FP7-ICT-611140 CARRE

Project co-funded by the European Commission
under the Information and Communication Technologies
(ICT) 7th Framework Programme

D.6.1. DSS Runtime Infrastructure

Rafał Kłoda, Jan Piwiński, Robert Ugodziński

December 2015

CARRE Contacts

Project Coordinator:	Eleni Kaldoudi	kaldoudi@med.duth.gr
Project Manager:	George Drosatos	gdrosato@ee.duth.gr

DUTH Democritus University of Thrace	Eleni Kaldoudi	kaldoudi@med.duth.gr
OU The Open University	John Domingue	john.domingue@open.ac.uk
BED: Bedfordshire University	Enjie Liu	Enjie.Liu@beds.ac.uk
VULSK: Vilnius University Hospital Santariškių Klinikos	Domantas Stundys	Domantas.Stundys@santa.lt
KTU Kaunas University of Technology	Arūnas Lukoševičius	arunas.lukosevicius@ktu.lt
PIAP Industrial Research Institute for Automation & Measurements	Roman Szewczyk	rszewczyk@piap.pl

Disclaimer

CARRE is a Specific Targeted Research Project partially funded by the European Union, under FP7-ICT-2013-10, Theme 5.1. “Personalized health, active ageing & independent living”.

Document Control Page

Project

Contract No.:	611140
Acronym:	CARRE
Title:	Personalized Patient Empowerment and Shared Decision Support for Cardiorenal Disease and Comorbidities
Type:	STREP
Start:	1 November 2013
End:	31 October 2016
Programme:	FP7-ICT-2013.5.1
Website:	http://www.carre-project.eu/

Deliverable

Deliverable No.:	D.6.1
Deliverable Title:	DSS Runtime Infrastructure
Responsible Partner:	PIAP
Authors:	Rafał Kłoda, Jan Piwiński, Robert Ugodziński
Input from:	All partners
Peer Reviewers:	Allan Third (OU), George Drosatos (DUTH)
Task:	T.6.1 DSS Runtime Infrastructure
Task duration:	9 months: 1 February 2015 to 31 December 2015
Work Package:	WP6: Domain specific empowerment & decision support services
Work Package Leader:	PIAP – Rafał Kłoda
Due Date:	31 December 2015
Actual Delivery Date:	14 January 2016
Dissemination Level:	PU
Nature:	R & D
Files and format:	Deliverable report: 1 pdf file Software source code available from project web site, see Annex 1: https://www.carre-project.eu/innovation/dss-runtime-infrastructure/
Version:	03
Status:	☐ Draft ☒ Consortium reviewed ☒ WP leader accepted ☒ Coordinator accepted ☐ EC accepted

Document Revision History

Version	Date	Modifications	Contributors
v01.0	1 December 2015	Deliverable – outline	Jan Piwiński Rafał Kłoda
v01.1	10 December 2015	VULSK input to DSS	Domantas Stundys
v01.2	29 December 2015	Draft version	Rafał Kłoda Jan Piwiński Robert Ugodziński
v01.3	8 January 2016	PIAP internal review	Roman Szewczyk
v02	11 January 2016	Reviewed version	Jan Piwiński
v02.4	13 January 2016	Updated after review (Allan and George)	Jan Piwiński Robert Ugodziński
v03	14 January 2016	Edited for uniformity and conformance to QA	Eleni Kaldoudi

Table of Contents

List of Figures

List of Tables

Executive Summary

Patient empowerment and decision support services (DSS) is a personalised service for disease progression management. This CARRE component is an event driven service, which supports and enable real-time decision support for patients. The DSS in CARRE requires, firstly the risk assessment of comorbidities probability occurrence based on input from risk association model and other variables, secondly suggests treatment guidance by providing alarms (based on a particular patient's observables) and education materials that are suitable to a particular patient's needs, thirdly provides identification of important changes, provide advices and personal life-style guidance to the patient, based on monitoring of current medical treatment data in order to manage risks for comorbidities or progression of disease to more severe stages.

Work package 6 "Domain specific empowerment & decision support services" aims to address the aforementioned challenges. The 6.1 task "Development of DSS runtime infrastructure" requires the development of integrated infrastructure to support the development and maintenance of real-time decision support and empowerment services, both for a particular patient and for the medical professionals. The platform will serve as an infrastructure for the further forecasting analytics, which are part of the objectives of this WP and aims to deliver system functionality aggregated into manageable and reusable modules, which ease communication with other CARRE system components.

This document is a deliverable report of 6.1 "DSS runtime infrastructure" of WP6 in CARRE project. In particular it covers tasks designated in Task 6.1. This deliverable report focuses on the design and initial implementation of DSS components in order to provide personalized services to both Patient Application and Medical Expert to be able to provide run-time decision based on current status of incoming data to the CARRE RDF Repositories.

About CARRE

CARRE is an EU FP7-ICT funded project with the goal to provide innovative means for the management of comorbidities (multiple co-occurring medical conditions), especially in the case of chronic cardiac and renal disease patients or persons with increased risk of such conditions.

Sources of medical and other knowledge will be semantically linked with sensor outputs to provide clinical information personalised to the individual patient, to be able to track the progression and interactions of comorbid conditions. Visual analytics will be employed so that patients and clinicians will be able to visualise, understand and interact with this linked knowledge and take advantage of personalised empowerment services supported by a dedicated decision support system.

The ultimate goal is to provide the means for patients with comorbidities to take an active role in care processes, including self-care and shared decision-making, and to support medical professionals in understanding and treating comorbidities via an integrative approach.

Terms and Definitions

The following are definitions of terms, abbreviations and acronyms used in this document.

Term	Definition
API	Application programming interface
CARRE repository	The backend major component of the CARRE platform responsible for storing, indexing and accessing the public and private RDF data
DoW	Description of Work
DSS	Decision Support Service
LOD	Linked Open Data cloud
PHR	Personal Health Record
RDF	Resource Description Framework: a standard model for data interchange on the Web.
RESTful API	A Web API that adheres to the REpresentational State Transfer architectural constraints
SPARQL	A RDF query language.
Triple	A statement in the subject-predicate-object expression
XML	Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

1. Introduction

This document reports on the infrastructure design and development of the Decision Support Service (DSS). This CARRE system component is a personalised service for disease progression management and is mainly responsible for supporting and maintaining of real-time decision support for patients.

Decision making in healthcare is a complex process in terms of number of parameters and variables, outcome possibilities and amount of information must be processed[1].

Decision support systems (DSS) can assist patients and provide to him advices, recommendations and diagnosis of problems in cardiorenal domain, where the optimal solutions for a given sort of data about the possible consequences are determined similar as human experts in the field.

A modern intelligent decision support system not only provides access to data and models. It is also a significant development in the field of analytical data processing, data warehousing and artificial intelligence-aided methods of knowledge discovery in databases i.e. data mining.

The present document is following up on project tasks and their reports as well as system solutions created until now and enable to understand how DSS environment work and how it will be implemented in ultimate CARRE system. Hereby, we discuss how the DSS is informed by already submitted CARRE deliverables[2]:

- *D.2.1 Domain analysis & use case definition.* The document gives an insight on the overall domain analysis, including medical domain analysis, overview on comorbidities management, patient`s empowerment as well as the definition of CARRE main scenario. All of the above have been taken into account in the design of CARRE DSS, especially following initial use cases dedicated to DSS.
- *D.2.2 Functional requirements & information model.* This document presents the first version of the CARRE ontology, by providing a general scheme for the description of data model for risk, intended to represent current medical knowledge regarding cardiorenal risk factors and their interactions. This document is considered as fundamental for DSS in terms of risk factors association as well as their relationships and their attributes (patient observables, observables conditions, ratio type, ratio value, medical evidence, diseases symptoms, treatment, sensor input, alarm, educational material, etc.). The CARRE information model has already been used by DSS infrastructure, having in mind the functional requirements dedicated to DSS that need to be met.
- *D.2.4 CARRE metadata scheme and & ontology.* This deliverable presents the first, complete version of the CARRE ontology as well as comprises the set of tools developed for crafting the CARRE vocabulary as well as software artefacts, which take into account interactions with CARRE services, like DSS.
- *D.2.5 CARRE Architecture.* CARRE architecture reports on the architectural design of the overall CARRE services environment. The CARRE architecture is described through the definition of the major software components and the description on how these components operate with each other. This deliverable provides an overall design for the CARRE integrated environment and takes into account the various data sources harvesting and information extracting components,
- *D.4.1 Semantic repository design & implementation.* The document presents the semantic repository developed for managing large volumes of data in the form of RDF triples - information stored in the CARRE repository, which act as the central point of information storage for DSS.
- *D.4.2 Schema mapping & metadata enrichment.* This report discusses the various external vocabularies used, together with the alignment of these vocabularies with the CARRE ontology. The document presents the data sources, ontologies and vocabularies used for annotating and enriching the data collected within CARRE, which are now under processing by DSS.

[1] Upkar Varshney: A model for improving quality of decisions in mobile health, Decision Support Systems, Volume 62, June 2014, pp. 66-77

[2] https://www.carre-project.eu/project-info/deliverables/

Figure 1 represents graphically aforementioned relations of DSS with previous deliverables. The DSS infrastructure is a basis for currently running tasks under WP6: *T.6.2 Development of personalized service for the patient*, which captures relationships among many variables and risk factors to allow to determine the potential decision based on particular set of conditions; *T.6.3 Development of personalized service for the medical expert*, where the user-friendly repositories explorer is planned to enable experts to prepare specific decision for particular patient; *T.6.4 Tools for adapting existing and creating new services*, which provide adaptive platform for customizing previously developed services to CARRE system during the integration phase of the project.

