methodology “A.Spektors, I. Auzina, R. Dragis, N. Gruzitis, P. Paikens, L. Pretkalnina, B. Saulite. (2016). Tezaurs.lv: the Largest Open Lexical Database for Latvian, LREC-2016, 23-28 May 2016, Portorož (Slovenia).”

2.1.10. Research of competitive technologies for semantic graph parsing based on SemEval-2015 competition, and integrating them in semantic analysis toolchain for Latvian language, as used in LETA and elsewhere. (2-nd period)

After a successful participation in SemEval-2015 competition as described in publication [1], we were able to integrate these semantic graph parsing technologies and also the approaches used by other competitors [2,3] in the Latvian language semantic analysis toolchain developed in the 1-st period of this project. Applying this research enabled a significant improvement in the accuracy of semantic frame extraction – an improvement of F1-score from 57.6% to 74.6% for semantic frame target word selection, and 70.4% to 77.0% for frame element classification.

A prototype of the system was approbated by LETA news agency.

2.1.10.1. Introduction

The described toolchain allows to perform information extraction on newswire text in Latvian language, using plain text source documents to update an entity knowledge base with facts identified in those documents. The main components are:

- "MelnāKaste" – a text analysis pipeline (described in the next section) that performs text preliminary NLP analysis (morphology, syntax, NER) and afterwards its semantic analysis and fact extraction;
- Semantic database – a knowledge base describing the entities identified in text (people, organizations, etc) and facts – semantic frames that link those entities according to the ontology shown in Fig. 2.1.10.1.
- Modules for adding new documents, handling the issues of entity linking and ambiguity, and consolidation of overlapping facts from different sources.

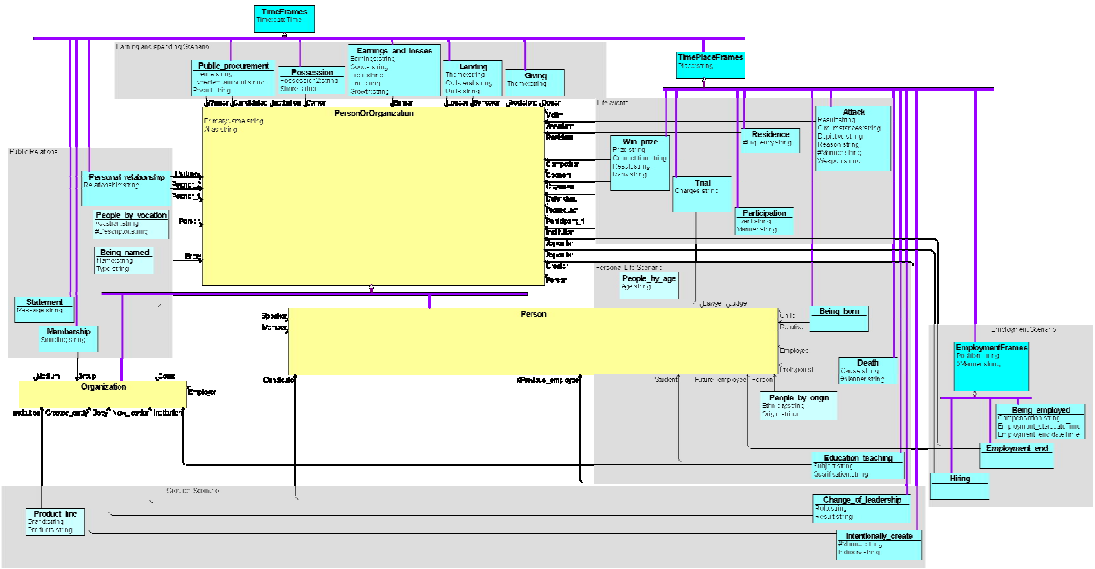

Fig. 2.1.10.1. The semantic frame ontology used in analysis.

2.1.10.2. The implemented semantic analysis pipeline

Text analysis in this pipeline is performed by a sequence of specialized modules. We apply the following analysis modules to the source text in order:

- Tokenization, morphological analysis and disambiguation/tagging;
- Named entity recognition;
- Sentence syntactic analysis to a dependency graph;
- Document-scope coreference resolution
- Semantic frame analysis.

Each layer of analysis is appended to a JSON-structured document describing all the annotation layers of each document. The system has been successfully been deployed to a highly parallized grid environment, enabling to process a large archive of historical news data efficiently.

The largest improvement to system accuracy comes from a modification of the semantic frame analysis layer. In our implementation, we augment our semantic frame analysis step with a run of a dependency parser (MALTparser in the current implementation) trained to parse semantic relations instead of the common syntactic relations, obtained by transforming the semantic graph to a dependency tree with extra information encoded in dependency lables, according to an algorithm described in detail in [1,2]. The results of this parser by itself are not sufficient for obtaining a usable frame semantics model (at least not with the limited number of training data available for the Latvian language), however, we then use these results as additional features to our C6.0 decision tree semantic classifier and obtain a significant accuracy improvement described in further sections.

2.1.10.3. System technical requirements and usage

The developed toolkit is supplied as two binary distributions – as a 32-bit Linux version and a 64-bit Linux distribution. The system is developed using Java and Python. The binary packages include an integrated OpenJDK Java 7 environment and Anaconda Python 3.4 to avoid the need for installing and configuring these environments. This allows the system to be installed on an appropriate Linux system by just unzipping the distribution files. The document analysis package can be launched by the following command:

$ ./run.sh [opcijas] <folder of input text documents> <output folder>

Configuration options:

—db-config=<path to db_config.py>

Location of database configuration file. The system will attempt to read a list of documents that need to be processed from table “dirtytexts” with the following columns: “filepath” (string), “status” (integer), “process_id” (string), “priority” (integer), “updated” (datetime); and will update the table with the processing status.

—list=<filename>

If this argument is provided, the system will read the list of documents that need to be processed with the configured file that should contain the relevant filenames only, one per line.

Additional options are described in the *readme.txt* file provided with the system distribution.

2.1.10.4. Quantitative improvements compared to previous version

In order to estimate system accuracy, we run the system on a set of manually annotated test data, and compare the results of this analysis with the semantic frames identified by human annotators. We then measure two particular types of classification:

1) Detecting the words that evoke a semantic frame ('target words' in frame semantics terminology), for example, the sentence elements that indicate that a "Hiring" event is described in that part of sentence.
2) Detecting each particular factual element of those semantic frames, for example, the entities describing the "Employer", "Position" or "Employee" in the abovementioned "Hiring" frame.

For each of those parts, we count the appropriate elements as annotated by humans ('gold count'), as annotated by our system ('silver count') and the number of exact annotation matches ('correct count'). From that we calculate the industry standard metrics of precision, recall and F1 score as the final estimate of system quality.

At the end of 1st period, the initial prototype was able to detect semantic frames with the following measured results:

ALL TARGETS Precision = 57.4%
ALL TARGETS Recall = 57.8%
ALL TARGETS F1 = 57.6%

ALL ELEMENTS Precision = 65.6%
ALL ELEMENTS Recall = 76.0%
ALL ELEMENTS F1 = 70.4%

After implementing the new methods for semantic analysis described above, we observe these improved results:

ALL TARGETS Precision = 68.9%
ALL TARGETS Recall = 81.3%
ALL TARGETS F1 = 74.6%

ALL ELEMENTS Precision = 85.4%
ALL ELEMENTS Recall = 70.1%
ALL ELEMENTS F1 = 77.0%

This shows a significant improvement in semantic frame analysis, especially in the recall ratio of semantic frame targets, and in reducing the false positive rate of semantic frame element identification.

2.1.7.5. Conclusion

Practical approbation of our earlier experiments with SemEval 2015 competition data showed that those methods can be applied not only for larger languages with a large amount of available training data, but in combination with decision tree based classifiers provide significantly improved results also for Latvian language, as tested incooperation with LETA news agency.

We provide a toolkit for analysis of Latvian language text that provides not only the simpler morphosyntactic and NER analysis layers but also a state of art system for semantic data extraction.

References

1. Guntis Barzdins, Peteris Paikens, Didzis Gosko. Riga: from FrameNet to Semantic Frames with C6.0 Rules. Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), pages 959–963, Denver, Colorado, June 4-5, 2015, Association for Computational Linguistics.

