Connecting the Dots: A Multi-pivot Approach to Data Exploration

Igor O. Popov, m.c. schraefel, Wendy Hall, and Nigel Shadbolt

School of Electronics and Computer Science,
University of Southampton,
SO17 1BJ, Southampton, UK
`{ip2g09,mc,wh,nrs}@ecs.soton.ac.uk`
`http://users.ecs.soton.ac.uk/{ip2g09,mc,wh,nrs}`

Abstract. The purpose of data browsers is to help users identify and query data effectively without being overwhelmed by large complex graphs of data. A proposed solution to identify and query data in graph-based datasets is *Pivoting* (or *set-oriented browsing*), a many-to-many graph browsing technique that allows users to navigate the graph by starting from a set of instances followed by navigation through common links. Relying solely on navigation, however, makes it difficult for users to find paths or even see if the element of interest is in the graph when the points of interest may be many vertices apart. Further challenges include finding paths which require combinations of forward and backward links in order to make the necessary connections which further adds to the complexity of pivoting. In order to mitigate the effects of these problems and enhance the strengths of pivoting we present a *multi-pivot* approach which we embodied in tool called Visor. Visor allows users to explore from multiple points in the graph, helping users connect key points of interest in the graph on the conceptual level, visually occluding the remainder parts of the graph, thus helping create a road-map for navigation. We carried out an user study to demonstrate the viability of our approach.

Keywords: Data browsing, graph-data, pivoting, interaction.

1 Introduction

Challenges in browsing large graphs of data are rich: large numbers of ontology concepts, high entropy and diversity in links between individual data instances, often makes it hard for users to understand both the overall content of a dataset, as well as understand and find the particular bits of the data that might be of interest. Such problems can often overshadow the benefits of interacting over large highly inter-connected data. The goal of data browsers has been, in part, to tackle the problem of making sense of such rich and complex domains. A common technique that has been adopted by a number of data browsers for exploring large graphs of data is *pivoting* (otherwise known as *set-oriented browsing*). Pivoting leverages the rich semantic descriptions within the data to extend the commonly used *one-to-one* browsing paradigm to a *many-to-many* navigation for data.

In this paper we focus on several commonly observed design patterns found in pivot-based data browsers: (1) exploration is often restricted to starting from a single point in the data, (2) navigation is typically supported in a single direction, and (3) immediate instance level exploration is regularly preferred without gaining familiarity with the domain or setting the exploration context first. We argue that these characteristics impose a number of limitations: in the case of (1) they reduce flexibility and therefore the ability to quickly find data that are related to the initial set multiple hops away, in the case of (2) they reduce the expressivity of the browser, and in the case of (3) the absence of an overview of the domain to be explored can often lead to difficulties in retracing exploration steps as well as make potential alternative exploration paths difficult to recognise. In this paper we introduce a novel approach we call *multi-pivot* which extends the traditional pivoting techniques to mitigate the aforementioned limitations. We approach we designed a demonstrator tool named *Visor* and carried out a user study to test the viability of our approach.

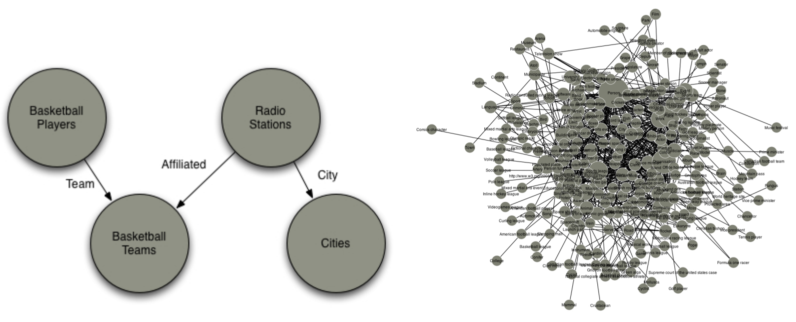
The outline of the paper is as follows. In the following section we examine related work in the area of pivot-based data browsers, and discuss these limitations in more detail. In Section 3 we discuss our approach, and lay out key design requirements in our approach. Section 4 describes Visor, a tool which we developed to test the multi-pivot technique. In Section 5 we carry out an evaluation study to test the viability of our approach and discusses the implications for design. The paper concludes with a summary of our work and planned future work.