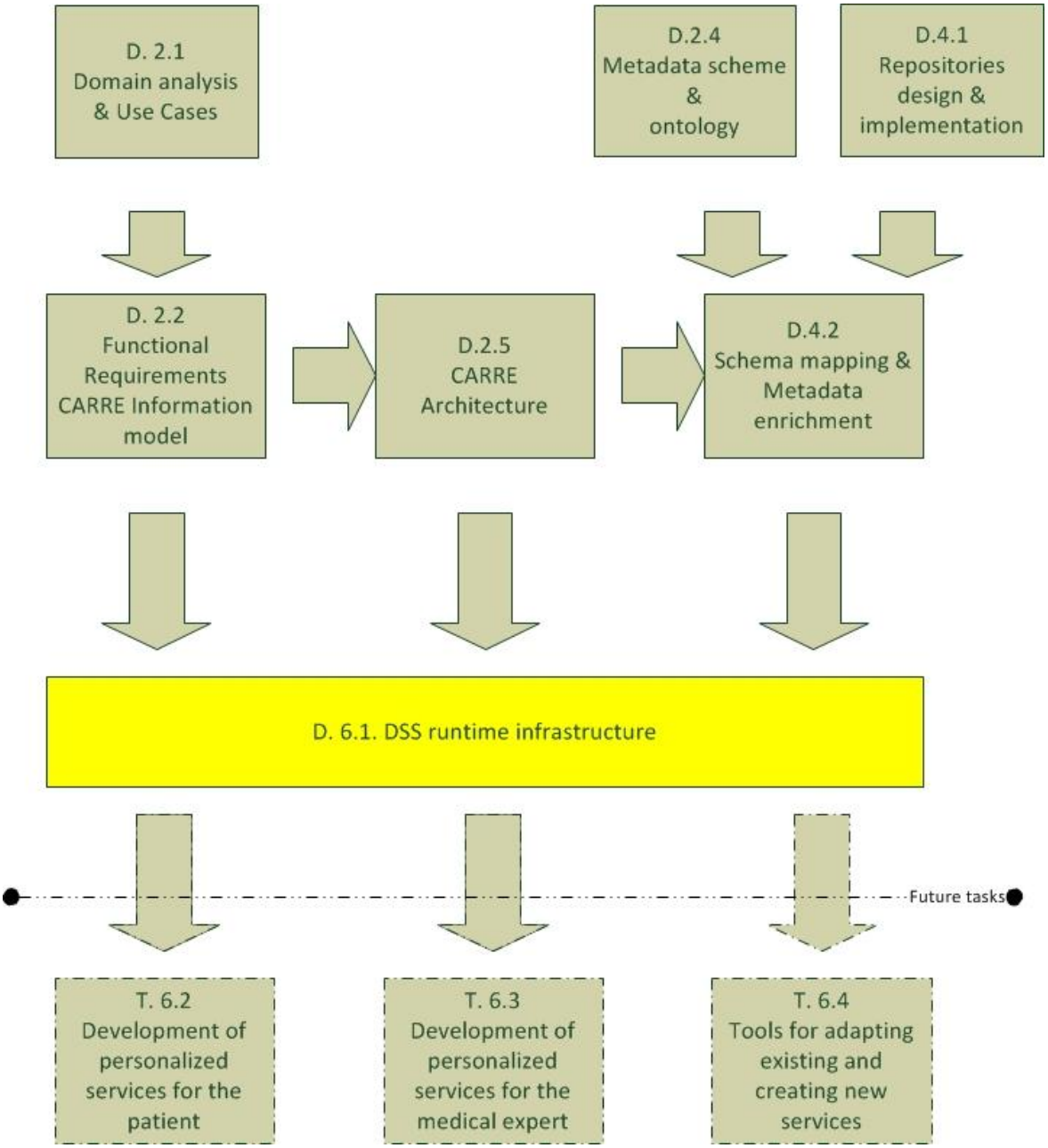

Figure 1. DSS relationships with other deliverables.

The current document aims to describe technical details of the DSS infrastructure and development activities. It defines the major software components that are crucial for meeting the requirements and use cases of DSS.

The remainder of this document is structured as follows: Section 2 presents the overall, high level DSS architecture in CARRE system. Section 3 describes the concept and methodology taken towards DSS development. Section 4 presents in detail DSS data sources and outputs to be stored again in CARRE repositories. Section 5 presents in detail each of the DSS components.

2. DSS in CARRE

In CARRE system decision support service will determine the optimal solution, predict future trends and patterns based on information data analytics and formal reasoning formed on ontologies, which are the main techniques supported by RDF Linked Data, which then will be the main source of decision recommendations for CARRE. Together with Interactive Visualisation Interface, DSS will support patient application, by providing user-friendly visualisation of the current disease status with appropriate personal recommendation and advices to his lifestyle.

In CARRE system the data to decision support service will be retrieved via RESTful web service APIs provided both by the public and private CARRE data repositories. After receiving the appropriate data the DSS will analyse the data to determine optimal recommendations and solutions for the patient to fulfil following users queries and interactions with this system component. Based on assessment of inputs from semantic data entry system the Personal Patient DSS should select educational materials based on current disease state and risks, suggest personal diet adherence and physical activities plan as well as provide alerting mechanisms and appropriate advises for changes.

All the above pieces of information will be send to private RDF Repository to be an input data to Interactive Visualisation Interface, by means of text, variables and recommendation to intuitive and user-friendly visualisation in patient application.

2.1. DSS Architecture high level view

The decision support service (DSS) is another data source, which provides real-time decision support data for patients. Secure APIs are used to provide encrypted communication, such as in the case of retrieving data from private repository. It is planned, for privacy reasons and not only, that DSS output will be stored in patient's private repository in RDF format. It is the most natural way, since RDF Repository is the central data storage of future CARRE system, so it enables to query the data for visualisation purposes in intuitive and user-friendly form.

Figure 2 below presents a high-level view of the overall DSS data flow illustrating the major identified components and how these are connected to one another.

Based on assessment of inputs from semantic data entry system the Personal Patient DSS should select educational materials based on current disease state and risks, create personal diet adherence and physical activities plan as well as provide alerting mechanisms of dangerous value of patient health condition and appropriate advice for changes.

DSS involves the development of decision support services for the medical expert. By means of this component medical expert, based on his expertise in given area and scientific literature as well as based on patient disease condition and calculated risks model for disease progression will propose the recommendation to particular patient life-style and provide guidance to treatment plan. For example, if patient's Systolic Blood Presure (SPB) is > 140 mmHg or Diastolic Blood Presure (DBP) > 90 mmHg, the risk factor data says that risk ratio (the likelihood of negative negative outcome to occur) of getting heart failure caused by hypertension is 1.58, therefore the DSS should recommend to patient to wait for 1-3 min and take a repeat reading; and if then patient's SBP remains > 180 mmHg or DBP > 110 mmHg, the DSS sould send the red alarm to call an ambulance.

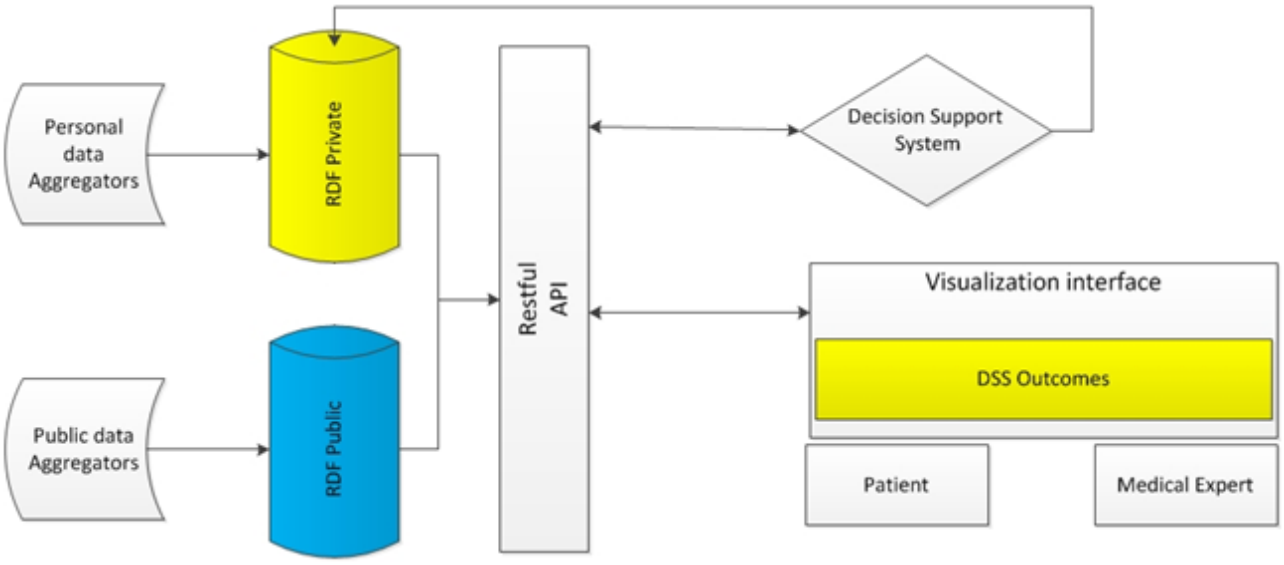

Figure 2. DSS high level architecture.

2.2. Use cases

The previous deliverable *D.2.1 Domain analysis & use case definition*, three high level DSS use cases have been identified, which should be taken into account throughout the development of DSS:

- UC_PE_10: The goal of this use case is to inform patients about their current health status and their risks.
- UC_PE_11: The goal of this use case is to inform end users about new medical evidence available about their wises.
- UC_PA_17: The goal of this use case is to inform patients that are in increased risk of acute episode.
- UC_PA_18: The goal of this use case is to inform patients to make the necessary and appropriate diet changes in order to reduce the risk of their disease.

3. DSS concept and promising methodology

In CARRE system decision support service will determine the optimal solution, by mining RDF repositories data to predict future trends and patterns as well as information data analytics and formal reasoning from ontologies, which are the main techniques supported by RDF Linked Data and ontologies.

