Adaptable Filter Graphs – Towards Highly-Configurable Query Visualizations

Florian Haag, Michael Raschke, Thomas Ertl

Visualization and Interactive Systems Institute (VIS)
University of Stuttgart
Universitätsstraße 38
70569 Stuttgart
{Florian.Haag, Michael.Raschke, Thomas.Ertl}@vis.uni-stuttgart.de

Abstract: Systems for browsing and finding information in the vast amount of data available on the web keep growing in complexity. To alleviate this complexity for queries, various visualization techniques have been developed in the past. We have examined the filter/flow concept. This paper presents a transformation schema for filter/flow graphs to make them even more intuitively usable and to support distributed collaborative use. For that schema, several possibilities for transformating the visualization and various factors that may influence that transformation are discussed. Finally, the schema is further explained based on a trip planning example scenario.

1 Introduction

The huge amount of information available in databases and on the web poses difficulties to retrieval systems. We are going to present our work in the context of a public transportation scenario, where end-users are often confronted with information retrieval tasks. Current interfaces for these tasks are not sufficient as they lack the flexibility to state complex queries.

In order to narrow down the number of results as much as possible, the users' understanding of queries can be improved by query visualizations. We have examined a variety of concepts for query visualization, which will be described in the Related Work section. It is noteworthy that most of these visualizations require the query to be expressed in a certain form (for example in CNF or DNF), or require some time to learn what the various shapes used in the visualization mean.

The filter/flow concept was presented in several early publications [YS93, Shn94]. An evaluation by Young and Shneiderman has indicated that users without any experience in boolean logic can understand the approach well [YS93]. Therefore, it is suitable for our scenario with untrained end-users and we have chosen the concept as a base of our work.

Filter/flow graphs are connected directed graphs that represent a query (usually based on boolean terms) and that have a defined start and end node. The nodes of the graph stand for atomic terms in the query. Each node may block data items that do not match the condition expressed by the term. Therefore, the nodes are called *filters*. The edges of the graph represent the flow of data.

The graph structure is based on the conjunctions and disjunctions that appear in the respective query expression. Hence, users can follow the edges starting at an initial node to the last node and thereby determine whether a given data item would reach and pass the last filter using any combination of subsequent edges (cf. Figure 1). The full dataset or single exemplary data items can enhance filter/flow visualizations by displaying a preview of the filtering results—for example, the number of intermediary results can be mapped to the thickness of the edges—, but as an additional benefit, the concept can also be used to visualize reusable query templates for future execution.

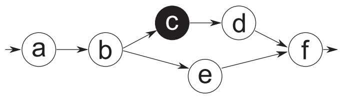

Figure 1: A visualization of boolean expression $a \wedge b \wedge ((\neg c \wedge d) \vee e) \wedge f$ according to the filter/flow concept as described in [YS93]: Boolean terms are represented by nodes, conjunctions as sequences of nodes and disjunctions as alternative paths. Negated terms are displayed in inverse colors. Starting on the left, users can move along the edges and check for each filter node on the path whether a given data item would be filtered out. If the right side of the graph can be reached with at least one path, the data item satisfies the boolean expression.

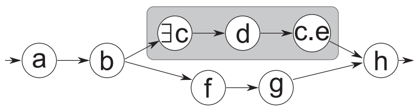

Figure 2: A visualization of the predicate logical expression $a \wedge b \wedge ((\exists c : d \wedge c.e) \vee (f \wedge g)) \wedge h$ according to the filter/flow concept: The scope of the existential quantifier is marked with a shaded background. Within that scope, one of the filters (node *d*) is independent of the quantifier, the other one (node *c.e*) depends on the entity bound by the quantifier.

As explained below, we have identified various difficulties when using basic filter/flow graphs with generic nodes. We hope to improve these by adapting the graph according to the situation it is used in.

We have examined various works that describe extensions of the filter/flow concept (which will be discussed in the Related Work section). However, each of them extended the concept into a pre-determined direction. They did not focus on the general adaptability of filter/flow graphs or parts thereof.

Our work aims to fill in these gaps by proposing a flexible transformation schema for filter/flow graphs that is not limited to, but especially suited for data found on the web. Based on this schema, filter/flow graphs can be transformed into what we believe is a more easily usable form of the filter/flow graph. This schema can be applied to a filter/flow graph before it is displayed to a user, thus taking advantage of any special requirements and preferences of the particular user and the device in use.

2 Related Work

Various concepts have been proposed in the field of query visualizations, for example Venn diagrams [Jon98], tables [HC03] or lists [PM00]. Also, approaches such as InfoCrystal [Spo93] sport less conventional basic looks. As explained above, these concepts require a transformation of the query itself into a standardized form (such as CNF or DNF), or require some effort to learn, so we have chosen to extend the filter/flow concept [YS93].

