Collaborative Network for Industry, Manufacturing, Business and Logistics in Europe

D3.3

Product and Service Search Engine and Search Mediator

Project Acronym	NIMBLE	
Project Title	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe	
Project Number	723810	
Work Package	WP3 Platform Technology Specification	
Lead Beneficiary	UB	
Editor	Shantanoo Desai	UB
	Marco Franke	UB & BIBA
Reviewers	Wernher Behrendt	SRFG
Contributors	SRDC, SRFG	ALL
Dissemination Level	PU	
Contractual Delivery Date	30/09/2017	
Actual Delivery Date	11/12/2017	
Version	V1.0	

Abstract

The NIMBLE project aims to perform research leading to the development of a cloud and IoT platform specifically targeted to supply chain relationships and logistics. Core capabilities will enable firms to register, publish machine-readable catalogues for products and services, search for suitable supply chain partners, negotiate contracts and supply logistics, and develop private and secure information exchange channels between firms. The intention is to support a federation of such NIMBLE instances, all providing a set of core services, and each potentially specifically tailored to a different aspect (regional, sectorial, topical, etc.).

This document explains the search capabilities of the proposed platform as implemented in the software prototype resulting from task 3.3.

This deliverable describes the design, intended use, and validation of the search functionality of the NIMBLE platform. It describes how the ontological representation of products, services and other catalogue specific information could be made searchable by users of the NIMBLE platform.

We start with a high-level description of the architecture and then give more detailed explanations of the addressed requirements and boundaries defined by the NIMBLE architecture. Common search scenarios are defined and it is shown how they are supported by the search capabilities. Finally, a summary and outlook to further work is given.

Document History

Version	**Date**	**Comments**
V0.1	11/08/2017	Table of Contents
V0.2	12/09/2017	First Draft
V0.3	21/09/2017	Revision of 1st draft by WB
V0.4	13/10/2017	Integrate contribution from SRDC
V0.5	19/10/2017	Change structure with respect to call with WB
V0.6	25/10/2017	Add requirement chapter, evaluation chapter and proofreading
V0.7	06/11/2017	Revision of final draft (WB)
V0.8	10/11/2017	Final Revision
V1.0	08/12/2017	Final Version

Table of Contents

List of Figures

List of Tables

Acronyms

Table 1: Acronyms

Acronym	**Meaning**
B2B	Business-to-Business
IoT	Internet of Things
IIoT	Industrial Internet of Things
SEMed	Semantic Mediator Front- end Backend
JSON	JavaScript Object Notation
POJO	Plain Old Java Object
QBF	Query By Form
DoA	Description of Action
FITMAN	Future Internet Technologies for Manufacturing Industries

Glossary (ALL)

- Angular – develop applications for different deployment platforms
- Apache Jena - Java framework for building Semantic Web applications, by handling storage and query capabilities of RDF triples
- ElasticSearch - high-performance indexing and search system that can be integrated with the CouchBase scalable data back-end.
- Eureka (Netflix) - Service registry and discovery component
- External applications – Independent applications that are developed and hosted outside the platform but internally make use of NIMBLE applications.
- NIMBLE applications – cloud and IoT applications, which make use of NIMBLE core Services, and are deployed within the NIMBLE cloud platform
- Platform-as-a-Service - a cloud computing concept which offers the developer and deployer of cloud based applications the infrastructure, both HW and middleware, needed for creating and deploying successfully such applications on a cloud environment.
- RDF - A standard model for data interchange on the Web. Used in NIMBLE for storing entities description in a searchable manner.
- SPARQL – an RDF query language.
- Intensional Queries – queries relating to the schema or conceptual structure of some database or object store. This is needed when the underlying semantic model is not yet known at the time of querying.
- Apache Solr – Enterprise search platform with features like full-text and faceted search and database integration

1 Introduction

NIMBLE is aiming at creating a B2B platform specifically geared towards improving the efficiency of supply chain creation and operations. At its core lay several services which enable companies to publish digital versions of their catalogues, containing the range of products they sell and business services (e.g. transportation, packaging and so on) they provide, and in turn enables other companies to efficiently search and find required counterparts. In addition, once potential partners find each other they are able to initiate a negotiations process through the platform and finally establish a supply chain relationship among them, including the creation of private information exchange channels.

The search for business partners is one of the core services of the NIMBLE platforms. It allows the look up of products, services, and associated suppliers from a wide range of heterogeneous catalogues. The total number of entities is large and spreading over many different categories. The conceptual structure of the categories is not flat but rather it contains multistage taxonomies and similar properties. The selection of a specific category or just entering a keyword would retrieve too many results, because specific products differ only in a few properties from the others. To include the formularisation of property based filters, additional search capabilities are required.

An appropriate search mechanism takes all information of an entity into account, which includes its location in a taxonomy as well as all entities' properties. To enable this, a formalization of the catalogues lies also in the focus of NIMBLE. Such integration of the different catalogues into NIMBLE will result in a common terminology. This common terminology is independent of the specific catalogues and allows the specification of search terms based on robust taxonomies.

To achieve these search capabilities, an efficient search is necessary which offers the power of established query languages as well as an intuitive user interface. For that purpose, the NIMBLE search is based on three requirements:

1. Use a query mechanism that covers search at different levels of granularity
2. Provide a graph-based QBF access method which is designed for users without technical skills
3. Apply the user context to support the search term specialization

To achieve this, two kinds of searches are in the focus of this prototype. Both types rely on formularized catalogues which are available in triple stores. First, a faceted search is presented which uses preconfigured indexes to define the possible amount of search terms. The framework Apache SOLR is applied for indexing. Second, two variants of search are presented, which use intensional queries to unfold the search space relating to the user input. The second option doesn't apply pre-configured indexes or static configuration. It visualizes the current amount of available concepts and properties. Each ontological change is visible to the user immediately. Therefore, it is applicable for all varying catalogues. In the following, both kinds of searches are presented in detail.

1.1 Requirements for NIMBLE search capabilities

NIMBLE should provide support for users to find new providers of supplies and services, and to initiate the specification of new supply chains that can be activated to become real business relationships. In the following, we map all stated requirements from D1.1 and from the DoA and show how the requirement is covered by the search capabilities that we provide.

The following table shows the mapping of functional and non-functional use case requirements to the relevant search functionalities / elements. The requirements in the first column are prefixed with "REQ" for Requirement, followed by an abbreviation indicating the partner / use case (e.g. MIC = Micuna; PIA = Piacenza; LIN = Linbäcks; WHR = Whirlpool) and a running number. After the prefix a short name for the requirement is given and followed by a description. The second column is the intended output that will support and fulfil the requirement.

Table 2: Requirements-to-Architecture Mapping

Requirement	**Respective search capability**
REQ_MIC_01 Provider search The manufacturer aims at finding providers of required materials and operations which cannot cover by its own resources or aims at improving at different levels (i.e.: operational costs, ultimate quality).	The search is capable to find every piece of information which is inserted into the common NIMBLE catalogue.
REQ_MIC_07 Search for supplier or logistics operator The company (Micuna) enters NIMBLE and searches the ecosystem introducing keywords related to seek resource. (Technical characteristics of required material or component defined by Micuna)	The different search types provide a keyword driven search which is partially or completely keyword driven.
REQ_MIC_16 Publication of product catalogue A company publishes part of their product catalogue in the platform, in order to be visible to other supplier and manufacturing companies for making business together or establish collaboration relationships. Information introduced in the platform may have a public and a private side, sharing the information of the private side only with selected potential collaborators.	The user driven access rights will be handled by the overall NIMBLE platform instead from each NIMBLE core service on its own.
REQ_MIC_28 Reliability If the platform collapses due to any cause, such as unexpected concurrent access, error connecting to databases, or temporal server downfall, the system should notify users about this situation to avoid bad user experiences. Server clustering or other techniques could be adopted to recover from these faults.	The search won't cache any search results and relies on the current available data. Therefore, no inconsistency of search results with respect to the NIMBLE catalogue could occur.

