Design and Implementation of LOD Explorer: A LOD Exploration and Visualization Model

Abstract

The quantity of data published on the Web according to principles of Linked Data (LD) is increasing intensely. However, this data is still largely limited to be used up by domain professionals and users who understand LD technologies. Therefore, it is essential to develop tools to enhance intuitive perceptions of LD for lay users. The features of LD point to various challenges for an easy-to-use data presentation. In this research, the technical details behind the LOD Explorer, which is a tool for exploring the Web of Data, is presented, the efficiency and scalability of the system has been tested using several different platforms, and the results of experiments indicated an outstanding performance.

Keywords: *Semantic web; linked open data; linked data browsers; RDF; SPARQL; graph; visualization; interaction*

I. Introduction

So far, the size of the LD has grown tremendously [1]. Consequently, a lot of LD projects are available for use and millions of triples have been put away in triple datasets [2]. However, from the opposite point of view, it is challenging to find exploring tools truly based on RDF standards that are capable to validate the efficiency of these standards [3]. Hence, an improved approach for presenting Web of data is developed so as to facilitate the human inspection of information accessible as LD. The developed system, which is called LOD Explorer , has been implemented with the aim of:

- RDF datasets exploration employing a dynamic visual graph
- using different RDF datasets to be used and connected with each other
- expanding the norm and standardization space of LD
- providing an easy application to be used by everybody for LD Exploration
- presenting data properties of LD resources
- searching within the resources to find its connections
- fetch and display an image of the resource
- providing flexibility for adding plugins

The fundamental idea of the LOD Explorer is to deliver an easy approach to discover, understand, and learn the published resources along with the W3C standards for Semantic Web

The novelty of the proposed approach is the capability to straightaway explore a SPARQL endpoint utilizing the greatness of JavaScript and its libraries without the need of a server side module which has been used by other systems presented in [2], [4]. LOD Explorer uses the technologies of JSONP calls to the constructed endpoints fetching JSON formatted data to be parsed by JavaScript and presents the LOD resources in an HTML5 web page. While the resource is fetched from the Dataset, all the provided functions such as searching within the resource and exploring all the details of the resource are possible even if the device is disconnected from the Internet. This is because the system has been developed using only client side technologies without the need of any server side modules.

The resources are presented as graph nodes while their properties as textual information with the aim of mixing the best of both worlds. Hence, this way, the significance of using SPARQL endpoints can be proved and promoted using triplestores to develop federated queries [5].

LOD Explorer processes RDF data in advance and organizes them for presentation. The system presents all existing materials in RDF datasets without hiding any of its portions. For instance, property types are used to group In/Out properties.

The exploration process can be initiated by querying the endpoint for a particular resource either by using a resource name or a resource URI. A couple of resource examples where one can start from are provided as well. Afterward, exploring the resource is easy as can be through an attractive information presentation and following the related incoming and outgoing connections. New resources can be added to the graph and each of the newly opened resources will automatically connect to the ones already opened, if and only if there is a semantic connection between them.

The system has been constructed using the following technologies:

- Pure JavaScript
- jQuery libraries
- jsPlumb toolkit to draw nodes of graph
- an HTML5 page

The working schema of the system is illustrated in Fig. 1.

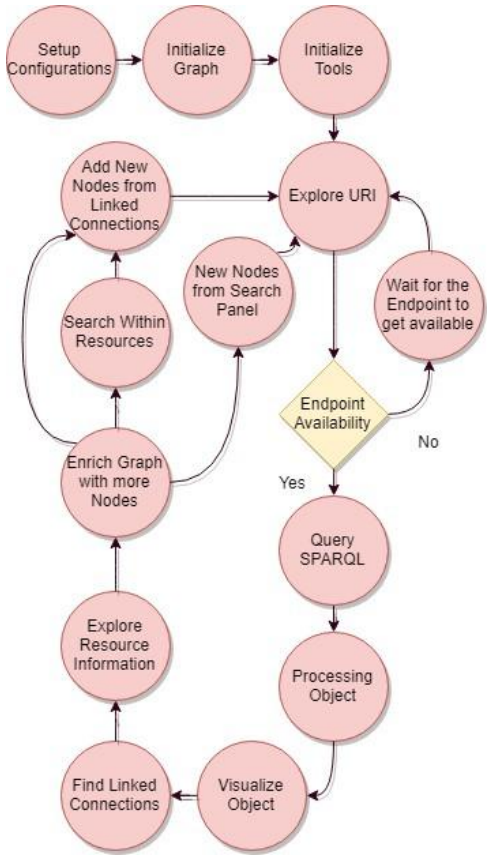

Fig. 1. Working Schema

II. SYSTEM MODULES

The system consists of several modules. The most important modules will be explained in this section.

