



FP7-ICT-611140 CARRE

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



D.4.1. Semantic Repository Design & Implementation

Z. Deng, G. Gkotsis, A. Third

February 2015

CARRE Contacts

Project Coordinator: Eleni Kaldoudi kaldoudi@med.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 Santariskiu Klinikos	Domantas Stundys	Domantas.Stundys@santa.lt
KTU Kaunas University of Technology	Arunas Lukosevicius	arunas.lukosevicius@ktu.lt
PIAP Industrial Research Institute for Automation & Measurements	Roman Szewczyk	rszewczyk@piap.pl

Disclaimer

This document contains description of the CARRE project findings, work and products. The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.

The content of this publication is the sole responsibility of CARRE consortium and can in no way be taken to reflect the views of the European Union.

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.4.1
Deliverable Title: Semantic repository design & implementation
Responsible Partner: OU – Allan Third
Authors: Z. Deng, G. Gkotsis, A. Third
Input from: OU, BED, KTU
Peer Reviewers: George Drosatos (DUTH), Vaidotas Marozas (KTU), Rafal Kloda (PIAP)
Task: T.4.1. Scalable semantic repository design and implementation & T.4.2. Data privacy and security issues
Task duration: 7 months: 1 July 2014 to 28 February 2015
Work Package: WP4: Data Enrichment, correlation and interlinking
Work Package Leader: OU – Allan Third

Due Date: 28 February 2015
Actual Delivery Date: 3 March 2015
Dissemination Level: PU
Nature: P
Files and format: Deliverable report: 1 pdf file
Software: 1 archive containing the RESTful API web application
Version: 08
Status: ☐ Draft
☒ Consortium reviewed
☒ WP leader accepted
☒ Coordinator accepted
☐ EC accepted

Table of Contents

Executive Summary.....	7
Terms and Definitions	8
1. Introduction	9
2. Semantic Repository Architecture	10
3. RDF management	11
3.1. <i>RDF graphs</i>	11
3.1.1. Private graphs	11
3.1.2. Public graph	12
3.2. <i>Accessing RDF through SPARQL</i>	12
3.3. <i>Performance</i>	13
4. Account authentication	15
5. CARRE RESTful API	18
5.1. <i>CARRE services</i>	18
5.2. <i>Dereferenceable URIs</i>	21
5.3. <i>Code Metrics</i>	22
6. Data privacy and security issues	23
Annex 1 RESTful Swagger documentation.....	25
Annex 2 Repository RESTful API Software.....	30
<i>What is CARRE Repository RESTful API?</i>	31
<i>Access</i>	31
<i>Download</i>	31
<i>CARRE Repository RESTful API is Open Source</i>	31

List of Figures

Figure 1. CARRE repository internal architecture.	10
Figure 3. Comparison of Query Mixes per Hour for different RDF databases [Morsey et al.]	14
Figure 4. CARRE's login form.....	15
Figure 5. CARRE's registration form.	15
Figure 6. CARRE's login authentication and authorisation workflow.	16
Figure 7. CARRE's logoff workflow.....	17
Figure 8. RESTful API homepage	19
Figure 9. UserProfile method invocation.	20
Figure 10. Screenshot of the Activity Digest integrated in the Mac OS calendar application.	21
Figure 11. An example webpage showing the output of the dereferenceable URIs.	21
Figure 12. CARRE security technology stack.....	23

List of Tables

Table 1. Code metrics of application constructor.	22
Table 2. Code metrics of CARRE services implementation.	22
Table 3. Code metrics of Virtuoso connection implementation.	22

Document Revision History

Version	Date	Modifications	Contributors
v01	3 February 2015	new content – outline	G. Gkotsis
v02	12 February 2015	content in sections 2,3 & 4	G. Gkotsis
v03	13 February 2015	content in section 5	G. Gkotsis
v04	20 February 2015	minor modifications - offline exporting of the document for formatting	G. Gkotsis
v05	21 February 2015	Content in section 6 from BED. Minor modifications.	A. Third, Z. Deng
v06	27 February 2015	Comments from reviewers, formatting & presentation	G. Gkotsis
v07	2 March 2015	Comments from reviewers	G. Gkotsis, A. Third
v08	3 March 2015	Editing towards layout uniformity	E. Kaldoudi, G. Drosatos

Executive Summary

This deliverable reports on CARRE's semantic repository design and implementation (D.4.1). CARRE's repository constitutes the backend of CARRE's overall architecture and is responsible for storing both public and private (sensor) data in an RDF format. It also realises a set of components that allow depositing and accessing this data for the CARRE applications under development. The report discusses the design and implementation decisions towards the above, taking into account various factors such as performance, security and robustness. The source code is also released (open source) as part of this deliverable in order to complement the information provided in this document.

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 document¹.

Term	Definition
CARRE repository	The backend major component of the CARRE platform responsible for storing, indexing and accessing the public and private RDF data.
Flask	A lightweight web application framework written in Python.
Linked Data	A method of publishing structured data so that it can be interlinked and become more useful.
Linked Open Data cloud	A collection of Linked Open Data repositories interconnected and publicly accessible through Semantic Web Technologies.
RDF	A standard model for data interchange on the Web.
RESTful API	A Web API that adheres to the REpresentational State Transfer architectural constraints.
SPARQL	An RDF query language.
Swagger	A specification and complete web application framework implementation for describing, producing, consuming, and visualizing RESTful web APIs (collections of web resources) and their implementations as web services.
Triple	A statement in the subject-predicate-object expression.
URI	A string of characters used to identify a name of a resource.
URL	A specific character string that constitutes a reference to a resource.
Virtuoso	A middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system.
Web API	An Application Programming Interface accessible on the Web.

¹ Terms and definitions are taken from W3C and Wikipedia.

1. Introduction

This document presents the semantic repository developed for managing large volumes of data in the form of RDF triples. The CARRE repository acts as the central point of information storage for all CARRE applications. It conforms to the principles of the Semantic Web and the guidelines of Linked Data. The Linked Data guidelines can be summarised as follows²:

- Use URIs as names for things
- Use HTTP URIs, so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF³, SPARQL)
- Include links to other URIs, so that people can discover more things.

Information stored in the CARRE repository consists of “RDF triples”. RDF is a standard format for representing semantic data on the Web; an item of RDF data is a *triple*, which corresponds to a statement of the form “subject predicate object”. Each term is a URI, often drawn from a standard vocabulary or ontology, making it easy to link triples from different sources – to allow *Linked Data*. RDF can be accessed through a *SPARQL endpoint*. SPARQL is a query language, much like SQL in syntax. The triples stored in the CARRE repository are either public or private. For private data, data privacy and security mechanisms have been deployed.

In addition to the above “de facto” standards, we have implemented and published the following web services:

- A CARRE-specific Web API used by CARRE applications that have been prescribed and undergoing development in currently active Work Packages.
- Dereferenceable URIs, that allow the seamless access and consumption of CARRE-generated data.
- Authentication and authorization services for CARRE users and applications.

Section **Error! Reference source not found.** gives an overview of the semantic repository architecture. Section 3 discusses how we have used and configured our database engine (Virtuoso) for managing all of the RDF data. Section 4 presents how the CARRE users and applications can manage their accounts with respect to CARRE's repository. Section 5 presents CARRE's RESTful API and section 6 discusses security and privacy issues that will be taken into account throughout the development of CARRE platform. Finally, Annex 1 presents the RESTful API's specification that can be used by swagger clients.

