7
Design Issues and Implementation

This chapter presents a detailed discussion about the design and implementation of a real exploration environment based on the formalization of the functional layer presented in chapter 4 and the separation of concerns approach presented in chapter 3. Here we present a novel approach to characterize the design space of exploration tools with special emphasis in the Interaction/Interface layer.

To illustrate the discussions of the design issues and possible solution alternatives, we use the case of the design and implementation of the XPlain environment, which provides higher expressivity when compared to state-of-the-art tools. Moreover, we also present comparisons between state-of-the-art tools in order to demonstrate the generality of the issues discussed. This chapter can be used as a guideline for designing new expressive exploration environments over semi-structured data.

7.1.Functional Layer

The functional layer contains the exploration operations of the proposed framework accessible through a specific DSL (Domain Specific Language) in Ruby language. As an application example of the DSL consider the case study "Evaluating a scientific paper**",** described in chapter 5, section 2. Subtasks 1, 2, and 3 are as follows: 1) Analyze the age of the citations by extracting the set of years of the citations and calculating the mean year; 2) Check potential relevant publications that are missing from the citations list; 3) find how many citations are publications from the same authors of the reviewing paper. These subtasks are represented as the following sequence of steps in our DSL. Let *d* be a variable containing a reference to the dataset and *p* be another variable referencing a unitary set containing the reviewing paper:

```
1  #Subtask1: mean publication year of the references
2  s1 = p.pivot{relation "cite"}
3  s2 = s1.pivot{relation "year"}
4  s3 = s2.map{mean}
5
6  #Subtask2: find missing relevant citations
7  s4 = d.refine{match_all "Semantic Web"}
8  s5 = s4.group{relation "cite"}
9  s6 = s5.map{count}
10 s7 = s6.rank{score_by_image level: 1}[0..19]
11 s8 = s7.diff s1
12
13 #Subtask3: finding the amount of self-citations of the paper
14 s9 = p.pivot{relation "isDocumentContext", "isHeldBy"}
15 s10 = s9.pivot{relation inverse("isHeldBy"), inverse("isDocumentContextFor")}
16 s11 = s1.intersect s10
17 s12 = s11.map{count}
```

Figure 32 - DSL representation of the solution strategy for the paper review case study presented in chapter 5, section 2

Each exploration set is referenced by a Ruby variable (s1…s12), where the user can apply an operation using the following rule:

<setVar>.<operation>{<ParamsSpec>}

The parameters of all but set operations are defined within ruby blocks, which can be delimited either by brackets or by the keywords “do end”. For example, the pivoting and grouping relation parameters are defined as “{relation :cite}”, where “relation” is the name of the parameter and :cite is the value for the parameter. The “relation” parameter also allows inversion of relations and relation paths, such as the execution presented in the line 15, where the user pivots backwards from authors to publications through the relation path :isDocumentContextFor:isHeldBy. Auxiliary functions are specified in a similar way. For example, the *mean* function for the *Map* operation is also specified in a block. Refer to Attachment A for a formal description of the DSL in the EBNF[13] metalanguage.

The result sets achieved by the script of Figure 32 are stored in an exploration session and each exploration set keeps a reference to input sets they depend on. Therefore, there is a dependency graph that can be shown in the interface as an exploration trail and further manipulated for reuse purposes.

In XPlain, a Web interface was developed on top of the DSL. The client sends DSL expressions to the server, which executes them, and returns the

[13] https://www.iso.org/standard/26153.html

resulting exploration sets. The next session presents a discussion of the design issues concerning exclusively visual and interaction aspects of the interface.

7.2.Interaction/Interface Design

Once the designer has a thorough view of the operators and their possible combinations for exploration tasks, she/he must focus exclusively on deciding which interaction paradigms and visual representations are more adequate. This section presents a discussion of interaction/interface issues separating its concerns from the exploration actions and compositions defined by the functional layer. Based on the concepts of the functional layer, the main goals of the interface are: (1) to present one or many exploration sets, where the items of each set may be hierarchically organized (nested); (2) Allow the user to select an operator/composition and specify its parameters; (3) Allow the user to visualize, manage, and browse the exploration trail. In order to leverage the discussion, we used the case study in the scientific publications field described in chapter 5, section 2. Figure 33 shows a screenshot of the main screen of the XPlain environment.

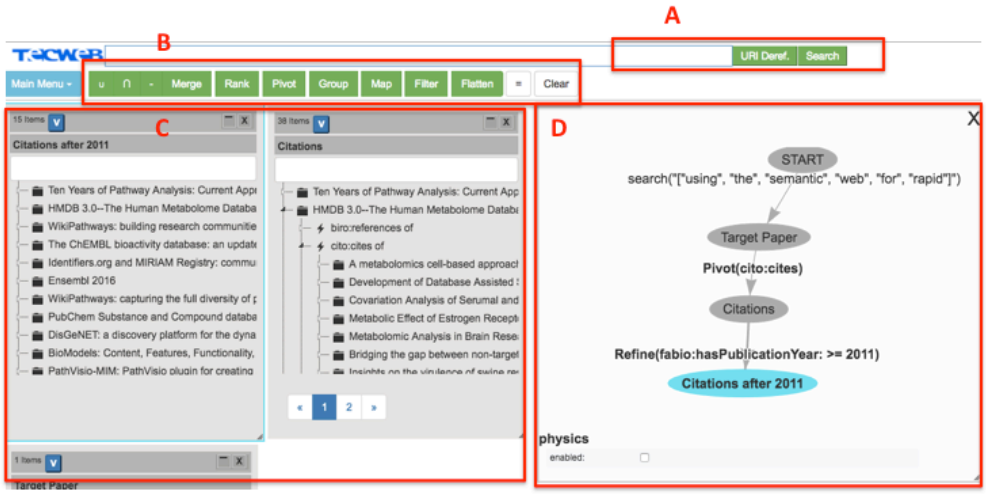

Figure 33 - The interface of the XPlain environment. (A) keyword search controls; (B) Exploration operations toolbar; (C) Exploration sets area; (D) Exploration trail view.

