Roith *et al. Visualization in Engineering* (2017) 5:20
DOI 10.1186/s40327-017-0057-y

Visualization in Engineering

RESEARCH Open Access

Supporting the building design process with graph-based methods using centrally coordinated federated databases

Johannes Roith[1], Christoph Langenhan[2*] and Frank Petzold[2]

Abstract

Background: Technological developments and globalized working processes have transformed the building process. However, current digital semantic building models no longer adequately represent the increasing complexity of modern building projects. The potential of combining agent-based and graph-based methods, for example for energy calculations or spatial research strategies, is not fully exploited.

Methods: In our system, users search for building floorplans, for example as sources of inspiration, by creating conceptual hand-drawn sketches of building parts on multi-touch devices. The sketch is analyzed and used to query a federated database system comprising a building information model server, the graph database neo4j and the content management system mediaTUM. Users interact with client applications that show and continuously update a list of floorplans by sending queries to a central coordinator service. In the paper, we describe the coordinator that enables our databases to appear as a single smart information system to search for digital information about buildings and visualize their floorplans. The application case comprises search by drawing the initial design idea of a building to find similar floorplans e.g. as source of inspiration.

Results: Our federated database system is queried using semantic building floorplan fingerprints, which are formalized as graphs and encoded in a common schema, like our AGraphML, to represent spatial configurations and perform graph matching. As graph matching is computationally expensive, the coordinator needs to analyze the queries, separate different fingerprints and metadata and pass it to specialized agents, implemented as microservices, for processing. The returned result sets are combined and the results are visualized. The use of multiple agents facilitates the recombination of the data from the underlying disparate data sources and also serves to support different execution strategies which are chosen using properties of the query and a set of predefined rules.

Conclusions: To deal with complex and dynamic queries, as well as dynamic results that are updated while some agents are still executing, a caching framework was developed which takes the similarity measures of the graph-based building representation into account when determining query equality.

Keywords: Research strategies, Building information modelling, Federated databases

Background

As the complexity of building tasks and requirements increases, designers often find themselves confronted with interdisciplinary problems that go beyond the specific challenges and methods of architecture and engineering. The iterative nature of the design process results in a continuous exchange between creative, analytical and evaluative activities, through which the designer explores and identifies promising design variants. The ability to compare and evaluate relevant reference examples of already built or designed buildings helps designers assess their own design explorations and informs the design process. Most computational search methods available today rely on textual rather than graphical approaches to represent information. However, textual descriptions are not sufficient to adequately describe spatial configurations

*Correspondence: langenhan@tum.de
[2]Department of Architecture, Technical University of Munich, Munich, Germany
Full list of author information is available at the end of the article

such as floor plans. To address these shortcomings, Langenhan et al. (2013) introduced a novel approach which facilitates the automatic lookup of reference solutions from a repository using graphical search keys. For the search key, the notion of a building floorplan fingerprint was introduced which describes the main characteristics of a building's design. This forms the basis for assessing the similarity of different reference solutions to a specified problem and serves, accordingly, as an index for the building model repository.

This paper describes an information retrieval system that allows users to search for building models that are similar to a hand-drawn sketch of a building or parts of a building on a multi-touch device to support the design process of architects. The design process is an iterative process (Buxton 2007) of searching for a plausible solution involving a continual back and forth in which potential solutions are developed by various means before narrowing down the selection to the most promising candidates. Architects typically work with traditional design tools in the early design stages, such as model-making, sketches and the use of reference examples. Using reference examples is an acknowledged method in both architectural education and architectural practice and helps in learning design principles and guiding the design process, as well as for inspiration or even as an explicit solution (Richter 2010). Digital equivalents have already been devised for several traditional design tools, however many of these do not fully translate the original strengths of the analogue methods into the digital world and fail to make full use of digital possibilities.

The approach discussed aims to overcome these shortcomings and proposes a new paradigm for designing with reference examples that takes into account the typical way a designer works in the early design phases and makes full use of current methods and technologies. Finding relevant reference examples with traditional technologies is often tedious, and at present there is a lack of appropriate IT support for this task. A central problem is finding a way of determining similarity between the designer's query and the data stock. The solution we are proposing (Langenhan et al. 2013) is based on graph representations that capture the topological relationships between spaces. For indexing and determining similarities, the use of semantic fingerprints (Langenhan and Petzold 2010) has been proposed as a way of describing and visualizing the arrangement of building floorplans in a manner analogous to the way that fingerprints can be used to identify a person. The system derives semantic fingerprints 'representing e.g. accessibility and adjacency as features for search criteria' from a reference solution that describes the spatial relationship of rooms extracted from building information models. In the application area described in this paper (see Fig. 1), the system assists the designer by analyzing sketches made

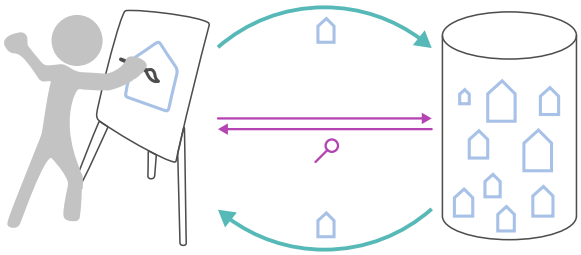

Fig. 1 An application scenario for knowledge-based design

in the early design phases and deriving a structure that can be compared with the fingerprints in the information repository. Using a graphical user interface, the designer can sketch spatial configurations as a bubble diagram, freehand sketch or schematic digital layout to visually explore the spatial configuration.

Floorplans are a way of representing the structure of a spatial configuration visually, which can be formalized mathematically in graphs. In our system, each floorplan sketch is first converted to a graph. Rooms are represented as nodes and doors/walls are represented as edges. The graph is stored as AGraphML, which is based on the GraphML standard (GraphML 2017). Next, *fingerprints* are computed which can be compared to a database of existing graphs. Users can define which fingerprints they want to use for matching or let the coordinating service select appropriate ones (Fig. 2).

There are six main reasons why we use coordinated microservices as our approach: First, we wish to support different types of fingerprints, each potentially implemented using its own microservice. Second, for each fingerprint, there may be multiple execution strategies available. This is particularly important for graph matching which often requires different strategies for different types of graphs in order to execute efficiently (Conte et al. 2004). It then becomes necessary to select the best service, e.g. based on a set of *rules*. Third, the building data is stored in a federated database system and it is useful to be able to access multiple databases in a single query. In our case the database system consists of the open source building information model server, the graph database neo4j, which stores the fingerprints, and the content management system mediaTUM, which holds pictures and metadata. Fourth, we want to be able to express complex queries that involve various microservices and obtain combined and ranked results. Fifth, since the computation of some fingerprints can be very expensive, it is necessary to reuse as much of the previous responses of the individual microservices as possible. Since reusability depends on the graph structure, an application-specific caching framework is needed. To address these problems while keeping the clients and microservices simple, we propose a central web service called the *Coordinator*

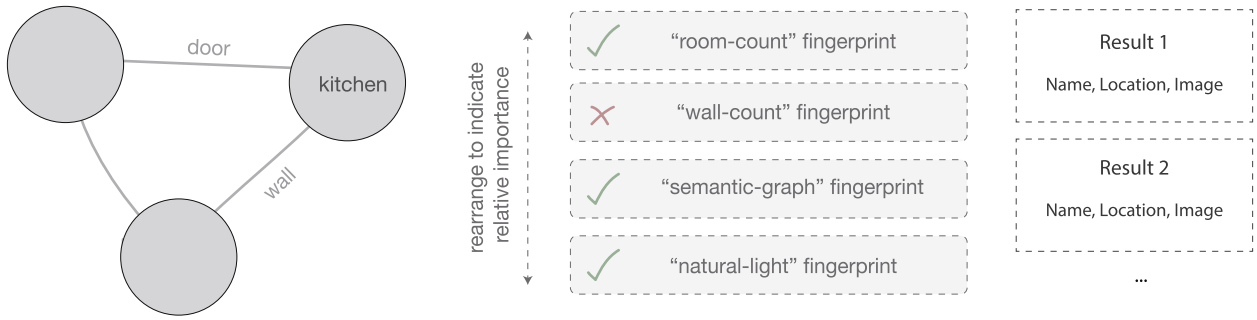

Fig. 2 The simplified graph (left) is extracted from a sketch and fingerprints are selected (right)

which processes queries and distributes their parts to the microservices. Sixth, the coordinator can precompute visualization data for the client applications.

Software architecture

To realize the semantic search engine, we propose a system of data integration[1] using a federated information system in which different autonomous sources of information (e.g. a BIM server, graph database and CMS) can be integrated using a common XML schema and queried as a single information source using a REST approach. This makes it possible to store and process information efficiently according to their specific properties. In contrast to other *data warehouse* approaches, a federated information system does not copy the various sources but rather queries the respective sources individually using processing components and bundles the results for further processing by the coordinated system of components in the client applications. For offline processing and extraction of building floorplan fingerprints, components are needed that a) analyze and augment unstructured data sources (CMS) and b) components that extend and attribute information to concepts in structured data sources (BIM). Figure 3 shows the software architecture of the information system which draws on the software architecture of the semantic search engine by (2012, p. 461). Topological information is extracted offline from formal (BIM) and informal (CMS) data sources using various methods and formalized in the form of graphs (fingerprints). The offline processing must structure the data in such a way that the user is presented with useful results within a reasonable time frame during online processing.