² <http://www.w3.org/DesignIssues/LinkedData.html>

³ <http://www.w3.org/RDF/>

2. Semantic Repository Architecture

Figure 1 illustrates the architecture of the CARRE repository and highlights the various components realised. Starting from left to right, *CARRE applications* and *users* are external to the CARRE repository and the primary clients of the repository's services. CARRE users are the users that are connecting on the Backend of the repository initially for creating an account. The *Authentication* component is responsible for certifying the users for the CARRE applications. After a *CARRE application* has identified and verified a user, it has the required authentication credentials to connect directly to the *RESTful API*. The *RESTful API* component exposes a number of CARRE-related services that allow accessing RDF data, both public and private. A dedicated component is built to implement *dereferenceable URIs* on top of the RDF repository.

Communication with the RDF repositories both by the API and where relevant the public LOD cloud is facilitated by the public and authenticated SPARQL endpoints accordingly. Finally, the actual RDF repositories are developed using the Virtuoso engine. The public and private RDF data are separated logically; however, they are both implemented under the same Virtuoso server with respect to the Linked Data principles.

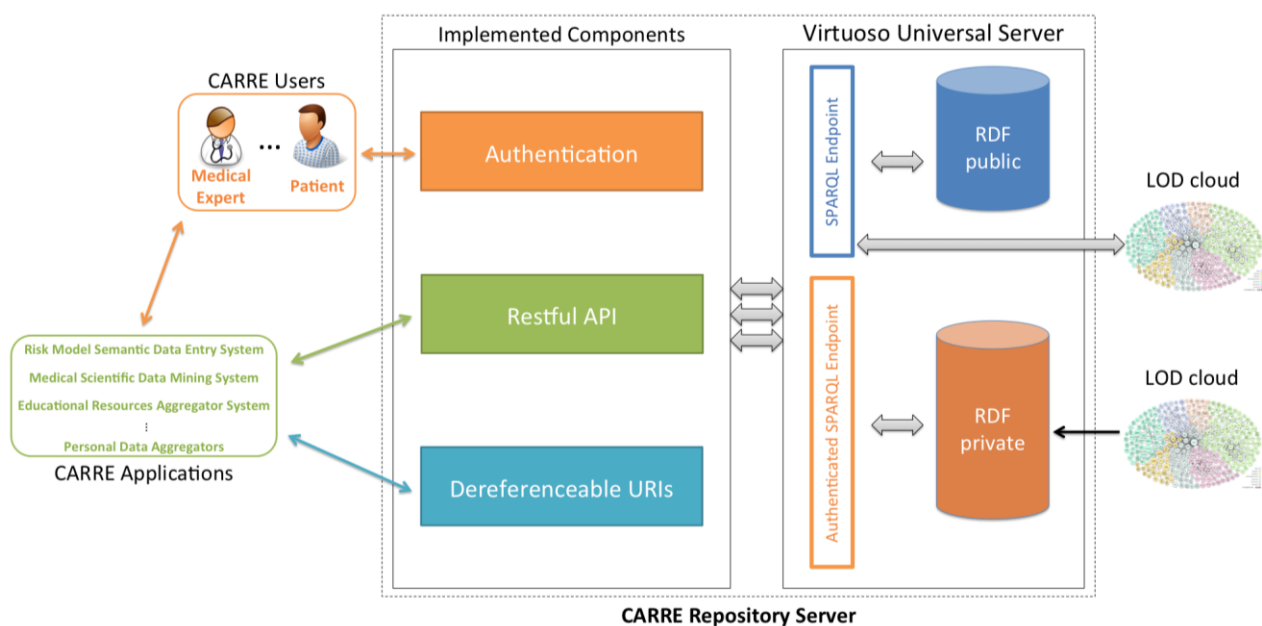


Figure 1. CARRE repository internal architecture.

The sections that follow present in more detail the implementation details for each component.

3. RDF management

Storage and management of RDF data is implemented using the open source version of *Virtuoso Universal Server*. More specifically, the OU team has installed the latest version of Virtuoso 6⁴ (6.4) on a 6.6 CentOS (Santiago) machine with 8GB of RAM and a dual core 2.50GHz Xeon with 30GB expandable Hard Disk space. Virtuoso is both a database and a middleware engine⁵.

From a database perspective, it is implemented following the relational database management system. Hence Virtuoso is fully SQL-92 compliant and supports several of the SQL 200n features. At the same time, Virtuoso can also be considered an XML and RDF database. It implements protocols that allow the management of RDF triples (hence “universal”) and allows loading of N3, Turtle, and RDF/ XML files. To achieve the above, the basic model of Virtuoso is an extension of the object-relational models. Furthermore, as a database, Virtuoso supports entity and referential integrity. Finally, Virtuoso offers a number of client libraries and drivers that permit database connection with a number of different solutions (e.g. ODBC⁶, JDBC⁷ and ADO.NET⁸).

As a middleware engine, Virtuoso follows a component-based architecture and allows the installation of components that are connected either locally or remotely to its database. These components vary, from services very tightly connected to its database (e.g. remote execution of stored procedures) to web services (e.g. SOAP, WSDL, and UDDI) and web applications (e.g. Wiki, AddressBook and Weblog).

In our approach, we exploit the following mechanisms of the Virtuoso server:

- We use the relational schema to programmatically create and register CARRE user accounts. Where needed, we made minor modifications to various stored procedures.
- We have installed the OAuth authentication module provided by Virtuoso in order for applications to execute user-level queries based on corresponding user/graph permissions.
- We use the RDF repository to implement the CARRE vocabulary defined in previous deliverables. Hence the RDF statements populated together with the CARRE vocabularies constitute a complete solution for the CARRE repository.

We discuss below details concerning the setup and configuration of the RDF repository.

3.1. RDF graphs

Virtuoso allows the storage of RDF triples under an arbitrary number of “named graphs”. A named graph, identified by a URI, can contain a set of RDF triples which can be stored and queried independently. In effect, each triple is extended to become a ‘quad’, with the first field being the graph name⁹. Triples need only be unique *within* a graph. The SPARQL standard allows the use of graph names to construct complex queries limited to particular graph(s) or drawing data from multiple graphs together. Hence Virtuoso allows accessing the content of these graphs through SPARQL queries.

3.1.1. Private graphs

Virtuoso’s named graphs support graph-level security¹⁰. That is to say, access to the triples within a particular graph can be restricted to a certain user or users - an important factor when dealing with confidential health-related data. Each user in the CARRE repository is given a unique private graph after her username in which

⁴ <http://virtuoso.openlinksw.com/>

⁵ http://virtuoso.openlinksw.com/virt_faq/

⁶ <http://support.microsoft.com/kb/110093>

⁷ <http://www.oracle.com/technetwork/java/javase/jdbc/index.html#corespec40>

⁸ <http://msdn.microsoft.com/en-us/library/aa286484.aspx>

⁹ This is the reason why Virtuoso is also described as a quad store.