7.2.1.Requirement 1: Manipulation of Exploration Sets and Items

The first challenge for the design of exploration environment interfaces is how to present the data being manipulated and its relationships. The biggest issue to be dealt with is handling the potential excess of information to be presented, as the number of items can be very large. Here, Shneiderman's visual information

seeking mantra “Overview first, zoom and filter, then details-on-demand” (SHNEIDERMAN, 1996) should be considered as a guideline.

Considering the conceptualization of the exploration process as a functional composition that results in multiple exploration sets, the design alternatives for presenting those sets are: show one exploration set at a time (unifocal) or show many sets at a time (multifocal). Unifocal interfaces have the advantage of reducing the amount of information shown at a given time and requiring less focus management interactions. However, they do not properly support operations that take more than one set as input, such as comparisons of alternatives (BUSCHBECK *et al.*, 2013). For example, imagine a user interested in comparing the publication profiles (e.g., venues in common) of two researchers in a certain period of time. S/he filters the publications of each researcher by the desired period, pivots by venue, and computes the intersection (or difference) of the two sets. In a unifocal interface s/he must apply this sequence of operations to each researcher one at a time, and somehow apply the intersection operation (if available in the functionality layer) on the results, which won’t be both available in the same interface. Note that if the functionality layer does not provide a set operation, s/he must annotate the results and then make the comparisons offline. For web browsers, a common strategy is to open two or more windows and organize the windows to support the comparisons, but this is limited by available screen real estate.

In a multifocal view s/he is able to visualize the two sets of venues simultaneously and compute the intersection straightforwardly. The drawback of multifocal interfaces is the need to design focus management controls, such as, maximization, minimization, restoration, and layout organization controls to avoid information overload. If comparisons between alternatives are not the case or the device is very restricted in screen size, unifocal interfaces may be more appropriate.

An additional design issue is the layout and presentation of the relationships between the sets on the screen. To illustrate one possible set of options, in XPlain we opted for a multifocal interface to better support operations over multiple sets, where each set presentation has minimization/maximization controls. Figure 33C shows two exploration sets in the workspace. The last generated set is always placed on top of the screen and the exploration trail presents the relationships

between the sets and also allows the user to navigate to intermediary sets by clicking on the corresponding node in the graph. After deciding between unifocal and multifocal presentations, it is necessary to define the organization and interactions for the exploration items and the relations they participate in, within the exploration sets.

According to our functional model, there are two types of item relations that must be considered: schema relations and computed relations. Computed relations are relations created along the exploration process, such as grouping relations or mappings, which not necessarily have an identifier, such as predicate URIs in RDF or column names. There are three common representations of schema relations in the literature: the tabular view, where each relation becomes a column and the related items are presented in rows; the graph view, where the relations are the edges between nodes; the list view, where the relations are presented in a list, which usually has multiple levels.

Tools presenting tabular views model the exploration process as a sequence of manipulations of rows and columns, where the interactions are very similar to those offered by electronic spreadsheets. However, depending on the amount of columns and rows required for the task, which is proportional to the range of concepts and dimensions under exploration, presenting all data in a single table may be unfeasible. Moreover, it neither leverages the visualization of the schema structure nor the join items, i.e., items related to two or more items. A tabular view may be easier for spreadsheet users while graph views favors the visualization of the schema structure and the joins between the items. Visor reconciles both graph and tabular views, where the graph view presents an overview of the RDF classes and their relationships, as Figure 34 shows. The tabular view in Visor presents detailed information of the items of the selected classes. Figure 34 shows the graph view of the schema implemented in Visor and Figure 35 shows the tabular view implemented in Liquid Query tool and in its follow up Search Computing.