2. Yantao Du, Fan Zhang, Weiwei Sun, Xiaojun Wan. Peking: Profiling Syntactic Parsing Tree Parsing Techniques for Semantic Graph Parsing. Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), pages 459–464, Denver, Colorado, June 4-5, 2015, Association for Computational Linguistics.

3. Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic and Zdenka Urešová. SemEval 2015 Task 18:Broad-Coverage Semantic Dependency Parsing, Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), pages 915–926, Denver, Colorado, June 4-5, 2015, Association for Computational Linguistics.

2.2. Development of approaches, methods and algorithms for knowledge structure transformations and analysis, and design methodology of semantic network services.

2.2.1. Development of methods, algorithms and their support software for analysis of knowledge structure models (1-st and 2-nd periods).

During accomplishment of this task research has been done on several knowledge structure models and their applications, including the structural modeling, intelligent agents and intelligent tutoring systems. Most important scientific and practical result is the development of several methods and algorithms for system's structure formalization and transformation necessary for knowledge structure models used in structural modeling, as well as implementation of software prototype I4S (also IFS) for model representation and analysis. Methods, algorithms and their support tool have been described in "IFS User Manual v.1.0" as the result of this research. The first year students of RTU Doctoral program "Computer Systems" M.Pudane, S.Skele and H.Grinbergs participated in I4S testing. During the second period, a novel formal knowledge structure transformation method and algorithm has been developed. It is proposed to transform a morphological structure model (MSM) into a functional structure model in a behavior space (FSM BS) which is used as an intermediate model supporting the next step of transformation – construction of functional structure model in a parameters' space (FSM PS). The latter supports evaluation of functional state of complex industrial control systems on the basis of problem domain experts' knowledge about changes of parameter values caused by different faults. The new functionality of I4S has been tested by the first year student of doctoral study program „Computer Systems" E. Urtans. Using the I4S the structural models (Morphological structure model (MSM) and Functional structure model (FSM)) of control system for winch handling system (CSWHS) for the company "ICD Software" (Norway) has been developed. The final goal – evaluation of functional state of abovementioned system has not been reached due to the fact that cooperation with experts from „ICD Software" was interrupted (the new management of the company decided to delay this research).

Detailed description of I4S functionality is given in "IFS User Manual v.1.0". The formal transformation method and algorithm which is implemented in I4S is based on Structural Modelling (SM) approach where the focus is centered on a continuous and unified view on the system under investigation. The main model is a morphological structure model (MSM) from which a functional structure model (FSM) is made. The following decomposition of FSM includes a functional structure model in a behavior space (FSM BS) and on the base of FSM BS a functional structure model in a parameters' space (FSM PS) is built. The topology of models provides consistency between different models and granularity of MSM, when decomposition is continued. The existing consistency between structure models allows to realize transformations

of them. In SM the transformation is the knowledge transfer or transition from one topological space to another, which is realized using algorithms. On the bases of the formal method of graph theory for directed graphs model transformations algorithms were implemented in I4S.

To obtain FSM topology in a space of functions, which further is transformed to FSM BS and FSM PS, from MSM, a transformation algorithm, which consists of 3 steps is used: 1) MSM is considered as undirected graph; 2) Nodes of FSM are acquired (transformation into a line graph); 3) Nodes of FSM are connected. The analysis of different technical and non-technical systems explored the necessity to invent changes in MSM notation. There were introduced also a logical operators for execution of flows. Therefore also the whole transformation algorithm was changed and automated logical operator transfer from MSM to FSM was implemented. New algorithm also consists of three steps: 1) MSM where logical operators are depicted is considered as an undirected graph; 2) Nodes of FSM are acquired; 3) Nodes of FSM are connected taking into account 3 rules:

1) All nodes of MSM are inspected sequentially, starting from the selected node;
2) If two incident arcs in MSM have opposite directions, then corresponding nodes in the FMS are disconnected;
3) If two incident arcs in MSM have the same direction, then logical operators and conditions are verified. Complying with imposed restrictions in FSM arcs between nodes are created.

Due to invented changes in the new transformation algorithm a different number of arcs was obtained (comparing to the previous transformation algorithm), second logical operators were represented. Consequently a correct set of arcs were obtained and incompatible connections were excluded from the model.

Considering the performed functionality of different systems it was observed that in the FSM exist several function connection variants that are acquired performing transformations. As a result there were described 5 different flow combinations in MSM and structure model transformation cases using logical operators:

1) one flow at the object's input side and one at the output side;
2) one flow at the object's input side and many at the output side;
3) many flows at the object's input side and one at the output side;
4) many flows at the object's input side and many at the output side;
5) one or more flows at the object's input side and no one at the output side or no one input flow and one or more output flows.

Further step was an implementation of formal transformation algorithm from MSM to the FSM BS. The algorithm consisting of three steps was created: 1) select the type of FSM BS (displaying behaviour space or separate behaviour states); 2) obtain behaviour state pairs that are connected with flows, and represent them in nodes; 3) connect nodes with arcs. Afterwards the FSM PS was derived from the behaviour model using transformation, and in the nodes parameter sets or parameters were represented. To acquire FSM PS (in which separated parameters are represented), following steps were performed:

1) Inspect FSM PS where are parameter sets;

2) Using expert`s knowledge each node is decomposed and parameters acquired, as well as defects that exist in the viewed parameter set;
3) Using expert`s knowledge, logical operators are added to the model.

The function structure model in a space of parameters that is acquired in the transformation is used to evaluate rejection of elements and consequences of faults and also to construct events tree that expand the FSM SP usage capabilities. Taking into account flow combination and logical operators, for structure models also production rules were created that support different reasoning mechanisms (structural, diagnostic, causal). Depiction of I4S application and automatically generated structural models for industrial control system is given in "Depiction of structural models of control system for winch handling system (CSWHS)".
Methodological and technological support for acquiring a topological model has been developed. Scientific results of this research provide formal description and analysis of different kinds of systems without complex mathematical formalisms allowing to reason about systems' functionality and possible faults. Models of nontechnical systems are applicable for software code generation. This would allow examining and using of theoretical results in practice.
The initial research phase has been carried out towards the development of formal method for evaluation of concept map complexity based on criteria used in Systems Theory. It is proposed to interpret for concept maps, which are one of knowledge structure representations, the four criteria applied for estimation of systems complexity – the number of system's elements and relationships between them, attributes of systems and their elements, and the organizational degree of systems, and use them for evaluation of concept maps. More details can be found in "Grundspenkis J. Initial Steps towards the Development of Formal Method for Evaluation of Concept Map Complexity from the Systems Viewpoint. Submitted to the 12th International Baltic Conference on Databases and Information Systems".
During research on knowledge structures used in intelligent agents, the mechanism for introduction of knowledge structure changes has been developed. This includes the general conceptual mechanism for introduction of changes, the ontology for experimental needs and the mechanism for adapting of existing rules to new/incomplete data. Analysis of methodologies for integrated design of multiagent systems and ontologies was done at introductory stage of this research [1]. For achievement of research goals and providing experiments the prototype of room cleaning multirobot system simulator which supports manual introduction of changes of knowledge structure into ontologies and rule bases of agents that simulate robots was developed during the first stage. Some research has been done to carry out the comparative analysis of knowledge structures used in knowledge bases and deductive data bases [2]. To enable intelligent agents to change their knowledge structures based on both user inputs and the results of machine learning the work on knowledge structures used in intelligent agents has been continued in the second period by developing more detailed architecture of the knowledge representation and learning framework. The rule based learning approach was added to the previously created ontology based knowledge structure. A concept of a multi-agent system management tool was introduced in the system architecture. The basic functionality of the tool is implemented, namely definition of the environment, configuration of the multi-agent system and the ontology used. A paper has been written reflecting the work done so far, namely the developed conceptual approach – "Lavendelis E. A Conceptual Approach for Knowledge Structure Update and Learning in Multi-Agent Systems.