2 Related Work

Pivoting, as an interaction method, has been adopted by a number of data browsers. Tabulator [3] can be considered an early pioneer of pivoting. Users browse data by starting from a single resource following links to other resources. Tabulator allows users to select patterns by selecting fields in the explored context and tabulate any results that are following the same pattern. Explorator [2] uses pivoting as a metaphor for querying, where users select subjects, objects and predicates to create sets of things, subsequently combining them with unions and intersections operations. The Humboldt browser [12] provides a list of instances and faceted filters from which the user can either choose to pivot or refocus. Parallax [10] shows the set of instances, accompanied by a list of facets for filtering and a list of connections showing the available properties that can be used in a pivoting operation. In VisiNav [7] users can drag and drop properties and instances in order to pivot and filter through results. A common characteristic of these interfaces is the notion that pivoting never occurs in branching i.e. a user cannot pivot with two different properties from the current focus and keep the context of both trails of exploration. Parallax, however, supports branching to some extent in the tabular view where generating a table allows this feature. gFacet [8] also mitigates the problem of branching; exploration starts from a set of instances and multiple properties can be selected to surface related sets of instances. The lists of instances, generated through successive pivoting operations, are used as facets and spatially arranged in a graph visualisation. Finding relationships between remote it typically unsupported by these browsers. RelFinder [9] allows finding relationships on the instance level, but not on the conceptual level. Fusion [1] offers discovery of relationships however the framework is designed more for programmers rather than non-technical end users.

2.1 Limitations of Pivot-based Browsers

In the following we expand our discussion of the aforementioned limitations. To better illustrate these, we consider the following example: a user exploring the DBpedia [4] dataset to find basketball players, their affiliation to radio stations, the radio frequency of these stations, and the cities from which they operate. Figure 1a depicts the subset of the domain that is needed to answer the given query. Figure 1b, on the other hand, depicts the entire domain of the dataset[1] from which the subset domain needs to be surfaced in order to answer the given query in our scenario. We highlight the specific problems through describing a hypothetical exploration process provided by a typical pivot-based data browser.



(a) Subdomain of Interest (b) Entire Domain

Fig. 1. (a) A portion of the DBPedia ontology showing how instances of the classes Basketball Players, Basketball Teams, Radio Stations and Cites are connected. (b) A graph visualisation of the DBPedia ontology showing which concepts are related. Each node represents a class and each arc means that there is at least one property connecting instances between two classes.

Exploration starts for a single point. In a standard pivot-based browser, exploration of a dataset begins with a particular set of instances. The initial set of instances usually pertains to instances from a certain class that is typically found through a keyword search. In our example, we can start our exploration

[1] `http://wiki.dbpedia.org/Ontology`

from either "Basketball players" or "Radio stations". Once the initial instances are shown, users are presented with a number of properties which can be selected to pivot and simultaneously get all the instances which are related to the instances in the first set through that property. In such a way navigation through the graph is facilitated. Let us suppose that "Basketball players" or "Radio stations" are commonly used keywords for our query, which would surface the corresponding two classes of instances. At this point we can choose to start with the instances of either class as our initial set. Suppose we choose, for example, "Basketball players". As can be seen from Figure 1a we need to perform two pivoting operations to get to "Radio stations". Since the two are not directly linked we need to do a little bit exploring to find out the ways "Basketball players" are related to "Radio stations". Unfortunately, no cues are given to guide us in which direction to start exploring so we can connect them with "Radio stations". In a situation where the domain is unfamiliar this presents a problem. Property labels, which are used to show to what is being navigated, do not hold any information about the path two or three arcs way of the current set of instances. The problem is further exacerbated when a high number of possible choices for pivoting is present and the number of choices increases exponentially if the relating instances of interest are multiple arcs way.

Navigation is uni-directional. The direction of pivoting in pivot-based browsers is often uni-directional i.e. navigation is enabled only from outgoing links from the instances in the current focus. The restriction can sometimes limit the query expressivity of the interface. In our example (refer to Figure 1a) we notice that whatever set of instances we start from ("Basketball players" or "Radio Stations" alike) we cannot pivot in a single direction to all the sets of instances we need, since the direction of the links we require for pivoting in "Basketball teams" are all incoming links.

Exploration and domain overview absence. Current pivoting practices are predominantly instance-centric. As such overexposing instance data, filtering and repeated pivoting operations, can often times result in lack of overview about the sub-domain being explored as part of the exploration. The general lack of overview that can often lead to unseen relations in the data and therefore contribute to a lack of understanding about the domain being explored. Research from the HCI community suggest that when confronted with complex information spaces, information-seeking interfaces should follow the Visual Information Seeking Mantra [14]: overview first, zoom and filter, then details-on-demand. The complexity and size of large datasets suggests that using such a paradigm might suitable for data browsers.

3 Multi-pivot Approach

In the previous section we pointed out several challenges to standard pivoting as an interaction technique for exploring graph-based data. The aim of our research was to ascertain whether we can fashion an interaction model that mitigates these limitations and test if the solution we propose will introduce any major usability problems for end users. We integrated our ideas into a tool that followed four design requirements (R1 - R4):

R1. Exploration can be initiated by selecting multiple items of interest. Rather than being limited in starting from a single point, we wanted users to be able to start from multiple points of interest, and discover how the selected points of interest are connected with each other. As an analogy, we considered a puzzle solving example. When solving a puzzle, the solvers can start piecing the puzzle from multiple points: they can select several different pieces, find pieces that match, create several greater pieces and then piece these together to slowly gain an understanding of the overall picture. Similarly, we wanted users to grab different portions of the domain simultaneously, navigate either back or forth using either normal links or back-links, build their own subset of the domain related to their interest. Since there was no central point where the exploration starts and users would be able to pivot freely from anywhere in any direction we named this approach a *multi-pivot.*

R2. Overview first, instance data on demand. We didn't want to overburden users by immediately exposing instance data during exploration. Rather we wanted them to always have an overview of their exploration path and be able to quickly access the individual instance data if required.