¹⁰ <http://docs.openlinksw.com/virtuoso/rdfgraphsecurity.html>

all data relating to that user is stored. More specifically, the setup of CARRE's RDF store was implemented as follows:

1. `DB.DBA.RDF_DEFAULT_USER_PERMS_SET ('nobody', 0);`

All graphs are declared private for all users. Hence, if a user or application queries a graph without providing the correct credentials, the result set is empty.

When a new user registers, the following actions take place:

2. `DB.DBA.USER_CREATE ('JohnSmith', 'PASSWORD');`

A new user is created using the above command. Usernames must be unique. A password can change using the following command:

```
USER_SET_PASSWORD('JohnSmith', 'NEWPASSWORD');
```

3. `GRANT SPARQL_UPDATE to "JohnSmith";`

The SPARQL_UPDATE role is by default assigned to all users. This is needed for step #4.

4. `DB.DBA.RDF_GRAPH_USER_PERMS_SET ('https://carre.kmi.open.ac.uk/users/JohnSmith', 'JohnSmith', 3);`

This last step allows the user to have read-write permission on her private graph. Notice that all private/user-defined named graphs are using the following prefix:

<https://carre.kmi.open.ac.uk/users/>

Therefore, the private graph for user JohnSmith has the URI:

<https://carre.kmi.open.ac.uk/users/JohnSmith>

Note that all user graphs are assigned an HTTPS URI, and therefore all HTTP interactions with a user graph or its contents are encrypted, as well as limited to authenticated users with the appropriate permissions.

3.1.2. Public graph

As already discussed, CARRE's RDF repository stores public data. This data is accessible to all users, both registered and anonymous. To address this need, all of the data are stored under a common graph, which we have defined as the "default graph" of Virtuoso. This is done by setting this option within the `virtuoso.ini` configuration file:

```
DefaultGraph = https://carre.kmi.open.ac.uk/public
```

A default graph in Virtuoso is a special graph since it allows users to execute SPARQL queries within this graph without providing the graph name. Finally, in order to make this default, public graph accessible to all users with read-only permission, the following command was issued:

```
DB.DBA.RDF_GRAPH_USER_PERMS_SET ('https://carre.kmi.open.ac.uk/public', 'nobody', 1)
```

A special set of users-applications are granted with write permissions on the public graph. This set refers to users administered by CARRE application developers in order to allow them to deposit RDF statements onto the public graph. For instance an "educational data aggregator user-application" may issue the following SPARQL query, as follows:

```
INSERT DATA
```

```
{  
  <http://example.org/subject> <http://example.org/predicate> <http://example.org/object>  
}
```

3.2. Accessing RDF through SPARQL

Having set up the public and private graphs, a developer/application may access the repository through SPARQL queries. Virtuoso SPARQL can be used through any SQL call level interface (CLI) supported by

Virtuoso (i.e., ODBC, JDBC, OLE-DB, ADO.NET, XMLA). Prefixing a SQL query with the keyword "sparql" will invoke SPARQL instead of SQL, through any SQL client API. We illustrate below a representative list of CARRE-related SPARQL queries:

- *Inserts* a triple into the private graph of user.

```
INSERT IN <http://carre.kmi.open.ac.uk/users/JohnSmith> {
<http://carre.kmi.open.ac.uk/ontology/sensors.owl#provider226866082>
<http://carre.kmi.open.ac.uk/ontology/sensors.owl#hasWeight> 65.9451 }
```

- *Removes* all triples from user's private graph.

```
WITH <https://carre.kmi.open.ac.uk/users/JohnSmith> DELETE { ?subject ?predicate ?object }
WHERE { ?subject ?predicate ?object }
```

- *Selects* all triples from user's private graph.

```
SELECT ?subject ?predicate ?object FROM <https://carre.kmi.open.ac.uk/users/JohnSmith>
WHERE { ?subject ?predicate ?object }
```

- *Selects* daily step count from user's private graph since January 1st, 2015.

```
SELECT      max(?v1)      as      ?value      xsd:date(?d)      as      ?d      FROM
<https://carre.kmi.open.ac.uk/users/JohnSmith> WHERE
```

```
{
  ?subject ?predicate ?object.
  ?subject <http://carre.kmi.open.ac.uk/ontology/sensors.owl#has_date> ?d1.
  ?d1 <http://carre.kmi.open.ac.uk/ontology/sensors.owl#has_value> ?d.
  ?subject <http://carre.kmi.open.ac.uk/ontology/sensors.owl#has_steps> ?v.
  ?v <http://carre.kmi.open.ac.uk/ontology/sensors.owl#has_value> ?v1.
  FILTER ( ( xsd:date(?d) >= '20150101'^xsd:date) )
}
GROUP BY xsd:date(?d)
```

3.3. Performance

CARRE's repository is implemented with large volumes of data in mind. Virtuoso has been reported in a number of studies as being robust and scalable. This scalability accounts both performance and the volume of data that can be stored. The website of the World Wide Web Consortium presents a thorough list of studies concerning RDF storage and benchmarks¹¹. We highlight some reports around performance aspects of Virtuoso:

- *Volume*

According to Virtuoso's FAQ¹² and based on our server's specifications, CARRE server can store more than 500M triples. Additionally, if needed, data can be partitioned to multiple servers multiplying the above limit proportionally to the supplementary servers deployed¹³.

- *Multiple connections*

CARRE's Virtuoso repository is multithreaded for all of its components and can easily support a large number of clients (i.e. users and/or applications). The most prominent example is the publicly-

¹¹ <http://www.w3.org/wiki/RdfStoreBenchmarking>

¹² http://virtuoso.openlinksw.com/virt_faq/

¹³ See <http://www.openlinksw.com/weblog/oerling/?id=1487> for a more detailed discussion.

accessible DBpedia SPARQL endpoint¹⁴ which serves thousands of requests per day on a single-node server.

– *Performance*

A paper published by Morsey et al.¹⁵ reports that “Virtuoso outperforms Sesame for all datasets” (Sesame¹⁶ is one of the biggest RDF store competitors). Moreover, in their paper, the authors propose a novel, pure RDF benchmark suite that can be applied for benchmarking real world data. The results of their evaluation show that “Virtuoso was clearly the fastest triple store” and was the only one able to complete all queries before the timeout period. This is also illustrated in Figure 2, where the Query Mixes per Hour (QMpH - a collection of mixed queries and different parameters) is presented for different RDF stores, over a dataset of approximately 154M triples. Finally, similar results are reported earlier by Bizer and Schultz in another paper¹⁷.