A. *Configuration module*

The system starts by using this module to configure and initialize all the required variables needed to fire up the system. The system is developed in a modular fashion as highlighted by [6], [7], therefore, the variables in this module can easily be changed to fulfil any further developments. The configuration module includes the setup of the triplestores to access and retrieve the data from. Currently, the system uses only DBpedia dataset to search and retrieve the RDF data, using the DBpedia Lookup Service API . However, the module is designed in a way that can allow further datasets APIs to be included and fetch the data from.

After initializing all the required variables, the module begins to initialize the services to be added to the application. One of the services is to initialize the SPARQL query service so as to handle the resources when they are queried from the SPARQL endpoint.

B. *Graph module*

This module is responsible mainly for the graph layout of the system. The module starts by initializing all the required variables for the layout such as nodes, node types and zoom ratio of the view, and then initializes the visualization engine by importing defaults into the jsPlumb instance. It creates a main graph unit for the whole system so to draw and display all the nodes of the graph.

Adding resources as nodes to the graph is the next mission of the module with the help of Graph Node and Linker Modules. This can be done by taking a resource ID, which is the resource URI itself, and finds the label of the resource, and adding to the graph as a new node. The process of adding a new resource is shown in Fig. 2. Since this module mainly takes care of the layout of the graph, adding new resources is performed by taking care of the available nodes on the graph as well. For instance, when opening a resource Details Box (or Details Panel) as shown in Fig. 5, and a resource is added from that Details Box of the opened resource, the newly added resource will be added around the corresponding resource. This is implemented by taking and keeping the position of each of the opened node. This way a better Human Computer Interaction (HCI) usability is achieved for the user to draw the nodes around the opened node.

A drawn node can be removed from the graph; this is performed in several stages. First, the system pushes this node, using its URI, into a stack called Node List. This list is used by the Undo function to redraw the removed node by the user. Second, the node is removed from the search container so that it is not included in any of the future search within resources method. Third, the node Detail Box is removed from the background when it is open. Fourth, all the endpoints/edges connected from/to this node are removed by jsPlumb instance. Finally, the graph node is detached from the graph using the jQuery detach method.

Deleting all nodes function is also possible from the system, which passes through the same steps of deleting a single node

except it clears all of the nodes, and all of them are redrawn at once when the Undo action is executed.

The Undo function, as it is clear from its name, is used to go back to the previous state. The function is triggered when a node is inserted into or removed from the graph. It works like a stack by going back step by step from the last performed actions to the first performed action. The method also goes through several steps, similar to deleting a node method, performing this action. When a node is inserted into the graph, it pushes to a list called Undo Action List. This list is used by this method to go to the previous state of adding that node. Similarly, when a node is removed, it is removed from this list as well with the same reason of node insertion. Undoing an inserted node, which means deleting this node, will do all the steps of deleting a node. However, Undoing a deleted node, will do a reversed step for deleting a node. Undoing a deleted node means inserting that node into the graph again, which means all the relations and properties of this node have to be available all over again for the system. This is more likely to be the case than as redrawing the node from the beginning, except remembering the position of the removed node to be redrawn at the same position of its deletion.

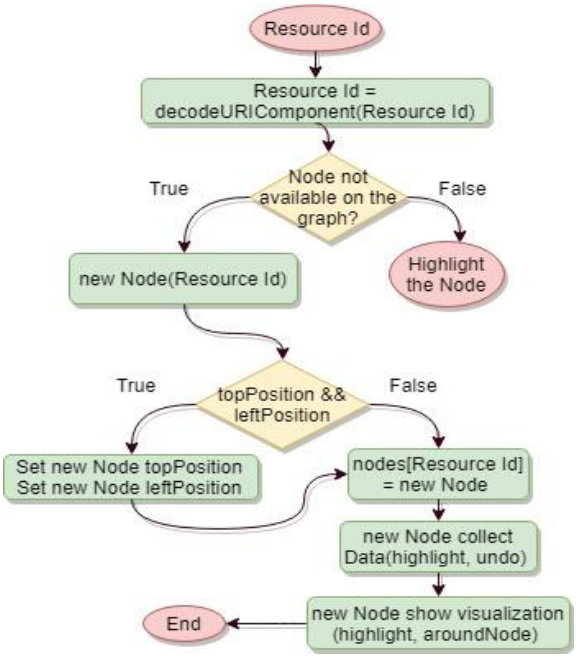

Fig. 2. Adding node to the graph

Highlighting the nodes while being clicked is attained from this module as well. When the Details Box for any of the resources is opened, that resource is highlighted with a different color with the purpose of HCI features as emphasized by [8]. Besides, this feature is used by the Explore method so as to get a configured number of new resources semantically related to the highlighted resource and add them to the graph.

C. Buttons module

This module is designed to aware of all the buttons, toolbar and navigation menu of the system. It starts by initializing a couple of variables needed for the system and then initializes the search form of the system.