The filter/flow concept has also been extended in other works. Kaleidoquery [MPG98] introduces special node and edge types to visualize the full range of elements used in query expressions for relational databases. While doing so, the flow metaphor of the original filter/flow concept is partially abandoned, as some nodes have an effect on other upstream nodes. Morris et al. also construct a comprehensive filter/flow-based query language for a special use case and define special filter nodes for that purpose [MAEGJ04]. The mashup framework Yahoo! Pipes features a variety of filter nodes designed for handling news reports and similar list items, though the focus of most nodes is rather sorting and modifying items than filtering them [SHT$^+$07].

Lark [TIC09] and Facet-Streams [JGZ$^+$11] transfer the filter/flow concept to a tabletop display. In Lark, the flow represents a visualization pipeline with options at various abstraction levels, and Facet-Streams enrich filters with tangible objects.

Finally, FindFlow [HSMT06] and DataMeadow [EST07] show that the flow does not need to run in one fixed direction on the display. Instead, the nodes can be arbitrarily placed on a plane as long as the edges clearly indicate the direction of the flow. Also, FindFlow puts an emphasis on showing a list of intermediate results after every filter. DataMeadow uses starplots for (numeric) filter visualizations. Moreover, the set operations performed when reuniting two or more separate inbound paths can be manually selected.

These examples of extensions of the filter/flow concepts also show that the concept can be used in a wide range of scenarios ranging from geo-information systems [MAEGJ04] over employee databases [MPG98] to topic areas roughly related to trip planning, namely hotel search [JGZ$^+$11]. All of these works provide valuable insight into particular extensions of the filter/flow concept. However, none of them focuses on making the graphs adaptable.

As for the transformation, we were inspired by adaptation concepts for user interfaces in general. For example, various works adapt generated interfaces for web services based on the device context [HY07, MM02]. An approach for modeling such adaptive interfaces is also taken in the area of ubiquitous systems, for example, when interactions are adapted based on contextual information [SK11]. Similarly, behavioral data about users can be used to adapt a user interface to make it more usable for a particular user [MRM05]. Approaches such as [LVM$^+$04] and [RPK$^+$11] create an adaptable interface model from scratch rather than building upon an existing concept.

Furthermore, there has been some research on transformation schemas related to particular areas of interest, such as for blind users [RKE05] or for specific UI elements [Ric05]. Transformations based on rules have been described in more recent works such as [BFKJ08] or [HSS07], though they concentrate on graphical user interfaces in general rather than the particular subject of graph-based visualizations.

3 Problem Description

In public transportation, queries to retrieve travel itineraries are already commonplace, yet we can imagine that they can be further enhanced if we combine that timetable information with other information sources on the web, such as data about local businesses. These data sources need not be connected as long as they can be joined based on their semantic annotations. Consider an exemplary query by an imaginary traveler:

The traveler is looking for a train from A to B on the 1st of December. One of the trains that the traveler is staying on for at least 30 minutes must have a restaurant car. Alternatively, a stopover of at least 40 minutes at a station with a restaurant is required. The traveler needs a return trip on December 6. On the return trip, if there is a stop in C before 6 PM, a stopover of two hours is required, because in that case, the traveler would like to visit a museum whose last admission is at 6 PM. As a little, but frequent, complication, let us assume that the customer can be granted a special discount by her travel agent, if the trip from A to B takes at least 2 hours.

Seeing this as a query, we want to select a set of itineraries from the set of all possible itineraries. The selected itineraries must match the above constraints. The constraints are defined by a traveler and a travel agent, that is, by (at least) two users.

3.1 Conventional Solution

Nowadays' connection search facilities for public transportation—such as connection finders on websites of traffic companies—are mostly restricted to finding only connections between two or more places. Additional information (such as hotel offers near the destination or the carbon footprint) are sometimes available, but it is already impossible to specify a query where the destination depends on a dynamic factor such as weather. For example, on the website of the Networked Traveler project [Cal] or on the German railroad website [DB], the user is asked for the settings in a series of relatively static forms. A similar example is the TripPlanner^{+} of New York's MTA, which allows to indicate the maximum distance that may be bridged by foot, but has no possibility of making this setting dependent on the terrain (part of the town, roads with public stores or just private houses, ...) that has to be crossed [Met].

Let us imagine to specify the aforementioned query in such a system. The basic connection attributes such as departure and return dates, start and destination can be entered without any problems. A restaurant car in one of the trains might be made a requirement, although defining something like a custom minimum time on the respective train is usually not possible. Yet, considering a longer stopover at a location with a restaurant as an alternative to a restaurant car is a rather arbitrary requirement that is usually not supported by fixed query interfaces: The restaurant might be requested, but defining that and another arbitrary requirement as mutually exclusive is typically not possible. At the latest when it comes to specifying a very user-specific requirement like the optional prolonged stopover in C before a given time, the query cannot be precisely entered.

