6
Evaluation of Exploration Tools

The evaluation framework addresses two main aspects of the exploration. The tactical aspect, which describes the most common atomic operations, and the strategic aspect that describes composition patterns of tactics to solve complex information problems.

The tactical analysis can help the designer in assessing what can be accomplished in the scope of a single action. However, it lacks semantics for describing exploration strategies, which requires the description of the possible sequences of actions in time. Therefore, we characterize the possible exploration strategies allowed by the tool using strategy analyzes. The following procedure can be used to derive evaluations and comparisons of exploration tools:

1. For each Operation in the framework
   1.1. Analyze how many are covered through the interface;
   1.2. For each Parameter analyze the range of arguments that can be passed according the established criteria;
   1.3. List possible specializations
2. Build a context-free-grammar to analyze the possible expressions;
3. Derive assessments and possible improvements.

6.1.Tactical Analysis

We describe the tactical aspects in terms of the signatures of each operation, comprising the description of parameters and result sets. Thus, we devised a set of attributes to assess the extent of arguments that can be attributed to the parameters of each operation, based on the formal description of the parameters. We organized the tactical analysis as a decision tree containing questions and possible answers for the operators. Figure 20 shows the decision tree for tactical analysis.

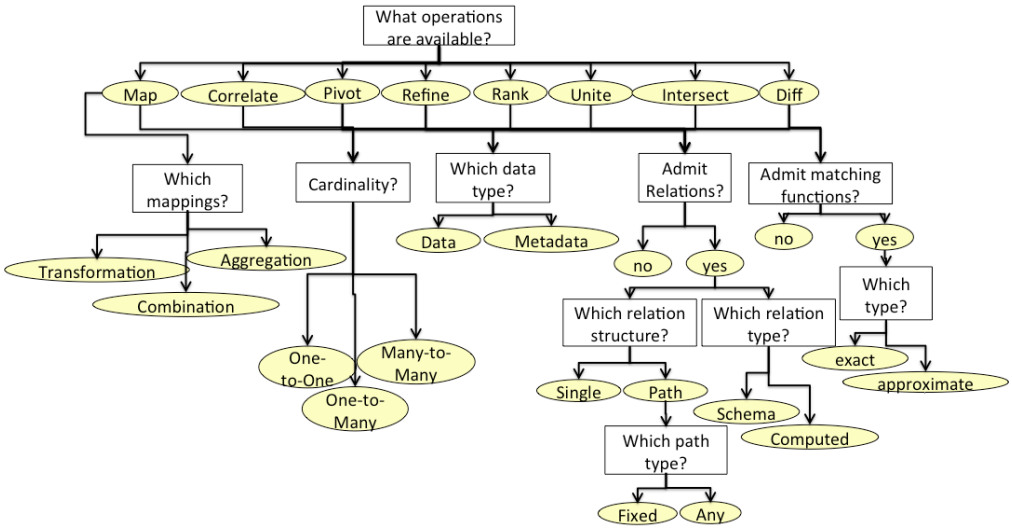


Figure 20 - Tactical Analysis Tree

The tactical analysis tree allows the assessment of what operations are available through the interface and which arguments they can receive, informing how they can be used in the exploration. The rectangles represent questions for each operation and the ellipses are the possible answers. The root of the tree is the main question "*What operations are available?*" and the possible answers are the framework operations. Next, there is a set of questions for each operation. As an example, some operations receive relations as arguments, such as *Pivot* and *Refine*. If the answer for the question "*Admit relations?*" is "*yes*", this leads to questions concerning both the relation structure, which can be a "*single"* relation or a relation "*path*", and the relation type, which can be a relation from the "*schema"* or a "*computed*" relation generated along the exploration process. The descriptions of the analysis questions and answers are as follows:

- *"Cardinality?":* characterizes whether the operation is a mapping from one item to another item ("*One-to-One"*), one item to many items ("*One-to-Many"*), or many items to many items ("*Many-to-Many"*);
- *"Which data Type?":* specifies whether the operation accepts "*data"* or "*metadata"* as input. Our model makes no difference between data and metadata, i.e., both types can be the input of exploration operations. The relevance of this characteristic is due to schema learning. As an example, consider a faceted search interface manipulating a dataset with hundreds of facets. If the facets could be ranked and refined by relevance, it could leverage learning for future steps.

- *"Which relation type?":* characterizes whether the operation accepts "*schema*" relations or "*computed"* relations, or both;
- *"Which relation structure?":* characterizes whether the operation accepts relation "*paths"* or only "*single"* relations. In case of relation paths, they can be *Any* or *Fixed*, where the former means that the user can use any property path of the dataset, while the latter informs that only fixed and preconfigured paths can be used.
- *"Admit matching functions?":* the refine operation receives matching predicates for items' comparisons that can be of two types: "*exact"* or "*approximate"*;
- *"Which mappings?":* This question characterizes the types of mapping functions available for *Map* operations. The possible types are "*aggregation"*, which maps a whole set of items onto a single aggregated value, "*combination"*, which combines two or more values, and "*transformation"*, which maps a set of items onto another equivalent set of items that are some kind of transformation of the original items. As examples of the latter, consider date format transformations and currency conversions.

As an example of tactical analysis, we choose gfacet. Figure 21 presents a gfacet screen with two interrelated sets of items.