R3. Allow navigation to be bi-directional. In addition to being able to start from multiple points, we wanted to support navigation in both directions. Additionally, we wanted to enable users to execute queries which paths include both forward and backward links.

R4. Creating custom spreadsheets as a way to query data. Once users have created and explored a sub-domain of the dataset they can query the sub-domain for instance data. Querying is done by assembling custom created spreadsheets from instance data by choosing among the concepts in the sub-domain and specifying the relationships between them. We choose spreadsheets as an representation, because its relative familiarity among non-technical data users. An additional motivator was that these spreadsheets can than be exported in a variety of formats which can be picked up and reused in different applications. For example, they can be published as an visualisation using ManyEyes [15] or published on the Web as a standalone dataset using an Exhibit [11].

4 User Interface

In order to test the approach outlined in Section 3 we developed a demonstrator tool called *Visor*[2]. Visor is a generic data explorer that can be configured on any SPARQL endpoint. For the purposes of testing and evaluating Visor we made an

[2] A demo version of Visor is available online at `http://visor.psi.enakting.org/`

initial deployment on the DBPedia[3] SPARQL endpoint. The following sections describe the user experience in Visor. Throughout the section we refer to specific areas where the description of the UI meets the design characteristics outlined in Section 3.

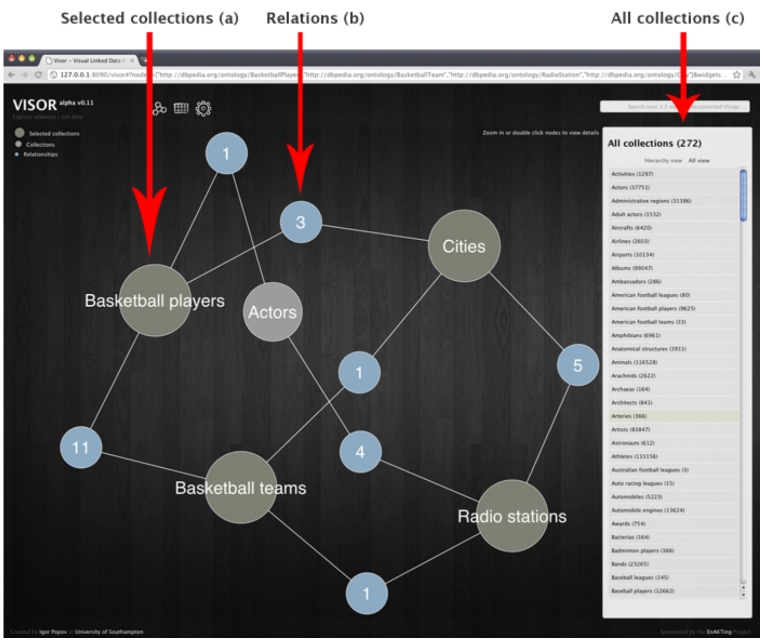


Fig. 2. Generating a subset of the DBPedia ontology generated by selecting concepts in the ontology in Visor. Selected concepts (e.g. "Basketball Players") are coloured in green, while suggested concepts are coloured in gray. Arcs between two collection with a blue node in the middle indicates links between items (e.g. "Basketball players" have a direct relationships with "Basketball teams".

4.1 Data and Ontology Exploration

In Visor, exploration starts by selecting ontological classes of interest (named *collections* in Visor). Users can choose from collections either by viewing an entire list of all the known collections or browse in the hierarchical view of the collections. The collections are listed in a panel on the right hand side of the user interface (Figure 2c). Alternatively a search bar is provided where the user can execute a keyword search to get results to both individual instances and collections in the data.

[3] `http://dbpedia.org`

Instead of choosing a single collection as a starting point for exploration, Visor allows users to select multiple collections simultaneously. The UI represents a canvas where a graph rendering of selected collections takes place. The graph rendering consists of the following nodes:

– **Selected collections**. Collections selected through the collections menu or searched are rendered with the title of the collection on top (Figure 2a).