Submitted to 16th International Conference on Applied Computer Science (ACS '16), Istanbul, Turkey, April 15-17, 2016".
During the third stage of the project a rule based reasoning mechanism will be implemented into multi-agent system for premises cleaning working in a simulated environment. Finally the machine learning mechanism will be implemented into the simulated environment and tested during the fourth stage. A technical report will be written at the end of the fourth stage describing the framework, all the methods and results achieved. The aim of the research is to provide a mechanism to significantly increase autonomy and adaptivity of multirobot systems which is a topical problem in practical use of multirobot systems.
The method is developed for the identification of the knowledge change source in the pedagogical model which is an integral part of the pedagogical module in intelligent tutoring systems. This method allows to choose more precisely and to adapt tutoring strategies to individual learners by analyzing both his/her current knowledge state and the influence of the tutoring situation on learner's emotional state. The architecture of agent-based affective tutoring system is proposed that involves emotion ontology sharing among agents simulating human-tutors and students for the evaluation of tutoring strategies adaptation "Petrovica S., Pudane M. Simulation of Affective Student-Tutor Interaction for Affective Tutoring Systems: Design of Knowledge Structure. Submitted to 7th International Conference on Education and Educational Technologies (EET '16), Istanbul, Turkey, April 15-17, 2016". During the next stage it is planned to develop ontology based hierarchical structure of pedagogical model for the implementation of dynamically adapted tutoring process (based on learners knowledge and emotions), as well as it is planned to continue research on proposed agent-based affective tutoring system by developing knowledge structure ensuring agent interaction and communication.

References

1. Svetlana Otto. Daudzaģentu sistēmu un ontoloģiju projektēšanas metožu integrēšanas metodoloģiju analīze Master thesis defended, 2015. Scientific advisor - prof. J.Grundspeņķis.
2. Emil Bikjanov. Zināšanu bāzu un deduktīvo datu bāzu salīdzināšanas analyze. Master thesis defended, 2015. Scientific advisor - prof. J.Grundspeņķis

2.2.2. Study of related works in the domain of structural compatibility between processes, enterprise architectures and knowledge, as well as development of initial draft for ideal linkage model (1-st period).

Several new theoretical results have been achieved during analysis of role of time dimension and knowledge structures describing time. They allow introducing time dimension in enterprise architectures and their management. New theoretical results have been reached, including extended and enriched time model which could be linked with enterprise architectures and Bunge ontology [7.1m]. During studies on processes important links which are necessary to provide continuous requirement engineering have been identified, which is important practical result for software development. More details in "Kirikova, M. Enterprise Architecture and Knowledge Perspectives on Continuous Requirements Engineering. Proceedings of REFSQ-2015 Workshops, Research Method Track, and Poster Track co-located with the 21st International Conference on Requirements Engineering: Foundation for Software Quality. Essen, Germany, March 23, 2015. CEUR-WS.org, Vol. 1342, ISSN 1613-0073, pp. 44-51".

2.2.3. Development of approaches and methods for identification of knowledge structure and process compatibility (2-nd period).

On the basis of research done in the previous period ("Kirikova, M., Penicina, L., Gaidukovs, A. Ontology based Linkage between Enterprise Architecture, Processes, and Time. ADBIS 2015 short Papers and Workshops BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD Proceedings. Poitiers - France, September 8-11, 2015. New Trends in Databases and Information Systems. CCIS, Vol. 539. Springer, pp.382-391. In database: SCOPUS. DOI: http://dx.doi.org/10.1007/978-3-319-23201-0_39") the understanding on knowledge structure and process compatibility was extended by analysis of related works where four compatibility criteria were identified, namely: "understandability" of the business process by the process users and information technology; transferability of the knowledge related to business process; acceptance of the technology used for supporting the business process; and reflection of the operating conditions by processes and their supporting information systems. While the criteria concern both tacit and explicit knowledge, in this period, the main focus was on explicit knowledge, when working on the compatibility identification approach that concerns the identified criteria and well known workflow data patterns. For moving towards compatibility identification methods, the FREEDOM framework was developed that relates operational business processes to its development and management functions, thus forming the basis for requirements engineering for continuous process development ("Marite Kirikova. Continuous Requirements Engineering in FREEDOM Framework: a Position Paper. Joint Proceedings of REFSQ-2016 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016), March 14-17, 2016, Gothenburg, Sweden. CEUR-WS.org, Vol. 1564."). The framework settles the template on what kind of processes and what kind of knowledge structures are to be analyzed for being able to support business processes continuously by needed information technology solutions. The framework includes such functions as future and current situation representation development, requirements engineering, fulfillment engineering, design and implementation, operations, and management. The framework prescribes continuous feedback acquisition, monitoring, analytics and audit among the afore listed functions.

Regarding fulfillment engineering, the main requirements problems in multi-project environment were analyzed and an approach for their handling was developed in order to enable continuous requirements engineering. More details can be found in "Anita Finke. Requirements Inheritance in Continuous Requirements Engineering. Joint Proceedings of REFSQ-2016 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016), March 14-17, 2016, Gothenburg, Sweden. CEUR-WS.org, Vol. 1564"

In the context of operational business processes, possibilities to reflect knowledge about business object states were investigated. The method for document compatibility analysis was developed and approbated on educational processes. It is described in "Artūrs Bartusevičs. The Development and Implementation of Model-Driven Software Configuration Management Solutions. Defended RTU Promotion Council P-07, September 2015, scientific supervisor - prof. L.Novickis".

The experiments with identification of security requirements patterns, as knowledge structure, and business process compatibility as well as possibilities to introduce continuous information security audit were done in cooperation with parallel international project ITSE: “Improvement of IT-Security in Enterprises based on Process Analysis and Risk Patterns (ITSE)”, involving university partners from: Estonia, Latvia, and Germany, URL: *http://hochschulkontor.lv/en/projects/247* ; thus gaining research synergy and promoting further collaboration with Tartu University. More details can be found in “Dmitrijs Kozlovs, Kristine Cjaputa and Marite Kirikova. Towards Continuous Information Security Audit. Joint Proceedings of REFSQ-2016 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016), March 14-17, 2016, Gothenburg, Sweden. CEUR-WS.org, Vol. 1564”. The FREEDOM framework is intended to be related to the integrated model of business processes and enterprise architecture based on Bunge Wand and Weber information systems ontology that was developed under the grant Nr. 342/2012. The connection is conceptually already made between the integrated model and the time dimension, (more in “Marite Kirikova, Raimundas Matulevičius, and Kurt Sandkuhl. The Enterprise Model Frame for Supporting Security Requirement Elicitation from Business Processes, submitted to the 12th Baltic Conference on Databases and Information Systems to be held 4-6 July 2016, Riga, Latvia.”), which, in related works and in our research [1], was recognized as important factor in knowledge structure and process compatibility. A considerable effort was made for finding appropriate modeling environment for further refinement of approaches and models. An ADOxx platform was investigated for this purpose. Information about DIMOD platform developed in the parallel project of SOPHIS program was acquired too, to consider it is an alternative for the modeling environment.

References

1. Andrejs Gaidukovs. Laika dimensija biznesa procesu un uzņēmumu arhitektūras modelēšanā. Master thesis defended, 2015. Scientific advisor - prof. M. Kirikova.

2.2.4. Development of demonstration prototype for integration of semantic network services into e-logistics portal (1-st period).

Several practical results are obtained during accomplishing this task. SADI (Semantic Automated Discovery and Integration) technology is used for implementation of semantic web services. It facilitates their implementation and maintenance. The service preparation template is developed to provide automated generation of plug-ins and additional files thus facilitating and accelerating a service development. Semantic web services which semantically annotate information received from traditional web services that are included in e-LOGMAR portal have been developed. Processed information is related to the logistics domain (available routes and their types, cargo expenses etc.). The logistics domain ontology which is built using OWL language supports annotation. OpenCalais is used in services to integrate natural language processing methods with semantic web technologies, allowing to retrieve semantic metadata from the text. The service returns semantically annotated document to the client. Semantic web technologies have been integrated also into model driven software configuration approach. More details in "Bartusevičs, A., Novickis, L., Lesovskis, A. Model-Driven Software Configuration Management and Semantic Web in Applied Software Development. No: Recent Advances in Telecommunications, Informatics and Educational Technologies: Proceedings of the 13th International Conference on Telecommunications and Informatics (TELE-INFO '14), Turcija, Istambul, December 15-17, 2014. Istambul: WSEAS Press, 2014, 108-116 p. ISBN 978-1-61804-262-0" and in "Bartusevičs, A., Lesovskis, A., Novickis, L. Semantic Web Technologies and Model-Driven Approach for the Development and Configuration Management of Intelligent Web-Based Systems. No: Proceedings of the 2015 International Conference on Circuits, Systems, Signal Processing, Communications and Computers, Austrija, Vienna, March 15-17, 2015. Vienna: 2015, 32-39 p. ISBN 978-1-61804-285-9. ISSN 1790-5117". In the future research it is planned to develop a general methodology based on past practical experience for introduction of semantic web services.