This method will search particular patient's observables and will assign risk factors and evidences as well as will show the probability occurrence of given risk factor.[3]

The proposed decision support service (DSS) that computes outcomes for patient and medical expert contains two key components:

1. Data pre-processing which is responsible for filtering and reduce the amount of information that must be processed.[4]

[3] The prototype of tools for this are presented in Section 5.

[4] There are 63 observables in CARRE risk model. Even one measurement of given observable per day would mean e.g. 30 data points needed to calculate given abserveble condition (for example BMI), whether the patient is moderately obese in the last month.

2. Reasoning which provide algorithms to generate appropriate alarms and the probability of risk factors occurrences based on given patient data, as shown in Figure 3.

The concept explained above doesn't model new risk factors, but requires that the inputs for the DSS are observables values, values ranges from patient's private RDF repository and risk elements existing in public RDF repository. The output are alarms and a set of *k* predicted probabilities, in the range of [0, 1] interval, reflecting occurrence of given risk factors for particular patient health condition. If probability value will be close to one that means – the given risk factor occurs and the patient is at given risk of cardiorenal disease or comorbidity.

Due to the fact that several the DSS necessary input (data sources) are under development our DSS is also our infrastructure is steadily improved and customizing to changes to CARRE systems components. Thanks to this other promising method for data pre-processing and connected with these algorithms are still considered.

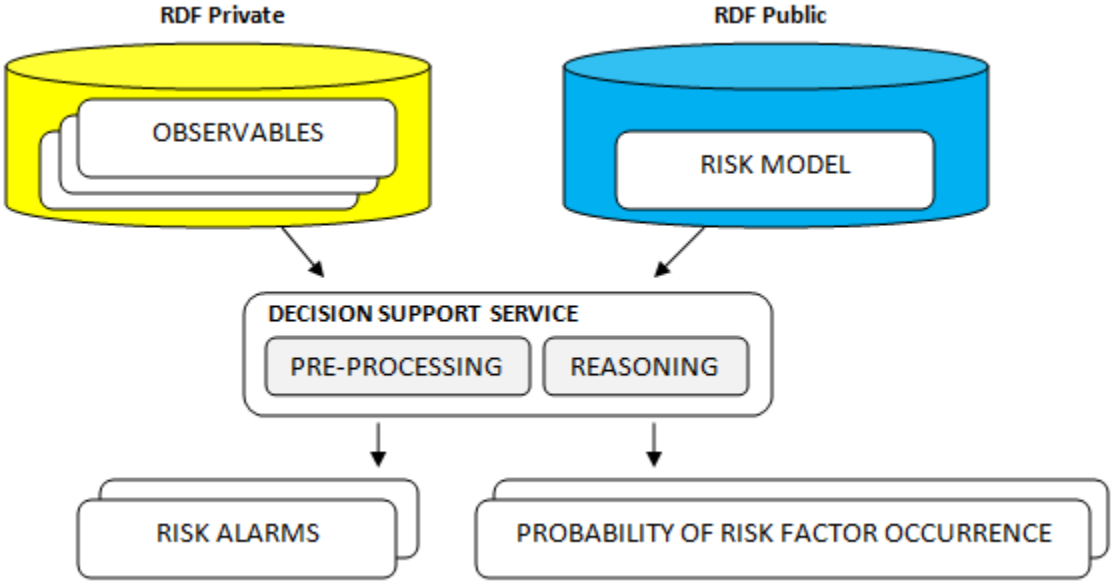

Figure 3. Functional schema of DSS.

Decision support can provide complex information to end user in terms of number of alerts and variables and outcome possibilities. The manner in which way information are presented has influence on proper visualization on the final interface. Output from DSS should consider following aspects:

- Firstly, focus on the most important information.
- Reduce the number of presented items and utilize various design enhancements.
- Colour coding of information (e.g. Risk Alarms presented in Table 1 in section 4.2.2).

DSS runtime infrastructure will provide:

- Framework and service to both patient and medical expert application.
- Forecasting models and analytics based on the risk model fulfilled by data in repositories.
- Run-time decision based on current status of incoming data.

DSS will support patient application and will be the main source of decision recommendations for CARRE. This includes the analysis of the generic and personalized risk model so as to allow the CARRE stakeholders to identify and assess critical medical conditions. Its aim will be to produce meaningful information that will be passed to the end-user interface with the synergy of the visualization component.

It should be stressed, that such large amount of information requires hierarchical presentation. Only in such case both easy access to all this data as well as intuitive operation is required. This will create the requirements for the development of logical interface.

Analyses of requirements and interaction design will be done on the base of direct consultations carried out among patients and medical experts in cardiac and renal diseases domain during the integration, customization and update towards patient's and expert's perspectives.

In addition to the classical view of the decision making process, there is an understanding of this process as a knowledge-based. This approach assumes that a decision is made based on fragments of knowledge describing the essence of information which is necessary to take decision. In this context, decision-making is a process of creating a new, previously non-existent piece of knowledge, which will be created by converting and combining pieces of existing knowledge available in both repositories.

4. DSS data sources and outputs

4.1. DSS data sources

In CARRE system the data for DSS will be provided via common interfaces and common data exchange method, which will ease the further integration. The DSS will communicate with CARRE RDF repositories over SPARQL (a RDF query language) to retrieve the RDF files from both repositories. For DSS service the CARRE semantic repository will be a RDF store accessible as a RESTful Web Service. The same approach will be used to access both repositories, with security built-in to the methods for accessing private data. This method was described in detail in submitted reports under work package *WP4 Data Enrichment, correlation and interlinking*.The server allows the storage of RDF triples (an item of RDF data) under a number of individual graphs. An access to the triples within a particular graph is restricted to a certain user, since it is confidential, health-related data. According to this, the private RDF repository stores the data related to individual patient, in a form of separate, restricted RDF graph. Particularly the RDF private repository relates to following data:

a) Sensors data and measurements, retrieved by 3rd party health market devices, which measure physical activity (steps, walking distance, burned calories, sleep quality), heart rate, blood pressure, blood glucose and weight. This data is collected via sensors manufactures internet application (see *D.2.3 Data source identification & description*) and stored in RDF Repository by the developed sensor aggregators, that are developed and described in *D.3.2 Sensors and Aggregators for Personal Sensor Data*.
b) PHR data. Personal medical data related to particular patient, retrieved by developed aggregator from the PHR system, intended for use by CARRE patients, particularly VivaPort[5] and Microsoft HealthVault[6] as well as PHR Manual Data Entry System[7], which is a browser-based application that allows patients to record their medical data by entering it manually (see *D.3.3 Aggregators for personal medical & lifestyle data from on-line sources*).

The public RDF repository is available for public querying without authentication, because it contains no confidential personal data for any patient. Particularly the RDF public repository relates to following data:

a) Educational and medical scientific metadata, which involve the development of aggregators for medical evidence data and patient educational content from on-line authoritative sources. These kinds of information are either openly available to public, such as some government medical advice sites, or access based on subscriptions, such as PubMED[8], and MedLinePlus[9]. More detailed description is provided in *D.3.4 Aggregators for medical evidence and educational material (meta)data from identified sites*.

[5] https://vivaport.eu

[6] https://www.healthvault.com/

[7] http://phr.carre-project.eu/

[8] http://www.ncbi.nlm.nih.gov/pubmed

[9] https://www.nlm.nih.gov/medlineplus/

b) Risk factor data representation, which is a main CARRE information model. Risk factor ontology describes the causal relationship between one or more risk elements, appearing as source(s) and another risk element, named as target. More detailed description is provided in *D.2.2 Functional requirements & CARRE information model*.

4.2. DSS outputs

4.2.1 Data stored to private RDF

DSS will use all data, stored in RDF repositories, to compute decisions for particular patients. Use cases for DSS will be firstly prepared by the medical experts in the CARRE consortium and then implemented in DSS as services, to be stored in private RDF Repository and to be showed in Interactive Visual Interface as a "DSS alarms".

Decision support service will provide links to educational resources and variety of health related alerting and changes identification mechanisms that are suitable to a particular patient's needs and are based on their personal data.

In this context, decision-making is a process of creating a new piece of data, previously non-existent, by using different methods of data modelling and processing to convert and combine pieces of existing knowledge in order to manage the risk for comorbidities or progression of disease to more severe stages.

4.2.2 Risk Alarms

The basic feature of DSS is the timely identification of major dangerous patient health condition levels, by means of sending alerting information about medical check-ups, monitoring, increased risk of disease progression and transition, the need to change diet in patient application.

This is a decision support object, which is the result of analysis and reasoning captured in CARRE ontologies. A basic feature of DSS is the timely identification of major dangerous patient health condition levels. Hence the DSS will provide the variables for the visualisation model on how to present given piece of information using various coloured based techniques – for example presentation of comorbidity threat level.

This refers to the output of the visualisation service. It includes an interactive graphical user interface and provides the set of pictograms (graphics), text and in the special cases other visual data. Basic feature of visualisation is rapid identification of major patient health condition levels (using various visualisation techniques). Table 1 shows some examples of the DSS alerts that are separated into three different health condition levels.

Table 1. DSS alerts.

Type of alert	**Red alert**	**Yellow alert**	**Green alert**
Blood Pressure	If your SBP is > 180 mmHg or DBP > 110 mmHg, wait for 1-3 min and t**ake a repeat reading.** If your SBP remains > 180 mmHg or DBP > 110 mmHg, call an ambulance	If your BP ≥135 and/or ≥85, remain checking your BP twice daily for 7 days**.** If your BP remains ≥135 and/or ≥85 during seven days, contact your family doctors, you may need the BP treatment correction.	If your BP ≥135 and/or ≥85, wait for 1-3 min and t**ake a repeat reading.** If your BP remains ≥135 and/or ≥85 for the second measurement, make sure: 1. You use your BP monitor correctly. 2. You took your BP treatment.