The search for resources is the most important section of this module which starts by selecting the search provider of the system. Currently, only the DBpedia Lookup service is set for the system as the search provider, but as mentioned above, the system is designed to accept further search providers by modifying the configuration module. An auto complete service is provided for sending requests to the SPARQL endpoint and receiving the responses for the searched term using AJAX technologies. This will help users to get results as soon as they start writing their queries. Resource labels are shown for the search results and they are pushed into a result container. When selecting a resource, the URI of the resource is returned from the endpoint and is sent to Graph Node module to add the resource and complete the process. Furthermore, a local database for the selected resource is prepared so as to assemble the related data of the selected resource for the search within the resource process.

Searching inside the resources is the next important unit of the module. When a resource is selected from the search section, this unit is activated so as to find the resources that are related to the selected resource. The method provides an autocomplete facility while searching for related data of the available resources and lists the results in a dynamic list. The related data in the dynamic list is displayed as subject, predicate and object considering the principles of LD, where the subject is the resource containing the searched data, the predicate is the property of that resource, and the object is where the searched data is found (the target). When the targeted resource is selected from the list, the details box for the resource containing the related data gets opened and the target resource is perceptible. The related data of resources are stored in a local storage, and whenever a resource is added or removed from the graph, this storage is refreshed so as to provide related data for all available resources on the graph.

One of the interesting methods in this module is the Explore utility. It is developed in order to assist users in exploring more semantic relationships to a specific resource. This method works as soon as there are resources activated on the graph. It takes a number of LD of the selected resource randomly to be drawn on the graph. The number of selected LD can be modified from the configuration module. Fig. 3, illustrates the dataflow of the explore utility.

Simple information of any resource can be displayed quickly in the Resource Information Box, which displays resource label, DBpedia URL of the resource, Wikipedia URL of the resource, and the abstract of the resource. The other facility that is provided by this module is the Zoom utility, where the drawn graph nodes can be zoomed in and out.

The search menu is designed with the HCI philosophies in mind, where the accordion effects are applied to the forms in order to give users a wider area for exploration to users.

D. Application module

Once the variables needed to drive the system are prepared, the application module takes the responsibility to initialize the methods of jsPlumb, graph module, button module, profile module, and other processes needed for the system. The jsPlumb methods include getInstance, which instantiates independent

instances of jsPlumb by taking an object which provides the defaults from the configuration module, registerConnectionTypes to set a collection of attributes such as paint style and hover paint style. This is a subset of the parameters one can set in an Endpoint or Connection definition, and registerEndpointTypes which registers all of the given Endpoint types on this instance of jsPlumb. Afterwards, the module initializes the FancyBox tool, which is used to display images of resources in a "lightbox" that floats overtop of web page.

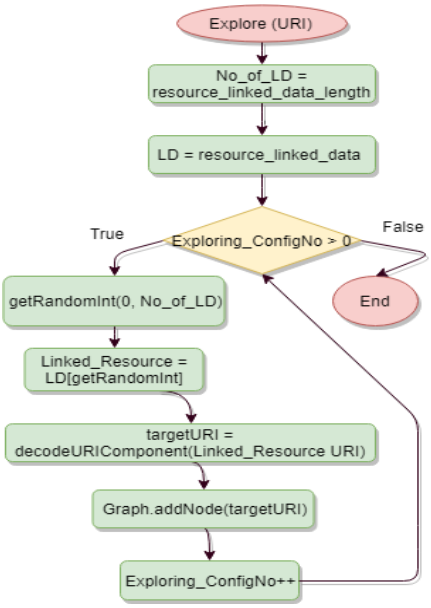

Fig. 3. Explore method

E. Graph node module

The module contains the most important parts of the system, where the nodes of the graph are spawned. The module begins by initializing a set of variables needed for the graph node such as its LD, presenting basic info of the node, images, and etc., the newly added node is checked with any of the available nodes of the graph to check whether there are relations between them or not. If there are linked relations among any of the available nodes, then the newly added node has to be linked with them. This process is done with the help of the relationships class of the system which takes the URI of the target, connection label, direction of the connection, and the endpoint label.

Collecting the contents of the resource is the next mission of the module. The LD of the resource is collected so as to be presented in the details box. The method is divided into three main parts: Literals, In connections, and Out connections. For each linked relation of the resource, the URI, the type of direction and the literal the linked resource converted to string are obtained and pushed into an array of elements so as to be used for the details box.

Each drawn node has an image representing the resource. This image is obtained using either the depiction property of FOAF or the thumbnail property of DBpedia. Afterwards, the image is sent to be stored in an images array in order to be used by Fancybox tool to show the resource image.

Since there is no server side scripts in the proposed approach, and we have no control over the server files we are querying to, hence using the technology of JSONP is a must. To collect the data of a resource, the module, with the help of Server-Linker class, starts the process of making queries and get the LD. It starts by initializing the API service for the dataset server to be queried from, which is the DBpedia dataset in our case. However, the method is flexible to accept other dataset APIs since it extracts the prefix of the dataset from the resource URI. Later, the SPARQL query is initialized to set the format and parameters of the query such as the service endpoint and query output as JSON. Then the JSONP method is used to set the URL of the service including resource URI, set the callback parameter and request data using the script tag to overcome the problems caused due to cross-domain policy. When the method successfully requests data, the return of the request is the JSON formatted data.