The components shown are groups of individual applications in the information cloud (servers) and on different end devices (clients). On the server-side the system is comprised of *data storage*, *processing* and *coordination* components. Possible clients include a desktop computer with mouse and keyboard (stationary workstation, e.g. in the office), tablet computers (transportable device, e.g. for client visits and meetings), smartphones (mobile, e.g. on-site use or site research) and a multi-touch table (stationary design environment).

The applications on the various end devices query the information cloud (Fig. 4). The information system for assisting the designer in the early design phases comprises a series of components for data storage and processing in order to store and process semantic building model data, graph data and text.

The application contexts and individual design processes differ from case to case. Different abstractions are therefore necessary to allow the designer to communicate his or her mental model as completely and accurately as possible. Because designers often only have a vague and incomplete notion of the design in the early design phases, the system needs to accommodate a high degree of freedom of expression and types of abstraction.

Different layers of abstraction

The system has to provide interaction approaches that allow the user to express a mental model that can be analyzed and compared with stored model schemes to produce search results. "Guidance is often required, especially for novice users, on what visualizations (scatterplot, parallel coordinates, treemaps, etc.) are appropriate for a given task; the focus should be on the problem to be solved rather than the type of raw data." (Keim et al. 2010, p. 123). To facilitate use and understanding, domain-specific presentation methods need to be provided along with strategies for improving the ability to describe and communicate ideas (Roth-Berghofer and Richter 2008). In addition, approaches are needed that present the user not just with results, but also show the often numerous and interdependent criteria in a transparent and understandable way (Gratzl et al. 2013) so that the user can adjust these as required. Keim summarizes these requirements as follows:

- "Progressive analysis: provide quick answers first, then make improvements incrementally or on-demand;

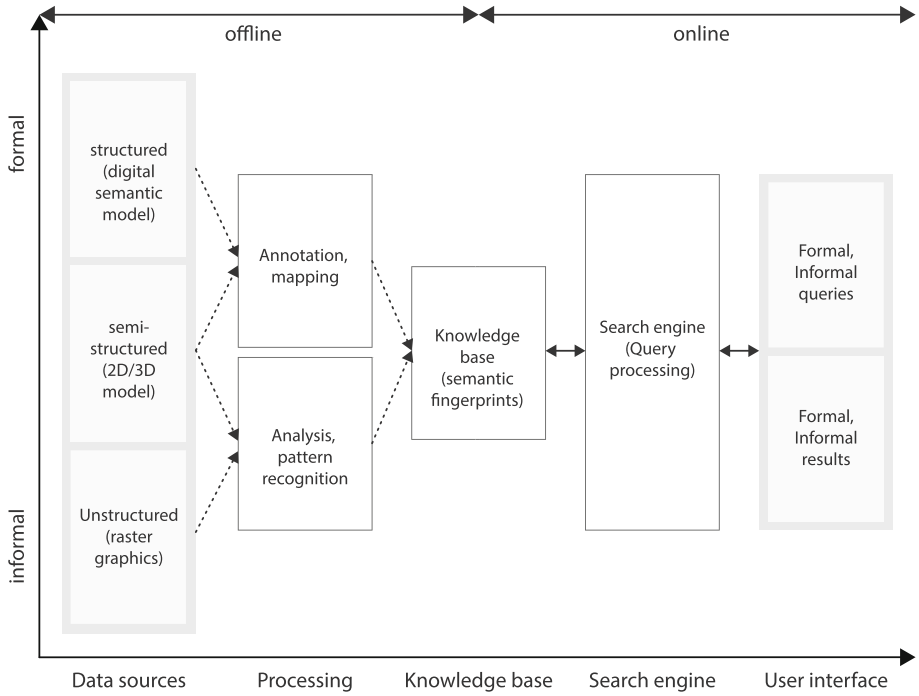

Fig. 3 Software architecture of semantic information systems based loosely on Dengel (2012)

- Management of dynamic data: incremental analysis instead of restarting it from the beginning;
- Steerable analysis: allow long-computations to be steered by users when possible." (Keim et al. 2010, p. 106)

The aim is to gradually narrow down possible solutions or reference cases by providing ongoing feedback so that the query can be successively steered and defined by the user. An example of this is shown in Fig. 5 for building information in the early design phases. The degrees of detail shown are the product of thinking processes that gradually transform a design and make it more concrete.

Corresponding interaction strategies for geometric, topological, geographic and lexical information are likewise required. In the information system we have implemented, we differentiate between five main representations for formalizing mental models according to their respective characteristics:

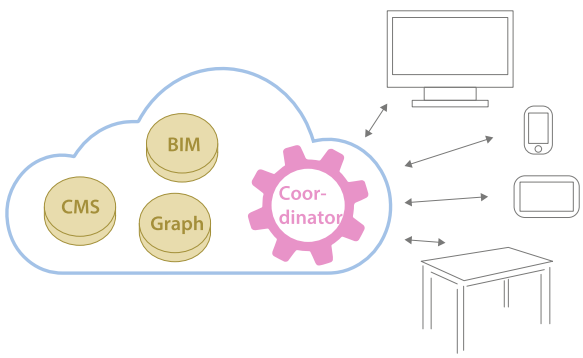

Fig. 4 Information cloud

- Texts (for example architect, year of completion, building costs, text search, descriptions, building typology)
- Tables (for example schedule of rooms, schedule of works, cost plan, specification of works, list of neighbors, access routes)
- Schemes (for example diagrams, spatial arrangement, zones, orientation, proportions, passage through a room, adjacent relation to other spaces)
- Freehand sketches (for example, arrangement of spaces, zoning, orientation, proportions, passage through a room, spatial delineation, floor plans, elevations, sections)
- 2D/3D drawings (for example, arrangement of spaces, zoning, orientation, proportions, passage through a room, adjacency relationships, cubature, floor plans, elevations, sections, perspectives).

The transformation between different digital formalization's and visualizations and their respective levels of abstraction is of particular help in the design process and is a topic of ongoing research. The aim is to examine the technical possibilities for computer-based interpretation of user input and the visualizations of building data.

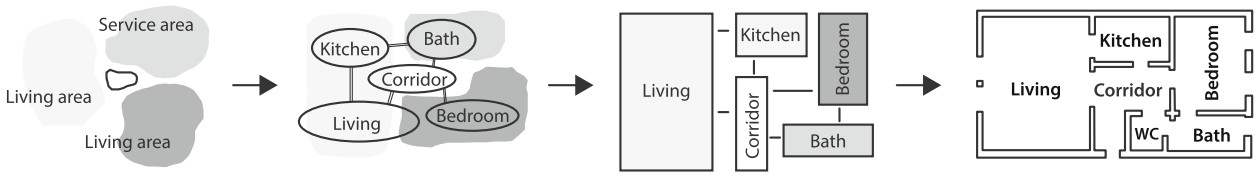

Fig. 5 Typical presentation schemes and their possible transformations, based loosely on (Hanson 1998, p. 308) and (Liebich 1994, p. 75)

Prototypes

Our prototypes enable users to interactively retrieve descriptions of similar building floorplans by sending queries which are constructed visually to a Coordinator middleware. The ability to add detail to or modify one's input, for example by adding a room to a drawing, makes it possible to formulate more granular fingerprint-queries and the mental model corresponds better to the designer's own mental model. By providing ongoing feedback, e.g. in the form of reference projects that match the fingerprint, the information system prompts the designer to modify and adapt his or her design idea. By increasing or decreasing the degree of specification, the set of possible search results can be expanded or focused to better match the problem at hand. What density of information is adequate and necessary depends on the specific application at hand and the user. For the general definition of spaces, their relationships and the passage between them, a simple graph can be sufficient. Graphs can take the form of diagrams or tables (Beck et al. 2014, p. 88). In our information system, we have used node and edge diagrams to make it easier to depict the location and relationship between rooms or nodes. Our user interface investigations looked at ways of providing vague input about spatial situations and constellations, i.e. their basic characteristics and arrangement. The corresponding formulation of this as a graph is shown to the user at different levels of abstraction.

The system has to provide interaction approaches that allow the user to express the mental model that can be analyzed and compared with stored model schemes to produce search results. Therefore, contextual coherence between the mental model and the representations of information is necessary. Accordingly, suitable strategies need to be implemented to achieve a beneficial outcome for the user.

Whenever the user modifies the sketch or the parameters of the filters, a new query needs to be generated and sent to the Coordinator. The sketch is encoded as an AGraphML floorplan graph and the filters are encoded as XML as well. After the coordinator has received the query, it executes it by distributing parts of it to the connected microservices and combining the results.

The ability to add detail to or modify one's input, for example by adding a room to a drawing, makes it possible to express more granular fingerprint-queries and the mental model corresponds better to the designer's own mental model. For this purpose, we implemented separate prototypes that use the same data stock to compare different input and retrieval strategies.

Our user interface investigations looked at ways of providing vague input about spatial situations and constellations, i.e. their basic characteristics and arrangement. The corresponding mathematical formulation of this as a graph is shown to the user at different levels of abstraction.