2.2.5. Analysis of the related studies and researches that are necessary to define the basic steps of the Semantic Web service development methodology (2-nd period)

All the tasks have been performed within the general Activity A 1.9: Development of Semantic Web Services Integration Framework and Methodology.

Several results are obtained during accomplishing this task.

1. Several solutions have been studied and analyzed before starting Semantic Web services development: OWL-S, Web Service Modelling Ontology (WSMO) and Semantic Automated Discovery and Integration (SADI). Following the results of study and analysis, SADI technology has been selected for implementation of semantic web services. It facilitates their implementation and maintenance. The service preparation template is developed to provide automated generation of plug-ins and additional files thus facilitating and accelerating a service development.

2. Semantic web services which semantically annotate information received from traditional web services that are included in eLOGMAR portal (www.elogmar.eu) have been developed. Processed information is related to the logistics domain (available routes and their types, cargo expenses etc.). General methodology consists from the following steps (described in more detail in "Novickis L., Vinichenko S., Sotnichoks M., Lesovskis A., Graph Models and GeoData Based Web Portal in Cargo Transportation. In : Scientific Journal of Riga Technical University. Applied Computer Systems, 2015/17, RTU Press, Riga, 2015, pp. 34-39. ISSN 2255-8683 (EBSCO, VINITI)":

> 2.1. For each departure and destination point (ports, railway stations, cities/warehouses) portal administrator has to define the areas around them by selecting a radius and fixing it. Firstly, portal administrator has to define GPS coordinates (Latitude – Lat., Longitude – Long.) for all points of departure and destination by clicking on the Google Map. Then he must define a radius and areas around points of departure and destination.
>
> 2.2. Actors from Transport Group using the set of interface forms entry to the portal database transportation tariffs and services for selected freight route.
>
> 2.3. Actors from Cargo Group create on-line transport request and receive data on cargo transportation rates and estimated transit time.

If corresponding request can't be provided with just one transportation mode (sea-sea, rail-rail, or road-road), then processing procedure for intermodal transportation is applied.

The procedure is based on the use of graph model of possible transportation routes between the points of departure and destination.

Intermodal transportation includes several services (freight route, tariff rates, expected transit time) which are connected by GPS coordinates of point of departure and destination.

Taking into account defined radius, points are united into one sector.

Each sector represents a node of route's graph.

2.4. The procedure searches the ways on graph model between points of departure and destination defined in on-line transport request.

2.5. Total transportation rates and estimated transit time are calculated. Data of separate services are sum up.

As a result of calculating all possible ways and their sections (lists of nodes) of transportation are presented in a table.

The logistics domain ontology which is built using OWL language supports annotation. OpenCalais is used in services to integrate natural language processing methods with semantic web technologies, allowing to retrieve semantic metadata from the text. The service returns semantically annotated document to the client. These results are described in "Novickis L., Mitasiunas A., Ponomarenko V. Towards Knowledge and Information Technology Transfer Concept and Its Validation. In: Procedia Computer Science, ICTE in Regional Development , 2015 , Valmiera, Elsevier, Volume 77, 2015, Pages 48–55".

3. Semantic web technologies have been integrated also into model driven software configuration management approach. Results are presented in "Artūrs Bartusevičs. The Development and Implementation of Model-Driven Software Configuration Management Solutions. Defended RTU Promotion Council P-07, September 2015, scientific supervisor - prof. L.Novickis".

Modified MTM (Model – Transformation – Model) approach based on MDA (Model-Driven Architecture) and Semantic Web Technologies is developed. It makes easier to implement repeated usage and integration of Software Configuration Management data. Research results are described in "Bartusevics, A., Lesovskis, A., Novickis, L. Semantic Web Technologies and Model-Driven Approach for the Development and Configuration Management of Intelligent Web-Based Systems. No: Proceedings of the 2015 International Conference on Circuits, Systems, Signal Processing, Communications and Computers, Austrija, Vienna, 15.–17. marts, 2015. Vienna: 2015, 32.–39.lpp. ISBN 978-1-61804-285-9. ISSN 1790-5117" and "Bartusevics, A., Novickis, L. Model–Based Approach for Implementation of Software Configuration Management Process. In: MODELSWARD 2015. Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, France, Angers, 9-11 February, 2015, pp.177-184. ISBN 978-989-758-083-3".

The future research is closely related to Activity A 2.6:

Validation and Enhancing of general methodology and integrated software framework based on practical experience for introduction of semantic web services in the fields

of e-Logistics (in cooperation with Logitrans Consult OU, Estonia) and software configuration management (in cooperation with Tieto Latvia).

Dissemination activities (A 4.2):

- International eINTERASIA conference in innovative IT solutions, knowledge and technology transfer was held in Almaty (Kazakhstan) in August 27-28, 2015 (prof. L.Novickis from RTU, Latvia was the Conference chairman). About 100 participants from EU and Central Asian countries took part in the conference. The SOPHIS results devoted to the development of semantic web services based Software framework and intelligent agents approach were presented at the conference (authors : prof. J.Grundspenkis, prof.L.Novickis, as.prof.E.Lavendelis from RTU). www.einterasia.eu
- The 4th International Workshop INTEL-EDU 2016 will be held in conjuction with 15th International Conference on perspectives in business informatics research BIR 2016 in Prague, September 2016. www.bir-conference.org
- Presentation at the conference The International Conference on Circuits, Systems, Signal Processing, Communications and Computers, Austrija, Vienna, March 15-17, 2015
- Presentation at the conference The 3rd International Conference on Model-Driven Engineering and Software Development, France, Angers, 9-11 February, 2015

2.3. Model based data visualization and real-time verification of business processes

2.3.1. Development of technologies for large scale NoSQL data base exploration and visualization.

New possibilities are researched in large scale data set analysis and visualization for new type of hardware – high resolution displays wall, consisting of many (more than 20) standard displays. In this research client-server environment is developed. This environment supports agent based modelling and relational data exploration and migration to NoSQL database with browser, that works with display wall.

First stage of research was devoted to create a prototype of display wall. Main research problems were compatibility with popular operation systems and to keep cost of display wall as low as possible. Different solution architectures were analyzed and display wall prototype was developed partially according to raised requirements. Results of the first stage are published in "Rudolfs Bundulis, Guntis Arnicans. Concept of virtual machine based high resolution display wall. In Information, Electronic and Electrical Engineering (AIEEE), 2014 IEEE 2nd Workshop on Advances in, pp. 1-6. IEEE, 2014. DOI: 10.1109/AIEEE.2014.7020317", "Rudolfs Bundulis, and Guntis Arnicans. Virtual Machine Based High Resolution Display Wall: Experiments on Proof of Concept. International Conference on Systems, Computing Sciences and Software Engineering (SCSS 14), Electronic CISSE 2014 Conference Proceedings, 2014." and "Rudolfs Bundulis, Guntis Arnicans, and Rihards Gailums. NVENC Based H.264 Encoding for Virtual Machine Based Monitor Wall Architecture, GPU Technology Conference, San Jose, Mart 17-20, 2015".