So far, the LD to the queried resource is available at the client side as JSON formatted object, hence, binding the results of the object is the next step to be performed. The method starts by taking the JSON object which starts to analyze it. If the JSON object is returned with no problems and the service of the endpoint is not empty, the time taken to process the resource is set to start. There are two main directions for the LD of the resource, which are In and Out directions. Thus, a list for each of them is created so as collect the related data for both directions. For each item in the object result a check for a self-connection is taken so as to prevent possible loops. The method then checks whether the item is Out or In. When the connection is Out, the image of the resource is stored and the item type is tested for whether it is a URI, Literal, or an unknown type. When the type is a literal, the item is added to the literals list of the resource by taking the language version of the literal.

Once the type is a URI, then it is decoded and examined with the URI of the current resource to flag any self-targeted connections, which creates a loop. Afterwards, the node type, endpoint label, and node URI are stored for the node. For instance, from the Out connections in the detail box of Iraq resource, Language as type, Kurdish language and its URI as endpoint label and URI can be found.

When the type is not a URI nor a literal, then the type is unknown yet. A similar process for the In direction is achieved for the item.

Up to now, the ontologies and properties of the resource is collected for both In and Out connections, thus, parsing them is the next turn so as to add connections for each property type, property URI and endpoint label of each resource.

Finally, the post parsing begins to set labels for both English and no language labels. After that, the search database gets refreshed to include the newly LD of the current resource so that they are available for the search within the resources method to find related data for the whole graph. Consequently, the whole drawn graph gets refreshed and the newly added resource gets inserted into a list of nodes. Finally, the undo action is logged to insert the node list into the undo action list.

F. Linker module

As it is clear from its name, the module's main duty is to form an environment for system files to link and connect with other tools used for the system, such as jsPlump toolkit and fancybox tool. Since the module communicates with the visualization tools, methods needed for visualizing nodes are employed in this module as well. Linker module consists of a wide range of methods, however, the most important methods will be explained in this section. When the data of a resource has been processed by Graph Node module, it is then required to be drawn on the ground. To do so, required information need to be sent to this module so that it is processed and sent to other tools to draw the resource as a graph node.

When the graph module initializes the graph, it uses this module to initialize the visualization engine by connecting to jsPlumb tool and run the jsPlumb Instance to import all the given defaults into this instance of jsPlumb.

As soon as the system is ready and the graph is initialized, resources can be added to the graph. If a resource has newly been added to the graph, a random position from an available empty ground area is taken to draw the node on. When a node has recently been deleted from the graph and needs to be added again, for instance from the Undo action, then the node is re-drawn at the same place where it was deleted since the top and left positions of the node are stored in the window variable for this purpose. If a node is added from the details box, the position of the newly added node will be close to the opened node with the aim of drawing nodes next to nodes having relationships with each other. These facilities are achieved by specific methods developed specifically for these purposes.

A method to show the node then runs to send the required data for the jsPlumbInstance object to draw the node. The draggable method of the object is used for the node so that each drawn node can easily be moved on the ground. To turn the entire elements into connection sources and targets, makeSource and makeTarget methods of the jsPlumb tool have been used. With these methods elements are marked as a source or target, and they provide an endpoint specification for the tool to use when a connection is established.

Images of the nodes are also set in this method be checking the resource data whether it has an image or not. In case the resource already have the image, which takes from the resource depiction or thumbnail properties, then the details of this image will be sent to Fancybox tool in order to set the image and make it visible within the node. In case the resource property does not provide any image details, then default images (sprites) for the node are employed to show icons for the node representing its type, for example for person nodes a person icon will be visible.

When more than a node is added to the graph, the probability of having relations with the available nodes has to be performed to connect the linked nodes. This module is responsible for this task as well, by taking the URI of the newly added resource and comparing it with each of the URIs of the available nodes on the graph. The getConnections method of the jsPlumb Instance is used to get all of the connections currently managed by the jsPlumb instance by taking the URI of the newly added resource, URI of the target resource, which is from the available nodes on the graph, and the connection label, which is the type of the connection between both nodes. Afterwards, jsPlumb Connect method establishes the link connection between the two nodes if and only if a connection with the same label and direction is not found.

During this process, the time taken from the first step of the creation of the node till the final step of the visualization is calculated and presented. Also, the time taken to arrange the details box such as ordering them into tabs of description, In connections, and Out connections is calculated and presented to the console window.

III. IMPLEMENTATION AND RESULTS

In this section, a brief introduction of the interface and the interaction model of the web application on the proposed framework: LOD Explorer, is the presented. Later on, real tests and performance measurements of the system is proposed, furthermore, the scalability of the system is verified. The exploratory search systems are sometimes referred to as Human-Computer Information Retrieval systems (HCIR) by the scientific community [9]. Hence, the position of the interactions for successful explorations is emphasized from this denomination. The interfaces have to be optimized to favor the users' engagement and to support them continuously and intensively from a cognitive point of view.