REQ_MIC_31 User-friendly Easy to follow processes and guidance on NIMBLE platform, e.g. by templates helping inserting appropriate data.	The search guides user in case of missing catalogue knowledge. Therefore, intentional queries are applied to show which kinds of products/services are available and which properties of a product/service could be applied for the search.
REQ_PIA_07 Share design and production data Company A (e.g. fabric supplier) can share design or production data with company B (e.g. its customer, a clothing producer).	The search uses as main data source the Apache Marmotta instance of each NIMBLE platform, which holds the catalogues. In case of necessity, a data integration solution is included to redirect search queries to common data sources.
REQ_PIA_12 Access to Virtual Catalogues and Services of Supplier Dynamic, real-time access to supplier virtual catalogues and services for fast design development.	The search is capable of finding every piece of information that is inserted into the NIMBLE catalogue.
REQ_PIA_13 Catalogue for Piacenza From the technical point of view (and requirements for the NIMBLE core system) we need to underline that the catalogue for Piacenza includes not only the products for the customer, but also the services for the collaboration platform.	The search is capable of finding every piece of information that is inserted into the NIMBLE catalogue.
REQ_PIA_31 User friendliness The platform must be user friendly and properly supported by instructions and tutorials, in main user languages. (connected to REQ_PIA_12)	The search guides user in case of missing catalogue knowledge. Therefore, intentional queries are applied to show which kinds of products/services are available and which properties of a product/service could be applied for the search.
REQ_LIN_01 Product Configurator The customer is able to make changes on the features and properties of a bath room that will be part of the flat in a future eco house. This will be realized by a bath room product configurator.	The search enables the user to look for potential products which could be replaced in complex products like a bath. Each search result could be used for a negotiation.
REQ_LIN_03 Access & Edit Database A user is able to add, update and remove database entries. The data itself is protected and owned by the user who pushes data into the database. The data can be shared,	The search is capable of finding every piece of information that is inserted into the NIMBLE catalogue. The user driven access rights will be handled by

when the data owner gives access rights (read-only, accessible, read-write). An optional approach could be the usage of roles a data owner can give to others.	the overall NIMBLE platform instead from each NIMBLE core service on its own.
REQ_LIN_22 Integrate quality data into Database Based on available information of "Rumsa" specification document, data already available from production (LPS-Lindbäcks Production System) and a complementary RFID solution to identify parts (up to modules) a tracing shall be realized on production and construction level.	The search is capable of finding every piece of information that is inserted into the NIMBLE catalogue.
REQ_WHR_13 Key Based Search The user will be able to enter two keys – product model and serial number. Based on the profile the user is now served with pre-filtered / dedicated information, that is based on their profile.	The different search types provide a keyword driven search which is partially or completely keyword driven.

2 Approach for Search Mechanisms

2.1 Types of Search

All search variants offer a keyword-based search as a starting point. Subsequently, the specification of the search term is offered via a Faceted Search and via an Explorative Search. The main difference between both search mechanisms is in the search scope, which is handled differently in each case. While the Faceted Search offers a single search scope and narrows it, the Explorative Search allows the user to define multiple scopes iteratively and each of them can then be narrowed again.

Both kinds of searches rely on product catalogues that must be accessible for this purpose. The Catalogue Ingestion process and appropriate search types are presented in Section 2.4 in detail.

As described in D3.2 Catalogue Ingestion and Semantic Annotation, published catalogue data is stored in different storages to be able to provide different search capabilities. In this section, we present the search modalities enabled by the ingestion mechanism illustrated in Figure 1.

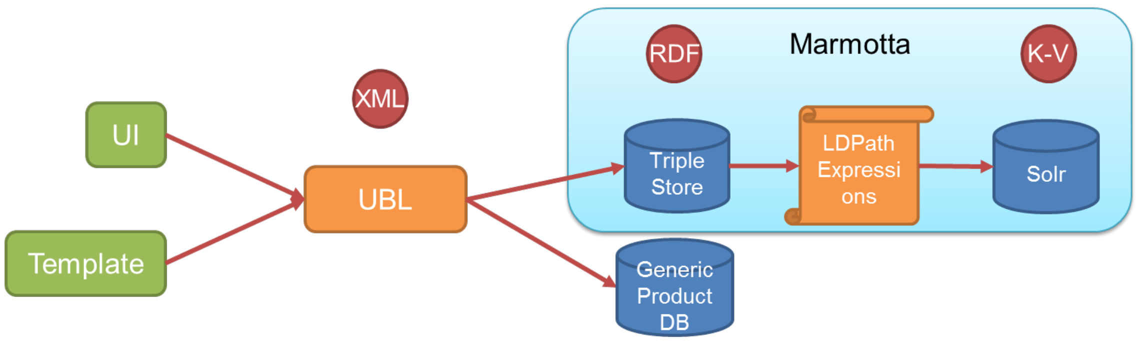


Figure 1 Catalogue Ingestion Mechanism

The cylindrical elements *viz.* Triple Store and Solr, Generic Product DB; show the components storing the published data. The data compliant to NIMBLE's UBL-based data model is first stored in a relational DB and also in a triple store as RDF triples. Furthermore, the RDF data is post-processed according to a certain configuration and stored in a free-text index using the Apache Solr tool[1]. Based on these storage modalities the following search modalities are available:

SQL Based Search: The relational database storage option provides SQL-based querying of the managed data. This kind of search is mainly used by internal components to gather information about a catalogue or individual products.

Keyword Search: Keyword-based search is the initial step performed by users who can then continue with the Faceted Search or the Explorative Search. It is supported by both the free text-index to retrieve the list of products matching with the given query term and by the semantic search component to retrieve matching companies, product or any other concepts.

Faceted Search: Faceted search is the de facto search mechanism of e-commerce platforms. Upon initiation with a keyword, the results are presented along with an applicable set of filters for narrowing/broadening the results.

Explorative Search: The goal of this search is to define iteratively in a user driven process which properties of a product or service are relevant for the current search request. Only the selected properties are used to show the user concrete products or services (search result). The explorative behaviour of the search is supported by a graph-based navigation, which allows the iterative exploration of the search space. In so doing, the user learns the available information (product structure and specific instances) for a product/service while he is exploring the search space. Intensional queries are intensively used to support this feature.

2.2 Federation of Information

Product and service catalogues may be heterogeneous in terms of syntax as well as semantics. To enable a search beyond the corporate boundaries of individual catalogues, a common terminology and dynamic data acquisition is necessary. The common terminology is being developed in T2.2 in the form of an ontology. The developed ontology will be deployed on the Apache Marmotta Linked Data Server in the NIMBLE platform. Therefore, the available product and service categories and their definitions could be requested by intensional queries from Marmotta's linked data repository.

The NIMBLE platform offers a manual functionality to upload product catalogues via Excel sheets. Therefore, the continuous data integration of catalogues in the NIMBLE Platform is possible. Apart from this functionality, the catalogue-search includes the Specific Enabler

[1] http://lucene.apache.org/solr/

SEMed[2]. It enables a search for information which is not located inside the NIMBLE Marmotta instance but rather the information is available in heterogeneous legacy systems.

In general, the FITMAN Semantic Mediator Front- and Backend (SEMed) is a mature middleware layer for semantic, virtual interoperability and integration specifically of item-level product lifecycle data. It facilitates a standards-based access to PLM data, for example through its support for the Open Group QLM Standard Open Messaging Interface (O-MI) and Open Data Format (O-DF). At the same time, it provides semantic interoperability for different kinds of common data sources like databases and file-based repositories. It introduces a layer of semantics on top of existing syntactic data structure descriptions to avoid semantic integration conflicts and allows a scalable, efficient and comfortable interoperability of product data across all of the stakeholders and IT systems.