Second stage of the research is devoted to improvement of the prototype of display wall. Main research problem is to optimize amount of data transferred between computer and display wall. Different software and hardware compression methods are explored. One of possible usage of the display wall was explored – development of agent based modeling and simulating environment. Results of the second stage are published in "Rudolfs Bundulis, Guntis Arnicans. Use of H. 264 real-time video encoding to reduce display wall system bandwidth consumption. In Information, Electronic and Electrical Engineering (AIEEE), 2015 IEEE 3rd Workshop on Advances in, pp. 1-6. IEEE, 2015" and "Ingars Ribners, Guntis Arnicans. Concept of Client-Server Environment for Agent-Based Modeling and Simulation of Living Systems. In Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on, pp. 83-88. IEEE, 2015. DOI: 10.1109/CICSyN.2015.25".

2.3.2. Business process runtime verification

Computerized system analysis and operation correctness evaluation during runtime in operational environment is understood as runtime verification in this research. Correctness evaluation can be done by tools built into system or by system events external monitoring. This research focuses on the last one. Verification is done according to each processes' verification description – model, where is defined events that confirm correctness of each process step, their execution sequence and execution time restrictions.

Prototype for runtime environment controlling system was developed in the first stage of the research. It fixes runtime environment events and via autonomous agents sent them to the controller. Controller monitors environment events and verifies them according to verification model.

In second stage of the research prototype developed in first stage was used for real life business process verification to measure additional workload to information system added by runtime verification process. Obtained measurements show, that additional workload for information system is negligible. It shows practical usability of proposed business process runtime verification mechanism.

Results of the research are included in two publications submitted to publishing.

2.3.2.1. Introduction

Information technologies provide unprecedented opportunities to automate many processes of human life. Actions which have been the preserve of human beings only a few decades ago can be executed by programmable equipment now. But the mankind's progress has also brought up new challenges. One of them is complexity of computing systems. The authors [1] refer to as „computing systems with complexity approaching boundaries of human ability". The IBM autonomic computing manifesto [2] claims: "It's time to design and build computing systems capable to manage themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them".

One of the possible solutions of this problem is to entrust at least some of complex IT supervisory processes to the systems themselves. Wikipedia [3] defines self-management in computer science as "the process by which computer systems shall manage their own operation without human intervention". Peter Van Roy [4] defines self-managing systems in the following way: "systems that can maintain useful functionality despite changes in their environment." IBM autonomic computing manifesto defines the self-management by four fundamental self–* features: self–configuration, self–healing, self–optimization and self–protection. Later [1] the "self–chop" was extended to eight self–management properties. Today the number of identified self–management properties reaches 20 and more [5], and it continues to increase.

There are two possible ways trying to implement the self-management in information systems: (1) to construct an "autonomous supervisor" – to develop autonomous information system for supervision of other computer systems, or (2) to implement the properties by adding "independent" components (or add-ins) to the

system.

The autonomous supervisor idea is consistent with the nature of the autonomic computing – independent autonomic components solve self–management problems even without knowing about existence of other components (like natural live organisms). Many of these solutions are implemented using “agents” that are able to provide information about specific events to the autonomous supervisors. This is rather a universal solution but unfortunately it is for the time being faced with serious difficulties and has not found wide application.

The implementation of self-management properties by adding “independent” components to information systems is a rather practically oriented approach. A group of specific solutions should be developed, partially built-in into the target systems, to enhance the non-functional features of information systems with self-management properties. Such solutions are called “smart technologies” [6].

We introduce four new self–management properties (see Fig.2.3.2.1):

- self–management properties to support information system operation: (1) run-time verification - control whether internal processes executing is compliance with business measures, (2) environment testing - control of interaction with the external environment;
- self–management properties for maintenance support (3) business model incorporation – built-in business process model lets to change the functionality of the information system by updating the business process descriptions, (4) self-testing – built-in component for internal system process control usable also in a productive mode.

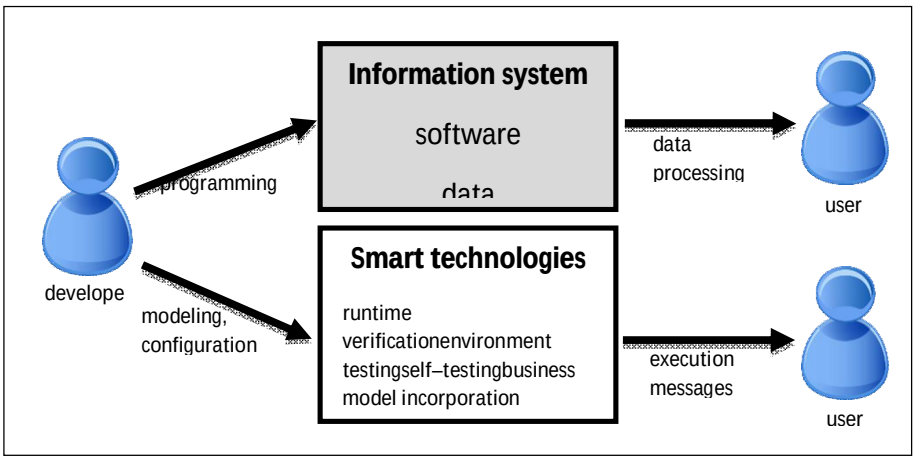

Fig.2.3.2.1. Smart Technologies (overview)

The approach of smart technologies offers to implement self–management properties into the architecture of system. Since each self–management property is simple to implement, the approach becomes rather applicable and it can be used even by rather small team of developers or companies (40-50 employees). At the same time it is the weak point of this approach: self–management properties can be developed practically from scratch for every information system. To minimize this impact the authors suggest developing self–management properties as framework applicable for wide spectra of systems. Thereby we discusse smart technology framework which brings software development towards the objectives of IBM autonomic computing manifesto.

This research is a continuation of the research described in [7] and [8]; therefore some of sentences from these papers are cited to keep completeness of the research.

2.3.2.2 Related work

Autonomic computing and smart technologies have a similar goal – to reduce the complexity of system use by delegating some part of user support functions to the information system itself. The autonomic computing manifesto declares a vision of fully independent computer systems that are able to self-management. The manifesto lists four aspects of autonomic computing:

- Self-configuration - automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly.
- Self-optimization - components and systems continually seek opportunities to improve their own performance and efficiency.
- Self-healing - system automatically detects, diagnoses, and repairs localized software and hardware problems.
- Self-protection - system automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent system wide failures.

In 2003 IBM extended the list to eight characteristic aspects [1], adding system's ability to "know itself" and manage its resources, system's ability to know its environment and the context surrounding its activity, and act accordingly – to adjust, operate in heterogeneous environment accordingly its open standards, as well as anticipate the optimized resources needed while keeping its complexity hidden. The fundamental concept in autonomic computing is the idea of self-regulation and the self-governing operation of the entire information system, thus disburdening users and administrators from complexity of system's use and maintenance. Achievements of autonomic computing movement during its first decade after publication of the manifesto have been explicitly demonstrated in [9].

Later the list of self-management features was extended with new ones, discussed by other authors [5]. The closest from the perspective of this paper is self–diagnosis – a system's ability to analyze itself in order to identify existing problems or to anticipate potential issues. Some of self-properties discussed in this paper are contained by self–diagnosis: testing of execution environment and run-time verification. These properties support system users during its runtime: at first users can verify if the system is running correctly in changing environment and then – whether all of business tasks are accomplished correctly.

Other two of smart technologies are aimed to the support of system maintenance process. The first of them is self–testing: it intends to include testing components into system itself. Practical experience shows that this solution helps to verify software correctness not only during software development likewise it would be done by traditional testing tools but also after the system is taken in production and the maintenance has been begun. However it helps to verify systems correctness in real productive environments.

The last one of smart technologies, business model incorporation, applies to the self–configuring: it is a system's ability to (re)configure itself by (re)setting its internal parameter values to achieve high-level policies or business goals [5]. The existence of this property potentiates implementing the principles of Model Driven Development (MDD). MDD provides business process model integration with computer system, thus allowing updating systems functionality by changing business processes definitions.As of now, manifesto's targets have been met only to some

extent. Paradoxically, to solve the problem — make things simpler for administrators and users of IT — you need to create more complex systems. Continuing efforts on autonomic systems include both, theoretical research and practical implementation [9]. Although many of self–properties are introduced, there is still place for innovative implementations [10]. Many of these are provided as individual compact solutions like smart technologies of this framework.