The different use cases (e.g. in the office, when visiting a client, in a meeting, out on site or while designing) are conceived as prototypes for the respective user. For the most common office drawing situation with mouse and keyboard, we propose using the *a.vista* concept previously developed by Christoph Langenhan in 2008 (Langenhan and Petzold 2010), which has different semantic ways of describing a building (level, unit, zone, and room).

The *ar.searchbox* (see Fig. 6) is a media reference collection from the *mediaTUM* (Langenhan et al. 2012) content management system that was devised as a research tool for buildings. The university library and computer science students maintain and enhance the system while students of architecture at the TUM enter and maintain data. It is possible to search using conventional textual search queries in the browser and the entries are linked to other information sources, such as the architects' website or a digital building model based on an analysis of the pixel images, building models and links.

For the *metis WebUI* (see Fig. 7), Johannes Bayer at the Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI GmbH) took a platform-independent 2D modelling approach and implemented, among other things, a means of enriching search queries (Bayer et al. 2015). In Fig. 7 one can see the spatial arrangement as a bubble diagram on the left and actual spatial arrangement with symbols for doors and windows on the right, demonstrating the different formalizations of levels of abstraction and detail.

An approach for formalizing queries as freehand sketches was developed by Markus Weber in 2009 (Weber et al. 2010) with *a.scatch* (Fig. 8). Doors are drawn as double lines and adjacent walls as single lines. The rooms (Fig. 8, left a) are recognized and the room type labelled by hand (Fig. 8, left c). The resulting room (Fig. 8, left b) and

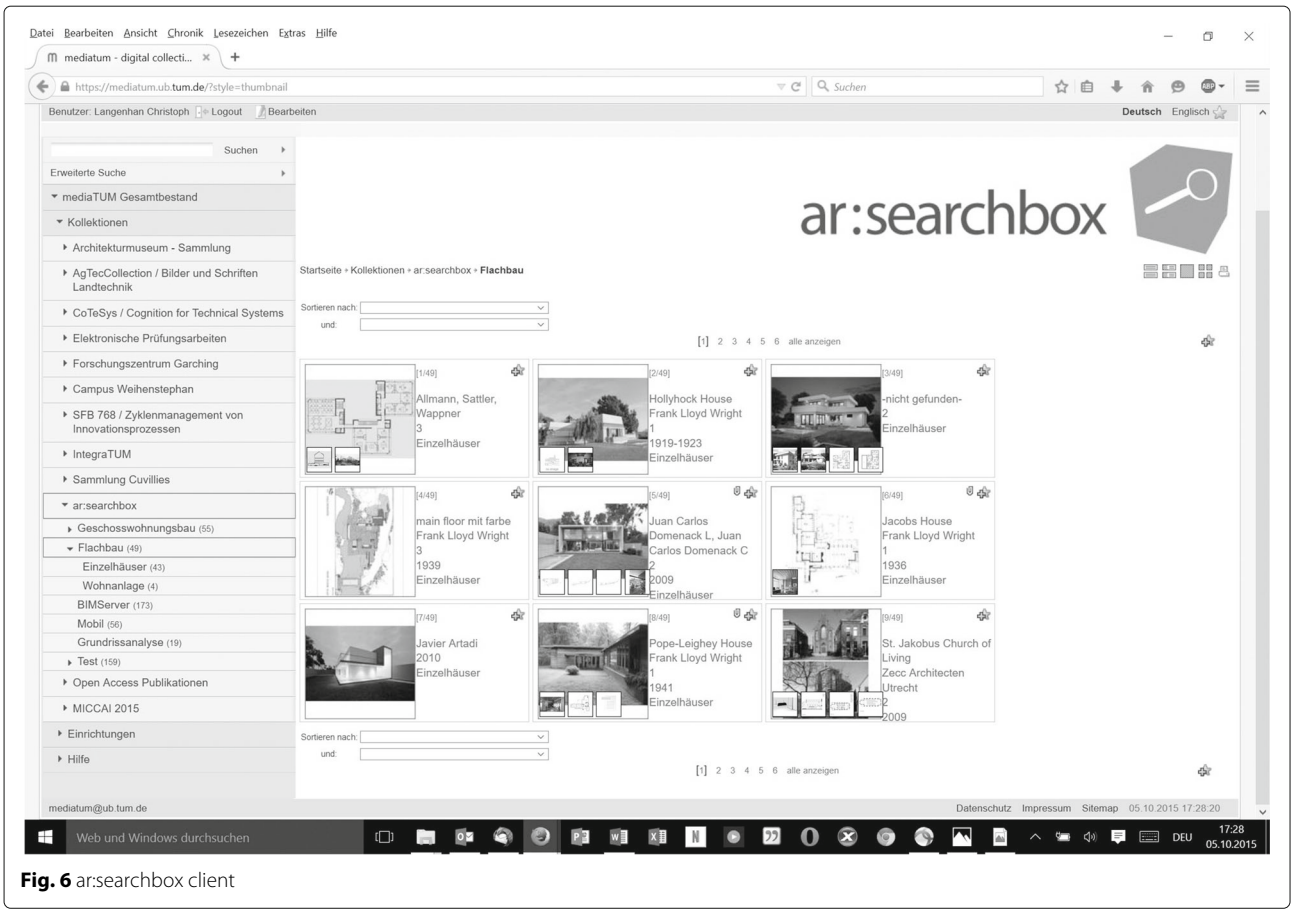

Fig. 6 ar:searchbox client

be used for the search query and a list of corresponding search results is then displayed (Fig. 8, right).

The application *Touchtect* Fig. 9 by Thomas Kinnen and Dario Banfi for a multi-touch table combines freehand drawing, geographic input and meta information from the ar:searchbox (Fig. 6). Here, however, the room type is selected from a predefined list, while the application *ar:searchDroid* (Fig. 10) by Sebastian Seitz is designed for a tablet computer for mobile research and can restrict search results to buildings that are in the vicinity.

All of the above applications for freehand drawings do not detect what has been drawn by visually processing the results but by detecting the movement of the stylus. As such, a polygon denoting a room needs to be closed in order to be recognised for use in the spatial query. To afford the user greater flexibility while drawing, the *cormorant* application (Fig. 11) by Dominic Henze uses a program library (Vatavu et al. 2015) for gesture recognition developed by the University of Washington. The application *ar:searchPad* by Jana Pejic (Fig. 12) does away with the input of geometric figures altogether. This application is based entirely around the input and comparison of bubble diagrams.

To incorporate such topological approaches in the design process, Thomas Stocker, Dario Banfi, Jana Pejic, Thomas Kühner, Markus Dausch, Bishwa Hang Rai, Dominic Henze, Arno Schneider and Johannes Roith have jointly developed an Add-on called *Dolphin* (Langenhan and Petzold 2015) for *Rhino3D* and its parametric design extension *Grasshopper3D*. The *Dolphin* Add-on provides a series of components for visual programming using *Grasshopper3D* that make it possible to query the information system using drawings in *Rhino3D* and additional meta information. The data can be exported for further use directly from the graph database using a service called *pigeon* developed by Leon Höß and Christopher Will. For example, AgraphML files can be imported with other components, either individually or as a collection, into *Grasshopper3D* for further use.

The Coordinator to prepare and visualize the results

The middleware described below is implemented as a RESTful web service. Clients pass queries to the Coordinator as XML via HTTP. The system executes a query using one or more agents. While the query is being executed, a result list is maintained on the server and continuously

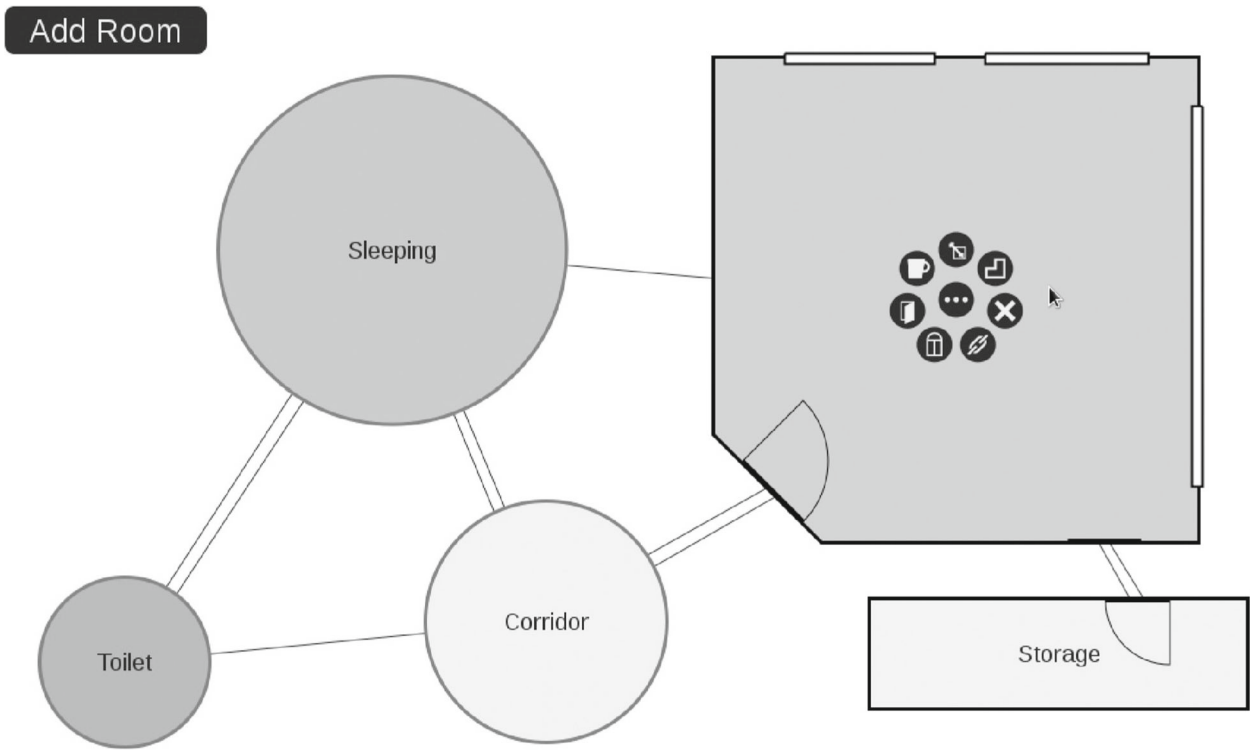

Fig. 7 Metis Web UI client

updated. The client receives a stream of change events until the execution terminates. Possible events include insert, remove and update events. The basic data flow is shown in Fig. 13.