Currently, the catalogue-search uses a local ontology or a triple store to retrieve the data, which includes all necessary information. As a consequence, SEMed is currently not necessary to get access to all relevant data. To enable the potential integration of potential future data sources, SEMed is integrated as libraries (Open Source licence (GPLv3)) into the search and is accessible via a web service method. Currently it offers just the functionality to configure the data sources and to retrieve the data with respect to a SPARQL query that uses a subset of SPARQL terms. In addition, SEMed could be extended to request the data from different catalogue data sources and add them automatically to a triple store like Apache Marmotta. For that purpose, SPARQL queries describing the information to be inserted, configuration files of the data sources and the appropriate access rights would be necessary. If everything is configured, the catalogue search could trigger SEMed to add new data each time the catalogue-search service is triggered.

2.3 Derivation of Search Request

The NIMBLE platform represents conceptual information (e.g. schemas, product categories, etc.) as connected ontologies. To make the available concepts accessible for user queries we combine traditional information retrieval methods with semantic technologies based on RDF and triple stores.

The design activities in T2.2 proposed the merging of ontologies that include many thousands of concepts and corresponding properties. For an appropriate search functionality, we cannot assume that the user knows all concepts and how they are interweaved. Moreover, the goal to add companies and corresponding products and services to the NIMBLE platform requires continuous extensions of the TBox and the ABox. That implies, that the body of terminologies varies over time. This is why we need to incorporate TBox-related (i.e. intensional) queries into the search or to add regularly predefined indexes.

The root of each ontology is the concept THING. Below the concept THING, a couple of concepts are usually applied to formulate the generic catalogue entities as well as the business processes. This basic structure is necessary but from the user perspective unnecessary to know or to use. On basis of this generic structure, a product and services taxonomy is inserted in the

[2] http://www.fiware4industry.com/?portfolio=semantic-mediator-semed

NIMBLE ontology. In consequence, the traversing from the root to any kind of a specific product or service includes a couple of concepts and properties which are meaningless for the end users. To skip the traversing of the ontological framework, direct access to the concepts is necessary.

The main idea is to achieve direct access using keywords. In all implemented search types, the user specifies a keyword. Then, a backend service determines which indexed data or which part of the ontology could be applied. The search result differs for the search types. While the Facetted Search by SOLR delivers a list of concrete products or service, the other search types require additional steps to estimate the appropriate search result. These additional steps protect the user from obtaining too many product entries and for each product too much information.

2.4 Usage Scenarios

A user of the NIMBLE platform searches for products, services, or a product in combination with a service. For each product or service, the user wants to apply filter to express his requirements for the desired product and service.

From the business process perspective, the following examples are common search requests from the MICUNA case, and are transferable to other companies' business processes:

1. "Search for a service with respect to a defined set of attributes and enable the negotiation for the best candidate"
    a. Micuna employee searches for a logistics provider who can transport some purchased products from Spain to France and then, starts the negotiation for the price.
    b. Micuna employee searches for a logistics provider who can deliver to France and is capable of deliver items that have a width of 3 meters and then starts the negotiation for the delivery time.
2. "Search for a specific product which has specific properties and order the product"
    a. Micuna employee searches for a MDF Board in yellow and wants to order 90 pieces.
    b. Micuna employee searches for a MDF Board in yellow where the manufacturer should be from Europe and should meet EU regulations for certain chemicals in the glue. Subsequently, the employee wants to order 90 pieces of the chosen product.
    c. A customer searches for a high chair that has a specific height. The customer wants to order one single piece.

The above listed common search requests have in common that a user starts from a specific product, supplier or direct service and navigates from this item to item-specific properties or linked other items. Linked items, for example, are the manufacturer of a product, the regulations that a product meets or specific properties of the logistics provider's transport vehicle.

These search types exhibit the task flow which is illustrated in Figure 2.

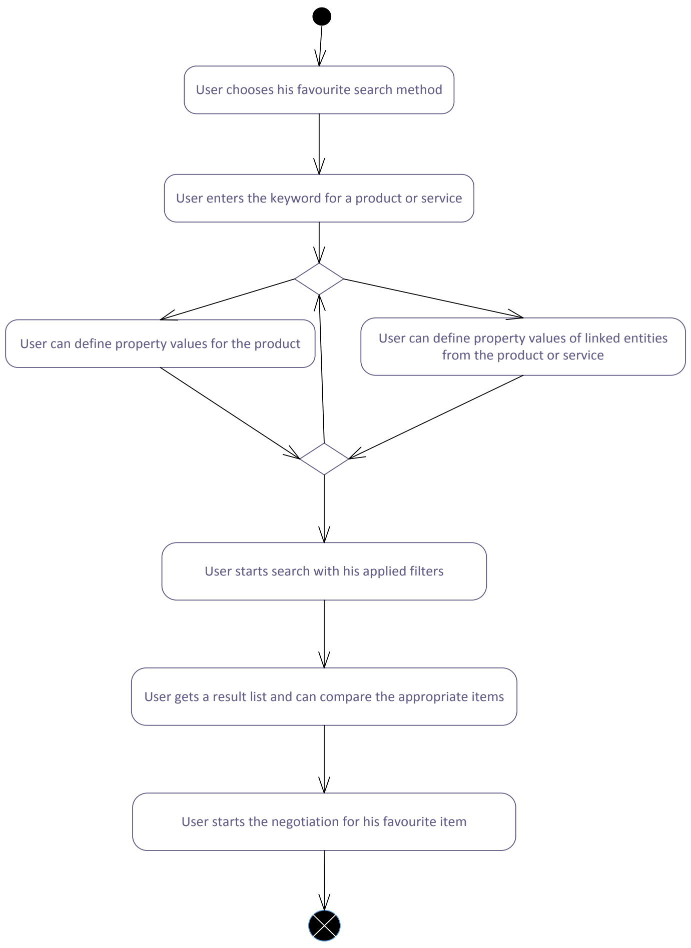

User chooses his favourite search method
User enters the keyword for a product or service
User can define property values for the product
User can define property values of linked entities from the product or service
User starts search with his applied filters
User gets a result list and can compare the appropriate items
User starts the negotiation for his favourite item

Figure 2 Search for a product or service

The following subsections describe the search for a product as shown in Figure 2 for the search types *graph based navigation*, *keyword based search* and *semantic query patterns based Search*. All presented usage scenarios apply the different search types on the basis of real data which has been provided by the use case partners and is uploaded into Apache Marmotta from the NIMBLE platform.

2.4.1 Graph Based Navigation – Explorative Search

Assumption

This specific search type assumes that the user has a product in mind, but only knows a very generic term for the product, e.g. "chair". This could be a domestic chair, it could be a high chair for children or it could be an office chair.

Goal

The goal of graph-based search is to give the user complete freedom of the product selection. The generic search model for the Explorative Search is more user-centric, keeping in mind that user is looking for a kind of product but is not sure about the specific properties of the product.

The figure below shows the user interface for the Explorative Search on the NIMBLE platform, which can be considered as the initial step towards the search query. The precondition is that the first step has been done successfully.

As the second step of the search process (see Figure 2) a search bar is provided, where the user enters the desired keyword for a product. Multilingual queries are supported. The default search is conducted in English, however for the current implementation the user can also search queries in Spanish. The multilingual feature within the search is introduced to cater to a larger diversity of users who shall benefit from the platform. The search history area is provided in order to keep track of keywords that were searched previously by the user.

User Interaction

The user for this scenario is looking for a chair, hence only the keyword *chair* needs to be entered in the search bar. The default language is English.