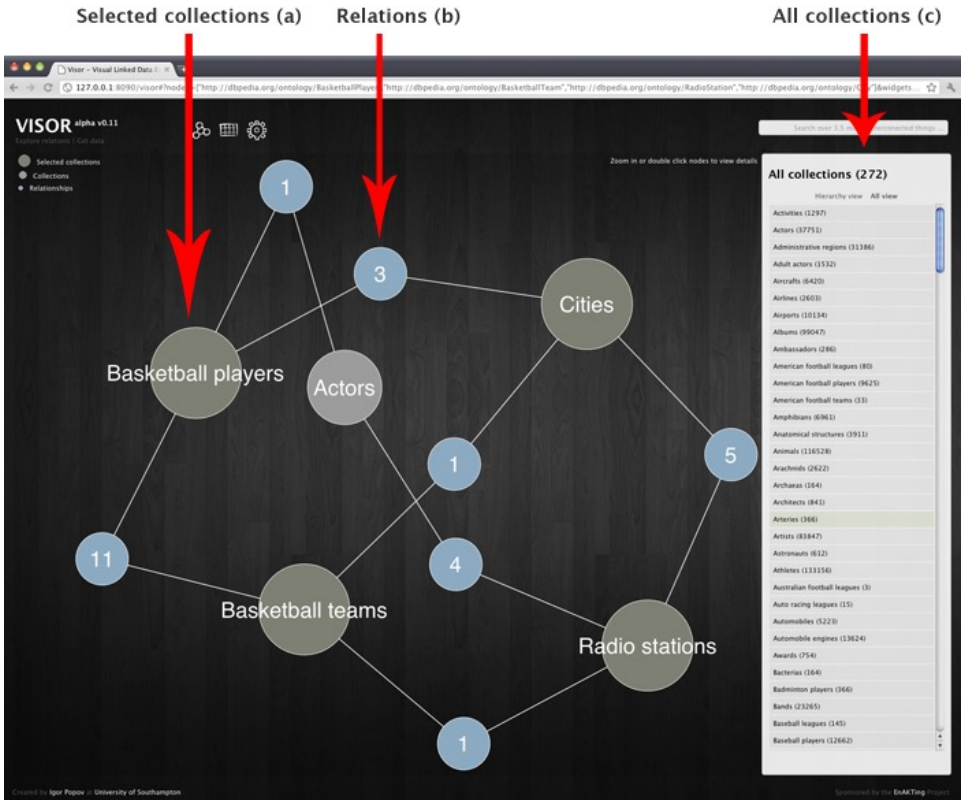

Figure 34 - Visor screenshot (POPOV *et al.*, 2011)

Concert						Hotel			
	Name	Address	City	Dist.(Mi.)	Start Date	Name	Address	Dist.(Mi.)	Ratin
☐	Jazz at the Rrazz Featuring Bruce Forman - Guitar With Mike Greensill	222 Mason St	San Francisco	0.76	12/21/2009	Hilton San Francisco	333 Ofarrell St	0.05	4.5
☐	Jazz at the Rrazz Featuring Bruce Forman - Guitar With Mike Greensill	222 Mason St	San Francisco	0.76	12/21/2009	Hotel Bijou	111 Mason St	0.07	4.5
☐	Jazz at the Rrazz Featuring Bruce Forman - Guitar With Mike Greensill	222 Mason St	San Francisco	0.76	12/21/2009	Serrano Hotel	405 Taylor St	0.12	4.5
☐	Ahmad Jamal	1330 Fillmore Street	San Francisco	0.76	12/10/2009	Metro Hotel San Francisco	319 Divisadero St	0.71	4.5
☐	Ahmad Jamal	1330 Fillmore Street	San Francisco	0.76	12/10/2009	Nob Hill Hotel	835 Hyde St	0.94	5
☐	Ahmad Jamal	1330 Fillmore Street	San Francisco	0.76	12/10/2009	Holiday Inn San Francisco-Civic Center Hotel	50 8th St	1.02	4.5
☐	Gaucho, Mitch Marcus Sessions	853 Valencia St	San Francisco	1.24	12/02/2009	Phoenix Hotel	601 Eddy St	1.67	4.5
☐	Gaucho, Mitch Marcus Sessions	853 Valencia St	San Francisco	1.24	12/02/2009	The Powell Hotel	28 Cyril Magnin St	1.88	4.5

Figure 35 - Tabular view of Liquid Query (BOZZON, ALESSANDRO *et al.*, 2010)

Alternatively to graph and tabular view, the list view usually shows items and relations in separate lists, where the relations list presents items' relations and their respective related values. This presentation is commonly found in unifocal faceted search systems, where the relations and their values are presented as filtering facets. The ordering of the related values in the list is often given by the

amount of items that participates in the relation. Figure 36 presents the implementation of a list view for items and facets in Rhizomer.

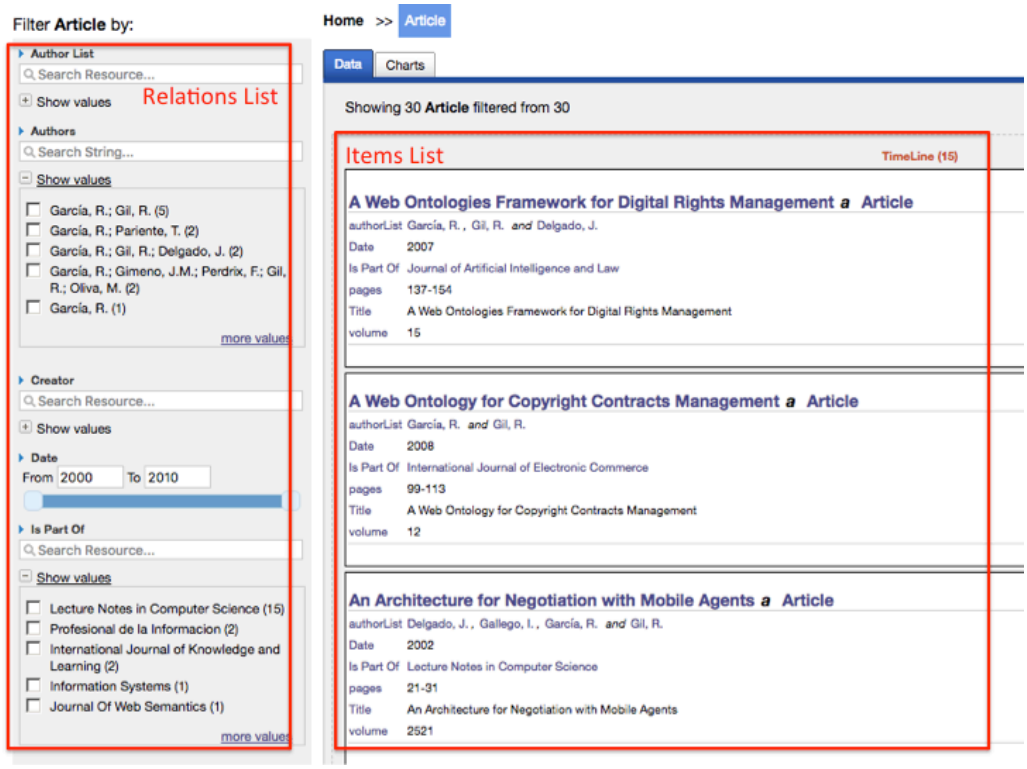

Figure 36 - Rhizomer list view

A variation of the list view presentation is usually found in tools that support facets that are hierarchically organized. Some examples of hierarchical facets are locations, organized in cities, states, and countries; Temporal facets, organized in days, months, and years; Classification facets with sub-class relationships, such as the International Patent Classification taxonomy. In order to cover this type of relation, the related items can be presented in a multi-level list, as implemented in Flamenco and /facet. Figure 37 shows the multilevel list for the Style/Period facet of artistic works in /facet.