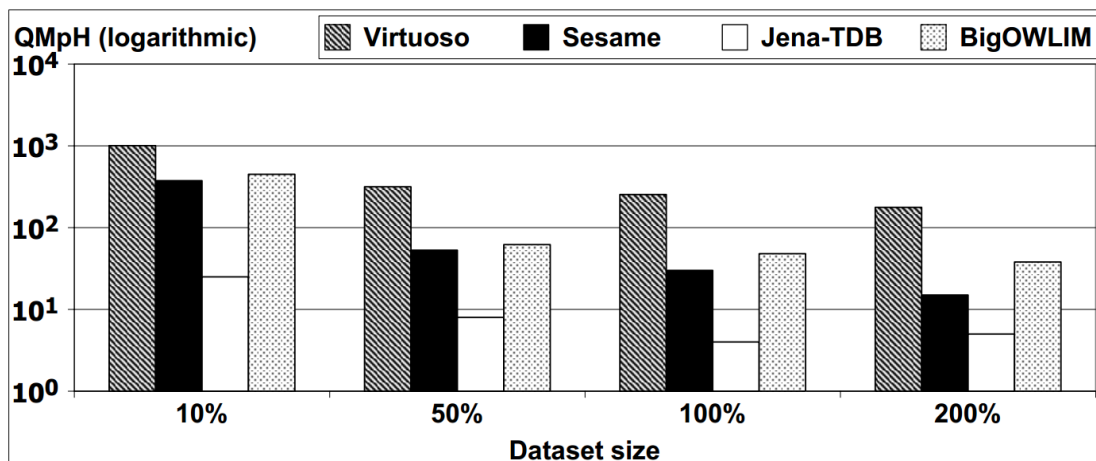


Figure 2. Comparison of Query Mixes per Hour for different RDF databases [Morsey et al.]

¹⁴ <http://dbpedia.org>

¹⁵ M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo. DBpedia SPARQL benchmark—performance assessment with real queries on real data. In *The Semantic Web—ISWC 2011*, pages 454–469. Springer, 2011.

¹⁶ <http://rdf4j.org/>

¹⁷ C. Bizer and A. Schultz. Benchmarking the performance of storage systems that expose sparql endpoints. *World Wide Web Internet And Web Information Systems*, 2008.

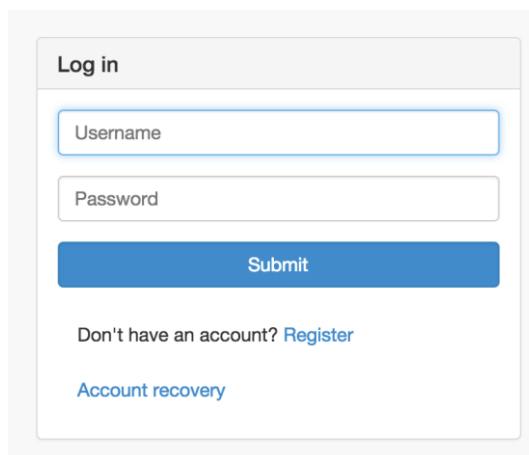
See also <http://www.openlinksw.com/weblog/oerling/?id=1484>

4. Account authentication

A user who wishes to create an account and use CARRE services must first register an account for accessing CARRE's repository. To accommodate the above, we have extended the sensor data aggregators website¹⁸, which is accessible in the following address:

<https://carre.kmi.open.ac.uk/devices>

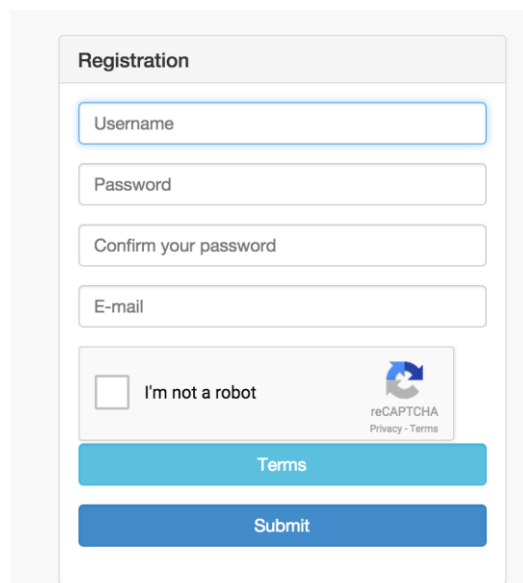
The website prompts the user for his username and password:



The login form is titled "Log in" and is contained within a light gray box. It features two input fields: "Username" and "Password". Below these fields is a blue "Submit" button. At the bottom of the form, there are two links: "Don't have an account? Register" and "Account recovery", both in blue text.

Figure 3. CARRE's login form.

New users follow the "Register" link to create a new account:



The registration form is titled "Registration" and is contained within a light gray box. It features four input fields: "Username", "Password", "Confirm your password", and "E-mail". Below these fields is a checkbox labeled "I'm not a robot" next to a reCAPTCHA logo. To the right of the checkbox is a link for "Privacy - Terms". At the bottom of the form, there are two buttons: a blue "Terms" button and a blue "Submit" button.

Figure 4. CARRE's registration form.

The user can then log in to the website and connect their sensor devices to CARRE's repository (for more information about this part, the reader may consult D.3.2). In addition to this, the "CARRE devices" website functions as a *Single Sign On* (SSO) server that allows users to sign in to CARRE applications. The authentication and authorisation workflow is illustrated in Figure 5. This workflow implements the OAuth 2.0 "meta protocol" enables secure authentication without the need to, for example, share passwords for every resource access.

¹⁸ Sensor data aggregators are reported in D.3.2.

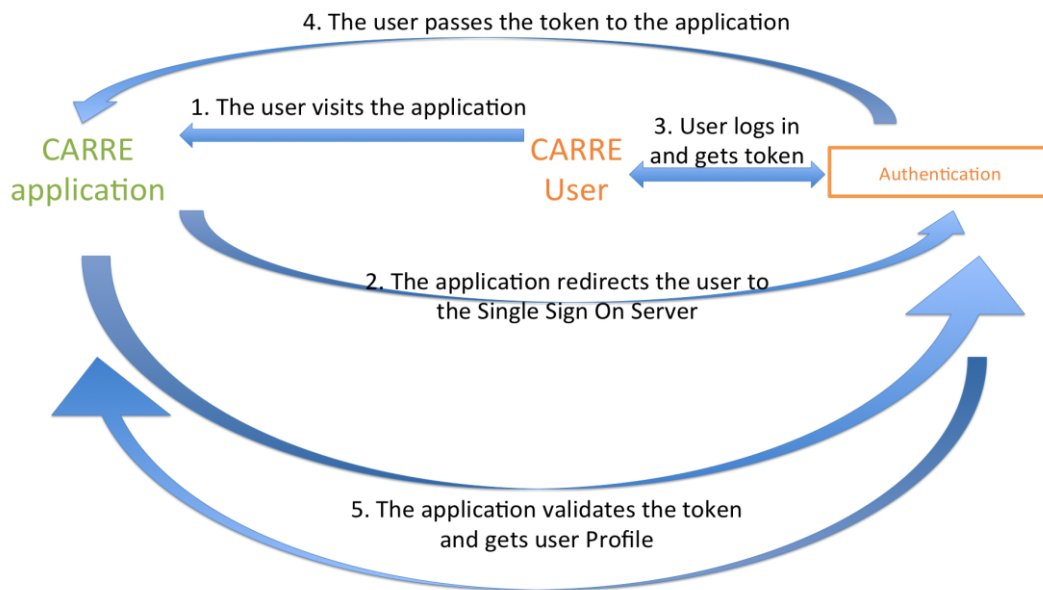


Figure 5. CARRE's login authentication and authorisation workflow.

1. User initially visits the CARRE application. At this stage, we assume the user is not logged on yet.
2. CARRE application redirects the user to the authentication/Single Sign On component of the CARRE repository. The redirection URL contains a callback URI in the following format:

<https://carre.kmi.open.ac.uk/devices/accounts/login?next=https://carre.application.url/listener>

The authentication component checks the host of the callback URI (i.e. `carre.application.url`) whether it is listed as a valid host name. If and only if the host is valid, step #3 follows.