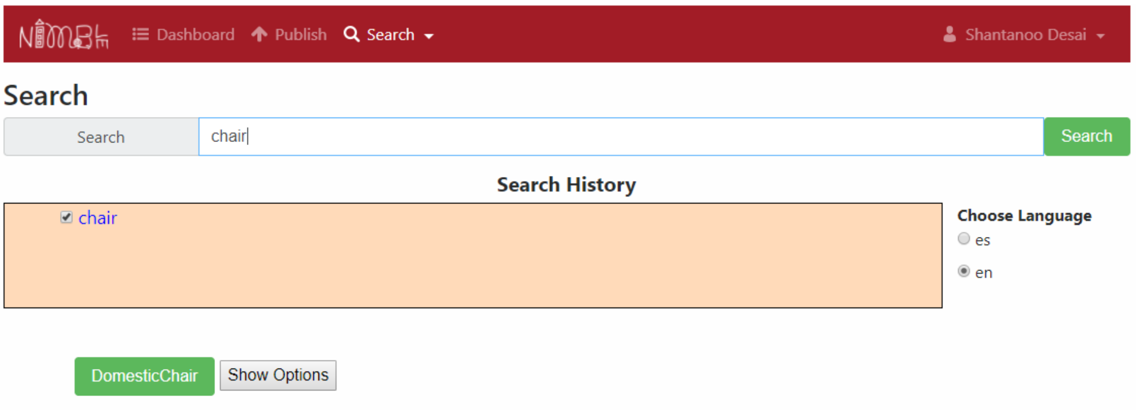


Figure 3 Initial Output for the Generic Search Keyword of Chair

Results

When the user enters the keyword and clicks the Search button, the query is processed in the backend and it provides the end user with the buttons shown in Figure 4. For brevity, the multiple buttons are not shown all at once, but the user can view all the options by clicking on the Show Options button. The keyword *chair* now also appears within the Search History area to indicate the choice the user has searched for. The figure below shows all possible search results for the keyword *chair* when the user clicks on Show Options button.

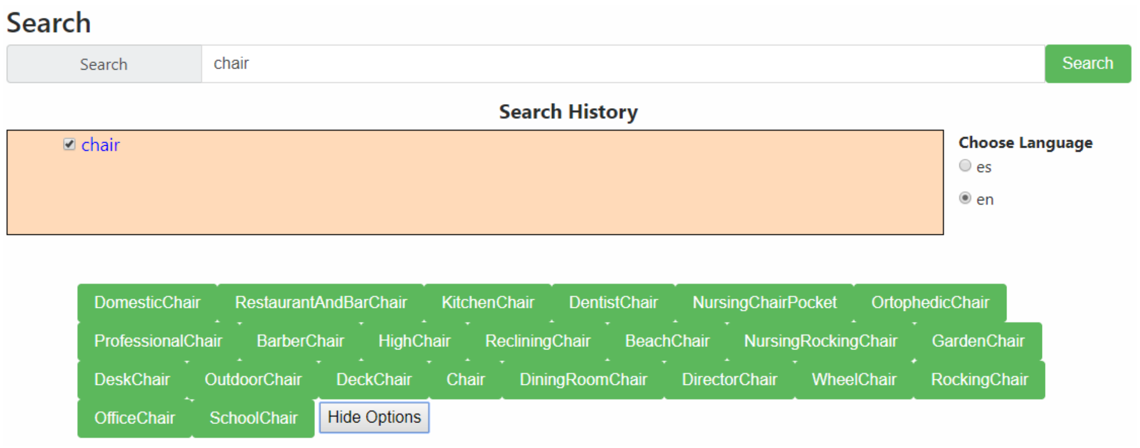


Figure 4 All Possible Outcomes for the Generic Search Keyword of Chair

As mentioned previously, the user inputs a very generic word like *chair* and the backend service provides all the possible products that are available with the word *chair* in it. For instance, the user might have a preference for buying a high chair rather than an office chair. The resulting options provided to the user are in the form of clickable buttons for ease of interaction.

The search is not limited to single keywords. The user can further decide to search for a generic term like *table* and obtain similar results and can choose which particular table product of interest can be further explored. However, it is worth mentioning that different search results are independent and would bear no relationship among each other. Hence, when a user searches for a chair and then a table, they are considered to be two independent search queries and search results.

Figure 5 illustrates the search outcomes for a keyword *table*. With all possible outcomes from the query of *table*, the user can obtain a wide range of products which can then help channel towards a specific product. The keywords searched by the user are available within the Search History area, which are initially checked with a checkbox. The checked value indicates that the user is still interested in the keyword. However, if the user wishes to hide or delete the outcomes of a searched keyword, it is possible by simply unchecking the keyword in the area. By unchecking the keyword, the corresponding search result is hidden and a delete button is provided, which upon clicking will finally remove the keyword and the outcomes from the User Interface.

Figure 6 describes the scenario where the user wishes to permanently remove the outcome for the search keyword of *table*. Upon unchecking the keyword, a red delete button appears next to it, and upon clicking the button the keyword in the area and all subsequent outcomes with respect to the keyword are removed from the User Interface. If the user enters the third keyword like *window*, it would appear in the history next to chair and the corresponding button list would appear below the list of keyword results for *chair*.

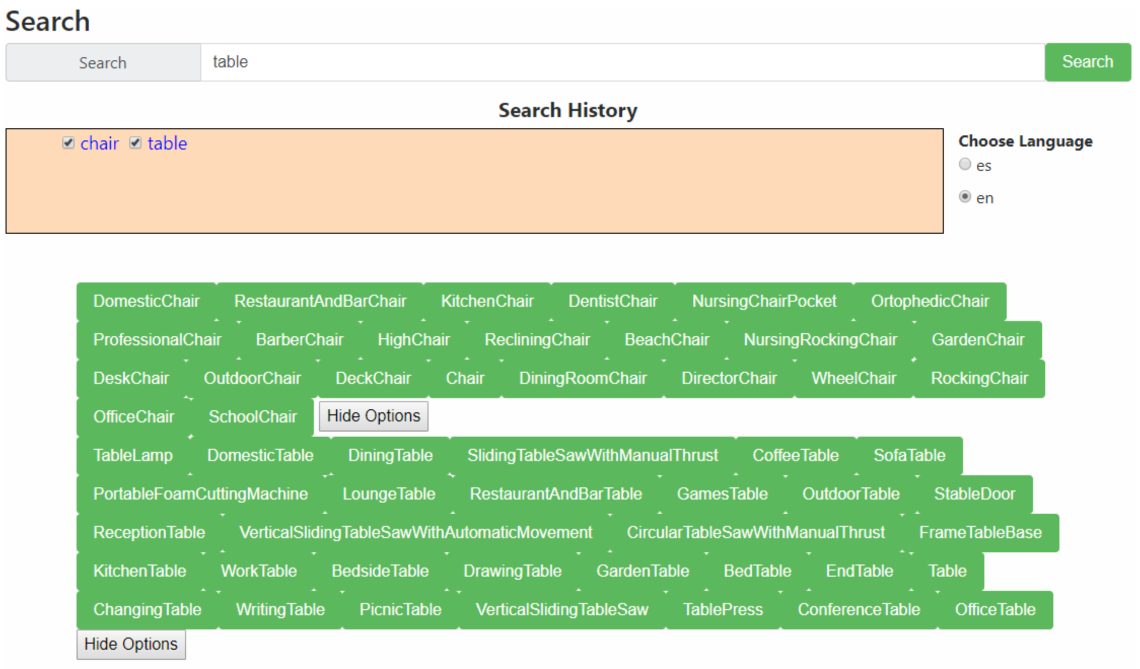

Search
Search
table
Search
Search History
☑ chair ☑ table
Choose Language
es
en
DomesticChair RestaurantAndBarChair KitchenChair DentistChair NursingChairPocket OrtophedicChair
ProfessionalChair BarberChair HighChair RecliningChair BeachChair NursingRockingChair GardenChair
DeskChair OutdoorChair DeckChair Chair DiningRoomChair DirectorChair WheelChair RockingChair
OfficeChair SchoolChair Hide Options
TableLamp DomesticTable DiningTable SlidingTableSawWithManualThrust CoffeeTable SofaTable
PortableFoamCuttingMachine LoungeTable RestaurantAndBarTable GamesTable OutdoorTable StableDoor
ReceptionTable VerticalSlidingTableSawWithAutomaticMovement CircularTableSawWithManualThrust FrameTableBase
KitchenTable WorkTable BedsideTable DrawingTable GardenTable BedTable EndTable Table
ChangingTable WritingTable PicnicTable VerticalSlidingTableSaw TablePress ConferenceTable OfficeTable
Hide Options