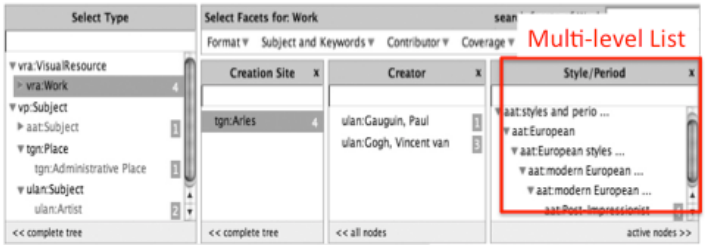

Figure 37 - /facet multilevel list view

In XPlain we took a different approach as we adopted the metaphor of a directory system, where items are mapped to directories and both schema and computed relations are nested directories, as shown in Figure 38. This choice

allows a natural representation of groups, where each group is represented as a separate directory. The drawback is the visualization of items that participate in more than one relation. If an item is related to two different nested items, it will appear in two “directories”.

Even in a unifocal interface, the amount of items within a single exploration set can be considerable. Therefore, the designer should weight choices about presenting the set using scroll and/or pagination controls.

It is also typically desirable to apply some natural ordering on the items. Although our model describes ranking as an independent exploration operator, it can also be used in conjunction with other operators. Thus, even when operators different than ranking are selected, such as keyword refine or grouping, the interface can also make the composition with a ranking function and send to the server in order to enforce a natural ranking for the result set. In XPlain we opted for pagination controls with a limit of twenty items per page and an alphabetical or numerical ordering of results.

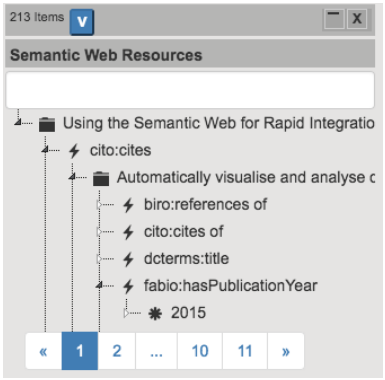

Figure 38 - Visual representation of an exploration set as a nesting of items and relations.

In summary, the interface design issues for the manipulation of exploration sets and items are:

1. Choosing between unifocal and multifocal view
    a. If multifocal: design appropriate focus management controls for the sets, such as maximization, minimization, restore, and layout organization controls.
2. Deciding where and how to show relationships between exploration sets;
3. Designing focus management controls for the items within the set e.g. pagination, scroll, or a combination of both.

4. Determining the best visualization for items and relations (schema and computed): graphs, trees, tables, lists, multi-level lists, etc.
5. Establishing a natural order for presenting exploration items and relations, possibly, adding sorting controls.

We can draw comparisons between state-of-the-art systems addressing interface and interaction features concerning the manipulation of exploration sets and items. Table 2 presents a summarized list of design issues and their implementations among exploration tools. Notice that some tools present more than one alternative for the same issue. For example, Visor allows the user to shift between tabular, graph, and list views of the exploration items.

Table 2 - comparison of design choices among exploration tools

Tool/Issue	Focus	Items View	Relations View	Sorting Controls	Natural Sort	Focus Management
/facet	Unifocal	List	Multi-level list	No	Yes	No
gfacet	Multifocal	List	List	No	Yes	Pagination
Parallel Faceted Browser	Multifocal	List	List	No	No	No
Rhizomer	Unifocal	List	List	Yes	Yes	Pagination
Relation Browser	Unifocal	Tabular	List	No	No	No
BrowseRDF	Unifocal	List	List	No	Yes	No
Sewellis	Unifocal	List	Multi-level list	No	Yes	Pagination
Visor	Multifocal	Graph, Tabular, List	Graph, Tabular, List	No	Yes	Pagination
Liquid Query	Partially multifocal	Tabular	Tabular, List	Yes	Yes	Pagination
SeCo	Partially multifocal	Tabular, List	Tabular, List	Yes	Yes	Pagination

7.2.2.Requirement 2: Applying Exploration Operations

The application of exploration operations presents another class of interface/interaction design issues, which concern both the selection and activation of an operator, and the definition of its parameters. The functional layer defines four types of arguments: exploration items, auxiliary functions, relations, and relation paths. Next, we argue that each argument type may require distinct interaction models.

To invoke an exploration action the user must assign the values to each input parameter of the invoked operation. Each assignment is a binding, i.e., a pair <Parameter, Value> that will be evaluated when an operation is executed. For example, pivoting requires two bindings: the definition of the input set and the pivoting relations. For binding definitions, the interaction issues are: defining the assignment order for parameters and defining the interaction that will support the binding definition. The latter issue depends on both the argument type and the operation.