3. CARRE authentication checks if user is already logged in (using session data of her browser). If the user is not logged in, a login box URL is presented (see Figure 3).
4. User successfully logs in (or is already logged in) and receives a redirection url to the carre application. The redirection URL passes through GET parameters user information the access token of his account.
5. CARRE application receives the token and sends it back to the Authentication module.

https://carre.kmi.open.ac.uk/ws/userProfile?token=TOKEN_VALUE

If the token is valid, a JSON response is produced as follows:

```

{
  "username": "JohnSmith",
  "graphName": "https://carre.kmi.open.ac.uk/users/JohnSmith",
  "email": "JohnSmith@example.com"
}
  
```

When and if the user himself or the CARRE application wishes to logoff the user, the authentication server must be informed as well. This is done similarly to the logon process, by visiting the following URL:

<https://carre.kmi.open.ac.uk/devices/accounts/logout?next=http://carre.application.url/logout>

Figure 6 illustrates the logoff process, which is a simplified version of the logon process. The workflow which is initiated by the application results in logging off the user from all other applications; hence this will only be used in cases that such thing is needed. For all other cases, CARRE application can choose to close the user's session, similarly to any standalone web application.

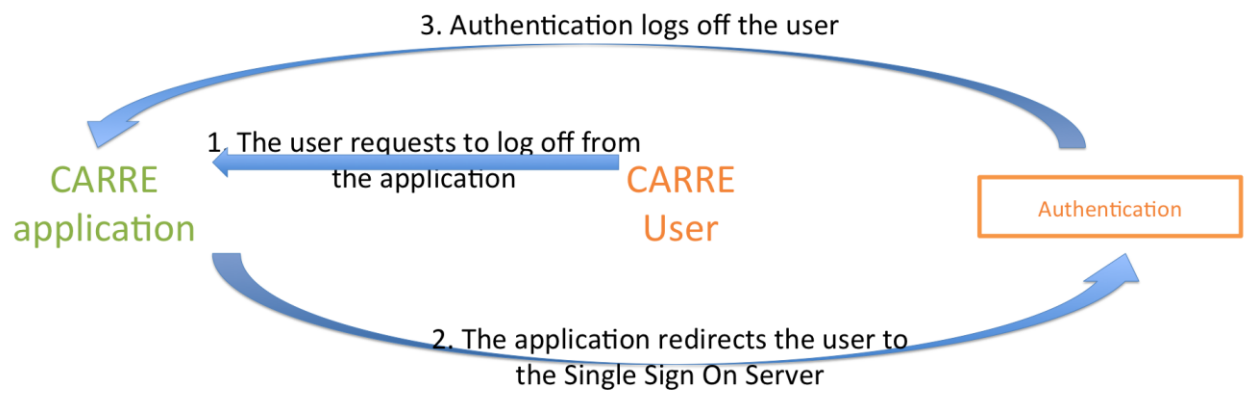


Figure 6. CARRE's logoff workflow.

5. CARRE RESTful API

The third component developed on top of CARRE's Virtuoso server constitutes a web service. This application aims to securely expose a number of CARRE-related methods. These methods are divided into two categories, which we present separately below.

5.1. CARRE services

This application is a RESTful API developed using Flask¹⁹, which is a Python-based web framework. In our approach, we also used the following components:

- FlaskRESTful²⁰, an extension for Flask that adds support for quickly building RESTful APIs.
- Flask-restful-swagger²¹, a wrapper for flask-restful which enables swagger²² support. Swagger constitutes an API description and representation framework that allows humans and machines to efficiently read and consume RESTful APIs.

The homepage of the above service is served as swagger web pages in the following address.

<https://carre.kmi.open.ac.uk/ws>

Figure 7 is a screenshot of the service homepage. The aim of this page is to list all methods available together with a documentation of each method. The same information is also provided as a JSON object²³, which can be used directly by modern swagger client-libraries²⁴. Currently, the methods implemented are the following:

- **query**. POST method that takes as input a *sparql* query and a *token* value. The response is a JSON object with the results of the query.
- **user**. POST method that can be used to remove a user from the system. Takes as input the *username* and the *token*. Token must match the username's token.
- **measurement**. GET method that takes as input the *name* of the measurement (e.g. Steps) and the *token* of the user. Returns a daily digest of the user with the values for the measurement passed.
- **userProfile**. GET method that takes as input the token of a user. If valid, returns user profile information (username, user's graphName, email, see Figure 8).
- **authenticate**. GET method that requires HTTP Basic authentication. If username and password are valid, the user's token is returned.
- **ical.ics**. GET method that requires HTTP Basic authentication. It returns an ical calendar that contains the activity digest of the user. Can also be accessed as a feed: webcal://carre.kmi.open.ac.uk/ws/ical.ics
- **icalToken.ics**. GET method similar to ical.ics above that does not use Authentication but takes as input the user's *token*. Can also be accessed as a feed: webcal://carre.kmi.open.ac.uk/ws/icalToken.ics?token=XXX (see Figure 9 for an example).

¹⁹ <http://flask.pocoo.org/>

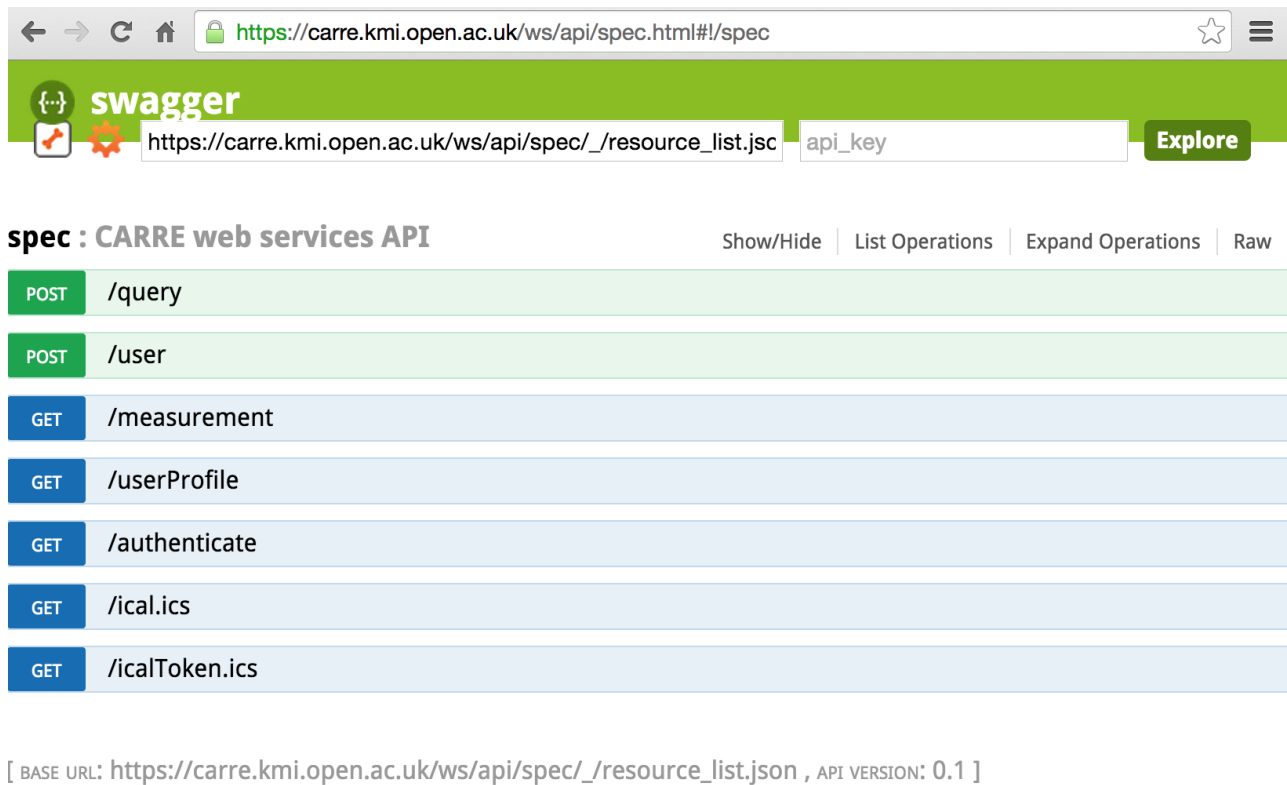
²⁰ <https://flask-restful.readthedocs.org>