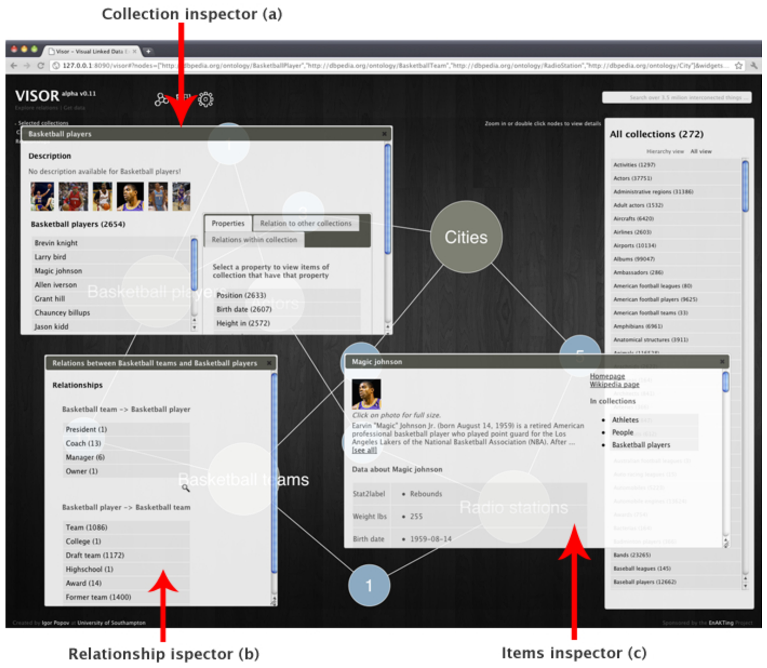


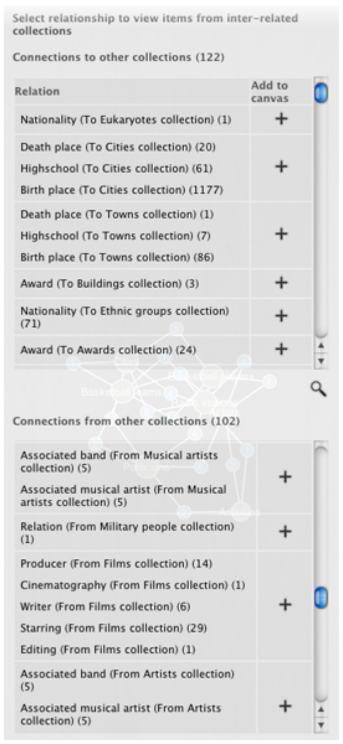
Fig. 3. Inspectors showing various information about the concepts in the ontology. Information about "Basketball Players" is shown in (a), and information about a particular basketball player (in this case Magic Johnson) is shown in (b). The various links that exist between "Basketball players" and "Basketball teams" is shown in (c).

– **Relations**. If there are properties linking the instances between two selected collections we indicate to the user that items from these collections are interrelated by displaying an arc with a blue node in the middle (Figure 2b). The number in middle of the blue node is a indicator of the total number of properties (named *relations* in Visor) that link instances between the two collections in either direction. We adopted this approach to mitigate generating a large and incomprehensible graph [13]. This is true in cases where datasets include classes with a large number of links between them.
– **Intermediary collections**. In some cases there are no properties linking two collections. In such a circumstance, Visor tries to find the shortest path

in the ontology by seeking an intermediate collection to which both selected collections can be linked from. If there is none, a path with two intermediary collections is looked up. The process is repeated until a path is found. Currently, Visor finds the first shortest path it can find and suggests it to the user by adding it to the current graph. While multiple shortest paths might exist, Visor recommends only the first one it finds. In cases where users want to find another path in the ontology they can simply select another collection, and the interface will attempt to link the last selected collection to all other selected collections. In such a way we ensure that whatever collections are selected the resulting sub-ontology is always connected and thus query-able. To distinguish selected and intermediary collections the latter are coloured in grey and are smaller in size.

The graph representation is rendered using a force directed layout and can be zoomed and dragged to improve visibility. Each node can be double clicked which opens up different inspector windows. These allow to view details about the sub-domain. In the following we describe the different kind of inspectors in Visor.

Collection inspector. Double clicking on the collections brings up a *collection inspector* (Figure 3a). The inspector shows the individual instances which are part of the collection, a description of the collection if one is available, and a list showing the possible properties that items from that collection can have. In Visor, object and datatype properties are listed separately. Object properties (or *relations*) are shown together with a corresponding collection to which they link (Figure 4). Furthermore object properties linking to and from other collection are show in separate lists (Figure 4). Users can than add these classes to the canvas. In such a way we support bi-directional set-oriented navigation, however in Visor we do so on the ontology level which serves as a potential roadmap for querying (Design requirement R3). Users can also view filters of items in the inspector by selecting any property (object or datatype). This shows the instances that only have that property and show the corresponding property value.