Figure 5 All Possible Outcomes for Multiple Generic Search Keywords

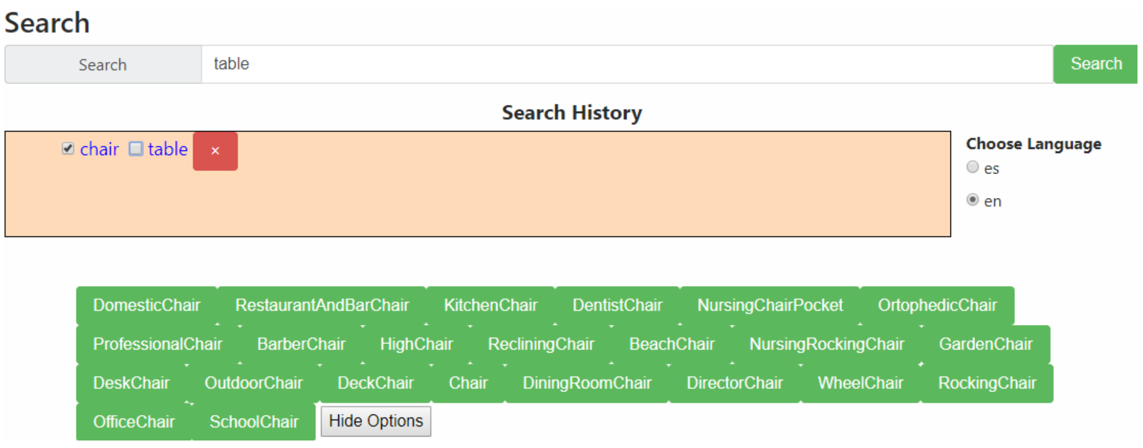

Search
Search
table
Search
Search History
☑ chair ☐ table ×
Choose Language
es
en
DomesticChair RestaurantAndBarChair KitchenChair DentistChair NursingChairPocket OrtophedicChair
ProfessionalChair BarberChair HighChair RecliningChair BeachChair NursingRockingChair GardenChair
DeskChair OutdoorChair DeckChair Chair DiningRoomChair DirectorChair WheelChair RockingChair
OfficeChair SchoolChair Hide Options

Figure 6 Removal Method of an Already Searched Keyword

2.4.1.1 Step 3: Define Property Values to Apply Filters

The scenario is taken further, where the user is interested in a chair, and from the obtained outcomes would like to look into High Chair as a product. Therefore, with results similar to Figure 4 the user clicks on the *HighChair* button for visualization. As opposed to textual context, providing the user with a more visual representation is also one of the goals for the Explorative Search feature. The implementation currently uses a radial layout scheme to describe main product and the available attributes that the product provides. Figure 7 illustrates the visualization features for the Product *HighChair*.

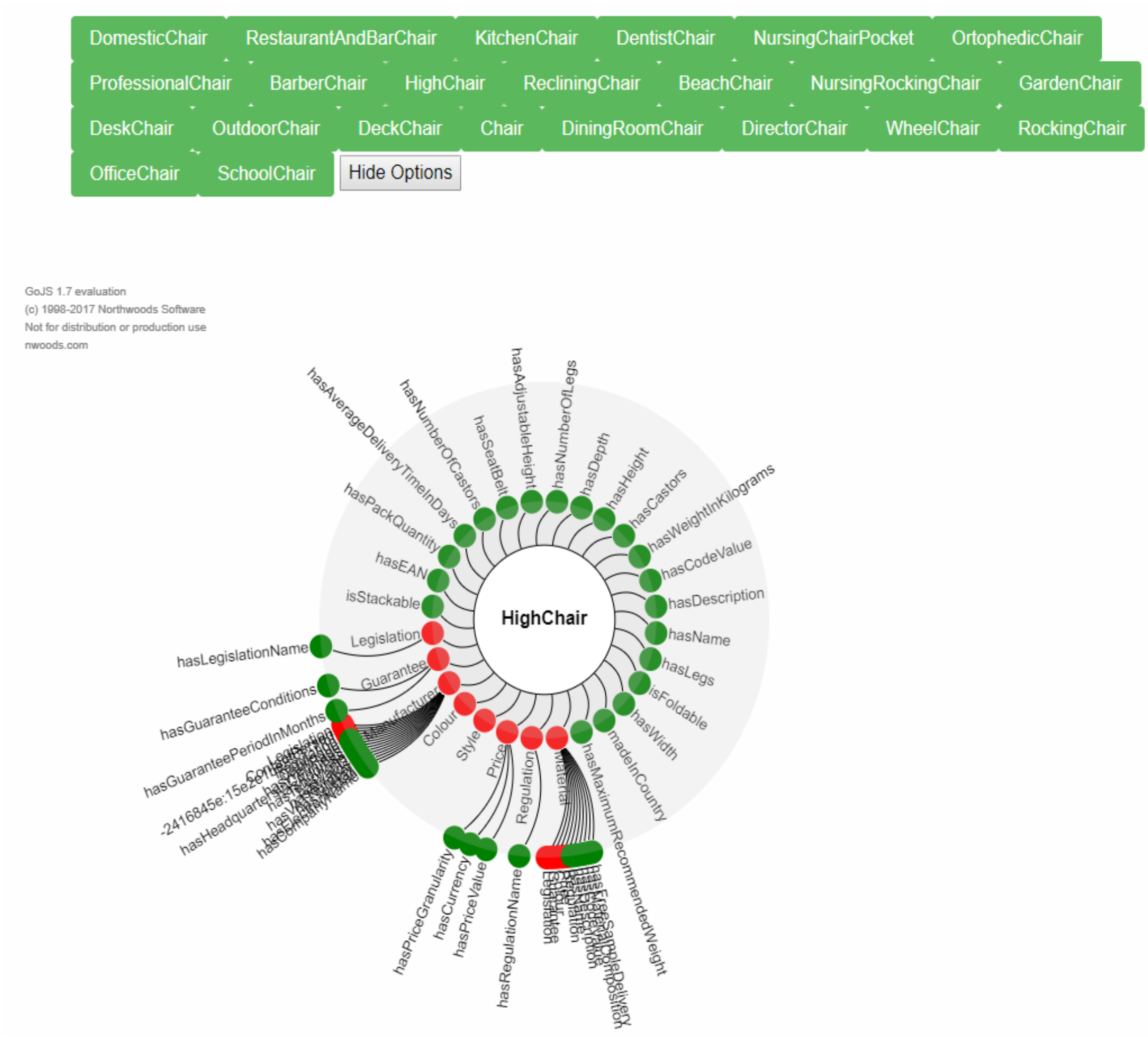


Figure 7 Visualization of High Chair as a Main Search Concept

Figure 7 provides a complete view of the concept and its essential attributes. For instance, a High Chair has height, width, etc. as important properties, to which the user can apply further filters in order to obtain a precise result. The initial diagram for any product search such as *HighChair, DomesticChair* etc. results in a two layer radial layout as above with green nodes on the diagram described as Datatype Properties and the red nodes described as Object Properties. The Datatype Properties represent properties which define a specific value in the range of a basic data types like string or integer. For example, the width of a chair would be a float describing the specific width of a chair. As opposed to this, an Object Property defines a reference from one individual property to another. For example, both the specific manufacturer of a product and the product itself could be requested by one search.