The result set will be unnecessarily huge. At the same time, it does not even include the discount yet. The travel agent might have an interface that applies generic discounts to offers for certain customers. However, the more complicated the conditions associated with a discount become, the harder it becomes for the travel agent to accurately specify the circumstances that activate the discount. Likewise, the special conditions that enable or disable the discount in her special case are difficult to show to the user.

3.2 Filter/Flow Solution

Let us imagine a connection search facility that lets users specify queries by using the filter/flow concept. As we are not looking at "flat" data objects but at data that may be hierarchically structured—an itinerary may contain various separate connections, a connection may comprise of several trains, and so on—pure boolean algebra will not suffice here. Therefore, we assume that the filter/flow expressions used in this system work on predicate logic. As shown in Figure 2, this is possible.

Figure 3 shows how this query could be represented using the filter/flow concept. Existential and universal quantifiers are used, as well as various filter nodes that represent basic comparison operations on some scalar data types. The query can be thought of as a query on the set of all possible itineraries.

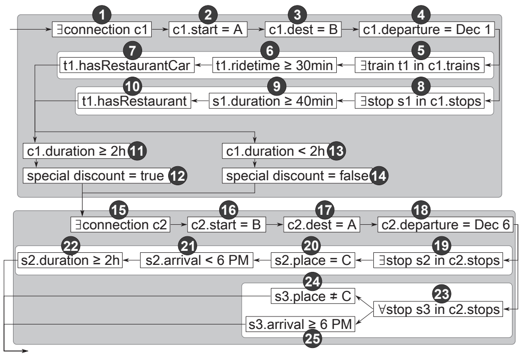

Figure 3: A filter/flow visualization of the example query presented in section 3.2: Connection (1) from A (2, 17) to B (3, 16) on December, 1 (4). Either, one of the trains in the connection (5) with a ride time of at least 30 minutes (6) should have a restaurant car (7), or one of the stopovers (8) of at least 40 minutes (9) should have a restaurant (10). Iff that connection takes at least 2 hours (11, 13), a special discount is applied (12, 14). On the return trip on December, 6 (15 to 18), if there is a stop (19, 23) in C (20, 24) before 6 PM (21, 25), the traveler wants to stay there for 2 hours (22).

By tracing the arrows, a user might be able to understand why a given connection (that she expected to be valid) would be ruled out by the visual query as it is structured. Possibly, the user might even be able to adapt the filter/flow graph in a way to improve the correlation between the actual results and the user's expectation. However, this requires a high degree of concentration and it is easy to get lost in the graph.

In order to identify shortcomings of the filter/flow visualization as it is, we will examine the filter nodes step by step.

The first four filters define some basic properties of the requested itinerary, namely the point of origin, the destination, and the date of departure. This information almost always has to be present when requesting an itinerary, yet the user already has to get an overview over four distinct nodes. Moreover, the point of origin and the destination have to be input again in filters 16 and 17 for the return trip. The connection from A to B and the connection from B to A are two distinct entities. Therefore, as long as only comparison nodes for basic data types are available, the two filters cannot be combined.

Note that, as a basic data type agnostic to the use case of the querying system, the location can only be specified as a text. Even if we assume that a data type for geographic coordinates exists, users have to select a location or even enter the coordinates. There is, however, no feedback whether the input location is near any location accessible by public transportation at all.

Filters 5 to 7 describe a requirement that may arise rather frequently. In particular, having a restaurant car does not make any sense without requesting a certain timespan on the respective train. Yet, the requirement uses three separate filters. Likewise, filters 8 to 10 might often appear in exactly the depicted combination. The groups of three filters just make the graph complex where a single requirement is expressed.

When looking at filters 11 to 14, it becomes evident that the condition of applying a special discount iff the connection takes at least two hours is put in an overly complicated way. The arrows can be traced for a given data item. However, natural language expresses the condition as an *if ... then ...* statement rather than explicitely stating the opposite case. Hence, creating such a graph structure in the first place will be very difficult for users.

Furthermore, the conditions of the discount, as expressed by filters 11 to 14, cannot be defined by the traveller, but only by the travel agent. Nonetheless, there is no means to keep the user from editing the respective part of the graph. It is interesting to note that some parts of the very same filter/flow graph should be editable only by some of the users who construct a query.

As noted above, filters 16 and 17 are redundant if it is known that connection *c2* should be the return trip for connection *c1*. In fact, travel search websites usually list the return trip as an option connected to the overall requested itinerary rather than as a separate journey. Again, users may have difficulties constructing the query if they have to insert as a separate filter what they expect to be an option of an existing one.