Select relationship to view items from inter-related collections

Connections to other collections (122)

Relation	Add to canvas
Nationality (To Eukaryotes collection) (1)	+
Death place (To Cities collection) (20) Highschool (To Cities collection) (61) Birth place (To Cities collection) (1177)	+
Death place (To Towns collection) (1) Highschool (To Towns collection) (7) Birth place (To Towns collection) (86)	+
Award (To Buildings collection) (3)	+
Nationality (To Ethnic groups collection) (71)	+
Award (To Awards collection) (24)	+

Connections from other collections (102)

Associated band (From Musical artists collection) (5) Associated musical artist (From Musical artists collection) (5)	+
Relation (From Military people collection) (1)	+
Producer (From Films collection) (14) Cinematography (From Films collection) (1) Writer (From Films collection) (6) Starring (From Films collection) (29) Editing (From Films collection) (1)	+
Associated band (From Artists collection) (5) Associated musical artist (From Artists collection) (5)	+

Fig. 4. Displaying properties in the collection ispector that link from or to items of a collection.

Item inspector. Clicking on any of the items in the collection inspector opens up an *item inspector* where all the data pertaining to the individual instance is shown (Figure 3c). A predefined lens is used to render the individual resource, including rendering images if any exist, a description of the item and the data associated with that item. If geographic coordinates are found for the item for example, a map is presented to the user. Additionally we show the collections that include the item. In the data panel, links to other resources opens the item inspector associated with that item. In such a way browsing from one item to another item is also supported in Visor.

Relations inspector. The relation nodes (the blue nodes in the visualisation) can also be inspected to quickly access properties that interlink items from two collections (Figure 3b). Clicking on any of the relations will display the items from both collections that are linked with that property.

With selecting collections users can create a subset of the ontology which is composed of concepts of their interest without restricting them to selecting a single collection and use navigation (Design requirement R1). With the inspector windows users can surface up the data on demand (Design requirement R2) to explore how collections are related, what are their individual instances, and if required inspect the instances themselves.

4.2 Spreadsheet Creation

Once a subset of the ontology is selected the user can query this information space by creating custom spreadsheets based on the selecting concepts and relations from the ontology subset (Design requirement R4). After the spreadsheet has been created users can export their custom made data collection in a format of their particular liking. Currently we support exporting data in CSV and JSON formats, however the system is extendable and multiple formats may be supported. In the following we describe the query interface and procedure for creating custom spreadsheets.

Main collection. The "Create a table" button located in the top menu of the UI opens up a query interface which guides users in selecting things from the previously explored domain (Figure 5).

The first step in creating a spreadsheet is selecting the *main collection* i.e. the collection that will be the focus of the spreadsheet (Figure 5a). This will instantiate a spreadsheet with a single column (the *main* column) composed of the instances from the main collection. All subsequent columns added to the table will be facets of the first column each created by specifying a path showing how the items of the newly created column are related to the items in the first column.

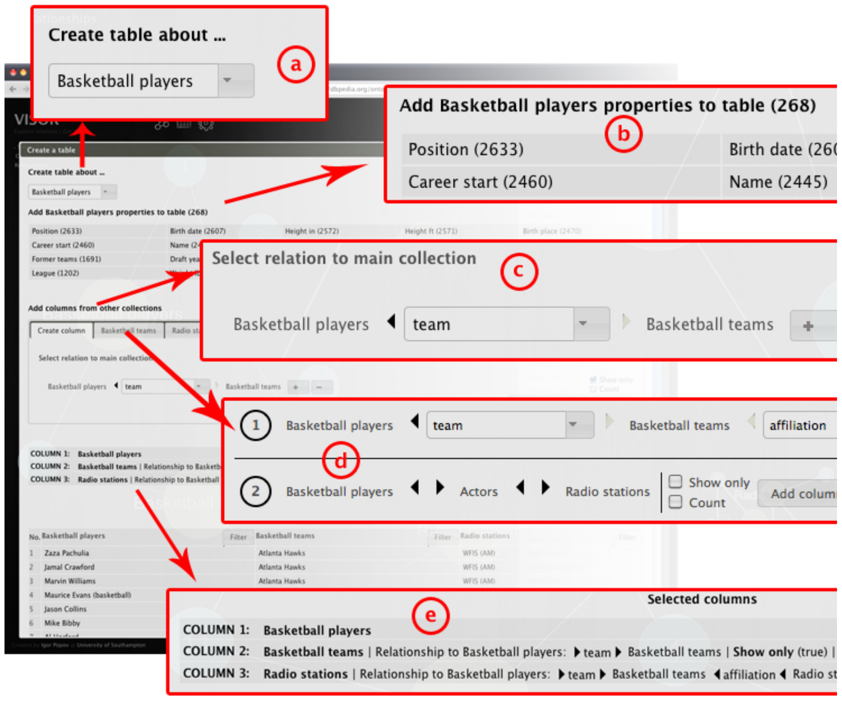

Create table about ...
Basketball players
a
Add Basketball players properties to table (268)
b
Position (2633)
Birth date (26
Career start (2460)
Name (2445)
Select relation to main collection
c
Basketball players
team
Basketball teams
1
Basketball players
team
Basketball teams
affiliation
d
2
Basketball players
Actors
Radio stations
Show only
Count
Add colum
Selected columns
e
COLUMN 1: Basketball players
COLUMN 2: Basketball teams | Relationship to Basketball players: team Basketball teams | Show only (true) |
COLUMN 3: Radio stations | Relationship to Basketball players: team Basketball teams affiliation Radio st