With regards to the order of the assignments, consider the pivoting action as an example. Some design alternatives are: the user selects the input set, activates the pivoting operator, and the system shows the relations for selection (e.g., interaction in SeCo (BOZZON, A *et al.*, 2013)); Alternatively, when the user selects the set, the interface could show all relations as selectable elements whose activation causes a pivoting over the selected relation. For tabular presentations, the first alternative may be better due to layout organization issues, however, for graph and list presentations the second option is closer to hypertext browsing, which may favor Web users. The second option is the solution adopted by the majority of faceted search tools with pivoting functionality.

Another example is the definition of bindings for the *Refine* operation, where the user should select the filtering relation, the comparison operator, and a value. One option is to simply allow the selection of values, where the relation is inferred and the comparison operator is always an equality test. Another option is to allow the selection of the relation, the value, and the filtering predicate, which may be different than equality comparison e.g., greater than or less than operators. Therefore, there can be many distinct interaction sequences for the definition of the bindings for each operation.

The next issue concerns specific interactions for different types of parameters. Considering the case of the *Refine* operation where the user must define bindings for the relation or the relation path, the comparison operator (e.g. =, <, >), and the restriction value for the relation, which is an exploration item. With regards to the relation, the interface has to reconcile the selection of relations and relation paths.

For example, in Open Citations, if we want to refine papers by venue names, we must bind the relation path *:isDocumentContextFor:isHeldBy:name* to the relation parameter, as shown in Figure 39B. One design option is to allow the user to pivot relation by relation in this path until reaching the next to last relation, which is the *:isHeldBy* relation. At this point, the interface can show the possible relations and values for the holders, which includes the *:name* relation and the actual venue names for selection. The computation is, therefore, carried out relation by relation until the desired path is achieved. Next, a selection of a venue name will cause the refinement over the path *:isDocumentContextFor:isHeldBy:name*. This is the most common interaction for path refinements found in faceted search interfaces, but considering the size of paths, the possibility of mistakes, and the amount of refinements required for the task, this design option can be cumbersome. Another option is to allow the visualization of relation chains on demand, where the user can explore and select relation paths without causing a context change (pivoting). XPlain implements this design option with relation nestings built in runtime, where when a relation *:x* is nested with a relation *:y* then there is relation path *:x:y* in the dataset. This design allows the user to browse the nestings in order to find the desired path, with reasonable performance. Figure 39B shows the nesting of *:isDocumentContextFor*, *:isHeldBy*, and *:name* relations.

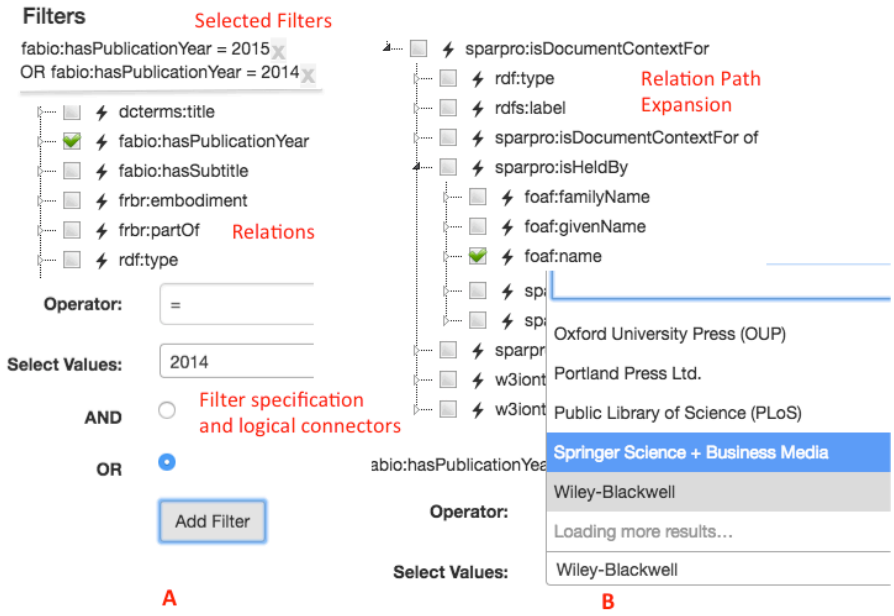

Figure 39 – Xplain's view for the *Refine* operation. The user selects relations (A) or relation paths (B) and restriction values for each filter. Filters can be disjunctive or conjunctive according to the selected logical operator.

Auxiliary functions, such as the comparison operators, the scoring function, and the mapping functions can be picked out in the interface from a pre-defined set. For example, the *Refine* modal dialogue presents a selection box with all comparison operators available, as Figure 39A shows. However, since it is very difficult to define a complete range of functions that covers all problem domains, the interface can also allow the user to describe the function in some computable language. Consider a user wishing to convert a set of measures to a different scale. The functional layer provides the *Map* function for such tasks, but, the desired scale converter is not among the available mapping functions. The user could simply type the formula and the interface creates the binding. Therefore, the interaction design should not only consider interface selection, but also textual inputs, with some validation in the case of function definitions, and filterable selection lists. The same issue also occurs for bindings of exploration items. XPlain's interface allows the definition of new auxiliary functions using a Domain Specific Language (DSL) implemented by the functional layer. We also choose filterable selection lists for the definition of the filtering values, as shown in Figure 39B.

Up to now we posed the interaction/interface design issues and possible solution ideas for the execution of single operations. However, for some recurring

functional compositions, there can be alternative interaction styles to the operation-at-a-time approach. An example of such compositions can be found in the expansion of an exploration item, shown in Figure 38. When a user double clicks an item in the exploration set, XPlain executes a composition of *Refine* and *Pivot* to respectively select the clicked item and pivot to its set of relations. The relations are shown as nested items that can also be expanded. This interaction allows the user to browse the graph of relations of an item in a follow-your-nose style without causing a context change or the addition of a new exploration set for each *Refine* and *Pivot* executed. Therefore, the designer can explore alternative interaction and interface designs for combinations of operators that are more appropriate for a given task context.