The user interface of the application consists of the following parts as shown in Fig. 5:

1) *Search panel (center)*
2) *Toolbox (top right)*
3) *Statistics Box (top right)*
4) *Description box (right)*
5) *Details Panel (left), and*
6) *the ground*

Statistics box, as shown in Fig. 4. The statistics that are presented in this box are as follows:

1. The total number of triples (piece of information) that have been processed by the system so far;
2. Information about the last inserted resource into the graph, which are detailed below:
   - The number of triples the resource owns
   - The time needed to visualize the node
   - The required time needed to processes the resource
   - The time spent to download the resource triples
   - The total time taken to insert the node into the graph including the downloading, processing, and visualizations.

A complete description of the application interface is described in [3].

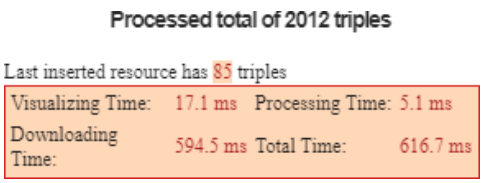

Fig. 4. Statistics Box

A. *Real tests and Validation*

Testing is the process of assessing a system or its module(s) with the aim of finding whether it fulfils the identified requirements or not. In other words, testing is executing a system so as to find any gaps, errors, or missing requirements in contrast with the actual requirements [10]. In this section, real tests for the proposed system are performed by taking several experiments. Since the system is fully implemented depending on the client side technologies, the system performance depends on the machine a user is using for the explorations. Hence, the test cases have been performed on different computing platforms. The platforms taken for the tests are shown in Table I:

1) *Test case 1*

In this case, the focus will be on athletics and sport clubs. Real Madrid C.F, Cristiano Ronaldo, Lionel Messi, La Liga 2010-2011, and La Liga 2011-2012 Football Leage Seasons are taken as resources. The graph for the above resources can be seen from Fig. 6. The number of properties and In and Out connections for each resource is shown in Table II:

TABLE I. RESOURCES WITH THEIR NO. OF PROPERTIES AND CONNECTIONS

Platform Code	Model	Processor	RAM	Hard disk	OS	Browser
P1	Dell Precision Mobile Workstation M6700	Intel Core i7-3720QM CPU @ 2.60GHz, 4 Cores, 8 Logical Processors	24 GB DDR3	256 SSD	Windows 10 Pro, 64-bit	Google Chrome V63, 64-bit
P2	MacBook Pro	2.2 GHz, Intel Core i7, 4 cores	16 GB DDR3	256 SSD	MacOS High Sierra V. 10.13.2	Google Chrome V63, 64-bit
P3	Dell Precision Mobile Workstation M6700	Intel Core i7-3720QM CPU @ 2.60GHz, 4 Cores, 8 Logical Processors	24 GB DDR3	256 HDD	Linux, Ubuntu 14.04 LTS, 64-bit	Google Chrome V64, 64-bit
P4	Lenovo Mixx 2 11 Tablet	Intel Core i5-4202 CPU @ 1.6 GHz, 2 Cores	8 GB DDR3	256 SSD	Windows 8.1 64-bit	Google Chrome V63, 64-bit

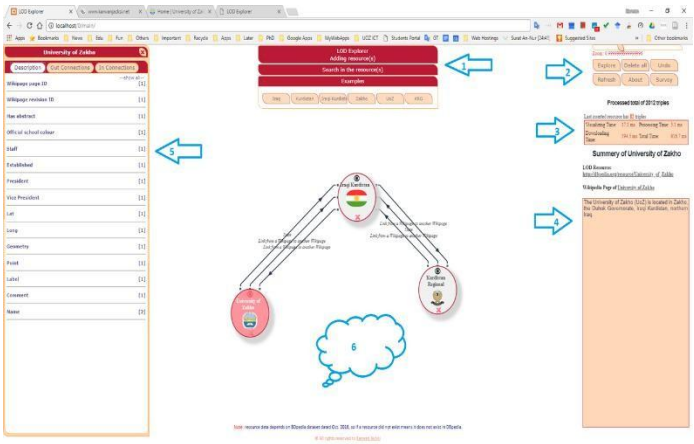

Fig. 5. User interface

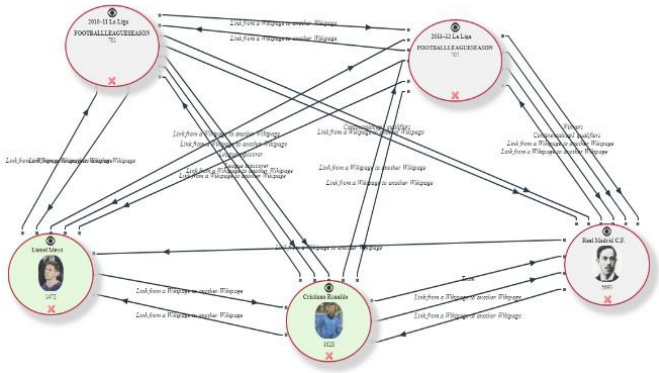

Fig. 6. Test case 1 resources

TABLE II. RESOURCES WITH THEIR NO. OF PROPERTIES AND CONNECTIONS

Resource Name	No. of Properties	No. of Connections (In/Out)
Real Madrid C.F	88	5693
Cristiano Ronaldo	61	1628
Lionel Messi	67	1475
La Liga 2010-2011	197	701