Fig. 5. Spreadsheet creating/query interface in Visor. Users start from selecting the main collection (a), datatype properties (b), columns other collections in the sub-ontology by specifying relations to the main collection (c) and (d). A preview of the columns in shown in (e).

Adding columns. Once the main collection is selected, adding additional columns is the next step. The first choice of columns are the datatype properties of the main collection shown in Figure 5b. Users can select a property and click on the "Add column" button to add the column to the table. By default when a column is added, Visor queries and tries to find a corresponding value for all the instances in the main collection. If such a value does not exist a "No value" cell informs users that the item in the main column does not have that property. The default option corresponds to generating a SPARQL query with an `OPTIONAL` statement. To filter for non-empty values users can check the "Show only" option before adding the column. In such a way user have the flexibility of selecting which columns are optional and which are mandatory to have a value in each cell. Additionally users can also choose the "Count" option to count the values in a cell in the corresponding to the item in the main column. Similarly, selecting the "Count" option corresponds to having a `COUNT` query in the SPARQL query.

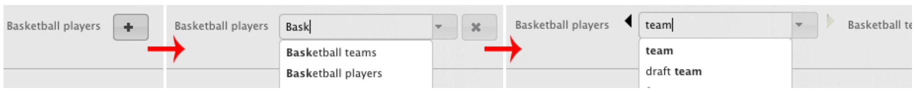


Fig. 6. Specifying a path from main collection to an arbitrary collection in Visor.

Defining column paths. Users can also select to add columns based on other collection in the current sub-domain. The query interface allows users to specify a path that connects items from the main collection to items in the newly added column. This can be implemented in two ways:

1. The first way is by using a path creation tool (Figure 5c). The path creation tool starts a path with the first element in the path being the main collection. Users can than select a collection that is related to the main collection using a drop-down list of available choices (Figure 6). Once a relating collection is selected, a property that links them is selected again from a selection of choices in a drop-down list. Then another collection can be chained to the previous one and again a property between them is specified and so on. When a column is added based on the specified path the column pertains to instances from the last collection in the path. We note that in the path creation tool enables users to connect the collections by properties going in both direction (the left and right arrows shown in Figure 5c,d).
2. To help speed up the process an alternative way of adding columns is supported in the UI. Each tab relates to a concept in the sub-ontology. Each tab panel contains suggested paths for reaching that node (Figure 5d). It basically list all the paths from the main collection to the collection specified in the tab. Then users need only to specify the properties in-between the collections. This will save users time, as well as give cues into all the different ways items from two collections can be related.

At any time users can update the current spreadsheet to monitor their progress. An overview of selected columns is shown to the user (Figure 5e) which allows to backtrack on choices made as well as rearrange the ordering. The spreadsheet also supports filtering for specific values in a column. Once users are satisfied with their custom spreadsheet they can choose to export it in a number of different formats.

4.3 Implementation

The implementation in Visor is composed of a front end (UI) and back-end system. The UI is based on HTML5 and Javascript together with the jQuery library[4]. For visualising the ontology we relied on the Protovis visualisation toolkit [5]. The Visor back-end server is a Python/Django application that serves data in a JSON format to the front-end by exposing a RESTful interface. Thus the UI side of the application does not rely on any raw SPARQL query generation or parsing SPARQL results.

[4] `http://jquery.com/`

5 User Study

In order to ascertain whether people will be able to learn and use Visor we conducted a user study. The purpose of our study was two fold: (1) we wanted to test if there was any major issues in the ability of users to comprehend and use our UI and (2) identify specific usability problems and areas where interaction can be improved. Thus, our goal was to test if our approach was viable.

5.1 Study Design and Procedure

For our study we recruited ten participants through an email advertisement among graduate students at the University of Southampton. Seven of the participants were male and three female and their ages ranged between 21-41. We wanted to have a diverse group of users with respect to knowledge of Semantic Web/Linked Data technologies and see if there were any particular difficulties among users with different skill levels. We asked them to rate their knowledge of Semantic Web/Linked Data technologies on a scale of one to three, one being *"very basic understanding or no knowledge"*, two being *"some knowledge and understanding"*, and three being *"high or expert knowledge"*. To gain further insight in their skills, we also asked them to rate their knowledge on the same scale to several specific areas: Linked Data application development, (2) Use of SPARQL, and (3) Ontology Engineering and/or data authoring. Calculating the averages of the responses by participants 50.25% or roughly half or the participants had no or very little understanding of Semantic/Linked data technologies, 30% or about 3 participants had intermediary knowledge and 17.5% or about 2 participants had expert knowledge.