Capturing common combinations that would require more appropriate interaction designs is likely difficult, since the combinations may not be obvious for some domains and contexts. However, we expect that these combination patterns should emerge with the continued use of the environment. Since the patterns are formally described and recorded, they can be mined from the environment log and analyzed from the perspective of differentiated interface and interaction models.

Another class of interaction issues concerns the possible sequence of actions the user can take. In order to discuss this aspect, we can approach the exploration process as a conversation between the user and the system using the interface language. A designer can use episodic models to structure the user-system dialogue independently of the interface controls, such as the Modeling Language for Interaction as Conversation (MoLIC) model (BARBOSA; GRECO, 2003) or the "Conversational Roles Model" (COR) used in (STEIN; MAIER, 2008). Here we select MoLIC to exemplify how this can be accomplished. MoLIC models the user-system interaction as a dialogue between the user and the system, considered as the designer's deputy. The interaction dialogue is organized in *Dialogue Scenes*, which represents a conversation about a certain topic. There are also *Transition Utterances*, which represent the turn taking between the interlocutors in the conversation. Figure 40 shows a MoLIC diagram for buying tickets for a theater, where the tag "d:" identifies a designer utterance, the tag "u:" identifies a user utterance, and the tag "d+u" identifies a conversation between designer's deputy and users about some topic.

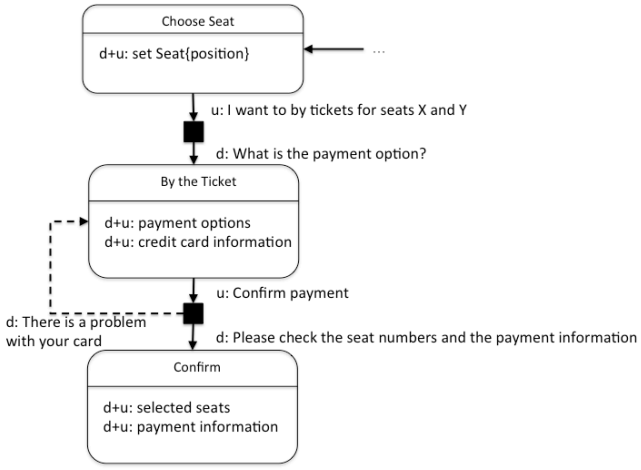

Figure 40 - Example of MoLIC diagram for the task of buying a ticket

In the scenario of Figure 40, there are three interaction scenes, represented as white boxes. The black boxes stand for system processing. Each scene is described both by a title and by the information required for the dialogue. For example, in the "Choose Seat" scene, the designer's deputy presents a set of available seats, identified by their respective positions, to the user (set Seat{position}). The user selects the seats, which causes a transition to the "By Tickets" scene. In this scene the designer's deputy and the user dialogue in order to define the payment option and the user also informs the credit card information. Next, the user asks the designer's deputy to confirm the payment. In case of problems with the credit card, the designer's deputy emits an error handling utterance and the dialogue returns to the "By Tickets" scene. If the credit card information is valid, the designer's deputy and the user go to the confirmation scene.

The diagram of Figure 40 describes the relationships between interaction scenes along with the information required for the conversation, but the details of the conversation within the scenes are abstracted. Each scene can be further detailed in a sequence of utterances when necessary. For a detailed discussion of the MoLIC language, refer to (BARBOSA; GRECO, 2003).

In the context of exploration environments, the interactions required for the defining bindings and executing operations/compositions can be modeled in one or many interaction scenes. Consider again the case of combinations of *Refine* and *Pivot*, which describes the majority of faceted search tools. There are two

dialogue structures that are most commonly found. Figure 41 A and B present these structures.

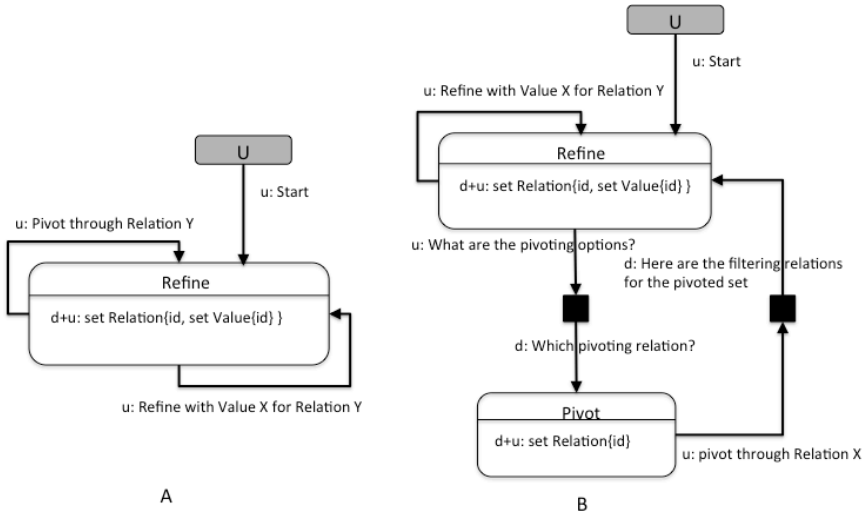

Figure 41 - (A) Pivot and Refine operations in a single scene; (B) Pivot operation defined in a different interaction scene.