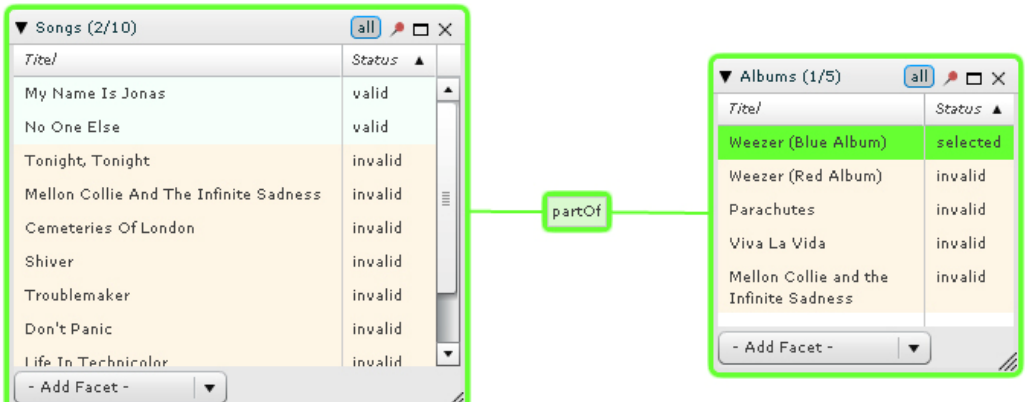


Figure 21 - pivot and refine in gfacet (HEIM; ZIEGLER; LOHMANN, 2008)

In gfacet the user starts with a keyword search and the result set is added to the workspace. Then, the user can select a facet, i.e., a relation whose domain is the items in the set, and pivot. The new set and a relation are also added to the workspace. Once having obtained many interrelated sets, the user can select an item from one set and gfacet filters the items that share the relation with the

selected item. In Figure 21 the user filters all items that are part of the album "Weezer (Blue Album)". The main feature in gfacet is the ability to filter sets separated by two or more relations. Figure 22 shows the tactical map of gfacet.

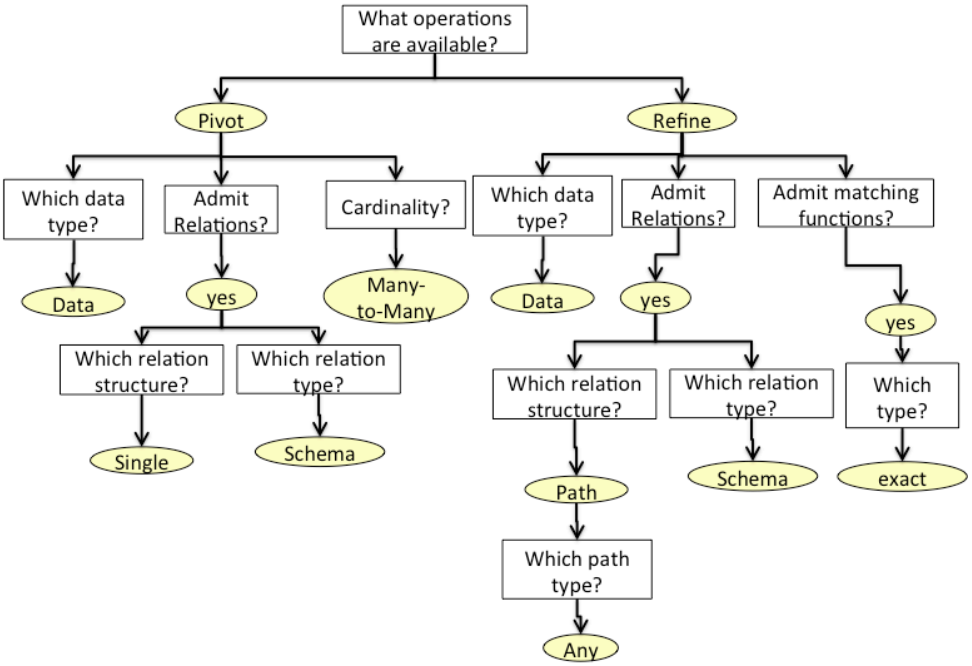


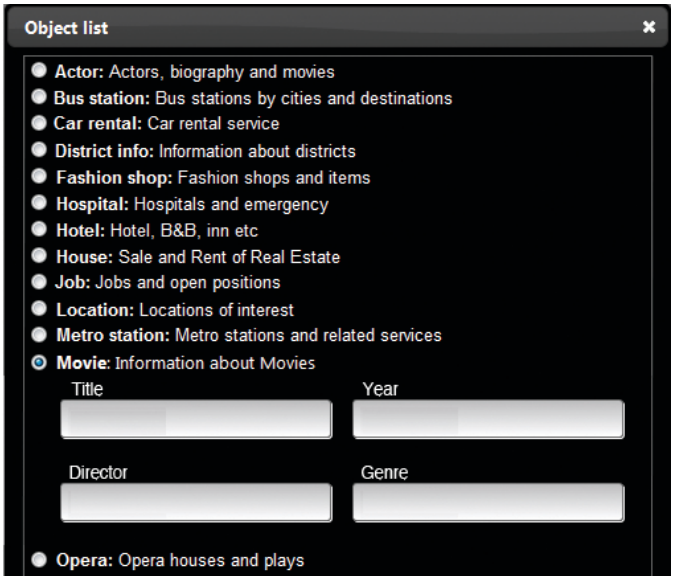
Figure 22 – gfacet tactical map

From the tactical tree we can draw some conclusions:

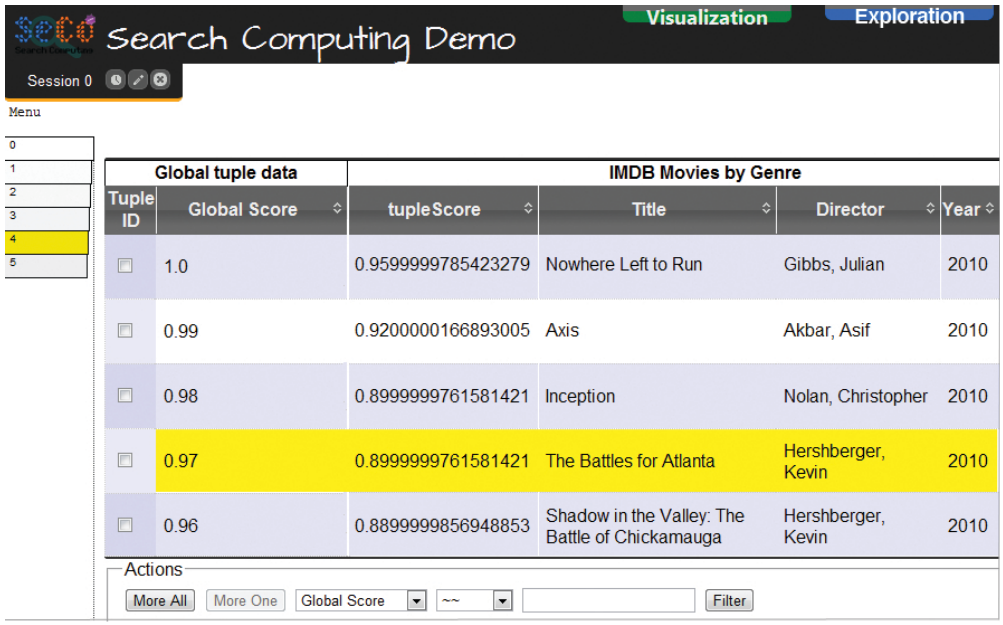
- Only two operators are available: *Pivot* and *Refine*;
- Pivot's cardinality is "*Many-to-Many*", hence, it is possible to navigate from multiple items to multiple related items;
- Since the relation structure of the Pivot is "*single"*, there is no possibility of pivoting through relation paths;
- Since the relation structure of the refine operation is a "*path"*, it is possible to filter items separated by two or more relations, i.e., indirectly related;
- It is not possible to refine using similarity measures, such as string distance, since the value of the match type attribute is "*exact"*, for the *Refine* operation;
- gfacet does not allow manipulation of metadata, which may hinder schema learning.