Physical activity			< 7000 steps/day, you need to increase your physical activity.
Body weight		If you gain 1kg (2-3 pounds) in a day or >2kg (5 pounds) in 1 week, contact your familly doctor. If the swelling of ankles or feet becomes worse, contact your family doctor.	
Blood Glucose	If your blood glucose levels are 20 mmol/L or higher and doesn't improve despite following your doctor's instructions for treatment, call an ambulance.	If your blood glucose levels remain high (>15mmol/l), contact your family doctor.	
Atrial fibrillation	If you noticed the heart rhythm disturbance, call an ambulance.		
Heart rate (HR)		If your HR> 90 at rest, contact your family doctor.	

4.2.3 Visual interface

One of the basic feature of the DSS is to provide the variables for the visualisation service on how to present given piece of information in user-friendly form, using various visualization techniques.

The visualisation service is a data-driven visualisation module which provides web-based interactive visual analysis of data stored in the public and private data repository, such as risk association relationship and disease progression data. The visualisation component is used by the patient application and the medical expert application, and communicates with the public/private RDF repository and the DSS in order to provide integrated interactive visualisation of the data.

The interactive visualisation client-side code will be downloaded online from the server side and run in web browsers in the patient application and the medical expert application.

One of the features of all visualisation components are scalability and adaptability in order to allow easy configurability of presented elements (interface) on both mobile devices as well as standard monitor display.

5. DSS components

5.1. Data mining app with rich functions API for Matlab

During early stage of development API for Matlab suite was created with many functions that will help external scientists use Risk Factor repository for their researches. Thanks to this functionality anyone can

explore all triples related to public and private CARRE RDF repositories. Brief of functions are listed in Table 2 and some indicative functions of them are presented as Matlab code in Annex 2.1.

Table 2. Rich functions API for Matlab.

Function	Description
[con] = rConnect(user, pass)	Create a structure with access token for public and private database based on provided user name and password
[data] = rQuery(con, query)	Send specified query in SPARQL format and return Matlab structure with query output.
[user_data] = r_userdata(conn)	Return user data for logged in by rConnect function user.
[types] = rGetTypes(con, from)	Get all available node types. Parameter “from” can be: 1 (default)- for getting types from both private and public repository 2 - for public only; 3 - for private only
[atributes] = rGetAtributes(con, from)	Get all available node attributes. Parameter “from” can have the same values as in function above.
[nodes] = rGetNodesByType(con, type)	Get all nodes of type “type”.
[nodes] = rGetNodesByAtribue(con, atr)	Get all nodes having attribute “atr”.
[atr] = rGetNodeAtributes(con, node)	Get all node attributes with corresponding attribute values of node “node”
[atr] = rGetNodeAtribute(con, node , atr)	Get specific value of attribute “atr” of node “node”.
[obsv] = rGetObservables(con)	Get all observables
[riskEl] = rGetRiskElements(con)	Get all risk elements
[riskEv] = rGetRiskEvidences(con)	Get all risk evidences
[riskFac] = rGetRiskFactors(con)	Get all risk factors
[terms] = rGetTerms(con)	Get all terms
[d, m] = rGetMeasurement(con, atr)	Get measurements specified by “atr”. Return values are stored in “m” with time stamp “d”.

As example of using this API data mining application (Figure 4) was created. Thanks to this application, user can explore the entire risk factor database, measurements in private repository, and search for particular triples.

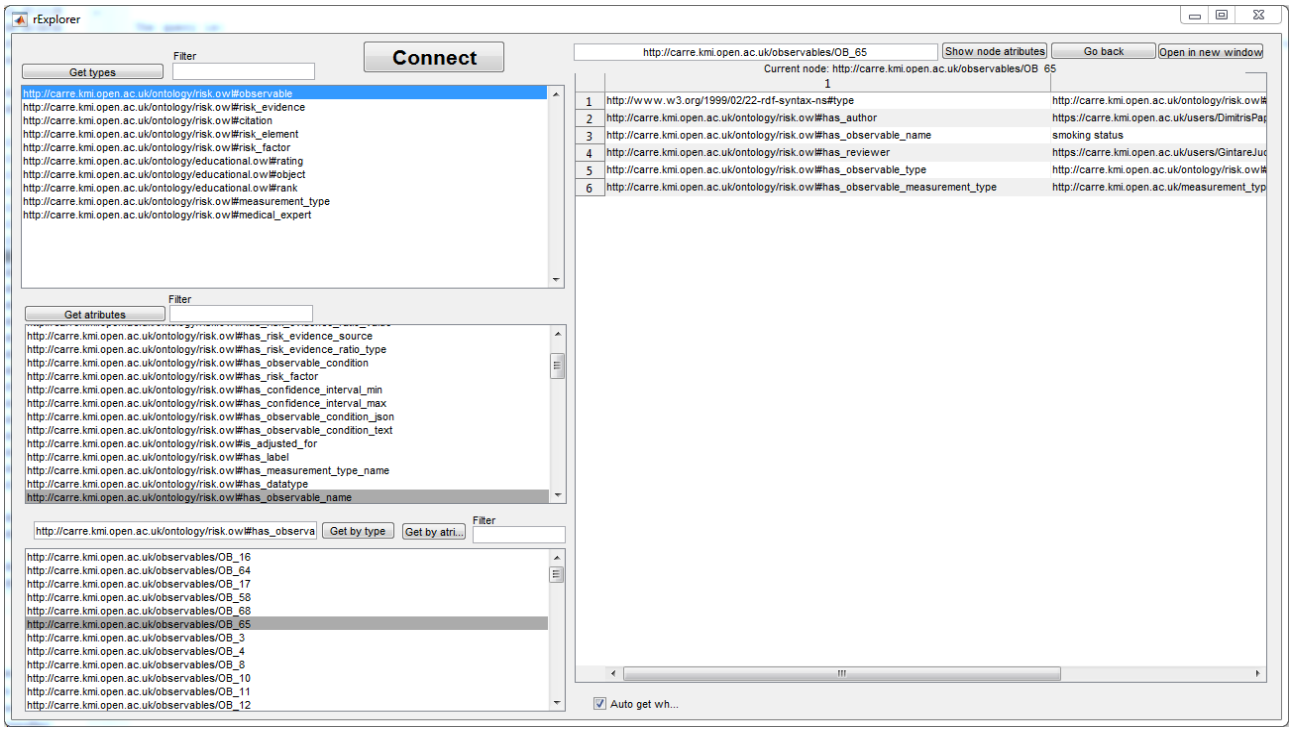

Figure 4. RDF data explorer Matlab GUI.

Examples of use:

- Find all observables
- Find nodes with particular measurement in private RDF store
- Find which devices are used by patient

5.2. Prototype of reasoning API for personal risk factor model for Matlab

Prototype of reasoning API was created for Matlab suite and partially implemented in DSS web service. Thanks to this user can calculate personal model for given observables values.

At stage of developing this API only manual reasoning was implemented, due to the fact that repository is under heavy development and conversion to computer readable format of risk evidences expressions was done recently. Figure 5 shows the reasoning API in command line interface during its execution in Matlab. Accordogly, Annex 2.2 presents some indicative Matlab code that is behind of reasing API.

```
>> p.printRaport

--- Associated observables ---

1: Sex = Male
2: Age = 40
3: Acute myocardial infarction diagnosis = diagnosed
4: BMI (Body Mass Index) = 30
5: Ischemic heart disease family history = yes
6: Smoking status = ex-smoker

--- Risk elements raport ---

Risk element nr 2: Acute myocardial infarction
  val:          diagnosed
  condition:    NONE
  type:         fullObsv
  revelant:     1

  Last observables state:
    1: Acute myocardial infarction diagnosis = diagnosed (up to date)

Risk element nr 3: Age
  val:          40
  condition:    N/A
  type:         fullObsv
  revelant:     1

  Last observables state:
    1: Age = 40 (up to date)
```

Figure 5. Prototype of reasoning API.

5.3. Web infrastructure for DSS

Web service for the DSS (as shown in Figure 6-Figure 9) will be central point of data processing for each patient. It connects and cooperates with both public and private RDF repository, where the medical knowledge is stored in public RDF repository and where sensors measurements and the output of DSS reasoning are store in private RDF repository.

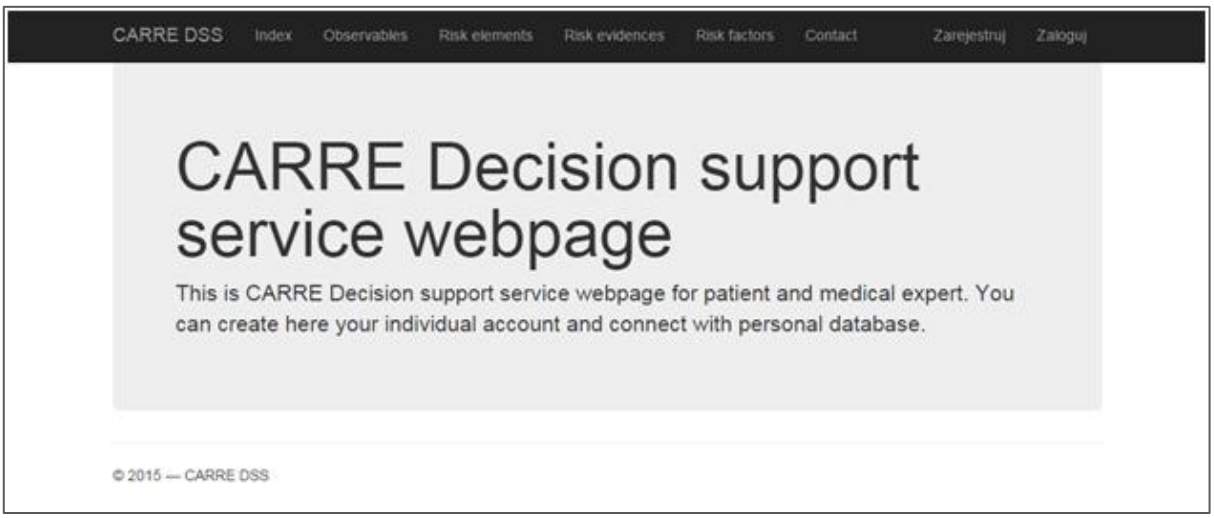

Figure 6. Web infrastructure for DSS.