²¹ <https://github.com/rantav/flask-restful-swagger>

²² <http://swagger.io/>

²³ <https://carre.kmi.open.ac.uk/ws/api/spec.json>, see Annex 1.

²⁴ <https://github.com/swagger-api/swagger-spec#additional-libraries>



spec : CARRE web services API

POST /query

POST /user

GET /measurement

GET /userProfile

GET /authenticate

GET /ical.ics

GET /icalToken.ics

[BASE URL: https://carre.kmi.open.ac.uk/ws/api/spec/_/resource_list.json , API VERSION: 0.1]

Figure 7. RESTful API homepage

The homepage of the service does not only serve as documentation. It allows developers to use the web services directly and test the behaviour of each method. For example, a developer may want to see the return object for the method `userProfile`. In that case, she can expand the corresponding method, fill in the input parameters and try the method, as shown in Figure 8.

GET /userProfile

Implementation Notes

this provides user profile information

None

Response Class

Model | Model Schema

JSON

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
token	[REDACTED]	OAuth token	query	String

Error Status Codes

HTTP Status Code	Reason
201	Executed.
405	Invalid input

[Try it out!](#) [Hide Response](#)

Request URL

[https://carre.kmi.open.ac.uk:443/ws/userProfile?token=\[REDACTED\]](https://carre.kmi.open.ac.uk:443/ws/userProfile?token=[REDACTED])

Response Body

```
{
  "username": "gkotsis",
  "graphName": "https://carre.kmi.open.ac.uk/users/gkotsis",
  "email": "gkotsis@gmail.com"
}
```

Response Code

200

Response Headers

```
{
  "Date": "Fri, 13 Feb 2015 11:42:20 GMT",
  "Server": "Apache",
  "Connection": "close",
  "Content-Length": "112",
  "Content-Type": "application/json"
}
```

Figure 8. UserProfile method invocation.

Finally, it is worth noting that the service described in this section is undergoing development. Our intention is to extend the number of web methods provided to developers as CARRE applications mature. For instance, the educational data aggregator developed by the team in DUTH is expected to request information related to risk associations. An example of a CARRE-related web method is the calendar-based provision of sensor data. This method is providing an iCalendar-formatted²⁵ feed that can be integrated into any modern calendar application. More specifically, the feed is providing a daily activity digest as recorded through the sensor data aggregators. Figure 9 shows a screenshot of the feed as integrated into Mac OS's calendar application.

²⁵ <http://en.wikipedia.org/wiki/ICalendar>

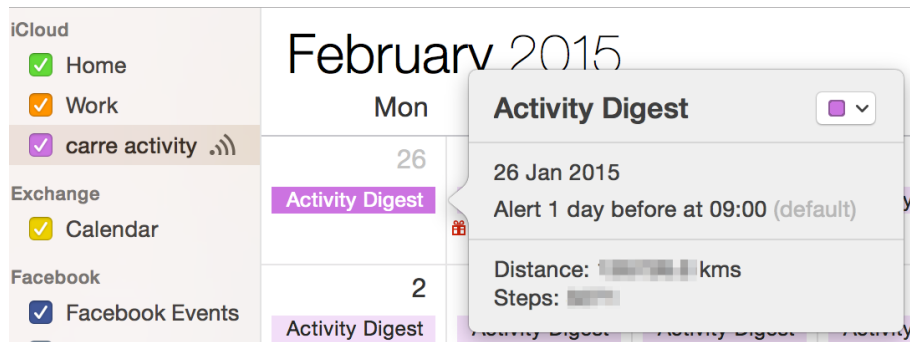


Figure 9. Screenshot of the Activity Digest integrated in the Mac OS calendar application.

5.2. Dereferenceable URIs

In addition to the above CARRE-related web services, a service has also been implemented for realising dereferenceable URIs, as prescribed by the DoW. A dereferenceable URI is “a resource retrieval mechanism that uses any of the internet protocols (e.g. HTTP) to obtain a copy or representation of the resource it identifies.”²⁶. That is to say, it should be possible to obtain a representation of an RDF entity identified by the URI <https://example.uri> by simply retrieving the contents of that URL via HTTP(S). To achieve this, a method is being implemented as part of the above flask application. This method is not exposed as a swagger method. For example, Figure 10 shows the webpage when accessing the private URI <https://carre.kmi.open.ac.uk/users/gkotsis>. The method identifies whether the **URI** to be accessed is a private and prompts the user for an HTTP Basic Authentication (over https). If the credentials passed are valid, a SPARQL query is passed on Virtuoso in the following form:

```
DESCRIBE <URI> FROM <https://carre.kmi.open.ac.uk/users/USERNAME>
```

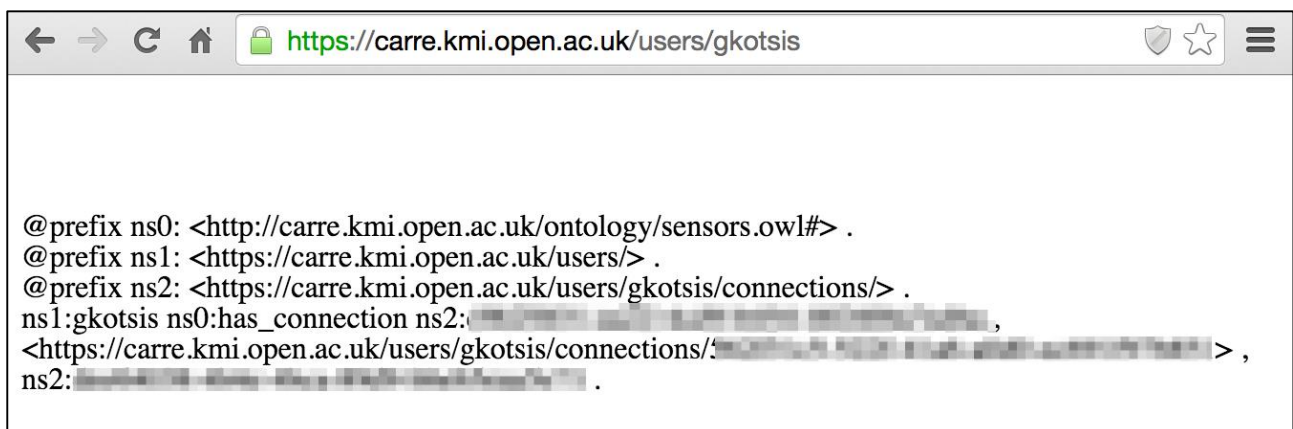


Figure 10. An example webpage showing the output of the dereferenceable URIs.