For our study we relied on a cooperative protocol analysis or "think aloud" method [6]. We choose this method because we wanted to pinpoint any potential usability issues introduced by the design requirements R1-R4 and get the users insight into what were the problems.

Each participant went through a study session that took approximately one hour to complete. A session was structured in the following way. First, the participant was shown a 6 minute video[5] tutorial of Visor. The tutorial explained the terminology of the UI and showed a complete example worked out in Visor. Second, the participants were handed 3 written tasks to complete. During this time the "think aloud" protocol was observed, and we recorded the users screen and audio. Finally, participants were required to fill in a questionnaire, in order to reflect and give feedback based on all the entire session with questions targeting specific portions of the UI.

Two of the tasks were structured tasks i.e. the users we given a concrete task with a clear result. The tasks were given with increasing difficulty: the first task required a three column table with generated with one-hop links in a single direction, while the second required more column, specifying a loop pattern, and setting paths with bi-directional patterns. One example task was the following:

[5] The video is available at: `http://vimeo.com/24174055`

In a history course you are required to find which royals from which countries have intermarried. You decide that you will need a table showing: All the royals you can find, which country these royals were born, the royals who are spouses of the royals, and the country the spouses were born.

The third task was unstructured i.e. we gave users a general area to browse and explore and come up with some data of their particular interest. The task was to find some data pertaining to *Scientists*, *Universities*, and *Awards*.

5.2 Results

During the task we focused primarily on three things: (1) Observing user behaviour during data finding tasks, (2) observing when users chose to view the actual instance data and for what reasons, and (3) observing what problems participants experienced when attempting to create their spreadsheets.

Data finding. When searching for collections to build up their sub-domain for answering their tasks most participants (nine out of ten) choose to use multiple collection selection rather than use navigation after selecting their first collection. Only when the the resulting connections contained intermediary nodes that did not meet the requirements of the sub-domain did they resort to navigating to other potentially useful collections. This was particularly the case during the exploratory task. Most users used a keyword search option to search for collections. Beyond using it for finding collections participants suggested additional ways search can be useful for finding additional data. For example, one user commented that the use of synonyms would be helpful to find collections. Another user, for example, wanted to search for a particular instance during the exploratory task because the user wasn't sure in which collection that particular item can be found. Beyond searching for instances and collection we observed that some users tried looking up things that we currently did not support e.g. searching for relations. For example, one user thought that there might be a collection named "Spouses" before realising that it might be a relation instead.

Showing instance data. We also observed users to see how much would they need to view the underlying instance data during various stages and across different tasks. We concentrated our observations on two things. One was to observe if users needed to examine the underlying data during the initial exploration phase when the user was building a sub-domain of the data. Second we wanted to test if they can specify relationships that were three or more hops away without seeing the intermediary data that relates them. During the structured tasks we found that users did spend very little time or no time exploring the generated sub-domains with the inspector tools. Six participants chose to directly open the spreadsheet creation tool mentioning that they felt confident they had everything they needed to answer the query. Three others noted that they just wanted to open up the inspectors to explore and but mentioned no particular reason except for just casual exploration. While we observed a slight increase during the exploratory tasks we did observe that the spreadsheet creation tool was used as an exploratory tool as well. When creating their spreadsheet more than half of the participants chose to view their progression with each added column. At the start of the sessions, novice and intermediary users reported difficulties in grasping how paths worked, but once explained they felt confident in generating paths two or more hops away without viewing the intermediary data.

Spreadsheet creation. As expected, users found the spreadsheet creation tool was the most difficult part of the interface the user to learn and use. Some suggested better integration with the visualisation by either selecting or being able to drag and drop directly from the graph into the header row of the spreadsheet. While for the two expert participants and one non-expert specifying direction of the relationship made sense, for most other participants, specifying the direction of the relationship seamed irrelevant. When asked what they would prefer, most of them responded that they would like a single list of how to relate two collections instead of two separate lists. However we did not observe that users had any difficulties in specifying paths where bi-directional patterns occurred. When faced with the choice of using the template paths or create the paths manually, the preference of participants were split among the two choices. Participants recommended, however, that rather than specifying a new path every time they would have been able to reuse paths from existing columns which were sub-paths of the new path.

Task completion and survey. Task completion was generally high: eight out of ten participants were able to complete the all the tasks and create spreadsheets to the specified requirements of each task. Overall we found that the users were able to easily learn and create their spreadsheets after the one hour session. After going through the tasks, participants were asked to submit a survey rate the overall difficulty of using the tool on a Likert scale of one to five. Two participants reported that the found the tool very easy (1) to use, six reported it easy (2), one user reported it average (3) and one user reported it difficult (4) to use. When asked to rate specific components of the UI most participants (8 of 10) reported that the graph visualisation useful and easy to use while they gave the spreadsheet creation tool an average (3.2) score.