Now we can use the same framework to compare gfacet against other tools. As an example, we analyze the tactical map of the Search Computing tool (SeCo) (BOZZON, A *et al.*, 2013).

The search computing tool (SeCo) is an exploration tool based on formal specifications. Two aspects make SeCo an interesting case of analysis. First, SeCo design was guided by Kuhlthau's cognitive model (KUHLTHAU, 1991), which is a well-known episodic model for describing exploration behaviors from the cognitive point of view. Second, SeCo presents a formal model of exploration operations, represented as the exploration query language SeCoQL. Figure 23 shows screenshots of SeCo.



(A)



(B)

Figure 23 - **(A)** Refinement of items by type and relations (movies by Title, Year, Director, or Genre). **(B)** Refined movies and relations presented in a tabular format.

In SeCo, the user can start by refining the items s/he wants to explore. The results are presented as a table that can be further processed by exploration operations. The user can add related items by selecting relations through the expansion mechanism. In Figure 24 the user selects related theaters and restaurants to expand the current table. Moreover, the rows can also be ranked and grouped.

IMDB Movies by Genre			Google Theatre by Addr					Yelp Restaurant		
Title	Director	Year	tupleScore	Name	Address	City	Start Time	tupleScore	Name	Address
Nowhere Left to Run	Gibbs, Julian	2010	1.0	Symphony Space	2537 Broadway	New York	8:00pm	1.0	Doaba Deli	945 Columbus Ave
Axis	Akbar, Asif	2010	0.99	Symphony Space	2537 Broadway	New York	6:00pm	0.99	Gandhi Fine Indian Cuisine	2032 Bedford Ave
Inception	Nolan, Christopher	2010	1.0	Symphony Space	2537 Broadway	New York	8:00pm	0.98	NY Dosas	50 Washington Sq S
The Battles for Atlanta	Hershberger, Kevin	2010	0.97	AMC Empire 25	234 West 42nd	New York	9.15 11.10am 12.30	0.97	Punjabi Grocery & Deli	114 E 1st St
Shadow in the Valley: The Battle of Chickamauga	Hershberger, Kevin	2010	0.98	Anthology Film Archives	32 Second Avenue	New York	4:00pm	0.96	Neerob	2109 Starling Ave
Sweet Science	Howell, Chris	2010	0.97	Anthology Film Archives	32 Second Avenue	New York	4:15pm	0.95	Dil-e Punjab Deli	170 9th Ave
15 Till Midnight	Meyer, Wolfgang	2010	0.98	Anthology Film Archives	32 Second Avenue	New York	4:00pm	0.94	King of Tandoor	600 Flatbush Ave
Double Negative	Johnson, Jesse	2010	0.95	Pavilion Cinema	188 Prospect Park	Brooklyn	7:20 9:40pm	0.9299999999999999	Shalom Bombay	344 Lexington Ave
GR30k	Falicki, Daniel	2010	1.0	Symphony Space	2537 Broadway	New York	8:00pm	0.92	Sapthagiri	804 Newark Ave
Sinners & Saints	Kaufman, William	2010	0.98	Bow Tie Cinemas	1450 East Avenue	Bronx	1:40 4:20 7:10	0.91	Vatan Indian Vegetarian	409 3rd Ave

Figure 24 - Movies table expanded with theaters and restaurants

Figure 25 shows the tactical map of the search computing.

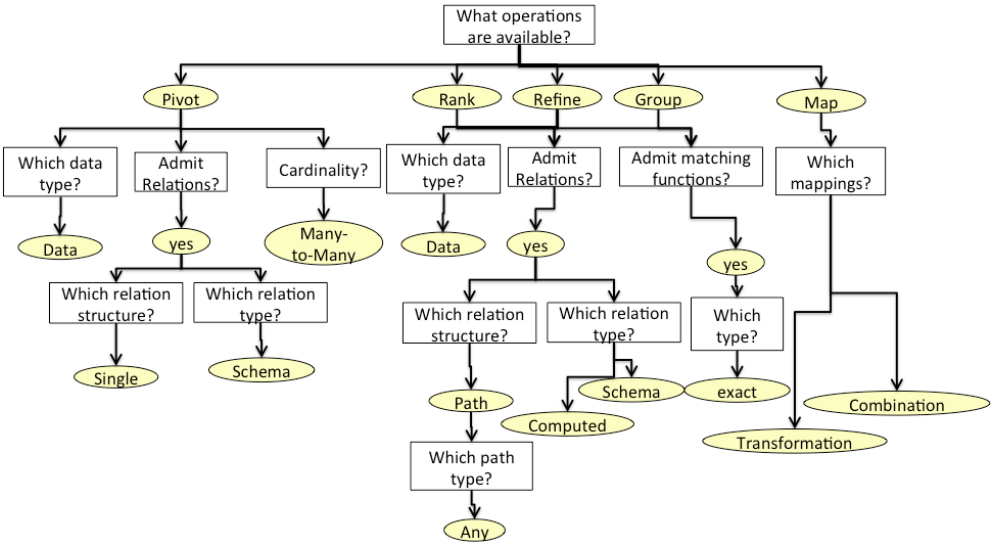


Figure 25 - Tactical map of SeCo

The conclusions we can draw from the map are the following:

- SeCo has a more complete set of operations than gfacet: *Pivot, Refine, Group, Rank*, and *Map*, noting that *Map* is uncommon among exploration tools.
- Despite the big difference in result set presentation between SeCo and gfacet (tabular vs graph based), the *Pivot* operation has the same characteristics in both - it does not admit relation paths and are restricted to schema relations.
- There is a difference in the relation type of the *Refine* operation. In SeCo, *Refine* accepts, not only schema, but also computed relations. This is due to the possibility of creating new columns, not provided by the schema, and using their values in filtering actions.
- SeCo allows mappings to combined fields based on two or more fields, such as "TotalPrice = Theatre.Price + Restaurant.AvgPrice" (BOZZON, A *et al.*, 2013). However, as far as we could determine, it is not possible to map using aggregation functions.

Faceted search interfaces all share many tactical aspects. Figure 26 presents a unified tactical tree of the faceted search interfaces analyzed, i.e., each interface is a subtree of Figure 26.

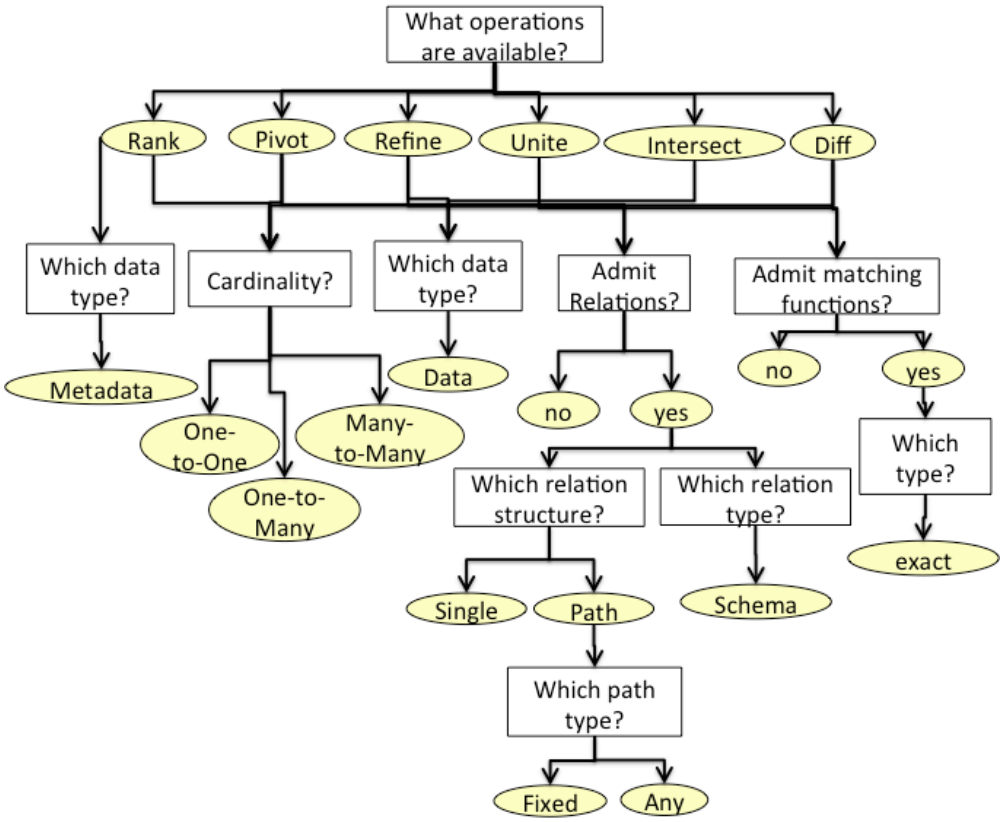


Figure 26 - Tactical map of faceted search tools

The first tool featuring the faceted search paradigm was Flamenco (HEARST *et al.*, 2002). Flamenco was restricted to *Refine* operations. Refinements were possible only through single relations and fixed paths, described by the branch "*Which relation structure?→Path→Which path type? →Fixed"* in the tactical tree. Many refinements could be nested, hence forming the conjunction of filters. Since *Pivot* was not available, the focus of the exploration was restricted to sequences of refinements over single type of item e.g. architectural images or Nobel Prize winners.

Later tools augmented the expressivity of *Refine* by allowing refinements over any relation path, described by the branch "*Path→Which path type? →Any"*. As an example, BrowseRDF allows the user to filter items based on properties of related items, such as, filtering "publications by venue's release year". A shortcoming of those tools is the impossibility of filtering different sets of items, such as changing from the set of publications to the set of venues and filter by release year and areas of interest, or change to the set of authors and filter by name. Therefore, tasks requiring manipulation of different types of items, carried out on heterogeneous datasets, were not possible in these tools under the same

configuration. Therefore, the *Pivot* operation was introduced to allow the user to change the filtering focus, for example, from a set of publications to a set of authors or publication venues. BrowseRDF, /facet, gfacet, and Rhizomer are examples allowing the coexistence of *Pivot* and *Refine* in faceted search. These tools only allow pivoting over single relations, described by the branch "*Which Relation Structure?→Single"*. More recent faceted tools, such as SemFacets and Sewelis, have included pivoting over relation path, captured in the branch *"Which Relation Structure→Path→Which path type?→Any"*.

Due to the increasing size and heterogeneity of public datasets available, *Rank* and *Refine* over metadata were introduced in some faceted interfaces, thus allowing the user to better organize big amounts of data types and dimensions. Facets could be ranked by some relevance measure, such as popularity or entropy (COHEN; SCHWABE, 2012), and also be the target of a keyword refinement (branch "*Rank→Which data type?→Metadata"*).

The *Unite* operation was also included to allow the expression of disjunctive filters. Henceforth, queries such as "authors whose birthplace is **UK or USA**" became possible.

The tactical map is helpful to analyze the operations set and their individual applications. However, it lacks expressivity for capturing the interactions between the operators and the possible compositions that can be formed during the exploration. We fill this gap with the strategic analysis of the operators.