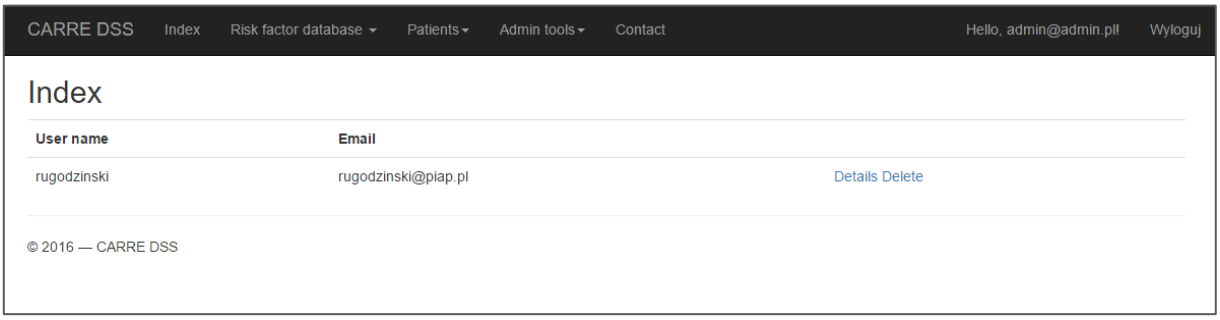

Figure 7. Patients view for medical expert.

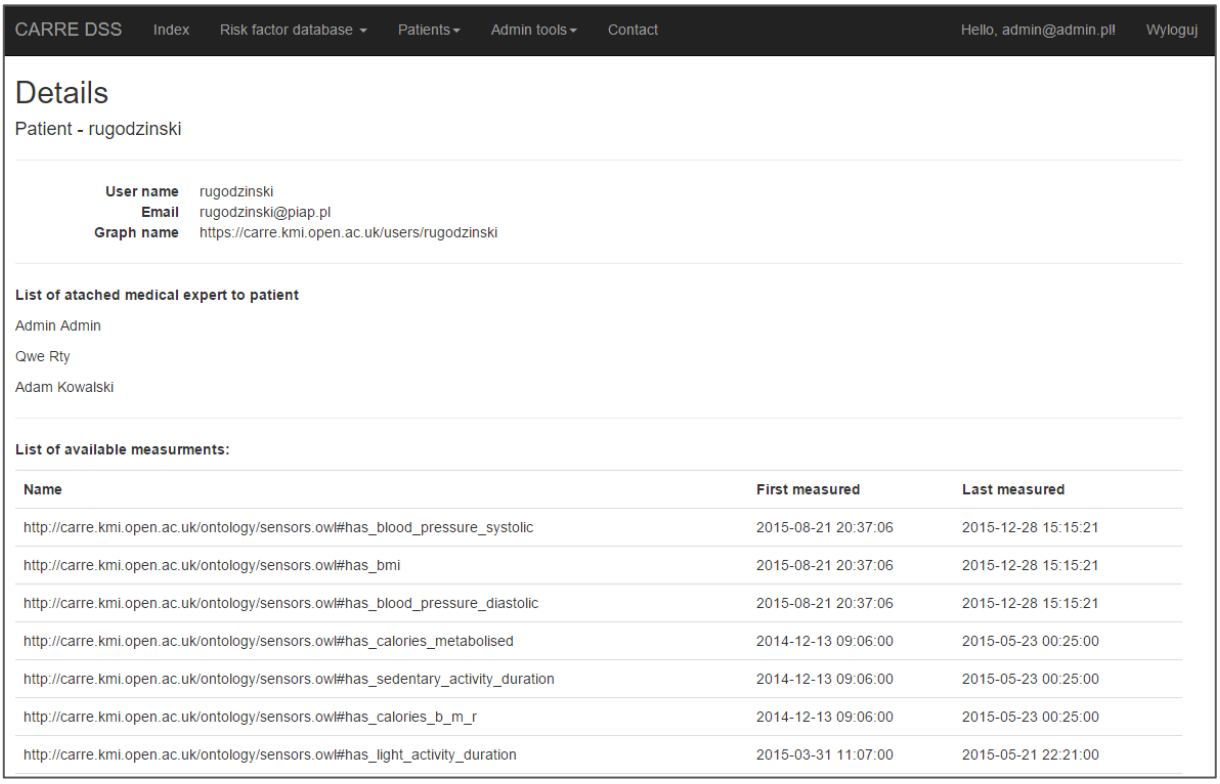

Figure 8. Details of the patient view for medical expert.

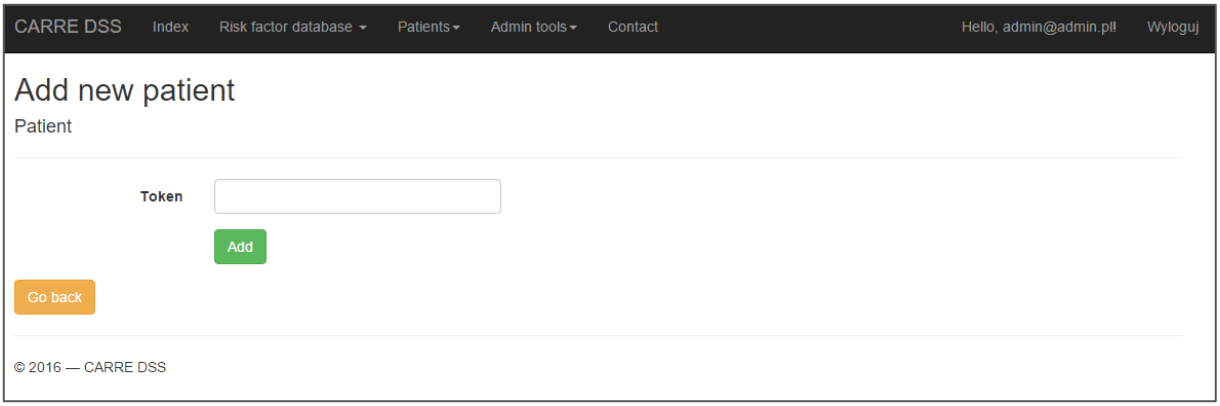

Figure 9. Attaching new patient to medical expert.

DSS web service is based on cloud architecture. That means that DSS doesn't need to know where the data is stored and connects to repositories by specified endpoints. The DSS architecture is designed to be portable across different systems and istallations, so that, for example, a hospital could deploy it in its own private installation in order to protect its patients privacy. Data storage diversity can be explained as below:

- Risk factor database: external public RDF endpoint
- Personal measurements and DSS output storage: external RDF private repository endpoint
- Medical experts user accounts and data: internal SQL database

Internally service is build using .NET technology. This is high level computer language similar to Java provided by Microsoft. Thanks to this technology scalable infrastructure can be created. Internally we are using Visual Studio Community Edition for Web integraded developing environment. As a framework to create full scalable service ASP.NET technology is used in cooperation with MVC (Model-View-Controller) methology. The code metrics of DSS web service is presented in Table 3.

Table 3. DSS web service code metrics (C# code).

Project	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
CARRE_DSS2	86	517	4	232	1112
CARRE_DSS2.Tests	72	13	1	18	41

6. Conclusion

This document reports on the Decision Support Service (DSS) infrastructure design and development, which is one of the main CARRE system components, which supports and maintenances of real-time decision support for patients. This CARRE system component is a personalised service for disease progression management and is mainly responsible for providing alerts depend on major dangerous patient health condition levels, advices and personal life-style guidance, based on monitoring of current medical treatment data in order to manage risks for comorbidities or progression of disease to more severe stages.

This document is a report of activities undertaken in task *T.6.1 Development of DSS runtime infrastructure*. This deliverable report focuses on the design and initial implementation of DSS components in order to provide personalized services to both patient application and medical expert application, which are in the process of development under currently running tasks in the work pachage WP6: *T.6.2 Development of personalized service for the patient* and *T.6.3 Development of personalized service for the medical expert*.

Annex 1

DSS Runtime Infrastructure Software

What is CARRE: DSS Runtime Infrastructure?

In CARRE system, **Decision support service (DSS)** determines the optimal solution, predict future trends and patterns based on information data analytics and formal reasoning formed on ontologies, which are the main techniques supported by RDF Linked Data, which then will be the main source of decision recommendations for CARRE. Together with interactive visualization interface, DSS supports patient application, by providing user-friendly visualization of the current disease status with appropriate personal recommendation and advises to his lifestyle.

In CARRE system, the data to Decision support service are retrieved via RESTful APIs web service provided both by the public and private CARRE data repositories. After receiving the appropriate data the DSS analyses the data to determine optimal recommendation and solutions for patient. Based on assessment of inputs from semantic data entry system the DSS should create educational materials based on current disease state and risks, create personal diet adherence changes and physical activities plan as well as provide alerting mechanisms and appropriate advises for changes.

All above pieces of information are send to private RDF Repository to be an input data to interactive visualization interface, by means of text, variables and recommendation to intuitive and user-friendly visualization in patient application.

Download

API for using CARRE RDF repositories, GUI Explorer of RDF and basic risk factor reasoning scripts:

v0.5 (Released 11 January 2016, Deliverable 6.1)

- Source for Matlab 2014b+ (503 KB): CARRE_DSS_matlab_v0.5.7z (Matlab code)

DSS web service:

v1.0 (Released 11 January 2016, Deliverable 6.1)

- Source for Visual Studio 2015+ (33.9 MB): CARRE_DSS_web_service_visualstudio_v1.0.7z (C# code)

The CARRE DSS Runtime Infrastructure is Open Source

CARRE DSS Runtime Infrastructure is Open Source and can be freely used in Open Source applications under the terms GNU General Public License (GPL).