Query processing

The server provides various types of conditions, such as a metadata conditions which can be used to filter results based on simple attributes such as the name of the architect. A very simple example would be the room-count condition which finds all building floorplans in the database that have the same number of rooms as the floorplan graph in the query. We have currently defined 12 fingerprints that check, for example, properties of nodes, edges or full graph isomorphism (Weber et al. 2011). Figure 14 shows some example queries. Below we shortly describe the query process. Detailed information can be found in Roith et al. (2016).

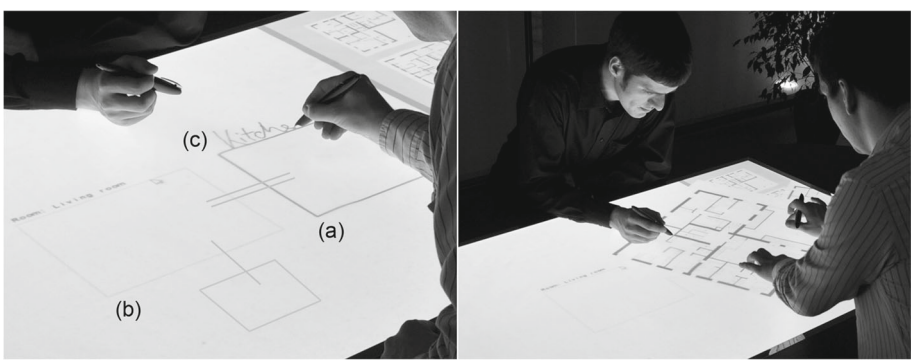

Fig. 8 a.scatch: Sketch-Based Retrieval for Architectural Floor Plans

Fig. 9 Touch tech client

Queries are executed in multiple phases. The basic process is shown in Fig. 15. The query arrives as XML at the service layer of the application and is parsed into a tree (AST, Abstract syntax tree). Since our queries are simple, we do not generate a query plan but simplify and normalize the tree using a set of rewriters: This also simplifies caching by making it easier to detect semantically equal[2] queries. The best agent is picked using a set of rules from a rule base, which is currently maintained manually by an expert. The simplified AST is then used to create *data sources* which are instances of agents (Fig. 16). Furthermore, a cached data source from a previous execution can often be reused. Next, all data sources are executed in parallel. Agents produce a stream of objects which represent building floorplans or storeys and can be uniquely identified. Time limits imposed by the configuration are used to abort agents that exceed their allocated execution time. Similarly, agents self-terminate or are aborted when they produce more than a certain number of results. While passing through the DAG,

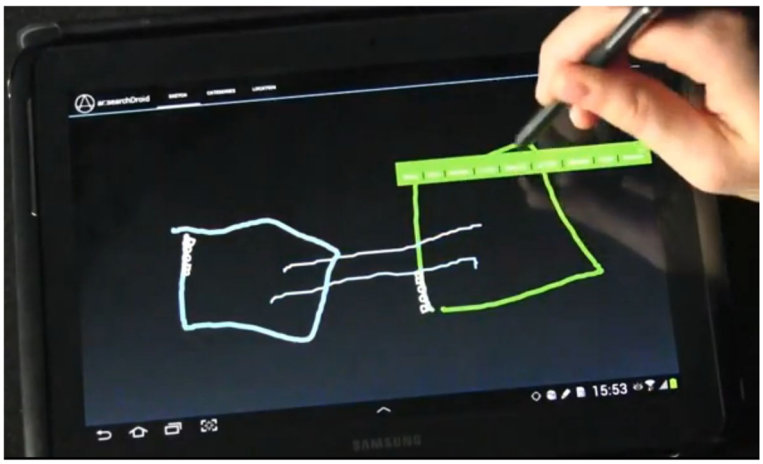

Fig. 10 User interface ar:searchDroid

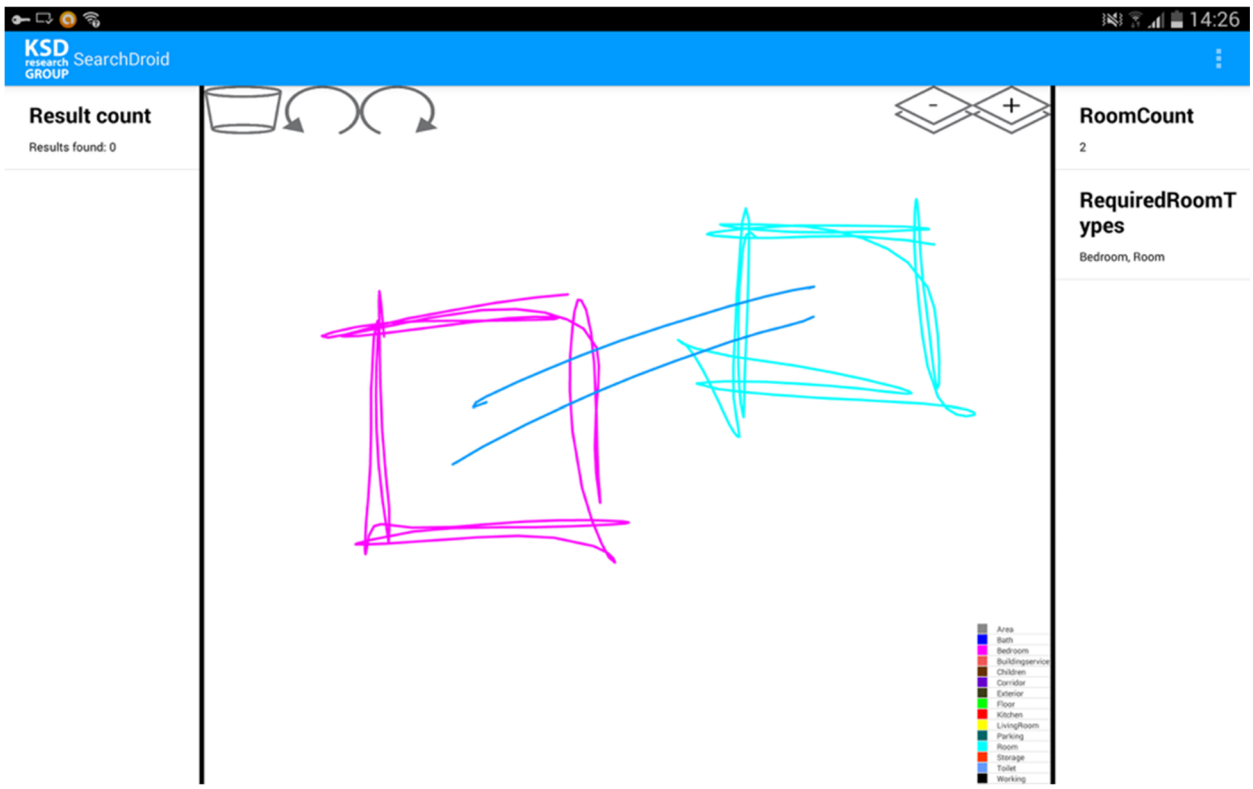

Fig. 11 User interface of 'cormorant'

results can be merged or dropped. Results that fulfill all conditions are inserted into a results list that is sorted by the similarity score of the results. Clients observing the list can specify whether they want to receive dynamic updates or a static list. Any changes made to the list, or to the content of items, are then propagated to the clients using appropriate change events. In the case of a static list, no events are sent until the execution has finished, resulting in an XML stream of only 'append' elements that can be interpreted as a document. Clients can specify which data they want to receive for the results. For example, each building/storey usually has associated metadata, a floorplan graph or some images. This data is not provided by agents. Instead the Coordinator has a concept of data model providers which fetch and provide any data content requested and deliver preprocessed data for visualization.