Figure 10 shows 3 triples that state that user “gkotsis” has one connected device. Each one of the links contained can be visited through an HTTP or HTTPS request, as appropriate, in order to continue navigation of information in a seamless way.

²⁶ http://en.wikipedia.org/wiki/Dereferenceable_Uniform_Resource_Identifier

5.3. Code Metrics

Table 1. Code metrics of application constructor.

__init__.py Python code metrics	
Number of Lines of Code (LOC)	392
Number of Logical Lines of Code (LLOC)	182
Number of Source Lines of Code (SLOC)	354
Number of Python Comment Lines	8
Number of Lines Representing Multi-line Strings	0
Number of Blank Lines	38

Table 2. Code metrics of CARRE services implementation.

carre.py Python code metrics	
Number of Lines of Code (LOC)	363
Number of Logical Lines of Code (LLOC)	263
Number of Source Lines of Code (SLOC)	309
Number of Python Comment Lines	12
Number of Lines Representing Multi-line Strings	0
Number of Blank Lines	54

Table 3. Code metrics of Virtuoso connection implementation.

virtuoso.py Python code metrics	
Number of Lines of Code (LOC)	173
Number of Logical Lines of Code (LLOC)	137
Number of Source Lines of Code (SLOC)	156
Number of Python Comment Lines	1
Number of Lines Representing Multi-line Strings	0
Number of Blank Lines	17

6. Data privacy and security issues

The CARRE system references the ISO/IEC 27002:2013²⁷ code of practice for comprehensive information security control and risk management. The system will incorporate controls which are aligned to ISO/IEC 27018:2014²⁸ best practices for protection of Personal Identifiable Information (PII).

Service Organization Controls (SOC)²⁹ 2 (security controls) will guide the trust principles of security, availability and processing integrity, which allows the CARRE system processing to be accurate, complete, fast and authorized.

The CARRE API security technology stack is guided by Neo-security Stack³⁰, illustrated by Figure 11.

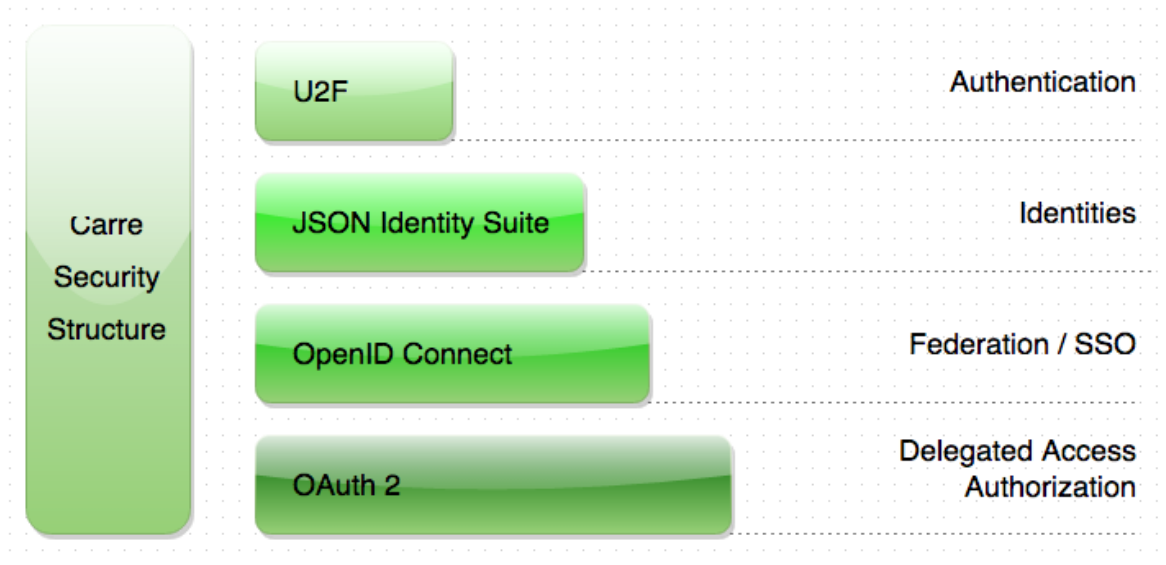


Figure 11. CARRE security technology stack.

The CARRE authentication system implements OAuth 2, which is a “meta protocol” providing a useful foundation for other protocols (e.g. OpenID Connect, NAPS and UMA). OAuth is very important in the CARRE API system because it features:

- Delegated access to third parties applications
- Reduce password sharing between users and third-parties
- Enable the revocation of access

OAuth 2 is used by the CARRE system for delegated access to the CARRE API.

HTTPS is always preferred in all CARRE system and third party web applications. The CARRE system maintains a list of allowed third parties, which includes key information:

- Callback URL which is used by OAuth 2 and SSO
- Third-parties contact and permissions (controlled by CARRE team)

Before deployment with patient data, it is proposed that the existing authentication model be extended with the Universal Second Factor (U2F) protocol. This allows the CARRE system to augment the security of password

²⁷ http://www.iso.org/iso/catalogue_detail?csnumber=54533

²⁸ http://www.iso.org/iso/catalogue_detail.htm?csnumber=61498

²⁹ http://en.wikipedia.org/wiki/Service_Organization_Controls

³⁰ <http://www.twobotechnologies.com/blog/2012/08/cloud-security-standards.html>

infrastructure by adding a strong second factor to user login. Google Authenticator³¹ on Android mobile device could serve as the second factor, which is a free application that has gained popularity recently.

The JavaScript Object Notation (JSON) based Identity protocol currently used in the security model of the CARRE system should also be extended. The JSON data format carries information with defined ways to encode tokens, symmetric / asymmetric keys and digital signatures. The JSON Web Token (JWT) specification defines the way to encode token in this JavaScript format; these lightweight tokens can be used in HTTP headers and query strings. The JSON Web Key (JWK) and JSON Web Signature (JWS) specifications define the way to encode encryption keys and digital signatures.

The implementation of federation and Single Sign On (SSO) is guided by OpenID Connect, which is essentially the third version of OpenID (however it is a complete rewrite, and not compatible with previous versions). OpenID Connect is an HTTP-based protocol that provides SSO. It is built atop of OAuth 2 and achieves higher Levels of Assurance (LoA) compared to other similar purpose protocols like SAML and WS-Federation.

CARRE's Virtuoso Universal Server implements privacy mechanisms for its users. In particular, as discussed earlier in Section 3.1, users' private data are kept in Virtuoso's private graphs, which by design do not allow queries by one user to access the data of another user. Also, the CARRE repository by design allows interlinkage between CARRE's public data and the Linked Open Data Cloud, whereas CARRE users' private data only allows this in one direction. More specifically, the user may join from external datasets in order to enrich their data semantically but external datasets may not query the users' private data.