Annex 2

DSS Matlab code examples

Annex 2.1: CARRE database API

rGetTypes

```
function [ types ] = rGetTypes( con, from)
%RGETTYPES Summary of this function goes here
%   Detailed explanation goes here

% FROM - from what repository?
% 1 - all (default): risk factors etc and from user grapf
% 2 - risk factors etc
% 3 - user grapf

if nargin == 1
    from = 1;
end
%http://www.w3.org/2000/01/rdf-schema#type
if from == 1
    query = [...
        'SELECT DISTINCT ?t '...
        'FROM <http://carre.kmi.open.ac.uk/public> '...
        'WHERE { '...
        '  { ?s a ?t . } '...
        '  UNION { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t . }
'...
        '  UNION { ?s <http://www.w3.org/2000/01/rdf-schema#type> ?t . } '...
        '} '];
elseif from == 2
    query = [...
        'SELECT DISTINCT ?t '...
        'WHERE { '...
        '  SERVICE <http://carre.kmi.open.ac.uk:8890/sparql> '...
        '    { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t . } '...
        '} '];
elseif from == 3
    query = [...
        'SELECT DISTINCT ?t '...
        'WHERE { '...
        '    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t .'...
        '} '];
else
    error('Wrong source');
end

[ data ] = rQuery( con, query );

types = {};
for k = 1:length(data)
    types{k,1} = data{k}.t.value;
end

end
```

rGetNodesByAtribue

```
function [ nodes ] = rGetNodesByAtribue( con, atr )
%RGETNODESBYTYPE Summary of this function goes here
%   Detailed explanation goes here

% searching for predefinded prefixes

carreSensors = 'http://carre.kmi.open.ac.uk/ontology/sensors.owl#';
carreRisk    = 'http://carre.kmi.open.ac.uk/ontology/risk.owl#';

if     strcmp(atr(1:2), 's:')
    atr = [carreSensors atr(3:end)];
elseif strcmp(atr(1:2), 'r:')
    atr = [carreRisk    atr(3:end)];
end

query = [...
    'SELECT ?n '...
    'FROM <http://carre.kmi.open.ac.uk/public> '...
    'WHERE { '...
    '   ?n <' atr '> ?x . '...
    '} '];

[ data ] = rQuery( con, query );

nodes = {};
for k = 1:length(data)
    nodes{k,1} = data{k}.n.value;
end

end
```

rGetNodeAtributes

```
function [ atr ] = rGetNodeAtributes( con, node )
%RGETNODESBYTYPE Summary of this function goes here
%   Detailed explanation goes here


query = [...
    'SELECT ?a ?v '...
    'FROM <http://carre.kmi.open.ac.uk/public> '...
    'WHERE { '...
    '  <' node '> ?a ?v . '...
    '} '];

[ data ] = rQuery( con, query );

atr = {};
for k = 1:length(data)
    atr{k,1} = data{k}.a.value;
```

```
    atr{k,2} = data{k}.v.value;
end

end
```

rGetAtributes

```
function [ atributes ] = rGetAtributes( con, from)
%RGETTYPES Summary of this function goes here
%   Detailed explanation goes here

% FROM - from what repository?
% 1 - all (default): risk factors etc and from user grapf
% 2 - risk factors etc
% 3 - user grapf

if nargin == 1
    from = 1;
end

if from == 1
    query = [...
        'SELECT DISTINCT ?a '...
        'FROM <http://carre.kmi.open.ac.uk/public> '...
        'WHERE { '...
        '  ?s ?a ?t . '...
        '} '];
elseif from == 2
    query = [...
        'SELECT DISTINCT ?a '...
        'WHERE { '...
        '  SERVICE <http://carre.kmi.open.ac.uk:8890/sparql> '...
        '    { ?s ?a ?t . } '...
        '} '];
elseif from == 3
    query = [...
        'SELECT DISTINCT ?a '...
        'WHERE { '...
        '    ?s ?a ?t .'...
        '} '];
else
    error('Wrong source');
end

[ data ] = rQuery( con, query );

atributes = {};
for k = 1:length(data)
    atributes{k,1} = data{k}.a.value;
end

end
```

rGetNodeAtribute

```
function [ atr ] = rGetNodeAtribute( con, node , atr)
%RGETNODESBYTYPE Summary of this function goes here
%   Detailed explanation goes here

carreSensors = 'http://carre.kmi.open.ac.uk/ontology/sensors.owl#';
carreRisk    = 'http://carre.kmi.open.ac.uk/ontology/risk.owl#';

if     strcmp(atr(1:2), 's:')
    atr = [carreSensors atr(3:end)];
elseif strcmp(atr(1:2), 'r:')
    atr = [carreRisk    atr(3:end)];
end

query = [...
    'SELECT ?v '...
    'FROM <http://carre.kmi.open.ac.uk/public> '...
    'WHERE { '...
    '  { <' node '> <' atr '> ?v . } '...
    '} '];

[ data ] = rQuery( con, query );

atr = cell(length(data),1);
for k = 1:length(data)
    atr{k,1} = data{k}.v.value;
end

end
```

rGetNodesByType

```
function [ nodes ] = rGetNodesByType( con, type )
%RGETNODESBYTYPE Summary of this function goes here
%   Detailed explanation goes here

% searching for predefinded prefixes

carreSensors = 'http://carre.kmi.open.ac.uk/ontology/sensors.owl#';
carreRisk    = 'http://carre.kmi.open.ac.uk/ontology/risk.owl#';

if     strcmp(type(1:2), 's:')
    type = [carreSensors type(3:end)];
elseif strcmp(type(1:2), 'r:')
    type = [carreRisk    type(3:end)];
end

% query = [...
%     'SELECT ?n '...
%     'WHERE { '...
%     '  {SERVICE <http://carre.kmi.open.ac.uk/sparql> '...
%     '    { ?n <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <' type '> .
}} '...
%     '  UNION { ?n a <' type '> . } '...
%     '} '];

query = [...
    'SELECT DISTINCT ?s '...
    'FROM <http://carre.kmi.open.ac.uk/public> '...
    'WHERE { '...
    '  { ?s a <' type '> . } '...
    '  UNION { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <' type '> .
} '...
    '  UNION { ?s <http://www.w3.org/2000/01/rdf-schema#type> <' type '> . }
'...
    '} '];


%     '  {SERVICE <http://carre.kmi.open.ac.uk:8890/sparql> '...
%     '    {{ ?s a <' type '> . }'...
%     '    UNION { ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <' type
'>}'...
%     '    UNION { ?s <http://www.w3.org/2000/01/rdf-schema#type> <' type '>}}}
'...
%     '} '];
[ data ] = rQuery( con, query );

nodes = {};
for k = 1:length(data)
    nodes{k,1} = data{k}.s.value;
end

end
```

rQuery

```
function [ data ] = rQuery( con, query )
%R_QUERY Summary of this function goes here
%   Detailed explanation goes here

url = [con.endpoint 'query'];

fprintf('\n');
disp('The query is:');
disp(query);

data = urlread(url,'Post', {'sparql', query, 'token',con.token});

disp('Data queried');

try
    data = loadjson(data);
catch
    data = [];
end

disp('Data parsed');

end
```

rConnect

```
function [ conn ] = rConnect( user, pass)
%R_CONNECT Summary of this function goes here
%   Detailed explanation goes here

endpoint = 'http://carre.kmi.open.ac.uk/ws';

if ~strcmp(endpoint(end), '/')
    endpoint(end+1) = '/';
end

path = [endpoint 'authenticate'];
text = urlread(path, 'Authentication','Basic', 'Username', user, 'Password',
pass);
conn = loadjson(text);
conn.endpoint = endpoint;

end
```

rGetTerms

```
function [ terms ] = rGetTerms( con )

% getting observables nodes from server
[ nodes ] = rGetNodesByType( con, 'http://www.w3.org/2004/02/skos/core#Concept'
);

%query each observables for atributes
terms = struct([]);
h = waitbar(0,'Wait');
for k = 1:length(nodes)
    waitbar(k/length(nodes), h, sprintf('Wait %d/%d', k, length(nodes)));
    [ atr ] = rGetNodeAtributes( con, nodes{k} );
    if ~isempty(atr)
        terms(end+1).plainTextAtributes = atr;
        terms(end).node = nodes{k};

        %getting label
        ind = find(strcmp(atr(:,1), 'http://www.w3.org/2000/01/rdf-
schema#label'), 1);
        if ~isempty(ind)
            terms(end).label = atr{ind,2};
        end

    end
end
close(h);
%sorting
[~, ind]=sort({terms.label});
terms = terms(ind);
```