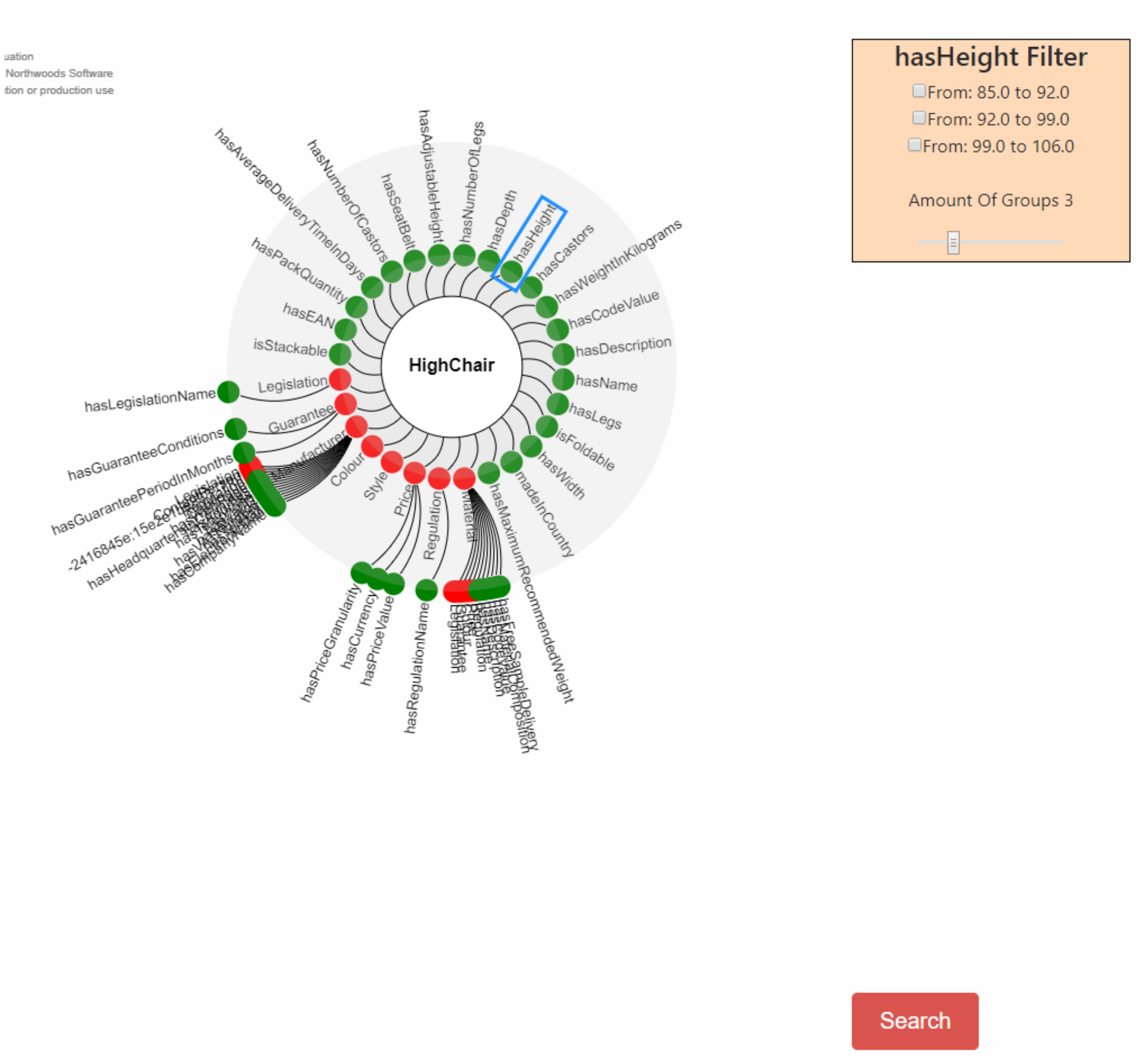
Different Usage Scenarios

The user can take advantage of further product filtering to a conclusive result through two scenarios:

1. Using a single selection of a property
2. Multiple selection of properties

In this scenario, it is assumed that the user wishes to obtain a search result for a High Chair with only a single attribute such as height or width in mind. It is also assumed that the user does not have particular values for he attributes in mind, for instance whether the High Chair needs to be of particular height or width. The point of interest is based only on a certain attribute that the High Chair as a concept is offering.

In a precise scenario a user wishes to obtain further knowledge of the High Chair product with Height as an attribute of interest. The user has to click on the *hasHeight* node on the diagram. This in turn, displays a filter of the same name on the right side of the diagram. However, it is worth noting that the filter selection is completely optional for the user. Figure 8 displays the filter for *hasHeight* attribute along with a Search Button when the user clicks on the respective node.



hasHeight Filter

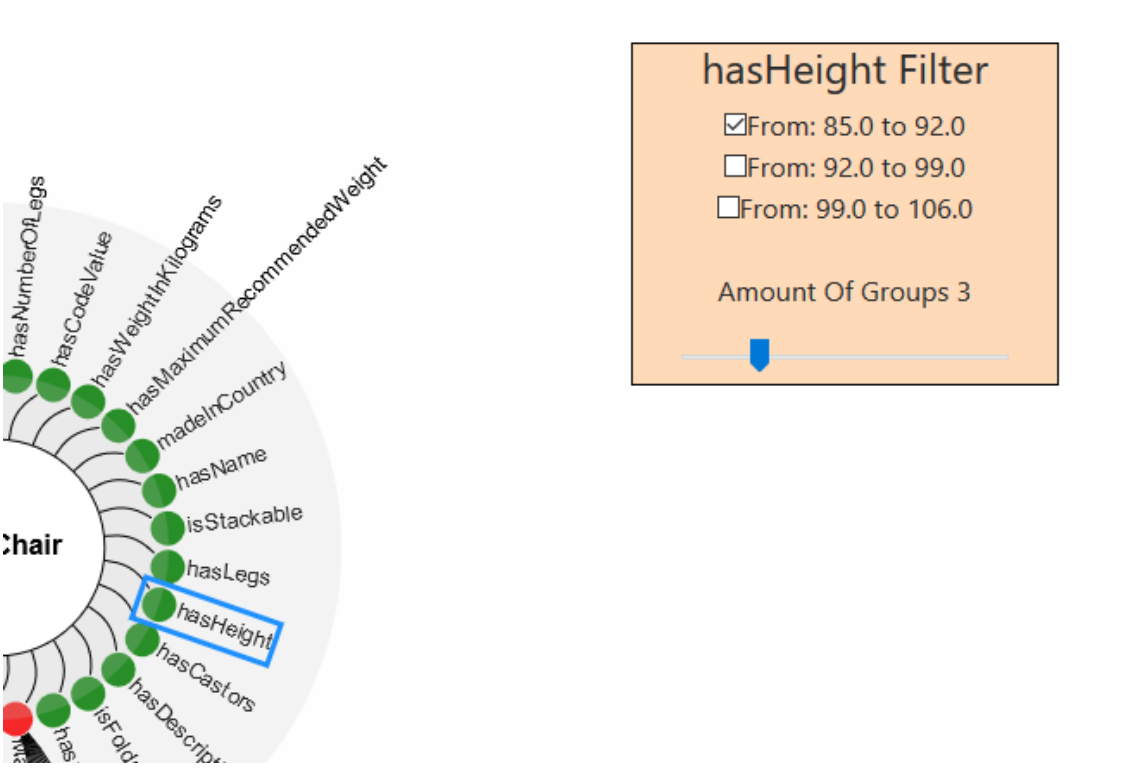
- ☐ From: 85.0 to 92.0
- ☐ From: 92.0 to 99.0
- ☐ From: 99.0 to 106.0

Amount Of Groups 3

Figure 8 Display of Filter and Search Button upon Single Selection of Property

To apply a filter, the user is assumed to have very specific search query in mind with respect to the product. The keyword search of *HighChair*, for instance, has the attribute of *hasHeight*, where upon clicking the particular attribute on the diagram produces a filter on the right side. Figure 9 illustrates the filter with available filter value options as checkboxes. The user, is interested in obtaining information for the *HighChair* with height parameter of 85, and hence would obtain information pertaining to the height parameter in a tabular form, as opposed to obtaining all available results.