All CARRE subsystems have their own security model, and only give the least permission necessary to other subsystems and/or users. The permission is assigned and controlled by specific members of CARRE team, and will be audited and reviewed regularly.

³¹ <https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2>

Annex 1

RESTful Swagger documentation

The Swagger documentation as a JSON object is provided in the following URL:

<https://carre.kmi.open.ac.uk/ws/api/spec>

```
{
  "description": "CARRE web services API",
  "swaggerVersion": "1.2",
  "produces": [
    "application/json",
    "text/html"
  ],
  "models": {},
  "basePath": "https://carre.kmi.open.ac.uk/ws",
  "apis": [
    {
      "operations": [
        {
          "parameters": [
            {
              "name": "sparql",
              "dataType": "String",
              "allowMultiple": false,
              "required": true,
              "paramType": "query",
              "description": "SPARQL statement to be passed to the RDF repository."
            },
            {
              "name": "token",
              "default": "JSON",
              "allowMultiple": false,
              "required": true,
              "dataType": "String",
              "paramType": "query",
              "description": "Authentication token"
            }
          ],
          "responseClass": "String in the desired format. JSON (default), XML, TURTLE, N3, RDF",
          "notes": "This is the simplest implementation possible. A SPARQL query is passed to the Virtuoso server and results are returned directly to the client.<br/>None",
          "responseMessages": [
            {
              "message": "Executed.",
              "code": 201
            },
            {
              "message": "Invalid input",
              "code": 405
            }
          ],
          "summary": null,
          "nickname": "upload",
          "method": "post"
        }
      ],
      "path": "/query",
      "notes": null,
      "description": null
    },
    {
      "operations": [
        {
          "parameters": [
            {
              "name": "username",
              "dataType": "String",
              "allowMultiple": false,
              "required": true,
              "paramType": "query",
              "description": "Username."
            },
            {
              "name": "token",
              "default": "JSON",
              "allowMultiple": false,

```

```

        "required": true,
        "dataType": "String",
        "paramType": "query",
        "description": "Authentication token"
    },
    ],
    "responseClass": "Error message if any",
    "notes": "Method for deleting a user<br/>None",
    "responseMessages": [
        {
            "message": "Executed.",
            "code": 201
        },
        {
            "message": "Invalid input",
            "code": 405
        }
    ],
    "summary": null,
    "nickname": "upload",
    "method": "post"
}
],
"path": "/user",
"notes": null,
"description": null
},
{
    "operations": [
        {
            "parameters": [
                {
                    "name": "name",
                    "dataType": "String",
                    "allowMultiple": false,
                    "required": true,
                    "paramType": "query",
                    "description": "Name of the measurement"
                },
                {
                    "name": "token",
                    "dataType": "String",
                    "allowMultiple": false,
                    "required": true,
                    "paramType": "query",
                    "description": "OAuth token"
                }
            ],
            "responseClass": "JSON",
            "notes": "this provides measurements<br/>None",
            "responseMessages": [
                {
                    "message": "Executed.",
                    "code": 201
                },
                {
                    "message": "Invalid input",
                    "code": 405
                }
            ],
            "summary": null,
            "nickname": "nickname",
            "method": "get"
        }
    ],
    "path": "/measurement",
    "notes": null,
    "description": null
},
{
    "operations": [
        {
            "parameters": [

```

```

    {
      "name": "token",
      "dataType": "String",
      "allowMultiple": false,
      "required": true,
      "paramType": "query",
      "description": "OAuth token"
    }
  ],
  "responseClass": "JSON",
  "notes": "this provides user profile information<br/>None",
  "responseMessages": [
    {
      "message": "Executed.",
      "code": 201
    },
    {
      "message": "Invalid input",
      "code": 405
    }
  ],
  "summary": null,
  "nickname": "nickname",
  "method": "get"
}
],
"path": "/userProfile",
"notes": null,
"description": null
},
{
  "operations": [
    {
      "parameters": [],
      "responseClass": "JSON",
      "notes": "this provides authentication tokens. Basic authentication is required.<br/>None",
      "responseMessages": [
        {
          "message": "Executed.",
          "code": 201
        },
        {
          "message": "Invalid input",
          "code": 405
        }
      ],
      "summary": null,
      "nickname": "nickname",
      "method": "get"
    }
  ],
  "path": "/authenticate",
  "notes": null,
  "description": null
},
{
  "operations": [
    {
      "parameters": [],
      "responseClass": "JSON",
      "notes": "Provides calendar, also accessible through <a href='webcal://carre.kmi.open.ac.uk/ws/ical.ics'>webcal://carre.kmi.open.ac.uk/ws/ical.ics</a>. Basic authentication is required.<br/>None",
      "responseMessages": [
        {
          "message": "Executed.",
          "code": 201
        },
        {
          "message": "Invalid input",
          "code": 405
        }
      ],

```

```

        "summary": null,
        "nickname": "nickname",
        "method": "get"
    },
    ],
    "path": "/ical.ics",
    "notes": null,
    "description": null
},
{
    "operations": [
        {
            "parameters": [
                {
                    "name": "token",
                    "default": "JSON",
                    "allowMultiple": false,
                    "required": true,
                    "dataType": "String",
                    "paramType": "query",
                    "description": "Authentication token"
                }
            ],
            "responseClass": "JSON",
            "notes": "Provides calendar, also accessible through <a
href='webcal://carre.kmi.open.ac.uk/ws/icalToken.ics?token=XXX'>webcal://carre.kmi.open.ac.uk/ws/icalToken.ics?token=XXX</a>.<
br/>None",
            "responseMessages": [
                {
                    "message": "Executed.",
                    "code": 201
                },
                {
                    "message": "Invalid input",
                    "code": 405
                }
            ],
            "summary": null,
            "nickname": "nickname",
            "method": "get"
        }
    ],
    "path": "/icalToken.ics",
    "notes": null,
    "description": null
}
],
"resourcePath": "/",
"apiVersion": "0.1",
"spec_endpoint_path": "/api/spec"
}

```

Annex 2

Repository RESTful API Software

What is CARRE Repository RESTful API?

The **Repository RESTful API** is a component developed on top of CARRE's Virtuoso server and constitutes a web service. This API aims to securely expose a number of CARRE-related methods.

The RESTful API is developed using Flask, which is a Python-based web framework. In our approach, we also used the following extensions:

- **FlaskRESTful**, an extension for Flask that adds support for quickly building RESTful APIs.
- **Flask-restful-swagger**, a wrapper for flask-restful which enables swagger support. Swagger constitutes an API description and representation framework that allows humans and machines to efficiently read and consume RESTful APIs.

Access

The homepage of the above service is served as swagger web page at: <https://carre.kmi.open.ac.uk/ws>

Download

Repository RESTful API v0.1:

- Source (10 KB): CARRE_Triple_Store.zip (Python code)
download from <http://www.carre-project.eu/innovation/repository-restful-api/>

CARRE Repository RESTful API is Open Source

CARRE Repository RESTful API is Open Source and can be freely used in Open Source applications under the terms GNU General Public License (GPL).

Copyright © 2015, CARRE Project, The Open University (OU), UK