2.3.2.3 Self-management for system operation

The studies of system life cycle [11] usually focus on the problems of system development. Usually software developer teams are IT professionals and experts therefore they have practically no problems with use of complex technologies. Many of them consider implementing of several non-functional features like self–management properties to be a waste of time and resources.

On the other hand the complex technologies of nowadays lead to complex solutions with awkward usability. Thereby information systems sooner or later are upgraded to improve their usability. The “ordinary” users of information systems become a target audience of self–management properties because they often have difficulties in overcoming of IT complexity. On this account the main focus of the next chapter discussing two smart technology features will be devoted to the support issues for better and easier information system’s operation (exploitation) instead of software development phases.

Runtime verification

Context

The business process runtime verification is the self-management property which allows verifying whether the business process is executed correctly and in compliance with all of time restrictions. This property is particularly useful when business process is supported by two or more loosely coupled information systems or some of business process steps are not automated at all. There are only few cases when system itself contains component verifying correctness of business process execution. Usually it is assumed that the development time testing ensures correct execution of process in the end user environment. However correctness of business process execution can be verified by checking all of process steps in all linked systems and rather often it cannot be done by users (time restrictions) or an individual system.

Likewise runtime verification is required to prevent conflicts of different processes and systems in collaboration, where one part of the process is done by people, and the other part is supported by software. The software can be designed to support particular processes in different environments at different time frames.

Runtime verification has been well known for years in the area of embedded systems. It is an approach to computing system analysis and execution based on extracting information from a running system and using this information to detect and possibly to react to observed behaviors satisfying or violating certain properties. Such defense mechanisms may be included in the system during its development or they may be included as independent controls from the base process. The independent character of such mechanism allows making later adjustments by adding or disabling the controlling component when a system is developed, and changes are made. These ideas can be applied in business process runtime verification, too.

Solution

The authors [12] propose a solution for business process runtime verification (see Fig. 2.3.2.2) using three objects - verification model/description, agents and controller:

- a verification description contains instructions about the correct execution of the base process;
- an agent is a software module for registering of base process execution events;
- A controller compares the events received from agents with the permissible ("correct") events described in the verification model. As a result, the controller may discover the incorrect behavior of base processes. If inconsistencies are detected, the controller sends messages to the user.

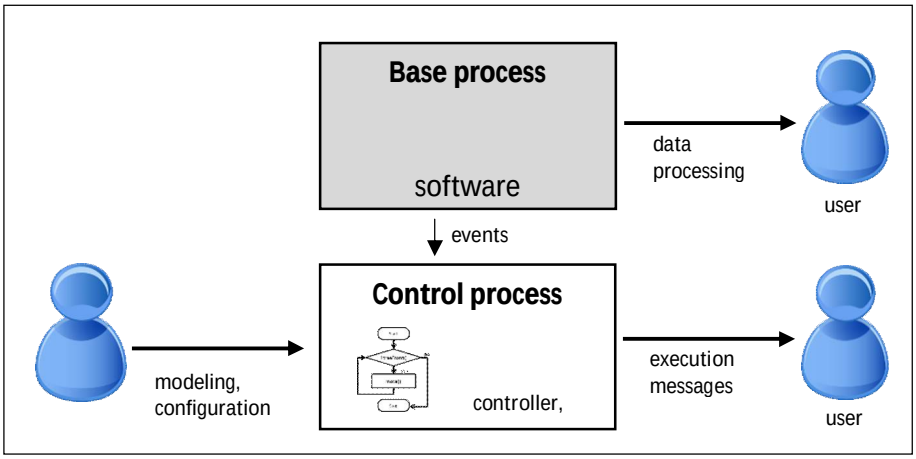

Fig. 2.3.2.2. Smart technology component: Runtime verification

If the base process is already described by a graphical model, the verification process can be created from the base model by indicating those process steps which will be carried out in the runtime verification process.

Results

The developed runtime verification solution was piloted in bank's electronic clearing system (ECS). It identified file processing process delays and some of processing bottlenecks. Some of these delays could be avoided if run–time verification would be introduced sooner.

The piloting results leads to 3 main conclusions – (1) solution provides convenient instrument for the tracking of business process execution, (2) the solution is able to detect business process execution defects, and (3) data processing system verification process creates a tiny extra load for the involved information systems infrastructure.

The solution provides a number of interesting possibilities, which bring us closer to the goal defined by ideas of autonomic computing:

- The verification process can be defined without modifying the base process - the base process can have more than one verification process so as to verify all of its various aspects;
- The verification process runs in parallel to a base process and does not interfere with it;
- Process verification can be added dynamically to legacy systems;

- Verification does not depend on modeling language used for process description; it depends only on possibility of verification agents to identity events of the base process.

Likewise, some solution limitations must be taken into account: verification mechanism can detect only those base process steps which leave some modifications in the computer systems „memory".

Environment testing

Context

The environment testing is the self–management feature for controlling and monitoring of operation (execution) environments to ascertain all involved operation environment fit to the requirements necessary for successful running of the information system. The requirements can relate to operating system, network characteristics, workstation parameters, etc. Discrepancy between the information systems requirements to external environment and the concrete execution environment may occur in several situations:

- Workstation may be incompatible in various means: insufficient memory or processor performance, inadequate network connection and other technical parameters. These are cases when execution environment verification may be done once.
- Workstation may use external resources and availability of these resources may vary during execution time. E.g., some of web services may be unavailable because of lost network connection on server downtime. Obviously in these cases execution environment should be verified continuously.
- Workstation settings do not comply with software requirements: directory structure does not contain all of required subfolders or required permissions, decimal separator must be the symbol ",", data base server must be reachable, etc.
- Developers sometimes assume that software, which works in development environment, will keep working after it is deployed elsewhere, hence encoding some assumptions about the environment into the program. As a result, when the software is installed in other environment, which is different from the development environment, the software may fail or work only partially correct.

Practical use of information systems shows that many incidents and failures are not related to the functionality of the information system itself, but rather are caused by inadequate infrastructure and the execution environment. It means information systems must be accompanied by automatic means for external environmental testing.

The authors do not deal with automated modification of external software environment as it is offered by other researches or tools [13]. It is assumed that one workstation may execute more than one system components simultaneously. Thereby automated external component updates for one system could affect execution of other systems.

Solution

The authors [14] propose a technology, which allows independent environment checks, performed by the software, named – "checker", in order to validate if the execution environment is suitable for normal execution (see Fig. 2.3.2.3). The proposed solution implies gathering these requirements in a "software profile" to be able to validate the execution environment before program's starting. Only if the results of all checks are satisfactory, the program can be considered prepared for work at a given environment, otherwise the session is stopped, giving the user an explanation, why it is not possible to perform work.
A program execution profile is a document achieved when all the requirement descriptions of software are combined together. The profile can be formalized as a separate document and supplemented to typical software deliverables such as code and documentation. The main, but not the only use of the profile is validation of execution environment during program use.

The practical environment testing task is carried out by "checker", which manages environment validation modules- drivers. Each driver is an atomic unit, which enforces validation of a single type of requirement; this is done by reading information from the environment and comparing it to reference values.

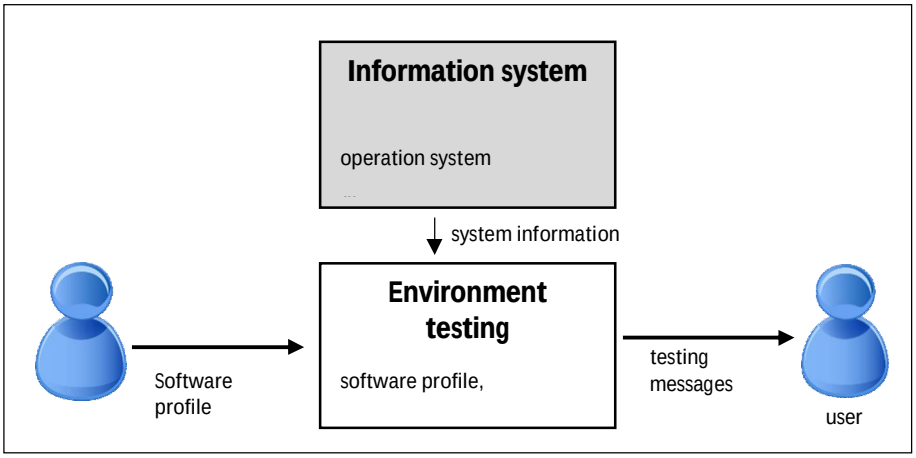

Fig. 2.3.2.3. Smart technology component: Environment testing

To be able to modify the set of checks to be performed without modifying the program code, information about the checks (both the algorithms and reference values) must be stored outside the code of base system – in the software profile. Domain specific language (DSL) was developed for effective software profile definition: it allows describing all external objects and their properties required by the developed software. Software profile is developed by qualified IT professionals as it requires deep knowledge of system execution environment. This concept differs from other approaches used in practice – both from the ones, which validate the environment straightaway after installation or updating, and from the others, which try to "hide" the checks in source code.