Figure 9 illustrates the user selection of the attribute of *hasHeight* with a particular value of 85.0 to 92.0 in mind as a filter value. Upon clicking the Search button, the user is given a specific result with the best available option from the backend rather than all possible outcomes.



hasHeight Filter

☑From: 85.0 to 92.0

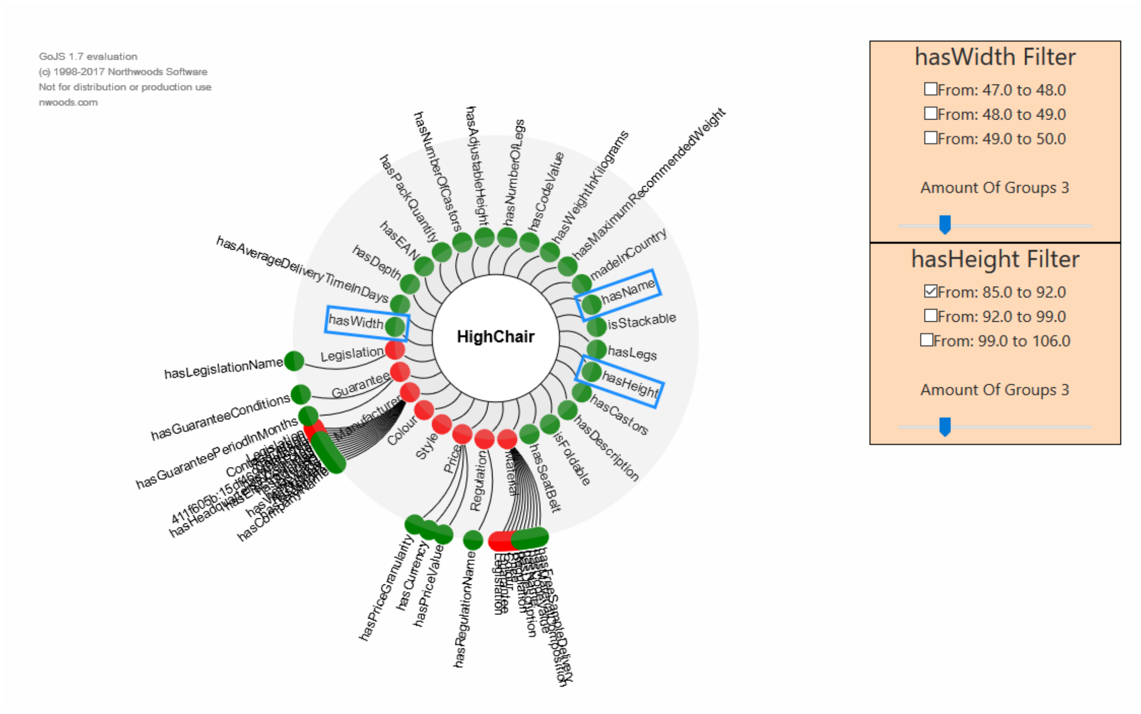
☐From: 92.0 to 99.0

☐From: 99.0 to 106.0

Amount Of Groups 3

Figure 9 Selection of Filter Value for Height Attribute of High Chair and Outcome

In terms of more precise search results, the user can check filter value for a particular attribute like *hasHeight* along with attributes without filters. Figure 9 describes this scenario of multiple selections along with different filter inputs in order to obtain the best case comprehensive result. Here the user selects *hasHeight*, *hasWidth, hasName* with the filter value of *hasHeight* checked from 85.0 to 92.0. The outcome is the best possible one provided from the backend with a particular product description. It is worth noting that *hasName* does not have a filter to display, because the filters are only available for attributes with value ranges or value sets, whereas "hasName" denotes a single value.



hasWidth Filter

☐From: 47.0 to 48.0
☐From: 48.0 to 49.0
☐From: 49.0 to 50.0

Amount Of Groups 3

hasHeight Filter

☑From: 85.0 to 92.0
☐From: 92.0 to 99.0
☐From: 99.0 to 106.0

Amount Of Groups 3

Figure 10 Multiple Selection of Attributes with Particular Selection of Height Filter for High Chair

To select properties that belong to linked items, the user just needs to click on the item and all item specific properties are shown in the graph. In the given example, a user would like to inquire about the manufacturing details of the product *HighChair*. The diagram of *HighChair* in Figure 11 provides a property *Manufacturer* and subsequent information about it. The user would prefer to access various aspects of the *Manufacturer* along with the attributes of the product (here, *HighChair*). This can be done with the Explorative Search diagram. Figure 11 shows the various properties connected to *Manufacturer* of the *HighChair*.

The *Manufacturer* can have many further attributes such as general information and further services such as its own catalogue or legislation it complies with. This can be critical for an end user who needs to abide by strict product and manufacturing standards. The search functionality helps the user to gain a deeper perspective of the product and crucial factors surrounding it.

In a usage scenario, the user is interested in the manufacturer's legislation attribute when searching for the *HighChair*. Hereby, all provided legislations from the manufacturer would be shown instead of a subset which is only related to the *HighChair*. For ease of use, when the user clicks on the *Legislation* node from the Manufacturer, the other attributes are hidden and the next connected properties for *Legislation* is displayed. Figure 12 describes the diagram alteration when the user clicks on *Legislation*, which displays on the subsequent attributes and hides all the other non-clicked attributes for ease of interaction. It is also worth noting that *Manufacturer* link to the *HighChair* also changes from a solid one to a dashed link. This signifies the user is traversing further through the layers of the Ontology and the path to root *HighChair* is displayed as *Manufacturer/Legislation*. A clear distinction here is needed since *Legislation* of the product itself also exists which is distinct from the *Manufacturer/Legislation* attribute.