The remaining filters in Figure 3 represent another condition. Here, users do not only have to explicitely state the opposite of a single filter. Instead, they have to negate a quantifier and its associated conjunction. This is only feasible for users with previous knowledge about predicate logic and boolean operations.

In general, several of the filters require a high level of abstract thinking from the user. For example, filters 21 and 25 express a time constraint as a numerical comparison. Users might not be aware that what is shown as *less than* (no matter whether it is displayed as the mathematical symbol or as a text) could be interpreted as *earlier than* or *before*. Also, quantifiers for picking a train (filter 5) or a stop (filter 19) from a connection are displayed the same way. Users might be confused by this uniformity of entirely distinct things and instead expect the selection facilities to be displayed very differently.

Overall, there may also be contextual factors that determine whether a part of the graph is suitable or not. These may be imposed by external circumstances, or simply influenced by personal preferences of the user.

Summarily, many parts of the filter/flow graph are not intuitively comprehensible for users, or they are only suitable for a narrow range of situations. Therefore, we see the lack of a schema that describes how to adapt a filter/flow graph to a given situation as a problem.

4 Basic Considerations

As a solution for the limitations described in the previous section, a filter/flow graph has to be represented in different ways based on the user and the way she uses the graph. Therefore, we are from now on going to distinguish the *filter/flow graph model* that defines the raw form of a filter/flow graph, and the *filter/flow graph representation* that is output and edited on a user terminal.

Also, the filter/flow graph model should be distinguished from the underyling *data model*. A graph model defines a particular filter/flow graph. The data model defines the structure of the data that the graph operates on (for example, a database schema, an ontology for a semantically annotated data source, ...). It may comprise of several parts if data from various disjoint sources on the web is queried. In other words, a filter/flow graph model is built out of elements defined in the data model.

As we wanted to define a general concept, we have analyzed the limitations pointed out above. We have concluded that sometimes, only single filter nodes have to be replaced with more suitable representations. On other occasions, whole connected subgraphs have to be replaced, either with other subgraphs that comprise of several nodes, or with single nodes that contain the combined functionality of all nodes in the original subgraph.

These transformations can be applied based on a variety of factors that we have grouped in four main categories:

Access Restrictions: Depending on the permissions of the current user, parts of the graph should be displayed as read-only or hidden altogether. The access rules can be saved along with the underlying data model [BL08]. Alternatively, if the data is semantically annotated, access might also be granted based on such annotations.

Generic: Some transformations can be applied generically, such as a “between” for ordinal values that replaces combinations of separate “greater than” and “less than” nodes.

Use Case-Specific: Certain transformations are only useful for particular use cases. For example, grouping the existential quantifiers for round trips (nodes 1 and 15 in Figure 3) is rather specific to queries for travel itineraries.

Context-Based: Different representations of subgraphs or filter nodes may be suitable depending on the context of a system. This encompasses the device context such as multitouch capabilities [HHV+09]—particularly significant when a filter/flow graph representation is available in a web-based way, so it can end up on a multitude of different devices—, the user context including the user's knowledge [Mil97] and cultural preferences [RB07], as well as environmental conditions [PLN04].

5 The Transformation Schema

The factors described above can be put into a fixed order. Based on this chain of transformations, we propose a transformation schema as depicted in Figure 4. It defines the transformation of a filter/flow graph model into a filter/flow graph representation in four steps. Information about the underlying data models of the employed data sources is transformed the same way. This information can be used when editing the graph.

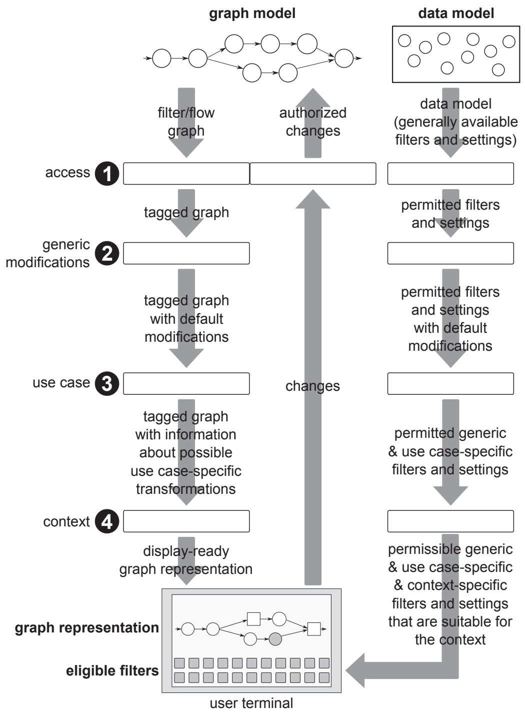
graph model
data model
filter/flow graph
authorized changes
data model (generally available filters and settings)
access 1
tagged graph
permitted filters and settings
generic modifications 2
tagged graph with default modifications
permitted filters and settings with default modifications
use case 3
changes
tagged graph with information about possible use case-specific transformations
permitted generic & use case-specific filters and settings
context 4
display-ready graph representation
permissible generic & use case-specific & context-specific filters and settings that are suitable for the context
graph representation
eligible filters
user terminal