Results

The proposed solution since 2009 is used in a number of local information systems in Latvia. An execution environment testing was usually performed when supplying a new version of the information system. The new version was installed only after the current execution environment was checked for its ability to run the new version. Also, receiving alarms from users about the systems malfunctions there was first tested if the execution environment of the concrete workstation meets the environment requirements. In many cases, missing or wrong components of the execution environment were the reason for malfunctions.
The described approach can also be used for other purposes, for instance to monitor the computer systems that are in use in company's internal network and to check the compliance of configurations with standards set by the company. The practical implementation showed that development of the proposed approach requires relatively little programming resources.

2.3.2.4 Self-management for system maintenance

This chapter is dedicated to the support, which may be provided to the information systems by self–properties during maintenance of these systems. After the first version of information system is deployed to the operation environment, it is will be updated or modified several times to comply with real user requirements. This leads to regular changes being introduced into the information systems.

In turn, this means that: (1) change requirements must be defined, (2) the software must be updated accordingly, (3) each of software versions must be tested (it should include regression tests and tests for new or updated functionality), (4) software should be deployed to the runtime environment and, if it requires, system data should be migrated. Furthermore usually system execution may be stopped just for rather limited period of time. The authors will discuss two self–properties introduced for support of software maintenance.

Business model incorporation

Context

Business model incorporation is a self-management feature allowing to adapt the functionality of information system without (or with minimal) coding effort, just by changing graphical business process descriptions. One of the implementation options is to apply Model Driven Development (MDD) principles for software development.
Model driven development provides a range of advantages for the system development, maintenance and execution [15], [16]. Same of main advantages are: (1) MDD provides high level of business process abstraction thus providing less error-prone description, meaningful validation and exhaustive testing, (2) MDD bridges the gap between business and IT, (3) MDD captures domain knowledge, (4) MDD results in software being less sensitive to changes in business requirements, (5) MDD provides up-to-date documentation since the models describe the essential issues of the information system’s usage. How should the system be developed to gain advantages provided by MDD?

Solution

At the beginning of information system development the business processes (see Fig. 2.3.2.4) should be described as the information system will be designed to support them. A set of business process descriptions are created using DSL, and it serves as business process model. Graphical representations like diagrams can easily be understood and used by domain experts (as a rule, non-IT specialists) for the business process description. After the business process model is created, the information from the diagrams can be transferred to the database of an information system, and it is a task for IT professionals. The business process descriptions are embedded into the information system, and the engine of the information system can interpret information born from the diagrams. Embedded business processes ensure that the information system behaves according to the business process model.

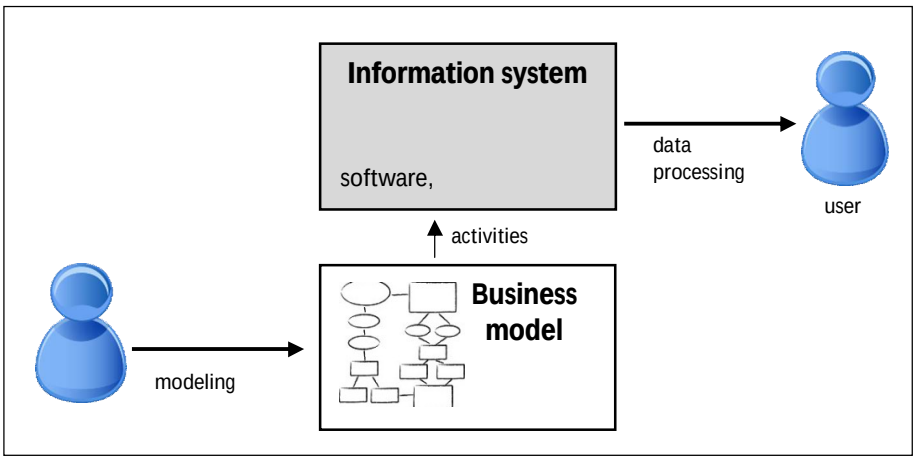

Fig. 2.3.2.4. Smart technology component: Business model incorporation

However, the proposed business process incorporation approach differs radically from the model driven architecture (MDA): MDA offers a complete application generation using business process specification described in unified modeling language (UML). If the business processes are changed, the changes must be implemented in the software specification and then new software should be generated.

The proposed approach of smart technologies provides business process execution engine running according business process definition (domain specific language or DSL is used for process description). It provides the possibility to develop flexible applications with user-friendly interfaces which can be implemented for each of systems independently. Furthermore, business processes can be updated without software modifications, and the functioning of the information system can be updated by changing of business process descriptions, without programming.

Results

The solution described in this chapter leverages use of DSL as language for business process definition providing user – friendly method for process description. Many years there were no sophisticated tools for DSL development available. Even if the DSL could be defined, the supporting tools did not provide user-friendly editor's generation for end users. This problem is solved now: There are tools allowing to define not only the DSL syntax, but also a graphical representation and to generate the corresponding graphical editor. The authors used the platform DIMOD for DSL

development [7] which is developed using the tools generation platform GrTP. More detailed GrTP review is given in [17] and [18].
As practice shows [19], it is possible to create a special tool for transfer of model's data to executable application relatively quickly. The API of the graphical editor can be used to access the model's repository, to gather the information and to transfer it to applications database. When business model is added to the system database, there is no more need for DSL editor repository in the system's execution environment. In turn, it provides numerous advantages for system performance and usability. Thus guaranties that the application operates according to the model developed in a graphical DSL. And the overall quality of the application – usability, reliability, security, performance etc. – is dependent on the application itself, not on the hypothetical ability of a code generator to create an application in the desired quality.

Self-testing

Context

Self-testing is the self-property providing the software with a feature to test itself automatically prior to operation. There is similarity between self-testing and hardware self-checking where computer tests its own readiness for operation when it is just turned on. The purpose of self-testing is to use a built-in support component for automated execution of previously accumulated tests cases not only in test environment, but also in operation (production) environment.

Solution

Self-testing contains two components:

- Test cases of system critical functionality that check the set of functions without which the software could not be used. Identification of critical functionality and designing of tests for it is a part of the requirement analysis and testing process.
- A built-in automated testing mechanism (regression testing) providing automatic execution of tests and comparison of results with benchmark values. These features are typically a part of traditional testing tools.

Implementation mechanism of self-testing approach [20] uses an idea and means of the software instrumentation, which is already known from the 70-ies. Testing operations are put by programmers into certain places of the source code; these points are named as test points. Testing operations allow to track the changing values and to compare them with a benchmark. Thus it is possible to check the correctness of the information system. Unfortunately, this solution is usable only for that software whose development is in the user's influence sphere.

Results

It should be noted that the idea of built-in support for program testing has been offered quite a while ago ([21], [22]) and it has been implemented in some projects. Regardless the system environment (Development, Test and Production) self-testing functionality can be used in the following system modes (see Fig. 2.3.2.5):

1. Test capture mode - new test cases are captured or existing tests are edited/deleted.

2. Self-testing mode - automated self-testing of software is done by automated execution of stored test cases.

3. Use mode - there are no testing activities – a user simply uses the system. The built-in self-testing mechanism can be used in emergency situations to find out the internal state of the system, which may facilitate the analysis of the causes and consequences of the emergency situation.

4. Demonstration mode. The demonstration mode can be used to demonstrate system's functionality. User can perform system demonstrations using use cases stored in storage files.