From the above graph, some interesting facts are observed. All of the resources have connections with each other with different connection types. For instance, Cristiano Ronaldo has a couple of connections with Real Madric C.F as one of the connections clarifies that the resource is the team he is playing for; he has two connections with Lionel Messi and La Liga season 2011-2012 from wikiPageWikiLink property; he has connections to La Liga 2010-2011 as seasons league top scorer. Real Madrid C.F is the winner of Spanish league season 2011-2012 but Lionel Messi is the league top scorer for the same league.

The time needed for the resources in the graph to be drawn on the ground are exposed in Table III and Table IV.

2) *Test case 2*

In this experiment, a graph with a larger number of nodes was the target. Fifty resources with a high number of

connections from different types have been added to the graph. The aim was to validate the scalability of system. The graph of this experiment is shown in Fig. 7.

B. Result discussions

The results of the above experiments are discussed in this section. Starting from test case 1, the nodes of the graph are tested and measured on different platforms, as shown in Table III and Table IV, using both localhost copy and the uploaded one to the online host.

Results of test case 1, where the graph nodes have more connections with each other, are presented in Table III and Table IV.

The results clarify that the visualization process does not depend on the amount of information and the number of connections a node has with other nodes of the graph, but instead, the processing time effects with this situation. Accordingly, the hardware and software specifications of the machine building the graph has a direct influence on the time of the graph creation. P4 has worst performance in all experiments for both online and local host cases due to the worse specifications the machine has among other machines.

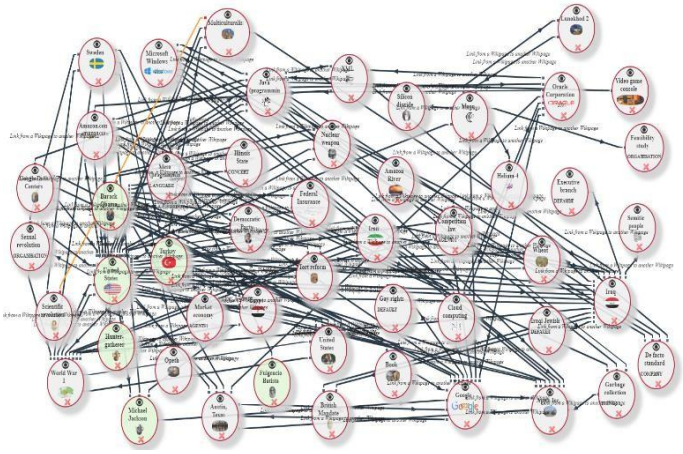

Fig. 7. A graph with 50 nodes

TABLE III. TEST CASE 1 RESULTS FROM LOCALHOST

Test No.	Resource name	No. of Properties	In & Out connections	Overall Time /milliseconds	Downloading Time /milliseconds	Processing Time /milliseconds	Visualizing Time /milliseconds	Platform Case
1	Real Madrid C.F	88	5693	3753.979	3626.060	116.376	11.543	P1
2				1580.713	1501.610	69.829	9.274	P2
3				2882.149	2567.175	282.941	32.033	P3
4				3429.790	2914.799	445.551	69.440	P4
5	Cristiano Ronaldo	61	1628	1437.209	1399.479	30.403	7.327	P1
6				696.206	671.925	18.553	5.728	P2
7				959.916	859.273	80.763	19.880	P3
8				2013.218	1811.451	171.731	30.036	P4
9	Lionel Messi	67	1475	1351.047	1313.228	29.354	8.465	P1
10				662.896	639.698	16.976	6.222	P2
11				873.006	802.203	54.347	16.456	P3
12				2296.171	2128.082	133.825	34.264	P4
13	La Liga 2010-2011	197	701	626.532	603.054	15.439	8.039	P1
14				1774.441	1758.331	7.523	8.587	P2
15				855.996	800.100	34.365	21.531	P3
16				1687.813	1580.636	64.949	42.228	P4
17	La Liga 2011-2012	191	707	709.336	686.588	15.638	7.110	P1
18				513.181	499.541	6.487	7.153	P2
19				1092.932	1037.034	32.727	23.171	P3
20				3264.312	3161.872	59.353	43.087	P4