5.3 Implications for Design

Based on the results we compiled a set of recommendations which we encourage future designers to consider when designing data-centred exploration interfaces.

Integrate keyword search with direct manipulation techniques. The approach in this paper gave more flexibility than standard browsers by enabling users to select multiple points of interest. We noticed that users not only took advantage of this flexibility, they even wanted more freedom when trying to find the portion of the data domain that is of their interest. We suggest that rather than being able to just add multiple collections, users should be able to search more freely for thing such as properties, instance data and view how they are relevant in the already explored data. Therefore we suggest integrating keyword search techniques with direct manipulation techniques as a possible way of providing flexibility in finding data during initial exploration. So far direct manipulation techniques have been mostly focused on supporting only direct manipulation techniques for navigating graph data, while interfaces supporting keyword search have focused on entity retrieval or question answering. We encourage future designers of data browsers to consider closely integrating keyword search with the direct manipulation features of the data browser.

Support bidirectional navigation. In our related work section we noted that a lot of interfaces support navigation in a single direction i.e. only form outgoing properties which limits the expressivity of the queries that can be answered by the data browsing UI. While we realise that from an implementation point supporting finding back-links on the open Web of Data is much harder task than when the data is contained in a single store, our study showed that from an interaction point of view supporting both does not have any significant impact on users when browsing.

Show data on demand. One of the thing our study showed was that users can browse and query for data without relying too much on always viewing the data. We recommend that when future UI designers of data browser develop their tools they use less screen real estate on showing too much instance data at once, or at least give several views to users. Retaining context while exploring or combining querying with visual aids can be utilised to give overview of the exploration path and make querying easier.Users should, however, always retain the option of viewing the instance data of current result of the exploration at any time.

6 Conclusion

In this paper, we examined some of the interaction challenges associated with pivoting as a exploration technique for data browsers. We also presented Visor, a tool for users to create custom spreadsheets by exploring a dataset using a combination of multiple selections as well as link navigation. With Visor we have shown a flexible way of finding data in large graphs and presented an overview-first data-on-demand approach to browsing data.

Our future work includes extending Visor based on the recommendations we laid out in this paper. We plan on adding additional features to further increase the flexibility for understanding complex data domains. Our plan is to support multiple potentially complementary ways of finding data including better integration with keyword search as well as more visual aids. We then intend to extend our evaluation with comparing and measuring support of different data-seeking tasks in UIs supporting different navigation and exploration techniques to better understand for what kind of task what exploration model works best.

7 Acknowledgments

This work was supported by the EnAKTing project, funded by EPSRC project number EI/G008493/1. Many thanks to Manuel Salvadores and Christos Koumenides for providing valuable feedback on this work.

References

1. S. Araujo, G.-J. Houben, D. Schwabe, and J. Hidders. Fusion - visually exploring and eliciting relationships in linked data. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.
2. S. Araujo, D. Schwabe, and S. Barbosa. Experimenting with explorator: a direct manipulation generic rdf browser and querying tool. In *VISSW2009*, February 2009.
3. T. Berners-lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *In Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06*, page 06, 2006.
4. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154 – 165, 2009. The Web of Data.
5. M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1121 –1128, 2009.
6. A. Dix, J. Finlay, G. D. Abowd, and R. Beale. *Human Computer Interaction*. Pearson, Harlow, England, 3. edition, 2003.
7. A. Harth. VisiNav: Visual Web Data Search and Navigation. In *Database and Expert Systems Applications*, pages 214–228. Springer, 2009.
8. P. Heim, T. Ertl, and J. Ziegler. Facet graphs: Complex semantic querying made easy. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, volume 6088 of *LNCS*, pages 288–302, Berlin/Heidelberg, 2010. Springer.
9. P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. Relfinder: Revealing relationships in rdf knowledge bases. In *Proceedings of the 4th International Conference on Semantic and Digital Media Technologies: Semantic Multimedia*, SAMT '09, pages 182–187, Berlin, Heidelberg, 2009. Springer-Verlag.
10. D. F. Huynh and D. R. Karger. Parallax and companion: Set-based browsing for the data web. In *WWW Conference*. ACM, 2009.
11. D. F. Huynh, D. R. Karger, and R. C. Miller. Exhibit: lightweight structured data publishing. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 737–746, New York, NY, USA, 2007. ACM.
12. G. Kobilarov and I. Dickinson. Humboldt: Exploring linked data. 2008.
13. m.c. schraefel and D. Karger. The pathetic fallacy of rdf. In *International Workshop on the Semantic Web and User Interaction (SWUI) 2006*, 2006.
14. B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336 –343, sep 1996.
15. F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. Manyeyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13:1121–1128, 2007.