6.2.Strategic Analysis

The strategic analysis step aims at assessing the expressiveness of the tool from the point of view of the range of strategies the user can employ to solve exploration problems. The range of different functional compositions that is allowed by the tool when we abstract interface and interaction details defines the possible strategies. Once the operations framework is defined, we represent the dependencies between the operations using context free grammars, enabling analyses such as which operations can be the input of another.

6.2.1.Notational Conventions

- **Nonterminal symbols:** we use capital letters for non-terminal symbols: S, R, O, etc.
- **Terminal symbols:** terminal symbols are operation names represented in lowercase and Italic: *intersect(), unite(), refine()*. Let "*s0"* be a terminal representing the starting point or a dataset reference.
- **Production rules:** defined as non-terminal-symbol replacements. Consider the following examples and some possible derivations (sentences in the grammar), respectively:
    - R → *refine*(R) | *s0*
        - *refine*(*refine*(*s0*))
    - S → *refine*(S) | *pivot*(S) | *s0*
        - *refine(pivot(s0))*
        - *refine(pivot(pivot(s0)))*
- **Operation arguments:** even though we use filters and relations to leverage the understanding of the examples, for the sake of simplicity, the grammars are described only in terms of the input arguments of the operations. Other types of arguments, such as refine filters or pivot relations are abstracted.
- **Refinements through compositions:** some faceted systems allow the user to pivot multiple times, using schema relations, and apply a restriction at some point. This restriction is also applied to the previous sets. Let *s0* be a set of publications. The following composition illustrates this case:

refine(pivot(pivot(s0, :Author), :BirthPlace), equals(:PartOf, "Brazil"))

The composition above expresses a user navigating from *s0* to the set of authors, and then, to their birthplaces. The birthplaces are restricted to those being a location in Brazil. In order to also restrict the set of authors born in the filtered birthplaces and, by propagation, the initial set of papers, we use the symbol "!" appended to the refine operation. Therefore, the following notation represents a back propagation of the birthplace restriction to all intermediary sets:

refine(pivot(pivot(s0, :Author), :BirthPlace), equals(:PartOf, "Brazil"))!

- **Branching:** some tools, such as gfacet and Parallel Faceted Browser, allow repeated applications of exploration operations over the same set of items, where the results of an application does not interfere in the results of the next, i.e., the applications are independent from each other. Consider the case of a user applying two independent refinements to a hypothetical set of publications *P* in our framework:
  - $P_1 \leftarrow P.Refine(equals(:Author, a1))$
  - $P_2 \leftarrow P.Refine(equals(:Author, a2))$

  Figure 27 shows a graphical view of the applications.

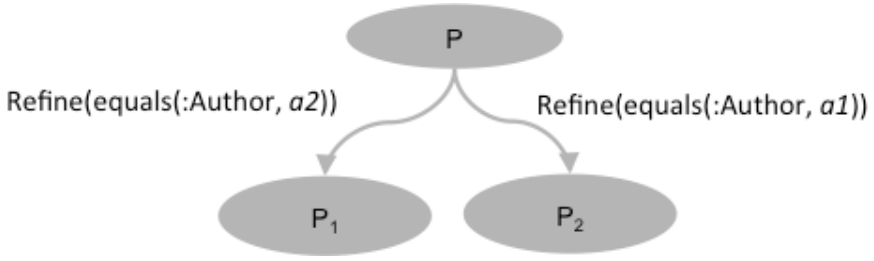


Figure 27 – Independent refinement executions.

The set P_1 contains publications of the author *a1* and the set P_2 contains the publications of the author *a2*. The application of the second refinement is independent of the application of the first refinement since both refinements have the set *P* as input. Moreover, both P_1 and P_2 are accessible through the interface for further analyzes. We introduce the operator *branch(inputExp* Exp_1, Exp_2*)* in the grammar to cover these cases. *branch(InputExp,* Exp_1, Exp_2*)* receives an input expression *InputExp* and two expressions Exp_1 and Exp_2, and returns two independent results. The input of both Exp_1 and Exp_2 is the result set of *InputExp*. Let the terminal *irs* denotes the result set of the input expression. The refinements of Figure 27 can be expressed in the following way:

branch(P, refine(irs), refine(irs))

The branching operator should not be confused with the set operators *unite, intersect,* and *diff*. The set operators receives two input expressions and return a single result set that is a combination of the two inputs.

6.2.2.Evaluation

In order to test our evaluation method, we choose a set of 13 faceted search systems and show how we can categorize them in terms of their strategy grammars. We organize them in increasing order of expressiveness.

For each grammar we present task examples and an assessment. Consider the following hypothetical dataset for the examples:

Let *s0* be a dataset of publications having the relations *:Author = {<p1, a1>, <p1, a2>, <p2, a3>}, :Year = {<p1, 2002>, <p2, 2005>, <p3, 2007>}, :Venue = {<p1, ISWC>, <p2, ISWC>, <p3, WWW>}*. The description of the grammars and the examples are as follows:

Grammar: faceted search version 1

The following grammar is shared by Flamenco, mspace, and most of the e-commerce websites:

S → *refine*(S | *s0*)

- **Example:** find papers authored by author *a1* that were published in ISWC in 2002:

```
refine(
        refine(
                refine(s0, equals(:Year, 2002)),
                equals(:Venue, ISWC)
        ),
        equals(:Author, a1)
)
```

- **Assessments:** The grammar of the most traditional faceted tools is restricted to sequences of applications of the *refine* operation. Thus, the task is described in terms of multiple intersections of refinements. The exploration language, therefore, is very limited since disjunctive restrictions are not allowed. Moreover, it is impossible to employ any kind of browsing due to the absence of pivoting.
- **Difficult tasks to Solve:**

- **Disjunctive Filtering:** e.g. find papers published in ISWC or ESWC;
- **Comparisons:** e.g. trace similarities and differences of the publication venues of two researchers;
- **Browsing hierarchies of concepts:** e.g. discover and describe all knowledge areas that are sub-areas of "Semantic Web" area.