TABLE IV. TEST CASE 1 RESULTS FROM ONLINE HOST

Test No.	Resource name	No. of Properties	In & Out connections	Overall Time /milliseconds	Downloading Time /milliseconds	Processing Time /milliseconds	Visualizing Time /milliseconds	Platform Case
1	Real Madrid C.F	88	5693	1767.134	1647.437	108.023	11.674	P1
2				4624.369	4542.649	70.790	10.930	P2
3				2081.305	1757.858	294.090	29.357	P3
4				2644.071	2202.772	384.248	57.051	P4
5	Cristiano Ronaldo	61	1628	831.020	786.494	37.234	7.292	P1
6				1428.337	1394.938	26.676	6.723	P2
7				1022.099	915.066	87.757	19.276	P3
8				1347.060	1190.267	131.338	25.456	P4
9	Lionel Messi	67	1475	804.165	768.413	29.556	6.196	P1
10				1200.215	1170.772	23.312	6.131	P2
11				917.607	836.276	65.677	15.654	P3
12				1383.593	1258.390	101.792	23.411	P4
13	La Liga 2010-2011	197	701	532.906	508.222	16.246	8.438	P1
14				1205.779	1186.719	11.093	7.967	P2
15				1480.762	1435.747	27.430	17.585	P3
16				1060.046	976.106	49.884	34.055	P4
17	La Liga 2011-2012	191	707	1116.425	1092.491	15.909	8.025	P1
18				1127.999	1110.316	9.161	8.522	P2
19				1029.644	982.155	25.857	21.632	P3
20				1435.325	1358.168	50.741	26.416	P4

For test case 2, where the aim was to test the system for the scalability, fifty nodes were inserted into the graph, as displayed in Fig. 7. These nodes were chosen to have as many connections as possible. As a result, 427,391 pieces of information were downloaded to the graph and were processed by the system to visualize the nodes and form the graph. The experiment in this case was tested using only P1.

The outcomes in Table V show that, for all the 50 nodes, the overall time spent to form the graph is around 126 seconds (2.52 seconds per node as average). However, most of the time was spent for downloading data from the dataset, which was about

115 seconds (2.3 seconds per node as average). The total time needed to process the graph which consisted of 427,391 piece of data was only 8.316 seconds (0.16632 seconds per node), and the time required to visualize all the graph nodes was only 3.177 seconds (0.06354 seconds per node) which are amazing results.

C. Validation

A manual testing is performed for the above experiments in order to investigate the effective performance of the system. The investigation is conducted so as to meet the goals of Software Development Life Cycle (SDLC). The system operated smoothly and met the requirements that conducted its design and development. The system appropriately responded to all kinds of queries, and its functions achieved in an outstanding time and free of bugs.

TABLE V. FIFTY NODE GRAPH STATISTICS

Node No.	Overall Time	Downloading Time	Processing Time	Visualizing Time	Node No.	Overall Time	Downloading Time	Processing Time	Visualizing Time
1	3715.52	3212.69	467.33	35.51	**26**	2433.98	2330.67	74.82	28.50
2	3970.68	3539.39	412.38	18.92	**27**	6300.63	5827.37	437.79	35.48
3	4729.00	4367.85	340.38	20.77	**28**	1201.66	1037.25	119.85	44.55
4	3158.73	2779.75	361.94	17.04	**29**	2803.04	2561.34	197.06	44.64
5	6958.82	6548.30	393.07	17.46	**30**	1110.53	1009.79	4.32	96.42
6	14077.16	13617.83	443.12	16.21	**31**	727.58	566.97	19.88	140.73
7	5508.63	5082.13	405.34	21.17	**32**	706.48	660.95	7.82	37.71
8	1893.47	1543.60	325.36	24.52	**33**	587.19	508.77	37.02	41.40
9	1608.53	1462.60	129.93	16.00	**34**	561.24	468.27	40.38	52.58
10	7610.72	7220.10	371.50	19.11	**35**	1391.80	917.79	387.93	86.08
11	2316.30	2037.29	257.31	21.70	**36**	567.95	390.62	13.45	163.89
12	2302.13	2078.09	206.78	17.26	**37**	928.57	798.08	100.01	30.48
13	734.93	651.36	62.23	21.34	**38**	814.92	652.23	5.82	156.87
14	1121.49	953.34	151.87	16.28	**39**	565.19	395.85	23.23	146.11
15	5190.13	4817.52	354.05	18.56	**40**	1361.69	1279.46	41.03	41.20
16	3110.86	2657.91	431.83	21.12	**41**	789.30	713.84	16.79	58.67
17	4812.16	4405.92	387.30	18.93	**42**	980.12	762.43	16.75	200.94
18	802.42	718.47	62.06	21.89	**43**	835.27	718.69	39.94	76.65
19	1709.67	1527.80	164.30	17.57	**44**	1319.71	1212.03	42.23	65.45
20	2285.41	2034.74	227.89	22.77	**45**	1081.70	887.32	22.44	171.94
21	1365.36	1238.99	103.40	22.98	**46**	990.27	704.04	54.16	232.08
22	473.75	430.46	11.85	31.45	**47**	790.90	720.29	12.61	58.00
23	293.25	268.17	3.20	21.88	**48**	680.26	448.63	11.42	220.22
24	5493.88	5136.42	340.22	17.23	**49**	3791.37	3549.73	18.85	222.79
25	5949.93	5785.98	124.04	39.92	**50**	1547.90	1329.52	31.98	186.41