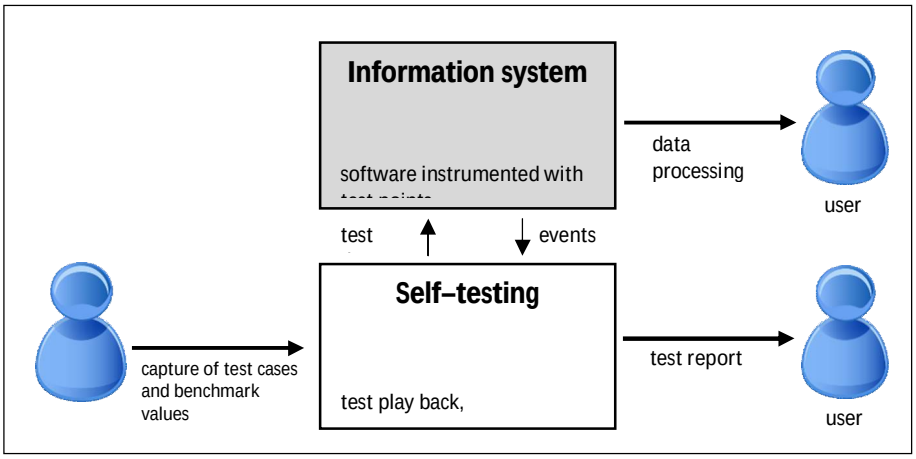

Fig. 2.3.2.5. Smart technology component: Self-testing

The implementation of self-testing feature can be done in ca. 3000 LOC in C++. Additionally, the source code of the particular system should be instrumented with testing activities like accumulating of test cases and executing of them. These investments are justified when the system is designed and developed for long-term use.

2.3.2.5. Conclusion

The approbation of the language also showed several additional features that could alleviate formulation of queries like subset definition feature and attribute definition feature. Such features would allow one to write queries in more concise way than it was seen in examples above. These and similar features are currently under development and require some technical work to be implemented. Another useful feature would be to obtain event distribution in time, which could further be analyzed in MS Excel using its time axis component.

There are four smart technology features provided and described in this research. These extends variety of software self–properties and allow to achieve goals similar to autonomic computing – facilitating the use and maintenance of systems by including support components in them. These technologies provide support for information system execution and maintenance:

- business process run–time verification allows to describe and perceive processes, regardless of systems supporting these processes, to verify

correctness of process execution and compliance with time restrictions; this self-property is used during system execution;

- execution environment testing provides verification of system's external environment requirements;
- business model incorporation within system allows to change implemented business processes without (or with minimal) software modifications; this self–property is intended for software maintenance;
- self–testing is intended to use during system maintenance and it allows to verify correctness of systems execution when some software changes or environment updates are done;
- building of smart technologies into information systems requires additional work; the proposed smart technologies have advantages when the information systems are used by many users without profound IT knowledge and the cooperation between the customer and the supplier is long-term;
- According to authors' experience smart technologies can be used even in a small to medium size IT company with 40-50 employees.

The smart technologies which are described above achieve the autonomic computing initiative goals only partially. There may be still a vast variety of smart technologies which would be useful to explore and implement practical systems. For instance, these would include – data quality control, access control, performance monitoring, availability monitoring which are easy enough to implement for a small/medium size organization. Authors are not discussing the research directions where no practical implementations of technologies are achieved yet.

References

1. Kephart, J., Chess, D. (2003) The Vision of Autonomic Computing, IEEE, Computer Magazine 36: 41-52, doi:10.1109/MC.2003.11600552003.

2. Horn, P. (2001) Autonomic Computing: IBM's Perspective on the State of Information Technology. IBM, 2001. http://libra.msra.cn/Publication/2764258/autonomic-computing-ibm-s-perspective-on-the-state-of-information-technology.

3. Wikipedia (2016) https://en.wikipedia.org/wiki/Self-management_(computer_science) (accessed 05.02.2016)

4. Peter Van Roy. (2007) Self Management and the Future of Software Design https://www.info.ucl.ac.be/~pvr/facs06VanRoyFinal.pdf

5. Lalanda, P., McCann, JA., Diaconescu, A. (2013) Autonomic Computing: Principles, Design and Implementation, 2013 – Springer, 288 p

6. Bičevska, Z., Bičevskis, J. (2007) Smart Technologies in Software Life Cycle. In: Proceedings of Product-Focused Software Process Improvement. 8th International Conference, PROFES 2007, July 2-4, 2007 (Münch, J., Abrahamsson, P., eds.), Riga, Latvia, vol. 4589/2007, 2007. pp.262-272.

7. Bicevskis Janis, Bicevska Zane (2015) Business Process Models and Information System Usability ScienceDirect, Procedia Computer Science 77 (2015) 72 – 79..

8. Bičevska, Z., Bičevskis, J., Oditis, I. (2015) Smart Technologies for Improved Software Maintenance. Preprints of the Federated Conference on Computer Science and Information Systems pp. 1549–1554.

9. Kephart, J. (2011) Autonomic computing: the first decade. ICAC 2011: 1-2.

10. Dobson, Simon; Sterritt, Roy; Nixon, P.; Hinchey, M. (2010) Fulfilling the Vision of Autonomic Computing, IEEE Journals, Volume: 43, Pages: 35 - 41, DOI: 10.1109/MC.2010.14

11. Roger S.Pressman. (2010) Software Engineering. A Practioner's Approach. The McGraw-Hill Comp., Inc., 2010.

12. Oditis, I., Bicevskis, J. (2015) Asynchronous Runtime Verification of Business Processes. In Proceedings of the 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), Riga, 2015, pp. 103-108.

13. Installshield (2016) Installation Information (accessed 05.02.2016) http://www.sevenforums.com/performance-maintenance/137478-installshield-installation-information.html

14. Rauhvargers, K., Bicevskis, J. (2009) Environment Testing Enabled Software – a Step Towards Execution Context Awareness. In: H.-M. Haav, A. Kalja (eds.), Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, vol. 187, IOS Press, (2009), 169–179.

15. Johan Den Haan (2009), 15 reasons why you should start using Model Driven Development

    http://www.theenterprisearchitect.eu/blog/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development/ 2009

16. Johan Den Haan (2015). Opening up the Mendix model specification & tools ecosystem

    http://www.theenterprisearchitect.eu/blog/2015/10/30/open-mendix-model-specification-and-tools-ecosystem/ 2015

17. Barzdins J., Zarins A., Cerans K., Kalnins A., Rencis E., Lace L., Liepins R., Sprogis A., GrTP: Transformation Based Graphical Tool Building Platform. [WWW] http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-297/ (accessed 01.08.2014)

18. Sprogis (2014) Configuration Language for Domain Specific Modeling Tools and Its Implementation Baltic J. Modern Computing, Vol. 2 (2014), No. 2, 56-74.

19. Cerina-Berzina, J., Bicevskis, J., Karnitis, G. (2011) Information systems development based on visual Domain Specific Language BiLingva Selected Papers from the 4th IFIP TC 2 Central and East Europe Conference on Software Engineering Techniques, CEE-SET 2009, Krakow, Poland, LNCS 7054 Springer (2011), 124-135.

20. Diebelis, E., Bičevskis, J. (2013) Software Self-Testing. In: Proceedings of the 10th International Baltic Conference on Databases and Information Systems, Baltic DB&IS 2012, July 8-11, 2012, Vilnius, Lithuania. IOS Press, vol. 249, 2013. 249 – 262.

21. Bichevskii, YY, Borzov, YV. (1982) Prioriteti v otladke bolsih programmnih sistem Programmirovanie, 1982, vol. 3, pp. 31-34 (in Russian). (Bichevskii Ya.Ya., Borzov Yu.V.. Priorities in debugging of large software systems. PROGRAM. & COMP. SOFTWARE. 8:33, 129-131, 1983).

22. Chengying, M., Yansheng, L., Jinlong, Z. (2007) Regression testing for component-based software via built-in test design. In: Proceedings of the ACM symposium on Applied computing, March 11 - 15, 2007, Seoul, Korea, 2007. pp.1416-1421.