Figure 41 A and B present two alternative dialogue structures for combinations of *Pivot* and *Refine* operations in faceted search tools. The boxes with gray backgrounds represent ubiquitous access, which are always available to the user. In Figure 41 A, the user starts the exploration using the ubiquitous access and the dialogue goes to the *Refine* scene, from which the user can either apply a restriction or pivot to a related set. The conversation in this scene involves a set of relations and their respective values, where the tag "d+u" informs that both interlocutors interact with the tagged information. When the user asks the designer's deputy to refine the set she/he informs the relation and the desired value for the relation – "…Value X for Relation Y". The pivoting is executed in similar way but the user only has to inform the relation. Either pivoting or refinement utterances lead to the *Refine* scene for next actions. Alternatively, the user can define the binding for the pivoting relation in a scene apart, as Figure 41B shows.

In the design option of Figure 41B, in order to pivot, the user asks the designer's deputy to present the possible pivoting relations, which causes a transition to the "Pivot" scene. Within this scene, the user dialogues with the designer's deputy about the relations until she/he finds the desired one. Finally, she/he asks to pivot through the selected relation, which causes a transition back to the "Refine" scene. Such design option may favor schema learning since

presenting relations in a separate scene can leverage the presentation of additional information, such as descriptions and information about their ranges and domains, avoiding information overload. The design option of Figure 41A can be found in Parallax, Rhizomer, /facet, Sewelis, and others. The design option of Figure 41B can be found in Liquid Query, SeCo, and gfacet.

In XPlain, the dialogue structure was modeled with the goal of being a full expressive environment, according to our framework of operations, which led to a richer dialogue structure. Figure 42 presents XPlain dialogue structure for *Refine* and *Pivot*.

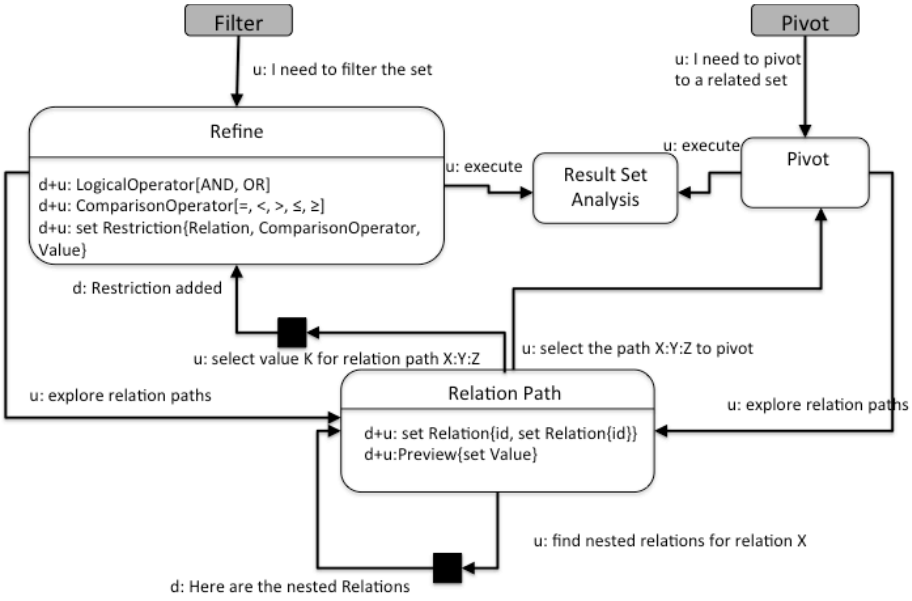

Figure 42 - XPlain interaction dialogue for combinations of Pivot and Refine

In the “Refine” scene, the user can, not only define values for relations, but also logical connectors and comparison operators. In order to define a restriction, the user asks the designer’s deputy to present the relations that apply to the items of the set under refinement, which causes a transition to the “Relation Path” scene. Within this scene, the user can both ask for nested relations (relations of the related items) and get a preview of the related items. When the desired relation and the filtering value have been found, the user asks for the designer’s deputy to add the restriction to the set of restrictions. When the user feels satisfied with the restrictions, she/he asks the designer’s deputy to execute the refinement, which leads the dialogue to the “Result Set Analysis” scene. The *Pivot* definition

depends only on the selection of a relation path, which is defined likewise the *Refine* operation.

The main goal of the presented conversational models is to describe the relationships between the operations from the interaction aspect. Although the models abstract previously described concerns, such as the order for binding definitions and specific dialogues for different types of arguments, they can be accurately described in the MoLIC scene-detailing phase, where the conversations within each scene are presented in detail.

The interaction/interface design issues presented for the specification of exploration operations are:

1. The ordering for the specification of bindings for each operation.
2. The interaction required for specifying bindings for each parameter, depending on the parameter type and the operation. Some possibilities discussed were: interface selection, textual inputs, filterable selects, computable specification for auxiliary functions, and navigation through relation paths for relation parameters.
3. The possibility of modeling differentiated interactions for specific combinations of operators, such as for combinations of *Refine* and *Pivot*, and for compositions of *Refine, Intersect, and Unite* that can be modeled as a faceted search interaction.

We emphasize that the goal here is not to determine which design option is better. There are many variables involved and proper user studies should be carried out for a final answer. We demonstrate, though, how the proposed separation of concerns approach, based on a formal functional layer, can leverage the definition of the scope of the interaction design space, and also leverage comparisons independently of the functional aspects.

7.2.3.Requirement 3: Exploration trail management and browsing

It has been recognized that exploration tools should allow the user to visualize the history of the exploration actions (WHITE; ROTH, 2009). The functional layer defines relationships between result sets, where the result set of a previous action can serve as the input for the next. Hence, the design issues at this point are how to present the exploration trail and how to allow its manipulation.

Some interface options for visualizing the exploration trail are lines, trees or graphs. For example, Figure 43 presents line and tree options.