Some variations of this grammar include branching, which allows the user to apply independent restrictions on a set of items for comparison purposes, as it happens, for example in Parallel Faceted Browser:

S → *branch*(s0, S, S) | R

R → *refine(*R | *s0*)

With the *branch* operation, we can apply two independent restrictions and draw comparisons more easily since the two sets will be available through the interface. Discovering similarities and differences between two sets of items can be very difficult without set operations. However, for small sets, if the tool allows branching, the user could generate two sets and draw the comparisons by visualizing the sets.

For example, let *s0={ISWC, ESWC, WWW, KESW}* be a set of venues, *:VenueOf = {<ISWC, p1>, <ESWC, p2>, <WWW, p3>, <KESW, p4>}* be a relation between venues and papers, and the relation *:Author = {<p1, a1>, <p2, a1>, <p2, a2>, <p3, a2>, <p4, a2>}* be a relation between the publications and their respective authors. The task of comparing the publication venues of two researchers, without set operations, can be solved more easily in the following way:

```
branch(s0,
        refine(irs, equals(:VenueOf:Author, a1)),
        refine(irs, equals(:VenueOf:Author, a2))
)
```

Since the two result sets of venues (venues of *a1* and venues of *a2*) are small and will be accessible through the interface, the user could draw the comparison more easily. Without branching, this task becomes more complicated since the user should: 1- apply the first restriction to obtain venues of *a1*; 2 - use

an external tool to record the venues; 3 - remove the restriction for *a1*; 4 - apply a new restriction to obtain the venues of *a2,* and finally draw the comparisons with the recorded venues of *a1*.

Grammar: faceted search version 2

The following grammar describes the possible strategies in /facet, tfacet, gfacet, and Rhizomer.

S → P | R | *branch*(S, P, P)

R → *refine(R | P | s0)*!

P → *pivot(R | s0 | P)*

Consider the following task examples that illustrate the grammar derivations:

- **Task 1:** find authors of papers published in ISWC in 2002:

```
pivot(
        refine(
                refine(s0, equals(:Year, 2002)),
                equals(:Venue, ISWC)
        ),
        :Author
)
```

- **Task 2:** find papers whose authors' affiliations are PUC-Rio. Let *:Abbr* be the relation between affiliations and their abbreviated names:

```
refine(
        pivot(
                pivot(s0, :Author),
                :Affiliation
        ),
        equals(:Abbr, "PUC-Rio")
  )!
```

- **Assessment:**

The possibility of combining *Pivot* with *Refine* augmented considerably the expressivity of the tools. Now, we can browse through different data types and relations and apply restrictions to more than one type of item. In the task example

2, the user changed the focus from papers to authors, and then to affiliations (*pivot(pivot(s0, :Author), :Affiliation)*). Next, the set of affiliations was filtered to the one having "*PUC-Rio"* as abbreviated name, and the restriction was propagated to the intermediary sets as a result of the operator "!" described in the previous section.

Although the same result could be achieved through a single step just by restricting the property path *:Author:Affiliation:Abbr* to "*PUC-Rio"*, pivot is strongly related to browsing and schema learning. Consider a user that does not know the data schema. His/Her strategy can be to start by browsing through the data types and relations and, once the connection between *papers* and *affiliations* is learned, a refinement over the relation path can be applied. In exploration tasks, the user is less likely to take the shortest strategy due to the inherent uncertainty and lack of knowledge.

- **Difficult Tasks:**
  1. Disjunctive filtering
  2. Comparisons between large sets of items due to the absence of set operations.

A relevant fact to notice about the task 2 is that, although some tools feature the same grammar of operations, the refine operation cannot be propagated to the intermediary sets i.e. the operator "!" is not present, which makes this strategy unfeasible. This is the case of Parallax and Humboldt.

Some faceted interfaces also feature a union operation, allowing disjunctive filters. The grammar with the *Unite* operation is as follows:

S → P | R | *branch(S, P, P)*

R → *refine(R | P | s0)*! | *unite(R, R)*

P → *pivot(R | P | s0)*

Consider the following examples of tasks using the *Unite* operation.

- **Task:** find ISWC papers published on 2002 or 2005 by PUC-Rio authors:

```
refine(
        pivot(
                pivot(
                        union(
```

refine(s0, equals(:Year, 2002)),
refine(s0, equals(:Year, 2005))
),
:Author),
:Affiliation),
equals(:Abbr, "PUC-Rio"*)*
)!

Grammar: faceted search version 3

The most expressive faceted search environments, to the best of our knowledge, are Sewelis and SemFacet. They extended the grammar by including the set operations *Intersect* and *Unite* and allowing complex expression on both inputs of the *branch*, *unite* and *intersect* operations:

S → branch(S, S, S) | O| R |R! | P

O → *intersect*(S, S) | *unite*(S, S)

R → *refine(*O| R | P | s0*)*

P → *pivot(*O| R |P | s0*)*

Consider the following example of a complex intersection:

- **Task:** find papers published in ISWC or ESWC whose authors' affiliations are PUC-Rio or UFRJ:

```
intersect(
        union(
                refine(s0, equals(:Venue, ISWC)),
                refine(s0, equals(:Venue, ESWC))
        ),
        union(
                refine(s0, equals(:Author:Affiliation, PUC-Rio)),
                refine(s0, equals(:Author:Affiliation, UFRJ))
        )
)
```

- **Assessment:**

Considering faceted search environments, the version 3 grammar is the most expressive, allowing any combinations of the operators and branching. The

limitations of this grammar are related to the tactical aspects, since operations such as, *Rank*, *Correlate*, *Map*, and *Group,* are missing. Table 1 presents the summary of strategies allowed by faceted search environments.

Table 1 – Summary of tools' exploration grammars.