Although clients can specify the list of possible or required fingerprint matches as part of the expression tree, it is also possible to allow the server to select appropriate fingerprints based on the query graph. In this case, clients include a special 'auto' match condition in the query tree. For example, if a floorplan graph lacks node or edge labels, a fingerprint that relies on those labels can't be computed and will not be included in the server-selected list. Fingerprints may also have parameters, for example to adjust the required precision of a match. For server-selected fingerprints, appropriate parameters need to be selected. In some cases, it may be appropriate to select multiple possible parameters and have multiple instances of the same fingerprint. For example, the room-area fingerprint is always parameterized by the room type. If created during server-side fingerprint selection, a fingerprint instance is added for each room type that is found in the query graph.

For each condition (and, by extension, fingerprint) an agent must be selected to execute it. If more than one agent is compatible with the condition/fingerprint type and parameters, a good one (that produces good results, ranks them well, executes quickly and consumes little resources) has to be picked. The rules usually look at certain features of the floorplan graph - for example the number of nodes - to make a decision.

If an AND or OR node contains multiple children that are assigned to different agents, the node itself does not form an assignable subtree but it is often possible to extract several children that share the same agent into a new node of the same type and make the newly created

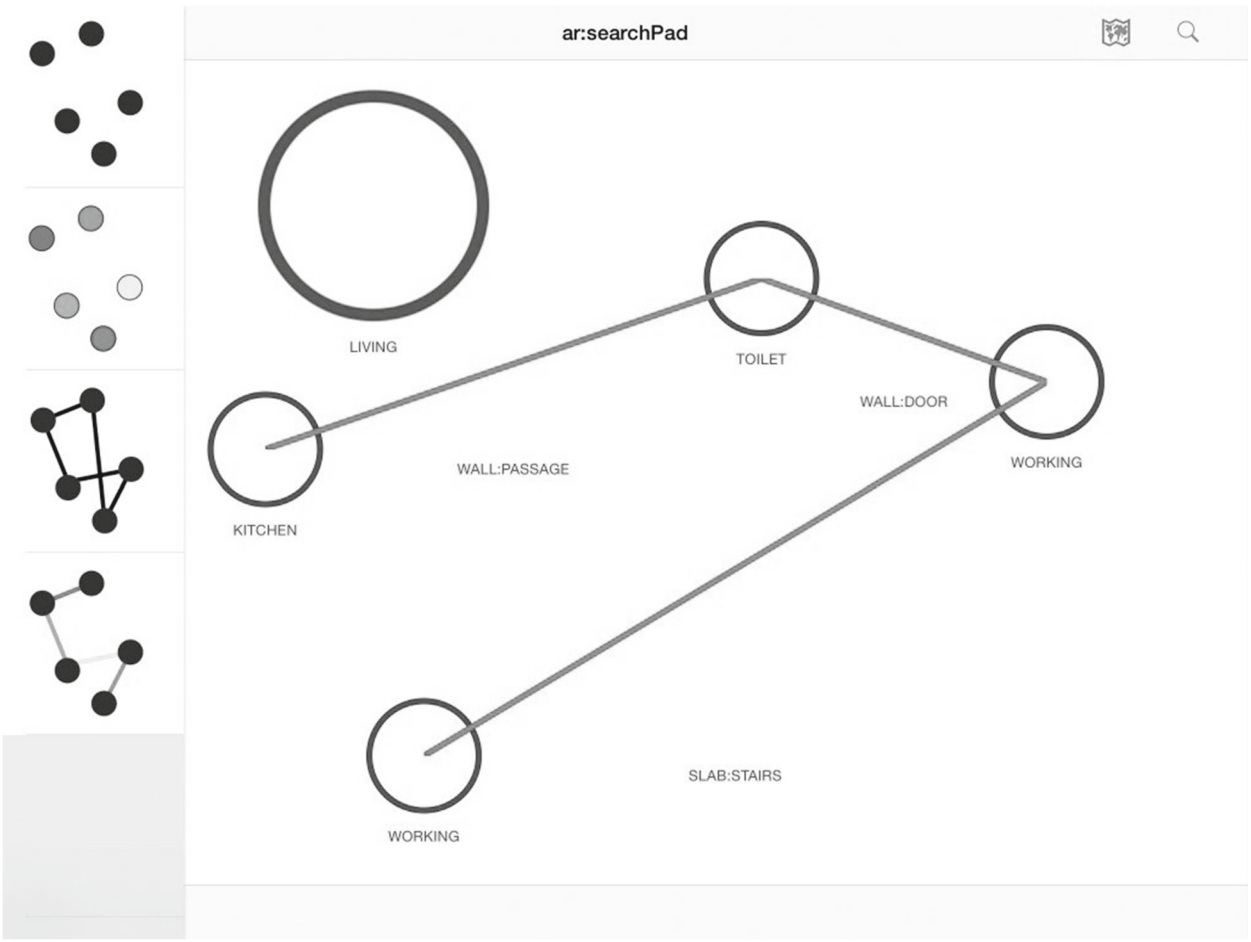

Fig. 12 Another client using bubble diagrams

node a child of the original AND/OR node. Some agents may support subtrees, but not for all types of conditions/fingerprints they support or not if those conditions occur in the same tree. When subtrees are formed via node extraction, agents receive the set of all children of the containing AND/OR node that have been assigned to that agent. They can then cherry-pick the nodes that can be executed together, returning zero or more compatible subsets which can be extracted as assignable subtrees (Fig. 17).

Query execution

After query preparation, all server- and client-side decisions that can affect the returned result set have been taken (based on rules and the server configuration) and specified as part of the rewritten query. The next phase deals with executing queries efficiently.

When combining intermediate result sets, duplicates need to be merged. All elements need to have a common identifier that is available in all data sources. These identifiers are encapsulated in packets, which can carry

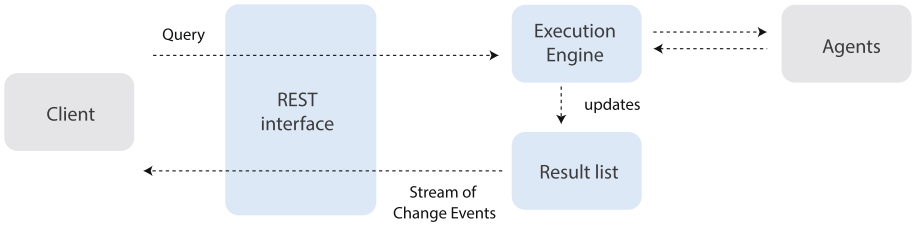

Fig. 13 Overview of system components

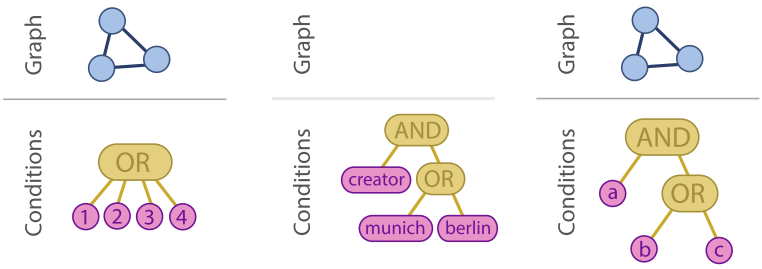

Fig. 14 Example queries: a query with conditions, a graph with fingerprints and a more complex combination of both

associated information, including the set of fingerprints matched so far and the current score of a result. Merges occur while results move through the execution DAG or when a result is added to the result list and an identical result is already present. For example, if two results with the same identity arrive at an AND node, they are combined before being pushed onwards. Packets support a content-specific merge operation. For example, the associated score values are merged into a new combined score.

All conditions in a query have two optional attributes 'fixed' and 'weight' which can be set to an arbitrary rational number by the client which sends the request. If a result is selected by a particular condition, both values are summed and added to the total score for that result. The 'weight' is multiplied by an agent-determined score between 0 and 1 that indicates the quality of the match. Clients can therefore use the 'fixed' value to indicate the importance of the condition independently of the quality, and the 'weight' part to affect the importance of the quality score. Formally, for a result x, the combined rank (before normalization) is currently computed as:

$$rank(x) = \sum_{i=0}^{n} \chi_i(x) \cdot \left(f_i + w_i \cdot s_i(x)\right)$$

where n is the number of conditions, $\chi_i(x)$ is the indicator function for the ith condition, f_i and w_i are the 'fixed' and 'weight' values defined as part of the query and $s_i(x)$ is a score computed by the agent.