TABLE VI. TABLE 5 IN TOTAL

No. of Nodes	Overall Time	Downloading Time	Processing Time	Visualizing Time
50	126,062.141	114,568.588	8,316.220	3,177.332

IV. CONCLUSION AND FUTURE WORK

The effortless process of uploading data online creates a giant volume of data. This causes the information retrieval systems to face new challenges in order to return relevant data. The evolution of the syntactic Web to the Semantic Web technology where the information can be understood by machines has raised high expectations. This technology increases the efficiency of searching approaches through presenting the automated data processing.

The amount of published data consistent with the standards of LD is growing dramatically. However, consumption is still limited for professionals who understand the technologies of LD. Thus, a tool for intuitive presentation of LD is crucial. LOD Explorer, an interactive and easy-to-use tool for exploring RDF resources, is presented. The application is made using pure JavaScript and jQuery libraries without the need to a server side software. Application performance and scalability has been measured using several different platforms, and the results have been excellent.

The future plan for the system is to enrich it with several further functions such as adding more RDF datasets, giving users the opportunity to select a desired shape for the nodes, and adding path finding feature so as to find the exact relation between two or more resources. Furthermore, adding advanced features to include semantic similarity for words and suggest semantically similar words to a query as in [11].

An interesting future work would be by building a custom ontology for a specific task such as e-learning and other ontologies as in [12], [13] and apply the LOD Explorer to retrieve triples and present the retrievals.

REFERENCES

[1] K. Jacksi and S. M. Abass, "Development History of the World Wide Web," *Int. J. Sci. Technol. Res.*, vol. 8, pp. 75–79, 2019.

[2] K. Jacksi, N. Dimililer, and S. R. M. Zeebaree, "A Survey of Exploratory Search Systems Based on LOD Resources," in *PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON COMPUTING & INFORMATICS*, COLL ARTS & SCI, INFOR TECHNOL BLDG, SINTOK, KEDAH 06010, MALAYSIA, 2015, pp. 501–509.

[3] K. Jacksi, S. R. M. Zeebaree, and N. Dimililer, "LOD Explorer: Presenting the Web of Data," *Int. J. Adv. Comput. Sci. Appl. IJACSA*, vol. 9, no. 1, 2018, doi: 10.14569/IJACSA.2018.090107.

[4] K. Jacksi, N. Dimililer, and S. R. Zeebaree, "State of the Art Exploration Systems for Linked Data: A Review," *Int. J. Adv. Comput. Sci. Appl. IJACSA*, vol. 7, no. 11, pp. 155–164, 2016, doi: dx.doi.org/10.14569/IJACSA.2016.071120.

[5] K. Jacksi, "Design and Implementation of E-Campus Ontology with a Hybrid Software Engineering Methodology."

[6] K. Jacksi, F. Ibrahim, and S. Ali, "Student Attendance Management System," *Sch. J. Eng. Technol. SJET*, vol. 6, no. 2, pp. 49–53, 2018.

[7] K. Jacksi, "Design And Implementation Of Online Submission And Peer Review System: A Case Study Of E-Journal Of University Of Zakho," *Int. J. Sci. Technol. Res.*, vol. 4, no. 8, pp. 83–85, 2015.

[8] S. R. M. Z. Adel AL-Zebari Karwan Jacksi and Ali Selamat, "ELMS–DPU Ontology Visualization with Protégé VOWL and Web VOWL," *J. Adv. Res. Dyn. Control Syst.*, vol. 11, no. 1, pp. 478–485, 2019.

[9] N. Marie, "Linked data based exploratory search," phdthesis, Université Nice Sophia Antipolis, 2014.

[10] M. Fagan, "Design and code inspections to reduce errors in program development," in *Software pioneers*, Springer, 2002, pp. 575–607.

[11] R. Ibrahim, S. Zeebaree, and K. Jacksi, "Survey on Semantic Similarity Based on Document Clustering," *Adv. Sci. Technol. Eng. Syst. J.*, vol. 4, no. 5, pp. 115–122, 2019, doi: 10.25046/aj040515.

[12] S. R. Zeebaree, A.-Z. Adel, K. Jacksi, and A. Selamat, "Designing an ontology of E-learning system for duhok polytechnic university using protégé OWL tool," *J. Adv. Res. Dyn. Control Syst.*, vol. 11, no. 05-Special Issue, pp. 24–37, 2019.

[13] A.-Z. Adel, S. Zebari, and K. Jacksi, "Football Ontology Construction using Oriented Programming," *J. Appl. Sci. Technol. Trends*, vol. 1, no. 1, pp. 24–30, 2020.