Tool	Grammar
Flamenco, Mspace, Faceted Wikipedia Search	S → *refine*(S | *s0*)
Parallel Faceted Browser	S → *branch*(s0, S, S) | R R → *refine(*R | *s0*)
Humboldt, Parallax	S → P | R R → *refine(P | s0)* | *intersect*(R , R) P → *pivot(R | s0 | P)*
/facet, gfacet, tfacet, Rhizomer, BrowseRDF	S → P | R | *branch*(S, P, P) R → *refine(R | P | s0)*! P → *pivot(R | P | s0)*
Sewellis, SemFacet	S → branch(S, S, S) | O| R |R! | P O → *intersect*(S, S) | *unite*(S, S) R → *refine(*O| R | P | s0*)* P → *pivot(*O| R |P | s0*)*

6.3.Evaluating Business Intelligence and Visualization tools

Our evaluation is not only useful to analyze faceted search systems but also visualization and business intelligence tools. In order to demonstrate this possibility we evaluate Tableau[12], which is a leading reference in the commercial business intelligence and data exploration tool market. It presents a rich set of operators, covering the majority of the exploration tactics that we defined in our framework.

[12] http://www.tableau.com/

Tableau is a visual processing tool, where data can be combined, aggregated, and refined among multiple dimensions. Figure 28 shows an example of the amount of sales aggregated by country and customers' type ("Corporate" or "Personal").

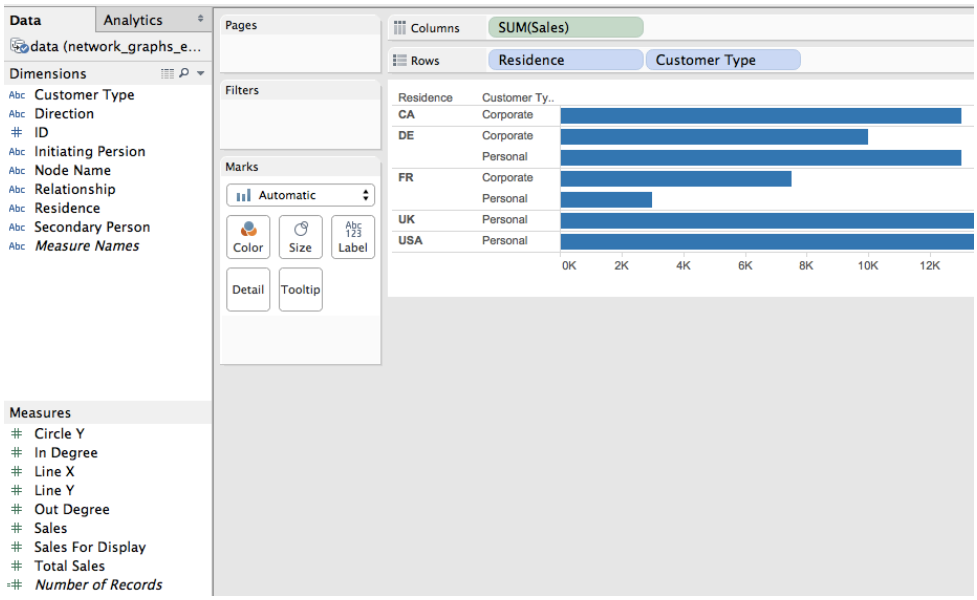


Figure 28 – Tableau's main screen

Tableau basically allows visual data analysis by dragging and dropping dimensions and measures to visualization panels. In Tableau "Dimensions" are literal data attributes and "Measures" are numerical fields that can be the input of aggregation functions, such as sum, mean, count, etc. When the user drags a measure to the row or column position of the visualization panel, the values are aggregated over the chosen dimensions. We interpret this feature as a composition of *Group* and *Map* exploration operations.

In Tableau, the user starts the exploration by loading the data from a data source and dragging a table to the view, such as the one presented in Figure 29A. From the starting view, the user can filter rows and/or join the table with related ones for browsing, as Figure 29B shows. The pivoting is carried when the user drags a related table and Tableau adds the join results to the table view.

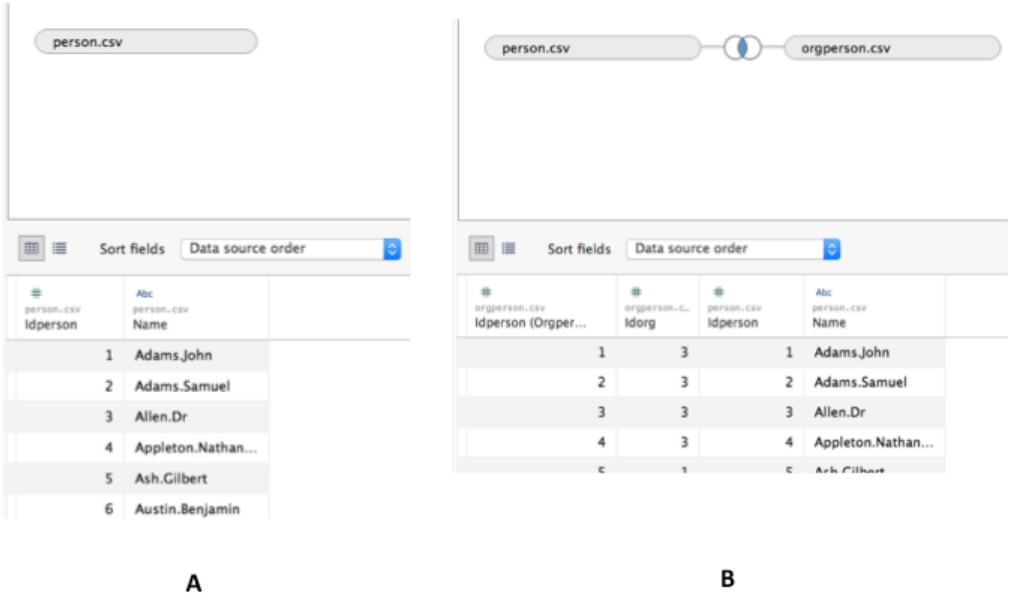


Figure 29 - Tableau's initial table view presenting the table *Person* (A); A join between the tables *Person* and *OrgPerson*.

It is also possible to refine the data by dimensions' values, as Figure 30 shows. By dragging a dimension to the "Filter" panel, the user can select filtering values and the visualization will be refined accordingly. Tableau also allows the creation of computed fields as combinations or transformations of fields in the original dataset. Computed fields can be used in the visualization and filtering panels in the same way.