Agents as data filters

Agents internally access the complete set of all buildings in the database and produce a subset. Instead of having agents operate on the underlying database set, it would be useful to get them to operate as filters on intermediate

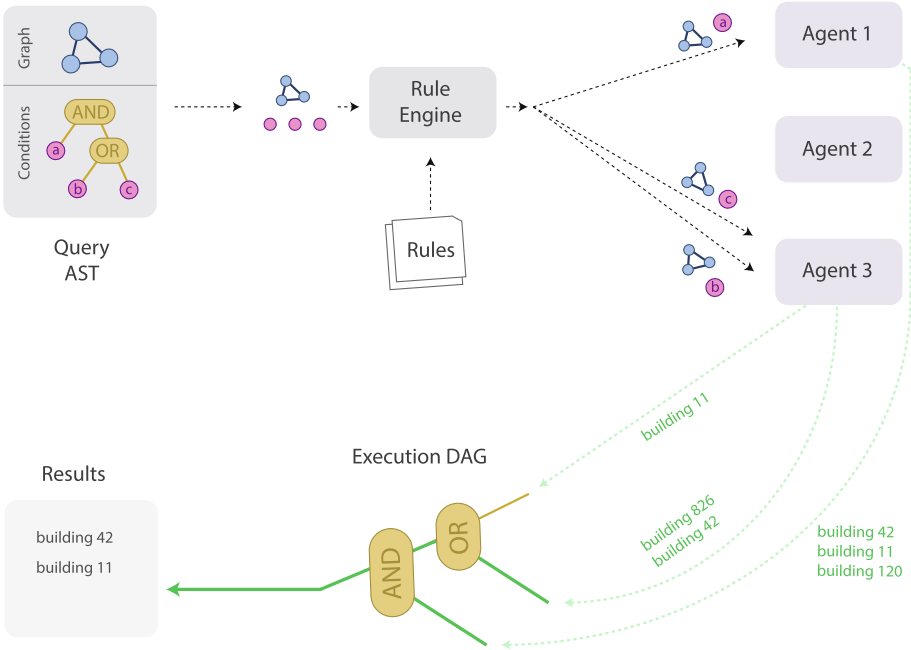

Fig. 15 Query Execution Overview

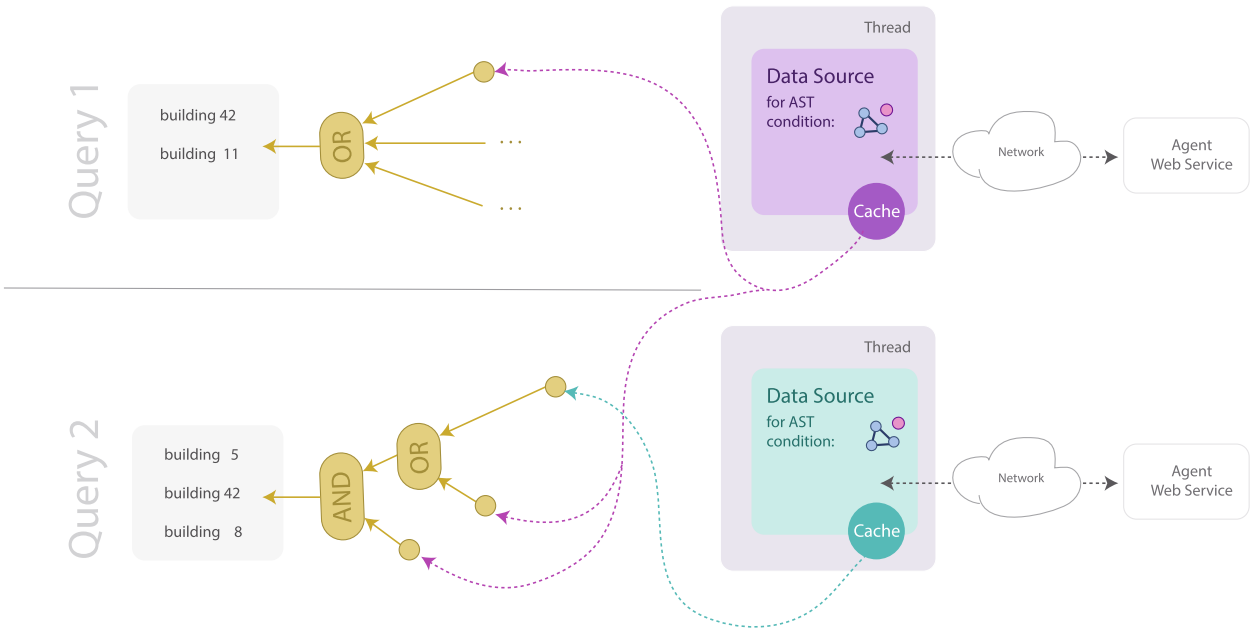

Fig. 16 Data sources are instantiations of agents that communicate with the agent microservices, can be shared across queries and push result candidates through the DAG

sets. Filters have two advantages: First, since intermediate input sets are likely to be smaller than the full underlying set, filters only have to operate on the smaller set and are likely to be faster. Second, filters could potentially depend on the structure of their input when used sequentially. For example, a fingerprint implemented using a filter might assume that its input set already has certain properties, due to being produced by another fingerprint. This can simplify the implementation of an agent. Since the data has to come from somewhere, some agents will always be data sources. Another limitation of filters is that the input set needs to be copied over the network into the agent. Finally, some agents cannot benefit from processing an input set, e.g. if they operate on pre-computed sets of the full database.

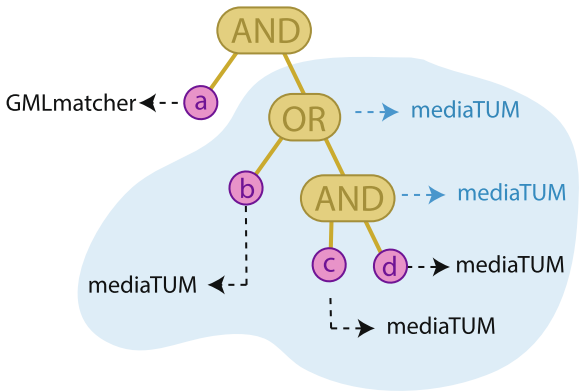

Fig. 17 Subtrees can be executed by a single microservice

Caching and query equality

Coordinator queries are dynamic and updates can happen in both directions: Clients frequently adapt queries in response to user input and result lists change during the execution of long-running queries. To simplify the interface and the clients, clients will always issue new queries. To achieve acceptable performance, the server needs to be able to determine which parts of previous queries are semantically equal and can therefore be reused. We support a query cache, which shares the result list of an existing query and only attaches a new change event listener. In order to determine if a query can be reused, the new query is transformed as usual and the transformed AST is compared to the transformed AST of previous queries. If the ASTs are equal, the results must also be equal and the result list can be reused.

Usually queries are not completely equal: Some parts change due to user interaction with the query sketch. Queries issued by different users will almost certainly be different, especially if they contain a floorplan graph. It is therefore important to also cache intermediate results that are produced by agents. If a matching data source is found, it is reused instead of creating a new one. The most difficult case is dealing with fingerprints. Fingerprints are a special type of condition and they are recursively checked for equality, just like normal conditions. Usually they have

no or few parameters, so the condition node in the AST will often be syntactically equal for fingerprints of the same type. However, fingerprints additionally always refer to a floorplan graph that was passed as part of the query and this graph must match, too. Moreover, the way it must match depends on the type of fingerprint. We implement the graph equality check by defining a pair of helper functions for each type of fingerprint that compute a hash code and provide a function to check equality of two AGraphML graphs respectively. In the example in Fig. 18, the intermediate result set computed by the 'room-count' fingerprint can be reused if the new graph has the same number of rooms. We assume that there is an agent that supports the 'room-count' and 'room-area' fingerprints and can execute them together.

In practice, the new subtree is not compared to all cached subtrees, but instead a hash code is computed and the AST subtree nodes and the graph equivalence helper functions contribute to this hash. The candidates with the same hash are then compared using the above procedure. Although the equality helper functions can be implemented by simply requiring syntactical equivalence of the compared graphs, such a strict comparison will reject many semantically equal graphs. Usually each fingerprint should have its own pair of functions that can perform better. Formally, each fingerprint must provide two functions for hashing and equality, respectively:

$$h_f(g) : G \to \mathbb{Z}$$
$$eq_f\left(g_1, g_2\right) : (G, G) \to \{\text{true}, \text{false}\}$$

where G is the set of all AGraphML documents. For correctness we require: if $eq_f(g_1, g_2)$ then the corresponding fingerprints f_1 and f_2 must produce the same results. This is the only requirement, but for optimal sensitivity fingerprints that always produce the same result sets should also satisfy $eq_f(g_1, g_2) = true$. Similarly for good sensitivity we want: $eq_f(g_1, g_2) \implies h_f(g_1) = h_f(g_2)$. Finally, the usual considerations for good hashing performance apply.

Visualizing results

Starting from the work processes in early design stages and keyword-based search, several methods and tools were developed, including design methods and problem solving strategies. The definition of search queries and the possibility to feed the results back into the design process was a primary concern and was investigated for each of the stages of the design process separately. The project takes a case-based reasoning approach with a special focus on the 'retrieve' step from the CBR cycle by Aamodt and Plaza. Since early architectural design stages focus on the description of the spatial situation primarily through rough sketches, graphical visualizations and drawings, the developed methods and tools also focused on visualizations, e.g. bubble diagrams and sketches. Additionally, the solution had to support certain workflows and interactions:

- Narrowing down the design and refining the graphical queries.
- Definition of multiple levels of abstraction, which include geometrical, topological and alphanumerical data.
- Supporting a flexible work process, which typically involves switching between multiple levels of abstraction when expressing queries. Additionally, users need to be able to start at a suitable point, depending on the stage of the design process.
- Precise specification through logical expressions, comparisons and set operations.
- Prioritizing the derived fingerprints.
- Deriving sequential and parallel search strategies.
- Support of a query history.