Annex 2.2: CARRE Reasoning API

rDataBase

```
classdef rDataBase < handle
    %RDATABASE Data base of CARRE Project repository.
    %   In future all queries should be done via this objectt.

    properties
        RiskFactors;
        RiskEvidences;
        RiskElements;
        Observables;
        Terms;
        Connection;
    end

    methods

        function obj = rDataBase(varargin)
            obj.RiskFactors = {};
            obj.RiskEvidences = {};
            obj.RiskElements = {};
            obj.Observables = {};
            obj.Terms = {};

            if length(varargin) == 1
                %connection structure passed
                obj.Connection = varargin{1};
            elseif length(varargin) == 2
                %create new connection
                obj.Connection = rConnect( varargin{1}, varargin{2});
            else
                %no connection structure
                obj.Connection = [];
            end
        end

        function obj = updateRiskFactors(obj)
            obj.RiskFactors = rGetRiskFactors( obj.Connection );
        end

        function obj = updateRiskEvidences(obj)
            obj.RiskEvidences = rGetRiskEvidences( obj.Connection );
        end

        function obj = updateRiskElements(obj)
            obj.RiskElements = rGetRiskElements( obj.Connection );
        end

        function obj = updateObservables(obj)
            obj.Observables = rGetObservables( obj.Connection );
        end

        function obj = updateTerms(obj)
            obj.Terms = rGetTerms( obj.Connection );
```

```
        end

        function obj = updateAll(obj)
            disp('updateRiskFactors');
            obj = updateRiskFactors(obj);
            disp('updateRiskEvidences');
            obj = updateRiskEvidences(obj);
            disp('updateRiskElements');
            obj = updateRiskElements(obj);
            disp('updateObservables');
            obj = updateObservables(obj);
            disp('updateTerms');
            obj = updateTerms(obj);
        end

        function saveDataBase(obj, filename)
            RiskFactors = obj.RiskFactors;
            RiskEvidences = obj.RiskEvidences;
            RiskElements = obj.RiskElements;
            Observables = obj.Observables;
            Terms = obj.Terms;
            save(filename, 'RiskFactors', 'RiskEvidences', 'RiskElements',
'Observables', 'Terms');
        end

        function loadDataBase(obj, filename)
            s = load(filename);
            obj.RiskFactors = s.RiskFactors;
            obj.RiskEvidences = s.RiskEvidences;
            obj.RiskElements = s.RiskElements;
            obj.Observables = s.Observables;
            obj.Terms = s.Terms;
        end

        function nodes = searchObservableByName(obj, obsvName, mode)
            %mode:
            % 1 (default) - find, case insersivity
            % 2 - find, case sensivity
            % 3 - exact
            if nargin <= 2
                mode = 1;
            end
            nodes = [];
            switch mode
                case 1
                    nodes = {obj.Observables(~cellfun(@isempty,
strfind(lower({obj.Observables.name})', lower(obsvName)))).node}';
                case 2
                    nodes = {obj.Observables(~cellfun(@isempty,
strfind({obj.Observables.name}', obsvName))).node}';
                case 3
                    nodes = {obj.Observables(strcmp({obj.Observables.name}',
obsvName)).node};
            end

            if length(nodes) == 1
                nodes = cell2mat(nodes);
            end
        end
```

```
        function obsv = getObservableByNode(obj, node)
            obsv = obj.Observables(strcmp({obj.Observables.node}', node));
        end

        function riskEl = getRiskElementByNode(obj, node)
            riskEl = obj.RiskElements(strcmp({obj.RiskElements.node}', node));
        end

        function riskF = getRiskFactorByNode(obj, node)
            riskF = obj.RiskFactors(strcmp({obj.RiskFactors.node}', node));
        end

        function riskEv = getRiskEvidenceByNode(obj, node)
            riskEv = obj.RiskEvidences(strcmp({obj.RiskEvidences.node}', node));
        end

    end

end
```