Figure 4: The transformation schema that transforms a filter/flow graph model into a graph representation: Filter/flow graph models and the underlying data model (that defines the available generic filters and their settings) are transformed in a stepwise way. The result is a representation of the filter/flow graph models that has been adapted based on the parameters outlined in section 4. Moreover, the result describes a set of filter nodes that are available for insertion. Any changes to the graph are checked for compliance with the access restrictions before being incorporated into the graph model.

The schema defines the general order of transformations. The concrete transformations can be specified as a set of rules depending on the target context of the transformation.

This way, a graph model can be collaboratively viewed and edited by several users on a variety of devices, each of which displays a graph representation that is suitable for the given device and user, as long as appropriate transformation rules have been supplied.

In the following sections, the transformation schema is described more in-depth.

5.1 Graph Transformation

The transformation steps conduct the following operations on filter/flow graphs:

1. Starting with the unmodified graph, the access restrictions are applied. This results in a graph representation that is annotated with tags that specify the access restrictions, if any, for filters and subgraphs. Any of the subsequent transformations on subgraphs can only be applied if all filters in the subgraph have equivalent access restrictions.
2. Then, some generic substitutions of filters and subgraphs take place. The result is a filter/flow graph representation with lists of possible filter and subgraph substitutions that would result in a graph representation that would be simplified, yet not adapted to any particular use case or context.
3. Subsequently, the information found in the graph representation is used to select possible filter nodes and subgraphs based on the use case. The substitution suggestion lists for filters and subgraphs are modified based on filters and subgraph templates available for the use case. An interesting thing to note is that the application that performs the transformation does not need to know about the use case. Semantic annotations in the source data as well as according semantic tags in the sibustitution rule definitions are sufficient.
4. Eventually, contextual information is first used to further adapt the suggestion lists. This might add new choices to the lists, but may also restrict the selection of eligible filters and subgraph substitutions. Eventually, if there are any lists with several possible substitutions left, one of them is selected based on fixed criteria. The result is a representation of the original filter/flow graph model that respects the access restrictions and is adapted to the context in which the representation is displayed.

The transformation is not as complicated in the other direction, as all filters retain some information about their meaning with respect to the original data model. Whenever there is a change to an existing filter or the graph structure is changed (for example, by adding an additional filter node), only the respective changes will be transferred from the terminal to the graph model. Before being applied to the graph model, an access layer checks whether the access restrictions are being adhered to.

Note that certain properties of the user (which would be, for example, data associated with a user account in a collaborative system) have an effect on various steps. The access layer requires some information about the user's general access rights and/or identity in order to determine which access restrictions are in place for the given data model. Reusable user-defined filters and subgraphs may be integrated into the list of available filters and subgraphs in various steps of the transformation process. That is because users might define their own substitution templates and preferences for arbitrary situations; use case-based, device-based or otherwise. Like this, the schema can also be applied in various scenarios, like those mentioned in the Related Work section: For any new use cases, additional filters and subgraphs can be defined.

5.2 Data Model Transformation

To make editing of the graph more intuitive, the data model that defines which filters and filter settings are available is processed by the same transformation steps as the filter/flow graphs themselves:

1. In the first step, only such items are left in the set of available generic filter nodes and filter settings that may be used according to the access restrictions in place.
2. Afterwards, the list of available filters and settings is expanded with generic filters and their settings.
3. Next, the list is expanded with use case-specific filters and their settings.
4. In the end, the list is changed according to which filter nodes are useful in the current context.

6 Example of Application

The transformation schema presented in Figure 4 and described in section 5 defines how to transform a filter/flow graph model. To illustrate the effect of the transformation pipeline defined in the schema, we are going to apply our schema to the exemplary filter/flow graph model described in section 3.2.

1. **Access Restrictions:** In step one, access restrictions are applied to the filter/flow graph model. In our example scenario, only the subgraph that defines the conditional discount is subject to any access restrictions, for the traveler must not modify the discount or its prerequisites.
   Figure 5 shows what the respective portion of the filter/flow graph representation resulting from step one of the transformation schema could look like.
2. **Generic Transformations:** Step two attempts to replace some of the filters and subgraphs with more intuitive versions that are suitable for any use case. Notably, this includes a substitution of the conditional subgraphs with more comprehensible compound nodes. Also, some single filters are replaced with alternative filters that

are more suitable for the given data type.
As described in section 5.1, the resulting graph representation features lists with various possible substitutions for filters and subgraphs. Figure 6 shows a possible representation after step two, by applying some of the selected substitutions.