Given these goals, requirements and criteria were derived for the formulation of spacial configuration queries. This was done by reviewing appropriate paradigms for human-computer interaction, especially in terms of drawing and sketching and investigating criteria for the

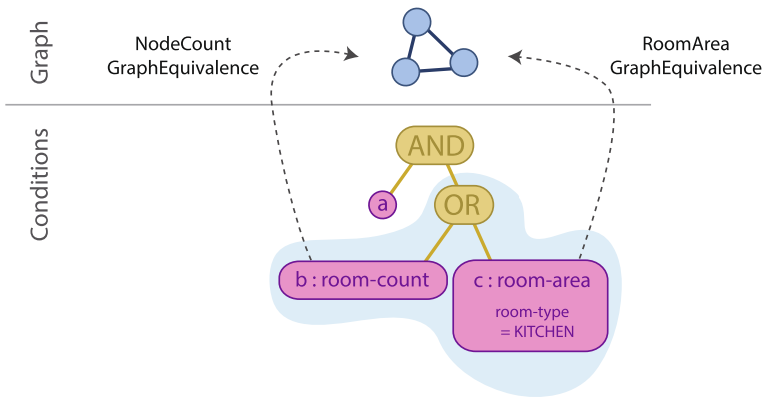

Fig. 18 Caching: Example with two fingerprints

user-oriented design of the user interfaces. An important challenge when developing user interface concepts are the individual, cognitive processes of users. The software has to provide users with a way to express their mental model and find comparable visualizations of the search results. As part of the project, ten prototype frontends were developed to elicit the requirements for the user interface concept and in particular to evaluate possible input methods and devices:

- text-based desktop applications: ar:searchbox (Fig. 6 (Langenhan et al. 2012)) and the mobile application ar:searchbox.app (Langenhan et al. 2013)
- pen-based applications (Figs. 8, 9, 10, 11): a.scatch (Langenhan and Petzold 2015), touchTect, ar:searchDroid, cormorant (Langenhan and Petzold 2016),
- geometrical modeling applications (Figs. 7, 12): ar:searchPad (Langenhan and Petzold 2016), metisWebUI (Bayer et al. 2015), flamingo (Thurow et al. 2016)

These evaluations resulted in multiple requirements for user interaction and user interface metaphors like:

- Interaction and input metaphor based on the desired information density, e.g. bubble diagrams and room volume
- Browsing and using search results as queries
- Suggestion of queries (trends, similar queries)
- User-specific weighting of search criteria

To enable the visualization, exploration and comparison of the results, visualization methods from architecture, visual communication, information and data visualization and interactive data analysis were considered. Understandability and readability were a major concern. Based on the particular fingerprints, the degree of abstraction of the input, and a manual review of the results, candidates like line graphs, scatter plots, and more interactive approaches like coordinated and multiple views, linking and brushing were analyzed using standard tools including Orange, Rapidminer, VisTrails and Aperture Tiles.

Throughout the design process, the spatial models, rooms, and their interrelationships become increasingly concrete in geometrical and topological terms. Which interactions are reasonable and required, depends on the use case and on the user. The topology of the rooms can be formalized as graphs and visualized using edge and vertex diagrams. The implemented system uses such diagrams, since they highlight the spacial relationships between rooms.

When implementing the fingerprint graph equivalence helper functions, it is worth paying particular attention to the common case where a user makes changes to the floorplan graph interactively on a client, which creates new queries after each change. Reusing computed data source sets in this case is very important to achieve good performance:

- *Identifying, adding and deleting nodes.* Some fingerprints need to identify nodes. This can't be done using a graph matching algorithm or graph canonicalization, since such an approach would defeat the purpose of caching. Instead simple heuristics need to be used.
- *Attributes are added, changed or removed for nodes or edges.* Such a change leads to cache misses if trivial implementations like the syntactical equality functions are used. Implementations can easily avoid this by providing a custom graph equivalence implementation and only comparing the attributes that can possibly affect the fingerprint.
- *Edges are temporarily removed by the same client and restored later.* If edges need to be identified on the server, this should not be done using their id, but using the ids of the connected nodes.
- *Nodes are temporarily removed by the same client and restored later.* Clients should try to pick the same node id for nodes that are likely to be the same as a node that previously existed. This is obviously the case for nodes that are restored via an undo/redo feature.

Sorted result list and change events

Packets eventually reach the root node of the execution DAG where they are inserted into an observable result list. The web service layer registers an observer for each query and translates the insert, remove, move, update, and error/warning events into an XML stream that is transferred to the client over HTTP one element at a time. Clients apply each event to a local list to synchronize it.

Clients

Clients can use the results to either show a global overview that does not visualize individual floor plans, but summarizes the resulting data with respect to the individual fingerprints or visualize each result separately. The visualization options range from simple bar and pie charts to complex mappings between query and result and the clustering of result sets. These include an overview of the found floor plans, the fingerprint-based mapping between rooms in the query and a selected result and the data analysis of the result set based on clustering. For example, the metis Web UI displays an overview of the results in the right column of the user interface by showing the floorplan structures from the CMS mediatum, sorted by similarity. Selecting such a floorplan produces

a mapping view, making it possible to evaluate the quality of this result by comparing the room configurations. The prototype client applications a.scatch, TouchTect, ar:searchdroid and cormorant accept input from freehand drawings which are created using a digital pen. The quality of the different visualizations and interactions was evaluated in a user study (Bayer et al. 2015). This initial proof of concept indicated that while most participants liked the prototypes, paper and pen were still the preferred method.

The required properties of information systems for the support of early design stages were elicited through a heuristic evaluation (see Fig. 19). The importance-satisfaction diagram shows, which properties a design tool should have in the early design stages and how well these properties can be realized in the described search scenario. Seven architects were asked to specific the importance of certain properties for searching based on freehand drawings from their point of view. Afterwards, a second questionnaire was used to assess the degree of implementability of the properties. Of the 24 properties (Fig. 19, see (Langenhan 2017, p. 88) for a complete list of properties), 17 (green area) were rated to be implementable or very implementable and further development was recommended. Two-dimensional visualizations (4.5), drawing of design ideas (4.2) and freehand drawings (3.1) had the highest priority for further development. Only the realization of schematic visualizations (4.1) required improvements (red area) and had to be revised. The aim of future user studies will be an assessment of the property 4.1 in the green area. Properties in the blue area were assigned a low priority for further development. Building information models (BIM) were rated as not important and not practicable and accordingly have the lowest priority.

Conclusions and future work

Our system selects agents using a database of rules when executing a condition in a query. The goal is to pick agents that are computationally efficient or produce good results. The rule-based approach is straightforward to implement and makes it easy to understand and to visualize why a particular decision was taken. A drawback is that it is difficult to determine good decision values for the various

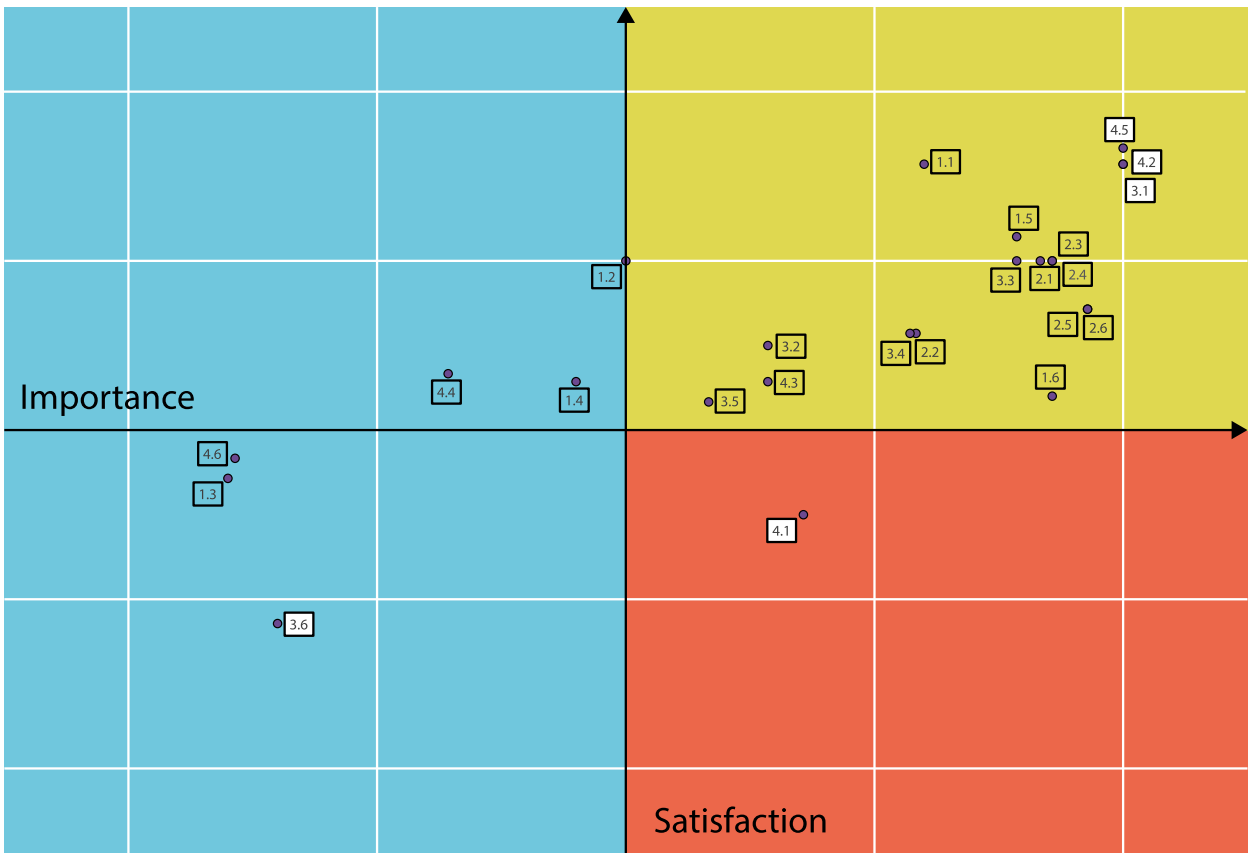

Fig. 19 Importance-Satisfaction diagram

features and that the rule database has to be maintained by hand. Instead of producing the rules by hand, it might be useful to learn them using machine learning. To keep the benefit of an intelligible system with easily traceable decisions, rule learning or decision trees are a good candidate (Flach 2012). A recent overview of rule learning is published in (Fürnkranz and Kliegr 2015). By keeping the intermediate results of executions and tracking the execution time of each agent, this approach could be taken further to learn and incrementally improve the rule set online. Clients could be extended to provide feedback on the quality of results, for example by allowing users to rate them.