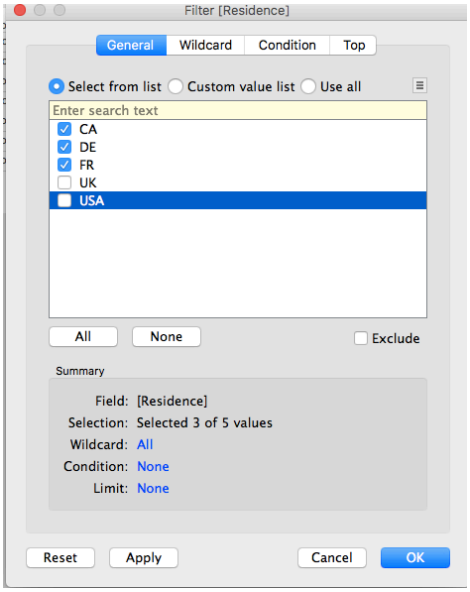


Figure 30 – Filters definition view in Tableau

The particularities on expressivity, though, are captured by a deeper analysis on the available tactics. The tactical map of Tableau is presented in Figure 31.

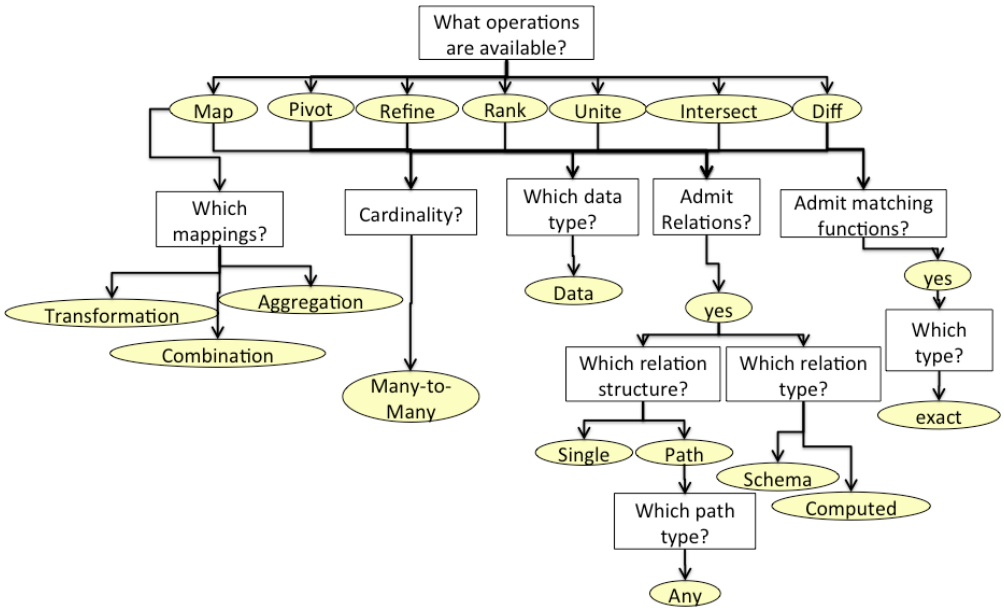


Figure 31 – Tableau's tactical map

The assessments that can be drawn from Tableau's tactical map are as follows:

- *Group*: although the grouping action in tableau is quite expressive since it is possible to group by multiple relations, which can come from the data model, be computed or a property path, it is not possible to group by approximated relations, such as distance relations for clustering;
- *Refine*: the matching predicates are only exact comparisons. Filtering by some sort of similarity measure is not possible;
- *Correlate*: there is not a correlation operator in Tableau. Therefore, discovering relationships between two sets of items is not possible. This operation could be useful when the user needs, for example, to discover join possibilities between two tables;
- The operations can only be applied to data items. Therefore, ranking or filtering meta-data items, such as relations, is not possible.

From Tableau's tactical map we can assess the tactical support to exploration tasks but it says nothing about how the tactics can be composed in order to form solution strategies. In order to do that we build a grammar that

precisely specifies the possible tactical compositions that can be achieved in Tableau. Tableau's exploration language is as follows:

S→ P | R | A

P→ *pivot*(P| R | s0)

R→ *refine*(P | R | s0)

A→*group*(A | s0) | *map*(A | s0) | *rank*(A) | *branch*(A, A, A) | *refine*(A) | T

T → U | I | D

U → *unite*(*refine(*A*), refine(*A*))*

I → *intersect*(*refine*(A)*, refine(*A*)*)

D → *diff*(*refine(*A), *refine(*A))

From Tableau's grammar we can identify some particularities. Since "s0" stands for the starting point, the user can start the exploration in four ways: by pivoting rows to a related set of rows using join keys (Figure 29B); by grouping items among dimensions, accomplished by dragging and dropping dimensions to the visualization panel; by refining the data or combining the refinement results with set operations; by mapping original fields onto computed fields.

The pivoting action in Tableau can only be carried out over the results of another Pivot or a Refine operation. These sequences of Pivot and Refine are carried out on the initial table view (Figure 29) as a data preparation step for the remaining operations that would be carried out in the aggregation view (Figure 28). When the exploration achieves the rule A of the grammar, the pivoting actions will not be available. Nonetheless, the user still can join tables for aggregation purposes (combinations of *Group* and *Map*)

The branching operation in Tableau is possible by creating new sheets from the actual state. By doing this, the user can apply new operations in the new sheets without altering the previous one.

An expressivity limitation in Tableau concerns the set operations. The union, intersection, and difference operations can only be applied to the results of refinements. Therefore, computing intersections and differences between distinct tables or groups for comparison purposes is not possible. Another restriction is the impossibility of applying disjunctions of filters on different relations and

conjunctions of filters over values of the same relation, such as the following derivations:

intersect(
    refine(*equals*(:Author, :Tim_Berners-Lee)),
    refine(*equals*(:Author, :James_Hendler))
)
unite(
    refine(*equals*(:fatherOf, :Albert_Einstein)),
    refine(*equals*(:GrandfatherOf, :Albert_Einstein))
)

In order to validate our evaluation method, we discussed a previous version in the work (NUNES; SCHWABE, 2016), where a comparison between Tableau and SeCo was presented. Moreover we discussed the evaluation of SeCo with their authors that confirmed the highlighted issues.