3. **Use Case-Related Transformations:** In the third step, the lists of possible filter and subgraph substitutions are extended with substitutions specific to travel planning. Travel planning-related templates for replacing groups of nodes that are commonly used in conjunction are applied. This refers to default parameters such as travel date, but also certain templates for frequent activities such as having a meal in a restaurant car and assuming a default duration for that. Some more single nodes may be replaced with more suitable filters. For example, travel planning-specific location filters might not simply show a geographical map, but rather a map with reference points from the public transportation network.
The subgraph modification may also include substitution templates defined by the user. Frequently employed sets of filters such as nodes specifying a 40-minute-break for having a meal on an intermediate stop could be defined in a reusable way. Alternatively, depending on the level of semantic reasoning applied to the data, the stop duration might be inferred from the requirement for a restaurant at the stop based on the underlying ontology.
Again, Figure 7 shows what a filter/flow graph representation could possibly look like when picking some substitutions selected unto here.

4. **Context-Related Transformations:** Lastly, some context-related transformations are applied. Based on the available context information, a final substitution is selected by the system from the list of eligible substitutions and applied to create a final graph representation.
For touch devices, touch-capable filters might be preferred. Depending on whether the user is in a public place or not, textual filters for the point of origin and the destination may be preferred to graphical ones, so as not to give away the users' home address at a single glance. Also, filters that display only the most important settings in an abbreviated way rather than the full information may be considered more suitable both for display on small screens and for situations in which the user has little time. In the travel use case, both may occur when changing plans during the journey.
Figure 8 shows some examples of what alternative filters might look like that are chosen based on special context parameters.

7 Conclusion and Future Work

In this work, we have discussed the difficulty of precisely specifying queries and referred to the filter/flow concept. We have demonstrated some shortcomings of the filter/flow concept in an example from the topic of public transportation. As a result, the usability of the filter/flow graph can be enhanced by adapting it based on access priviledges, generic, use-case-specific and context-based rules. We have proposed a transformation schema that should be applied to a filter/flow graph and its underlying data model when displaying and editing the graph on an arbitrary device. The result of the transformation are context-dependent filter/flow graph representations adapted for their respective use case and supporting access restrictions, hence suitable for multi-modal, distributed collaborative use. Depending on the degree of annotation in the source data, the transformation rules can be further decoupled from an implementation of the schema, thereby making the schema particularly useful for querying in the data web. The effectiveness of the transformation schema is shown by its application to the public transportation example graph.

Based on the schema presented in this paper, we are currently implementing a prototypical system. This system will be used to evaluate our schema in user tests. Among others, we would like to compare user performance in transformed graphs to user performance in graphs where only the generic nodes of the basic filter/flow concept are available.

Our future plans are to examine more closely the various substitution possibilities that result in structural changes to the graph, thereby defining a larger set of transformation rules to choose from. Moreover, we would like to consider different options for access restrictions and their implications to the way the graph can be collaboratively edited by users. Eventually, we also hope that our findings related to transformations of filter/flow graphs will also be beneficial for other types of query visualizations. Likewise, we hope that our transformation schema inspires similar schemas for use on other types of graph visualizations.

8 Acknowledgments

This research was funded through the IP-KOM-ÖV project (German Ministry of Economy and Technology (BMWi) grant number 19P10003N).

References

[BFKJ08] Pascal Bihler, Merlin Fotsing, Günter Kniesel, and Cédric Joffroy. Using conditional transformations for semantic user interface adaptation. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 677–680, New York, NY, USA, 2008. ACM.

[BL08] Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *The VLDB Journal*, 17:603–619, 2008. 10.1007/s00778-006-0023-0.

[Cal] California PATH, Headquarters, University of California, Berkeley, Institute of Transportation Studies. Networked Traveler. http://www.networkedtraveler.org.

[DB] DB Vertrieb GmbH. DB Bahn. http://www.bahn.de.

[EST07] Niklas Elmqvist, John Stasko, and Philippas Tsigas. DataMeadow: A Visual Canvas for Analysis of Large-Scale Multivariate Data. In *IEEE Symposium on Visual Analytics Science and Technology, 2007*, pages 187–194, 2007.

[HC03] Jiwen Huo and William B. Cowan. KMVQL: A Graphical User Interface for Boolean Query Specification and Query Result Visualization. In *Proc. of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 96–97, Washington, DC, USA, 2003. IEEE Computer Society.

[HHV$^+$09] Thomas E. Hansen, Juan Pablo Hourcade, Mathieu Virbel, Sharath Patali, and Tiago Serra. PyMT: a post-WIMP multi-touch user interface toolkit. In *Proc. of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 17–24, New York, NY, USA, 2009. ACM.