Goldschmidt and Smalkov have posed the underlying question of whether visual thinking is derived from mental processes or from preceding visual images (Goldschmidt and Smolkov 2006, p. 549). The history of architecture shows that design tools have influenced how the built environment was made. The influence of tools on the thinking process is, however, hard to measure. Gaenshirt has argued that every software contains a hidden ideology (Gänshirt 2007, p. 193) which conditions every object constructed with them. It might be useful to keep a history of query results and possibly also intermediate sets permanently, for example to support the machine learning case described above or other data mining scenarios. The persisted data needs to be versioned to account for changes in the database and in the involved services.

We have described the design and implementation of the Coordinator, a middleware to support graph-based floorplan retrieval. The system accepts queries in a unified query language and executes them using agents which are selected using a set of rules. Furthermore, queries can be updated and the Coordinator achieves efficient execution using a domain-specific caching framework. Future work includes automatically learning the rules, providing support for online analysis of materialized datasets and improving the prototype implementation.

Endnotes

[1] "Data integration is the problem of providing unified and transparent access to a set of autonomous and heterogeneous sources, in order to allow for expressing queries that could not be supported by the individual data sources alone." (Keim et al. 2010, p. 23)

[2] Semantically equal means that the queries describe the same set of results, but are phrased slightly differently.

Acknowledgements
Not applicable.

Funding
The work presented was supported by the German Research Foundation (DFG) as part of the 'metis' research project.

Availability of data and materials
Not applicable.

Authors' contributions
JR is responsible for the computer-science related parts of the paper. CL and FP are responsible for the architecture-related parts of the paper. All authors read and approved the final manuscript.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1] Department of Computer Science, Technical University of Munich, Munich, Germany. [2] Department of Architecture, Technical University of Munich, Munich, Germany.

Received: 11 November 2016 Accepted: 25 September 2017
Published online: 24 October 2017

References

Bayer, J, Bukhari, S, Langenhan, C, Eichenberger-Liwicki M, Althoff KD, Petzold, F, Dengel, A (2015). Migrating the classical pen-and-paper based conceptual sketching of architecture plans towards computer tools: Prototype design and evaluation, In *GREC 2015*. Springer International Publishing. doi:10.1007/978-3-319-52159-6_4.

Beck, F, Burch, M, Diehl, S, Weiskopf, D (2014). The State of the Art in Visualizing Dynamic Graphs, In *Proceedings State of the Art Reports (STARs)*. 1605.08485 (pp. 83–103).

Buxton, W (2007). *Sketching User Experience. Getting the Design Right and the Right Design*. San Francisco: M. Kaufmann.

Conte, D, Foggia, P, Sansone, C, Vento, M (2004). Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(03), 265–298.

Dengel, A (2012). *Semantische Technologien*, (p. 461). http://link.springer.com/10.1007/978-3-8274-2664-2.

Flach, P (2012). *Machine Learning. The Art and Science of Algorithms that Make Sense of Data vol. 2010*, (p. 396). New York: Cambridge University Press.

Fürnkranz, J, & Kliegr, T (2015). A Brief Overview of Rule Learning. In N Bassiliades, G Gottlob, F Sadri, A Paschke, D Roman (Eds.), *Rule Technologies: Foundations, Tools, and Applications SE - 4*. Lecture Notes in Computer Science. http://link.springer.com/10.1007/978-3-319-21542-6_4, 9202 (pp. 54–69). Berlin: Springer.

Gänshirt, C (2007). Werkzeuge Für Ideen. Einführung Ins Architektonische Entwerfen., Basel. http://www.gbv.de/dms/hebis-darmstadt/toc/187535930.pdf.

Goldschmidt, G, & Smolkov, M (2006). Variances in the impact of visual stimuli on design problem solving performance. *Design Studies*, *27*(5), 549–569.

GraphML (2017). The GraphML File Format. http://graphml.graphdrawing.org/. Accessed 4 May 2011.

Gratzl, S, Lex, A, Gehlenborg, N, Pfister, H, Streit, M (2013). LineUp: Visual analysis of multi-attribute rankings. *IEEE Transactions on Visualization and Computer Graphics*, *19*(12), 2277–2286.

Hanson, J (1998). *Decoding Homes and Houses*. Cambridge; New York: Cambridge University Press.

Keim, D, Kohlhammer, J, Ellis, G, Mansmann, F (2010). *Mastering the information age: Solving problems with visual analytics*. Goslar: Eurographics Association.

Langenhan, C (2017). Data Management in Architecture - Investigating the organisation of design information in IT infrastructures and identifying potential uses in knowledge-based systems. Dissertation. Technical University of Munich. ISBN 978-620-2-32014-6. Online verfügbar: https://mediatum.ub.tum.de/doc/1335437/1335437.pdf.

Langenhan, C, & Petzold, F (2010). The fingerprint of architecture: sketch-based design methods for researching building layouts through the semantic fingerprinting of floor plans. *International electronic scientific-educational journal: Architecture and Modern Information Technologies*, *4*(13), 1–8. http://www.marhi.ru/eng/AMIT/2010/4kvart10/langenhan/abstract.php.

Langenhan, C, & Petzold, F (2015). BEYOND THE BUBBLE – Computer-aided topological analysis and parametric de-sign of room configurations in university education, In *eCAADe 2015 – 33rd Annual Conference 16th–18th September 2015. Vienna*: eCAADe (Education and research in Computer Aided) Architectural Design in Europe) and Faculty of Architecture and Urban Planning, TU Wien.

Langenhan, C, & Petzold, F (2016). Perspectives on architecture: Different abstraction layer of building information imply special working methods and interaction metaphors to support a variety course of action, In *Sigradi, 2016. XX Conference of the Iberoamerican Society of Digital Graphics*.

Langenhan, C, Sahm, A, Seifert, A, Teichert, A, Petzold, F (2013). Mobile application to collect information about architecture to obtain a collective knowledge base, In *ASCAAD 2013*.

Langenhan, C, Seifert, A, Teichert, A, Petzold, F (2012). ar:searchbox: Knowledge management for architecture students, In *eCAADe*: eCAADe (Education and research in Computer Aided) Architectural Design in Europe) and ČVUT, Faculty of Architecture.

Langenhan, C, Weber, M, Liwicki, M, Petzold, F, Dengel, A (2013). Graph-based retrieval of building information models for supporting the early design stages. *Advanced Engineering Informatics*, *27*(4), 413–426.

Liebich, T (1994). *Wissensbasierter Architekturentwurf. Von Den Modellen des Entwurfs zu Einer Intelligenten Computerunterstützung*. Weimar: VDG, Verlag und Datenbank für Geisteswissenschaften.

Richter, K (2010). *Augmenting Designers' Memory. Case Based Reasoning in der Architektur*. Berlin: Logos-Verlag.

Roith, J, Langenhan, C, Petzold, F (2016). Supporting the building design process with graph-based methods using centrally coordinated federated databases, In *ICCCBE*. Osaka.

Roth-Berghofer, TR, & Richter, MM (2008). On explanation. *Künstliche Intelligenz*, *22*(2), 5–7.

Thurow, T, Langenhan, C, Petzold, F (2016). Assisting early architectural planning using a geometry-based graph search, In *eCAADe 2016*.

Vatavu, RD, Anthony, L, Wobbrock, JO (2015). $P Point-Cloud Recognizer. https://depts.washington.edu/aimgroup/proj/dollar/pdollar.html.

Weber, M, Langenhan, C, Roth-Berghofer, T, Liwicki, M, Dengel, A, Petzold, F (2010). a.scatch semantic structure for architectural floor plan retrieval. In I Bichindaritz & S Montani (Eds.), *ICCBR*, 6176 (pp. 2010–6176510524). Berlin: Springer.

Weber, M, Langenhan, C, Roth-Berghofer, T, Liwicki, M, Dengel, A, Petzold, F (2011). Fast subgraph isomorphism detection for graph-based retrieval. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6880 LNAI*, 319–333.