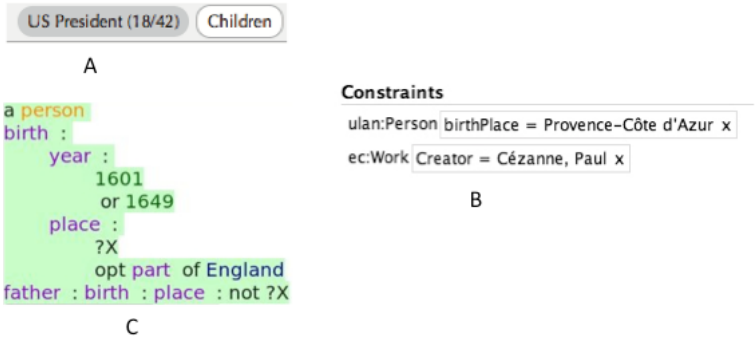

Figure 43 - Linear implementations of Parallax (A) (HUYNH; KARGER, 2009) and /facet (B) (HILDEBRAND; OSSENBRUGGEN; HARDMAN, 2006), and tree view of Sewelis (C) (FERRÉ; HERMANN, 2012)

Figure 43A presents a linear trail for pivoting actions in Parallax, Figure 43B presents a linear trail of refinement constraints in /facet, and Figure 43C presents a trail as a tree view implemented in Sewelis, which shows both pivoting actions and refinement constraints in the same view.

Linear representations, although simple, lack semantics for the visualization of branching actions, such as the parallel constraints in Parallel Faceted Browser. For these cases, tree and graph representations are more appropriate, since the exploration trail can become very complex. Tree representations also have the advantage of allowing the user to collapse or expand the branches, which may be a good option considering the "details-on-demand" rule of the information seeking mantra (SHNEIDERMAN, 1996). However, since the functional layer presents operations that receive two sets as input (e.g., unite, intersect, and diff), tree representations present a drawback because the result set of these operations must be repeated in two branches. In the tree representation is not easy to perceive these join nodes, i.e., sets resulting from combinations of two input sets. For example, the variable "?X" in Figure 43C is a way to refer to a specific result set in different points of the task and allow such operations in Sewelis. The "not ?X" restriction in Figure 43C is a join node between the branches.

In XPlain we choose a graph representations in order to enhance the perception of join nodes. For the following examples, consider the case of a user reviewing a paper. One revision strategy is to find relevant papers of the same area of the reviewing paper that were not referenced. Figure 44 shows an

exploration trail example for the case study of “finding relevant and not cited papers”. The join node is the set difference operation.

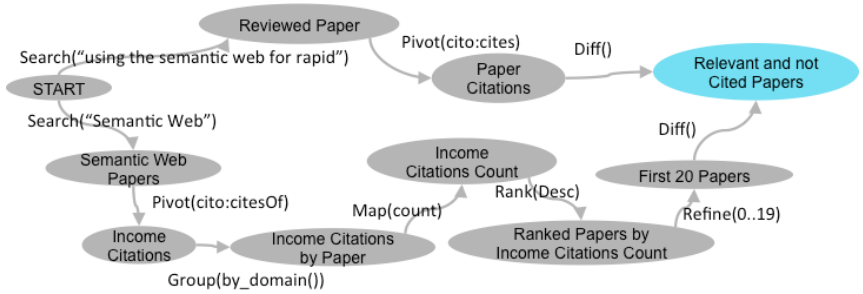

Figure 44 - Graph representation of the functional composition for the task “finding relevant and not cited papers”.

The graph in Figure 44 is a visual representation of the sequence of operations applied along the exploration process, where, each node is a result set and the arrows represent the operations applied. The “START” node represents the whole dataset and the highlighted node “Relevant and not Cited Papers” is the result of the difference between the citations of the paper being reviewed and the top 20 most relevant papers of the Semantic Web area, according to the number of incoming citations.

The graph in Figure 44 is more than just a visual representation of the exploration trail - it can also be used as a first-class object, where the user can parameterize the operations and reevaluate dependent branches. For example, the user could replace the set “Semantic Web Papers” in the exploration trail in Figure 44 by a set of papers in another research field and reevaluate the entire branch, thus reusing an exploration trail for different papers of distinct research fields. In other words, it is possible to reapply strategy used to solve a task as represented by the exploration trail.

Once we recognize that an exploration is, in the end, also a function, the interaction issues for allowing the reevaluation of a functional composition become quite similar to the issues concerning the definition of bindings for the operations presented in the previous section. The additional step is to consider the union of bindings from all operations in the composition as bindings of the exploration. Therefore, the reevaluation of the functional composition requires the redefinition of one or many bindings of some operations. The interface could show the bindings and ask which ones must be replaced for the reevaluation. The

same design decisions adopted for the definition of bindings for each argument type also apply for the redefinition of bindings of functional compositions[14].

The interaction/interface design issues for exploration trail management and browsing are:

1. The visual encoding for the exploration trail, which includes both exploration sets and their dependencies;
2. Allow the user to browse the exploration sets from the nodes of the exploration trail;
3. Allow the user to access the bindings for specific operations/functional compositions;
4. Define the interaction for binding redefinitions and reevaluations from the exploration trail.

In summary, we have shown how separating the concerns of interaction/interface design from the operations of the functional layer, together with use of the functional layer as a guide for what the interface should provide for specific task contexts guides the discussions of interaction possibilities. Since the main concern of this chapter is to discuss the design space of exploration environments, in the light of the separation of concerns approach, the XPlain interface is one possible interface and interaction model for the functional layer that, even though it presents full expressivity, it may not be efficient for all exploration contexts and users.

[14] This feature is currently under development