[HSMT06] Tomoyuki Hansaki, Buntarou Shizuki, Kazuo Misue, and Jiro Tanaka. FindFlow: visual interface for information search based on intermediate results. In *Proc. of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, pages 147–152, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[HSS07] Andreas Hildisch, Jurgen Steurer, and Reinhard Stolle. HMI generation for plug-in services from semantic descriptions. In *Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems*, SEAS '07, pages 4–, Washington, DC, USA, 2007. IEEE Computer Society.

[HY07] Jiang He and I-Ling Yen. Adaptive User Interface Generation for Web Services. In *IEEE International Conference on e-Business Engineering, 2007*, pages 536–539, 2007.

[JGZ$^+$11] Hans-Christian Jetter, Jens Gerken, Michael Zöllner, Harald Reiterer, and Natasa Milic-Frayling. Materializing the query with facet-streams: a hybrid surface for collaborative search on tabletops. In *Proc. of the 2011 annual Conference on Human Factors in Computing Systems*, CHI '11, pages 3013–3022, New York, NY, USA, 2011. ACM.

[Jon98] Steve Jones. Graphical query specification and dynamic result previews for a digital library. In *Proc. of the 11th annual ACM Symposium on User Interface Software and Technology*, UIST '98, pages 143–151, New York, NY, USA, 1998. ACM.

[LVM$^+$04] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins, and Daniela Trevisan. UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. In *Proc. of the ACM AVI'2004 Workshop Developing User Interfaces with Xml: Advances on User Interface Description Languages*, pages 55–62. Press, 2004.

[MAEGJ04] Andrew J. Morris, Alia I. Abdelmoty, Baher A. El-Geresy, and Christopher B. Jones. A Filter Flow Visual Querying Language and Interface for Spatial Databases. *GeoInformatica*, 8:107–141, 2004.

[Met] Metropolitan Transportation Authority. MTA NYCT Trip Planner$^+$. http://tripplanner.mta.info/MyTrip/ui_web/customplanner/tripplanner.aspx.

[Mil97] Maria Milosavljevic. Augmenting the User's Knowledge via Comparison. In *Proc. of The 6th International Conference on User Modelling*, pages 119–130, 1997.

[MM02] Nikola Mitrović and Eduardo Mena. Adaptive User Interface for Mobile Devices. In Peter Forbrig, Quentin Limbourg, Jean Vanderdonckt, and Bodo Urban, editors, *Interactive Systems:Design, Specification, and Verification*, volume 2545 of *LNCS*, pages 29–43. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-36235-5_3.

[MPG98] Norman Murray, Norman Paton, and Carole Goble. Kaleidoquery: a visual query language for object databases. In *Proc. of the Working Conference on Advanced Visual Interfaces*, AVI '98, pages 247–257, New York, NY, USA, 1998. ACM.

[MRM05] Nikola Mitrović, José Alberto Royo, and Eduardo Mena. Adaptive user interfaces based on mobile agents: Monitoring the behavior of users in a wireless environment. In *1st Symposium on Ubiquitous Computing and Ambient Intelligence*, 2005.

[PLN04] Tim F. Paymans, Jasper Lindenberg, and Mark Neerincx. Usability trade-offs for adaptive user interfaces: ease of use and learnability. In *Proc. of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pages 301–303, New York, NY, USA, 2004. ACM.

[PM00] John F. Pane and Brad A. Myers. Improving user performance on Boolean queries. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, pages 269–270, New York, NY, USA, 2000. ACM.

[RB07] Katharina Reinecke and Abraham Bernstein. Culturally Adaptive Software: Moving Beyond Internationalization. In Nuray Aykin, editor, *Usability and Internationalization. Global and Local User Interfaces*, volume 4560 of *LNCS*, pages 201–210. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73289-1_25.

[Ric05] Kai Richter. A transformation strategy for multi-device menus and toolbars. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, pages 1741–1744, New York, NY, USA, 2005. ACM.

[RKE05] Martin Rotard, Sven Knödler, and Thomas Ertl. A tactile web browser for the visually disabled. In *Proc. of the Sixteenth ACM Conference on Hypertext and Hypermedia*, HYPERTEXT '05, pages 15–22, New York, NY, USA, 2005. ACM.

[RPK$^+$11] David Raneburger, Roman Popp, Sevan Kavaldjian, Hermann Kaindl, and Jrgen Falb. Optimized GUI Generation for Small Screens. In Heinrich Hussmann, Gerrit Meixner, and Detlef Zuehlke, editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 107–122. Springer Berlin / Heidelberg, 2011.

[Shn94] Ben Shneiderman. Dynamic queries for visual information seeking. *Software, IEEE*, 11(6):70 –77, nov 1994.