rPatient

```
classdef rPatient < handle
    %RPATIENT Patient object

    properties
        db;
        associatedObservables;
        riskElements;
        riskFactors;
    end

    methods
        function obj = rPatient(database)
            obj.db = database;
            obj.associatedObservables = struct('node', {}, 'val', {});

            %type - 'noObsv', 'missingObsv', 'fullObsv'
            %val - if empty -> lack of answer or missing data
            obj.riskElements = struct('riskElNode', {}, 'knownObsv', {},
'unknownObsv', {}, 'type', {}, 'val', {});

        end

        function obj = setObservable(obj, obsvNode, value)
            %first check existence of observable in local associated obsv
            ind = find(strcmp({obj.associatedObservables.node}', obsvNode), 1);
            if isempty(ind)
                obj.associatedObservables(end+1).node = obsvNode;
                obj.associatedObservables(end).val = {value};
            else
                obj.associatedObservables(ind).node = obsvNode;
                obj.associatedObservables(ind).val = {value};
            end
        end

        function obj = deleteObservable(obj, obsvNode)
            ind = find(strcmp({obj.associatedObservables.node}', obsvNode), 1);
            if ~isempty(ind)
                obj.associatedObservables(ind) = [];
            end
        end

        function printRaport(obj, mode)
            % mode = {'full', 'reduced'};
            if nargin == 1
                mode = 'reduced';
            end
            if ~isempty(obj.associatedObservables)
                fprintf('\n--- Associated observables --- \n\n');
                for k = 1:length(obj.associatedObservables)
                    n =
obj.db.getObservableByNode(obj.associatedObservables(k).node);
                    name = n.name;
                    val = obj.associatedObservables(k).val{1};
                    if ~ischar(val)
                        val = num2str(val);
                    end
```

```
                    fprintf('%d: %s = %s\n', k, name, val);
                end
            end

            if ~isempty(obj.riskElements)
                fprintf('\n--- Risk elements raport --- \n\n');
                for k = 1:length(obj.riskElements)

                    [type] =
obj.statusOfRiskElement(obj.riskElements(k).riskElNode);
                    if strcmp(type, 'noObsv') && ~strcmp(mode, 'full')
                        continue;
                    end
                    obj.printRiskElement(obj.riskElements(k).riskElNode);
                        fprintf('\n');
                end
            end

            if ~isempty(obj.riskFactors)
                fprintf('--- Risk factors raport --- \n\n');
                for k = 1:length(obj.riskFactors)
                    obj.printRiskFactor(obj.riskFactors(k).rfNode);
                end
            end

        end

        function printRiskElement(obj, node)
                    rInd = find(strcmp({obj.riskElements(:).riskElNode}, node));
                    rElement = obj.riskElements(rInd);
                    if isempty(rElement)
                        return;
                    end
                    r = obj.db.getRiskElementByNode(rElement.riskElNode);
                    rName = r.name;
                    rCondition = r.condition;
                    if isempty(rCondition)
                        rCondition = 'NONE';
                    end
                    rCondition = strrep(rCondition,sprintf('\n'),' ');
                    val = rElement.val;
                    if isempty(val)
                        val = 'NONE';
                    end

                    [type, revelant] =
obj.statusOfRiskElement(rElement.riskElNode);

                    fprintf('Risk element nr %d: %s \n  val:\t\t\t%s\n
condition:\t%s\n  type:\t\t\t%s\n  revelant: \t%s\n\n', rInd, rName, val,
rCondition, type, num2str(revelant));

                    if ~isempty(rElement.Obsv)
                        disp('  Last observables state:');
                        for k2 = 1:size(rElement.Obsv, 1)
                            obs =
obj.db.getObservableByNode(rElement.Obsv{k2,1});
                            oName = obs.name;
```

```
                            oVal = rElement.Obsv{k2,2};
                            if isempty(oVal)
                                oVal = 'UNKNOWN';
                            elseif ~ischar(oVal)
                                oVal = num2str(oVal);
                            end

                            if revelant(k2)
                                revStr = 'up to date';
                            else
                                revStr = 'out of date';
                            end

                            fprintf('    %d: %s = %s (%s)\n', k2, oName, oVal,
revStr);
                        end
                    end
        end

        function obj = processRiskFactors(obj)
            obj.riskFactors = struct('rfNode', {}, 'rElements', {},
'rEvidences', {}, 'outputRatio', {});
            for k = 1:length(obj.db.RiskFactors)
                % check risk elements
                condition = true;
                rElements = {};
                for k2 = 1:length(obj.db.RiskFactors(k).source)
                    ind = find(strcmp({obj.riskElements(:).riskElNode},
obj.db.RiskFactors(k).source{k2}), 1);
                    if isempty(ind) || strcmp(obj.riskElements(ind).val,
'_noanswer') || strcmp(obj.riskElements(ind).val, '_noInfo')
                        condition = false;
                    end
                    rElements{k2} = obj.riskElements(ind).riskElNode;
                end

                if condition
                    % add risk factor to local db
                    obj.riskFactors(end+1).rfNode = obj.db.RiskFactors(k).node;
                    obj.riskFactors(end).rElements = rElements;
                    obj.riskFactors(end).outputRatio = NaN;

                    obj.riskFactors(end).rEvidences = struct('rEvNode', {},
'observables', {}, 'type', {}, 'val', {});
                    for k2 = 1:length(obj.db.RiskFactors(k).evidence)
                        rEvNode = obj.db.RiskFactors(k).evidence(k2);
                        obj.riskFactors(end).rEvidences(k2).rEvNode = rEvNode;
                        obj.riskFactors(end).rEvidences(k2).val = NaN;
                        rEv =
obj.db.getRiskEvidenceByNode(rEvNode);%obj.db.RiskEvidences(strcmp({obj.db.RiskE
vidences(:).node}, rEvNode));
                        rEvObs = {};
                        for k3 = 1:length(rEv.obsv)
                            ind =
find(strcmp({obj.associatedObservables(:).node}, rEv.obsv(k3)));
                            if any(ind)
                                rEvObs{k3,1} = rEv.obsv{k3};
                                rEvObs{k3,2} =
obj.associatedObservables(ind).val{1};
```

```
                            else
                                rEvObs{k3,1} = rEv.obsv{k3};
                                rEvObs{k3,2} = [];
                            end
                        end
                        obj.riskFactors(end).rEvidences(k2).observables =
rEvObs;
                        [type] = statusOfRiskEvidence(obj,
obj.riskFactors(end).rEvidences(k2));
                        obj.riskFactors(end).rEvidences(k2).type = type;
                    end


                end

            end

        end

        function printRiskFactor(obj, node)
            rfInd = find(strcmp({obj.riskFactors(:).rfNode}, node));
            rFacL = obj.riskFactors(rfInd);
            if isempty(rFacL)
                return;
            end
            rFacDB = obj.db.getRiskFactorByNode(rFacL.rfNode);
            rName = rFacDB.name;
            rVal = rFacL.outputRatio;
            rTargetName = obj.db.getRiskElementByNode(rFacDB.target).name;
            rSourceNamesAndVals = {};
            for k = 1:length(rFacL.rElements)
                rSourceNamesAndVals{k,1} =
obj.db.getRiskElementByNode(rFacL.rElements{k}).name;
                rSourceNamesAndVals{k,2} =
obj.riskElements(strcmp({obj.riskElements(:).riskElNode},
rFacL.rElements{k})).val;
            end
            fprintf('Risk element nr %d: %s\n', rfInd, rName);
            fprintf('  Target:\t%s\n', rTargetName);
            fprintf('  Ratio:\t%.2f\n', rVal);
            fprintf('  Sources:\t\n');
            for k = 1:size(rSourceNamesAndVals,1)
                fprintf('    %d: %s = %s\n', k, rSourceNamesAndVals{k,1},
rSourceNamesAndVals{k,2});
            end

            fprintf('\n');
            for k = 1:length(rFacL.rEvidences)
                obj.printRiskEvidence(rFacL.rEvidences(k), k);
                fprintf('\n');
            end
        end

        function printRiskEvidence(obj, rEvidence, num)

            if nargin == 3
                numStr = sprintf(' nr %d', num);
            else
```

```
                numStr = '';
            end

            rEvNode = rEvidence.rEvNode;
            rEv = obj.db.getRiskEvidenceByNode(rEvNode);
            [type, revelant] = statusOfRiskEvidence(obj, rEvidence);

            expression = rEv.expression;
            if isempty(expression)
                expression = 'NONE';
            end
            expression = strrep(expression,sprintf('\n'),' ');

            val = rEvidence.val;

            fprintf('  Risk evidence%s:\n    type:\t\t\t%s\n    risk
type:\t\t%s\n    risk ratio:\t\t[%.2f,%.2f,%.2f]\n    expression:\t\t%s\n
val:\t\t\t%d\n', ...
                numStr, type, rEv.radioType, rEv.ratio(1), rEv.ratio(2),
rEv.ratio(3), expression, val);
            if ~isempty(rEvidence.observables)
                disp('    Last observables state:');
                for k2 = 1:size(rEvidence.observables, 1)
                    obs =
obj.db.getObservableByNode(rEvidence.observables{k2,1});
                    oName = obs.name;
                    oVal = rEvidence.observables{k2,2};
                    if isempty(oVal)
                        oVal = 'UNKNOWN';
                    elseif ~ischar(oVal)
                        oVal = num2str(oVal);
                    end

                    if revelant(k2)
                        revStr = 'up to date';
                    else
                        revStr = 'out of date';
                    end

                    fprintf('      %d: %s = %s (%s)\n', k2, oName, oVal,
revStr);
                end
            end

        end

        function [type, revelant] = statusOfRiskEvidence(obj, rEvidence)
            emptyObsv = 0;
            numOfObsv = size(rEvidence.observables,1);
            revelant = [];
            for k = 1:numOfObsv

                oVal =
obj.associatedObservables(strcmp({obj.associatedObservables(:).node},
rEvidence.observables{k,1}));

                if ~isempty(oVal)
                    oVal = oVal.val{1};
```

```
                end

                if isempty(rEvidence.observables{k,2})
                    emptyObsv = emptyObsv+1;
                    revelant(k) = isempty(oVal);
                else
                    if isempty(oVal)
                        revelant(k) = false;
                    elseif ischar(oVal)
                        revelant(k) = strcmp(oVal, rEvidence.observables{k,2});
                    else
                        revelant(k) = (oVal == rEvidence.observables{k,2});
                    end
                end
            end

            if emptyObsv == 0
                type = 'fullObsv';
            elseif emptyObsv == numOfObsv
                type = 'noObsv';
            else
                type = 'partObsv';
            end

        end

        function obj = processRiskElements(obj)
            % clear data - for temporary propose
            obj.riskElements = struct('riskElNode', {}, 'Obsv', {}, 'val', {});
            for k = 1 : length(obj.db.RiskElements)
                obj.riskElements(k).riskElNode = obj.db.RiskElements(k).node;
                for k2 = 1:length(obj.db.RiskElements(k).obsv)
                    if any(strcmp({obj.associatedObservables(:).node},
obj.db.RiskElements(k).obsv(k2)))
                        obj.riskElements(k).Obsv{end+1,1} =
obj.db.RiskElements(k).obsv{k2};
                        oVal =
obj.associatedObservables(strcmp({obj.associatedObservables(:).node},
obj.riskElements(k).Obsv{end,1})).val{1};
                        obj.riskElements(k).Obsv{end  ,2} = oVal;
                    else
                        obj.riskElements(k).Obsv{end+1,1} =
obj.db.RiskElements(k).obsv{k2};
                        obj.riskElements(k).Obsv{end  ,2} = [];
                    end
                end
                obj.riskElements(k).val = '_noanswer';
            end
        end

        function answerQuestions(obj)
            % process Risk Elements
            obj.updateRiskElements;
            fprintf('Answer questions.\nType nothing if some information to
answer the question are not provided.\nType ''exit'' to stop answering questions
and return to command line\n');

            for k = 1:length(obj.riskElements)
```

```
                [type, revelant] =
obj.statusOfRiskElement(obj.riskElements(k).riskElNode);
                if strcmp(type, 'noObsv')
                    continue;
                end

                if ~strcmp(obj.riskElements(k).val, '_noanswer') &&
all(revelant)
                    continue;
                end

                fprintf('\n');
                obj.printRiskElement(obj.riskElements(k).riskElNode);
                fprintf('\n');
                an = input('Type value of risk element expression: ', 's');
                if isempty(an)
                    if strcmp(obj.riskElements(k).val, '_noanswer')
                        obj.riskElements(k).val = '_noInfo';
                    end
                elseif strcmpi(an, 'exit')
                    return;
                else
                    obj.riskElements(k).val = an;
                end
            end

            %process risk factors
            obj.updateRiskFactors;
            for k1 = 1:length(obj.riskFactors)
                for k2 = 1:length(obj.riskFactors(k1).rEvidences)
                    if isnan(obj.riskFactors(k1).rEvidences(k2).val)

obj.printRiskEvidence(obj.riskFactors(k1).rEvidences(k2));
                        an = input('Type value of risk factor (0,1,NaN): ');
                        obj.riskFactors(k1).rEvidences(k2).val = an;
                        fprintf('\n\n');
                    end
                end
            end
            obj.updateRiskFactors;
        end

        function [type, revelant] = statusOfRiskElement(obj, node)
            rElement = obj.riskElements(strcmp({obj.riskElements(:).riskElNode},
node));
            revelant = [];
            emptyObsv = 0;
            numOfObsv = size(rElement.Obsv,1);
            for k = 1:numOfObsv

                oVal =
obj.associatedObservables(strcmp({obj.associatedObservables(:).node},
rElement.Obsv{k,1}));

                if ~isempty(oVal)
                    oVal = oVal.val{1};
                end

                if isempty(rElement.Obsv{k,2})
```

```
                    emptyObsv = emptyObsv+1;
                    revelant(k) = isempty(oVal);
                else
                    if isempty(oVal)
                        revelant(k) = false;
                    elseif ischar(oVal)
                        revelant(k) = strcmp(oVal, rElement.Obsv{k,2});
                    else
                        revelant(k) = (oVal == rElement.Obsv{k,2});
                    end
                end
            end

            if emptyObsv == 0
                type = 'fullObsv';
            elseif emptyObsv == numOfObsv
                type = 'noObsv';
            else
                type = 'partObsv';
            end

        end

        function updateRiskElements(obj)
            %check status of revelacne of assiosiated observables and those
            %in risk elements

            for k = 1:length(obj.riskElements)
                [~, revelant] =
obj.statusOfRiskElement(obj.riskElements(k).riskElNode);
                if ~any(revelant)
                    obj.riskElements(k).val = '_noanswer';
                    % update assiociated observables values
                    for k2 = 1:size(obj.riskElements(k).Obsv,1)
                        obj.riskElements(k).Obsv{k2,2} =
obj.associatedObservables(strcmp({obj.associatedObservables.node},
obj.riskElements(k).Obsv{k2,1})).val{1};
                    end
                end
            end
        end

        function updateRiskFactors(obj)
            for k1 = 1:length(obj.riskFactors)
                obj.riskFactors(k1).outputRatio = NaN;
                for k2 = 1:length(obj.riskFactors(k1).rEvidences)
                [~, revelant] =
obj.statusOfRiskEvidence(obj.riskFactors(k1).rEvidences(k2));
                    if ~isempty(revelant) && ~any(revelant)
                        obj.riskFactors(k1).rEvidences(k2).val = NaN;
                        % update assiociated observables values
                        for k3 =
1:size(obj.riskFactors(k1).rEvidences(k2).observables,1)
                            obj.riskFactors(k1).rEvidences(k2).observables{k3,2}
= obj.associatedObservables(strcmp({obj.associatedObservables.node},
obj.riskFactors(k1).rEvidences(k2).observables{k3,1})).val{1};
                        end
                    end
```

```
                    if ~isnan(obj.riskFactors(k1).rEvidences(k2).val) &&
(obj.riskFactors(k1).rEvidences(k2).val == 1)
                        ratio =
obj.db.getRiskEvidenceByNode(obj.riskFactors(k1).rEvidences(k2).rEvNode).ratio(2
);
                        if isnan(obj.riskFactors(k1).outputRatio)
                            obj.riskFactors(k1).outputRatio = ratio;
                        else
                            obj.riskFactors(k1).outputRatio = max(ratio,
obj.riskFactors(k1).outputRatio);
                        end
                    end
                end
            end
        end

        function rebuild(obj)
            % reprocess all data, answer questions etc
            obj.processRiskElements;
            obj.answerQuestions;
            obj.processRiskFactors;
            obj.answerQuestions;
        end

        function savePatient(obj, file)
            associatedObservables = obj.associatedObservables;
            riskElements = obj.riskElements;
            riskFactors = obj.riskFactors;
            save(file, 'associatedObservables', 'riskElements', 'riskFactors');
        end

        function loadPatient(obj, file)
            s = load(file);
            obj.associatedObservables = s.associatedObservables;
            obj.riskElements = s.riskElements;
            obj.riskFactors = s.riskFactors;
        end

        function riskFactorRaport(obj)
            for k = 1:length(obj.riskFactors)
                if ~isnan(obj.riskFactors(k).outputRatio)
                    rFacDB =
obj.db.getRiskFactorByNode(obj.riskFactors(k).rfNode);
                    rVal = obj.riskFactors(k).outputRatio;
                    rTargetName =
obj.db.getRiskElementByNode(rFacDB.target).name;
                    fprintf('Possibility of %s with risk ratio %.2f\n',
rTargetName, rVal);
                end
            end
        end

    end

end
```