[SHT$^+$07] Pasha Sadri, Ed Ho, Jonathan Trevor, Kevin Cheng, and Daniel Raffel. Yahoo! Pipes. http://pipes.yahoo.com/, February 2007.

[SK11] Thomas Schlegel and Christine Keller. Model-Based Ubiquitous Interaction Concepts and Contexts in Public Systems. In Julie Jacko, editor, *Human-Computer Interaction. Design and Development Approaches*, volume 6761 of *LNCS*, pages 288–298. Springer Berlin / Heidelberg, 2011.

[Spo93] Anselm Spoerri. InfoCrystal: a visual tool for information retrieval & management. In *Proc. of the Second International Conference on Information and Knowledge Management*, CIKM '93, pages 11–20, New York, NY, USA, 1993. ACM.

[TIC09] Matthew Tobiasz, Petra Isenberg, and Sheelagh Carpendale. Lark: Coordinating Co-located Collaboration with Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15:1065–1072, November 2009.

[YS93] Degi Young and Ben Shneiderman. A graphical filter/flow representation of Boolean queries: a prototype implementation and evaluation. *JASIST*, 44:327–339, July 1993.

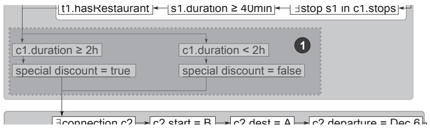

Figure 5: A part of the filter/flow visualization from Figure 3 after step one of the transformation schema has been applied, as seen by the traveler: A transparent gray overlay (1) symbolizes that the subgraph it covers is write-protected and can only be examined by the traveler.

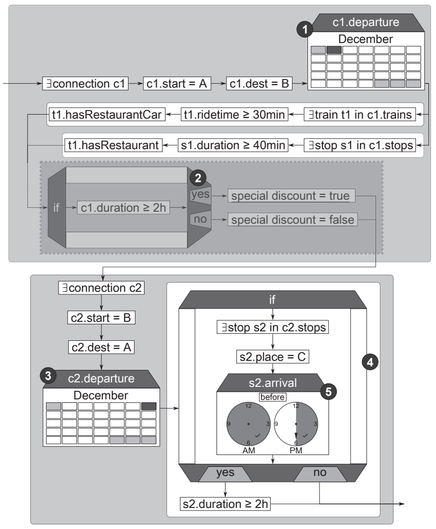

Figure 6: The filter/flow visualization from Figure 3 after step two of the transformation schema: The date filters have been replaced with calendar controls (1 and 3). Conditional subgraphs have been replaced with more comprehensible compound *if*-filters (2 and 4). The comparison operation with a time of the day is represented in a more intuitive way (5).

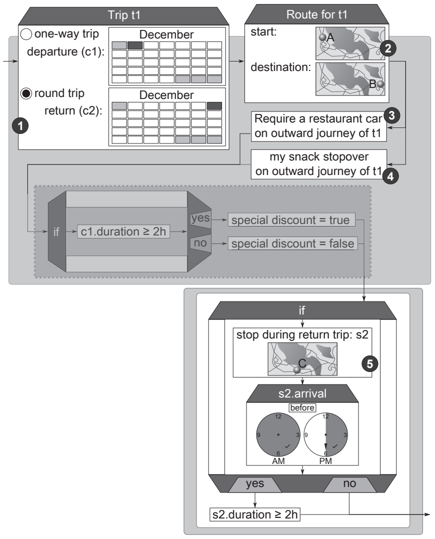

Figure 7: The filter/flow visualization from Figure 3 after step three of the transformation schema: The basic settings for the itinerary have been grouped in use case-specific compound filters for the travel time and route (1 and 2). The subgraphs that require an opportunity to have a small meal during the trip have been substituted with single filters provided as use case-specific templates (3 and 4). Likewise, the quantifier for a stop on the return trip and the filter for restricting the stop location have been merged into a single node (5).

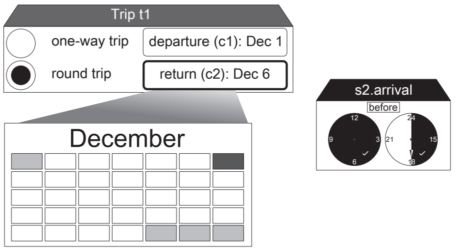

Figure 8: Some alternative filters for those shown in Figure 7, that might be chosen based on the context: The filter on the left side is optimized for touch devices. It sports extra-large input controls and—due to the higher spatial requisition—provides a calendar view only as a temporary popup. The filter on the right side may be chosen based on the user context, if the user's culture prefers 24 hour times rather than AM/PM designations. Also, it may be preferred due to environment context as it features a strong color contrast and is hence suitable for inconveniently lit environments.