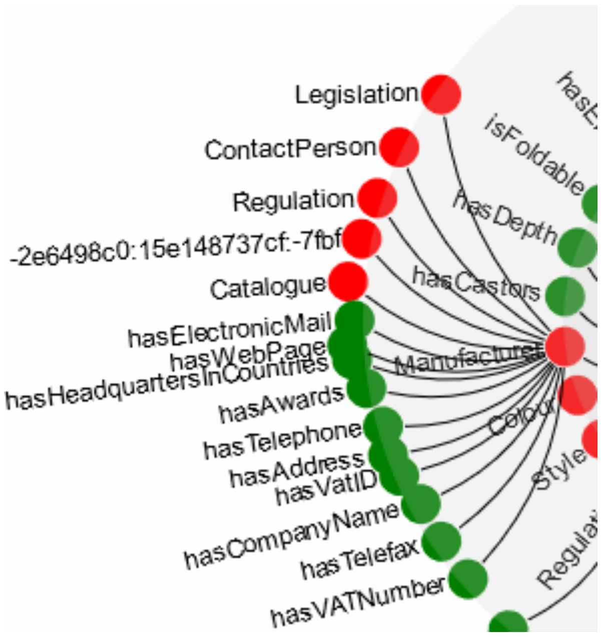


Figure 11 Attributes for the Manufacturer for High Chair as a Search Keyword

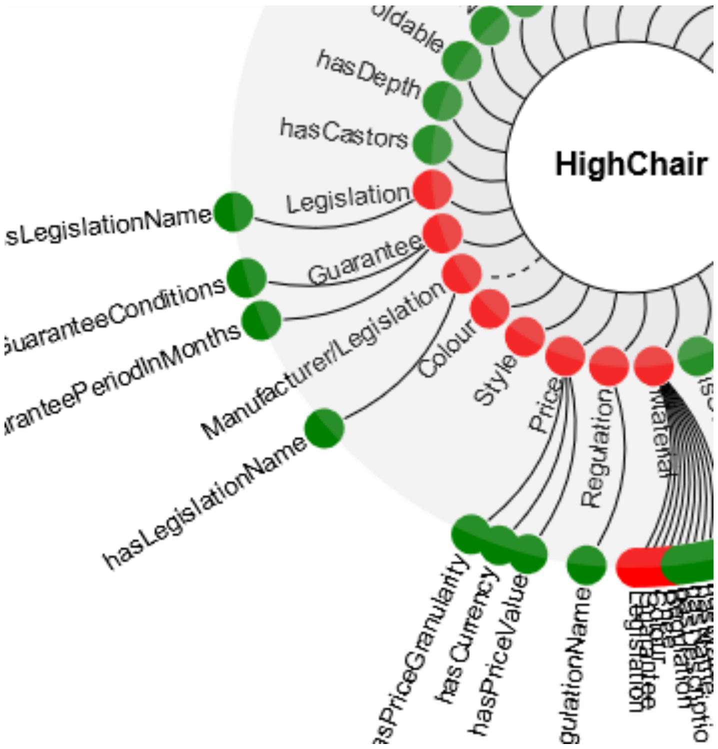
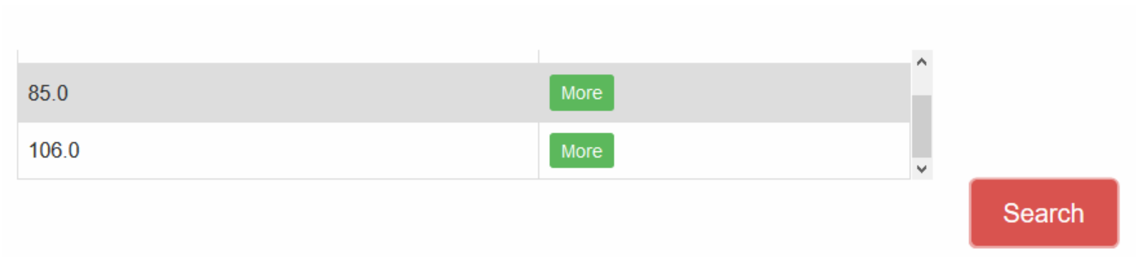


Figure 12 Traversing through Various Layers of Product with Explorative Search

2.4.1.2 Step 4: User Starts his Search

After the user has selected all relevant properties and applied his filters, he clicks on the search button and the search looks for appropriate items. A relevant item is a specific product or service which meets the category as well as the applied filter. Items that have no value for selected properties will be removed from the search result. All found items will be visualized in a table. Each entry contains only the selected property values to allow a quick overview and manual evaluation.



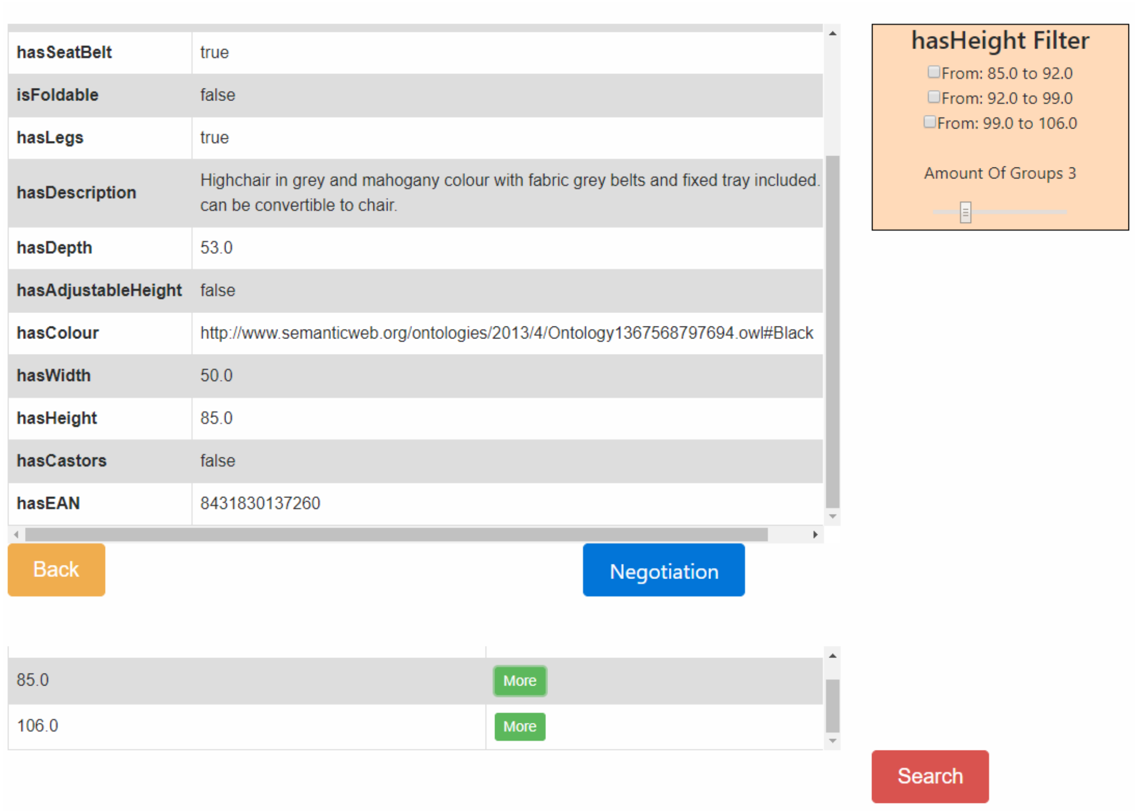
85.0	More
106.0	More

Figure 13 Search Outcome with Height Attribute for High Chair

Figure 13 shows the results table with values of 85.0 and 106.0 as available height attribute for High Chair when the user wishes to filter out the product based only on the Height attribute.

2.4.1.3 Step 5 & 6: User Compares All Search Results and Starts Negotiation

Figure 13 provides the relevant filtered results of the property values that have been selected for the main product (here, *HighChair*). However a complete description of the product is also of importance for the user. The complete detailed information for the filtered search result can be obtained by clicking on the *More* button for any of the filtered results. Figure 14 illustrates the detailed table of description for a filtered *HighChair* with height 85.0 as a value.



hasSeatBelt	true
isFoldable	false
hasLegs	true
hasDescription	Highchair in grey and mahogany colour with fabric grey belts and fixed tray included. can be convertible to chair.
hasDepth	53.0
hasAdjustableHeight	false
hasColour	http://www.semanticweb.org/ontologies/2013/4/Ontology1367568797694.owl#Black
hasWidth	50.0
hasHeight	85.0
hasCastors	false
hasEAN	8431830137260

85.0	More
106.0	More

hasHeight Filter
- From: 85.0 to 92.0
- From: 92.0 to 99.0
- From: 99.0 to 106.0

Amount Of Groups 3

Figure 14 Result Table for Single Selected Attribute for the Concept of High Chair

The detailed overview allows two actions, namely, to go back to the visualization of the product under consideration or to start the negotiation. If the user clicks on the negotiation button, the user is rerouted within the platform to the negotiation process service for ordering and price negotiation. The Back button at the bottom of the Table will take the user back to the radial layout diagram for further selection.

2.4.2 Keyword Based Search

Assumption

It is assumed that the user is familiar with the NIMBLE catalogues and is experienced with the possible search terms. In the following example, he wishes to obtain information regarding MDF Boards using the keyword-based search. With suitable information the user would like to subsequently place an order for a particular kind of MDF Boards using the search feature.

Goal

The usage of predefined terminology for the keyword search enables the specification of complex queries via keywords or tuples/triples of keywords. This kind of search does not require the user to navigate through terminologies.

The precondition is that the user has successfully logged in and has found the corresponding search form (step 1 of the general search process. See Figure 2 for more details). The following search also follows the described steps of Figure 2.

2.4.2.1 Step 2: Keyword to Define the Desired Product

The user can provide the term *mdf* as an input keyword in the provided search bar in order to obtain all relevant information for the MDF Board and all other results related to the keyword itself. The corresponding input form is shown in Figure 15. After the user had entered his keywords and clicked on the search button, the found products and services are listed. No filters or other kind of restrictions have been applied to reduce the number of relevant items.

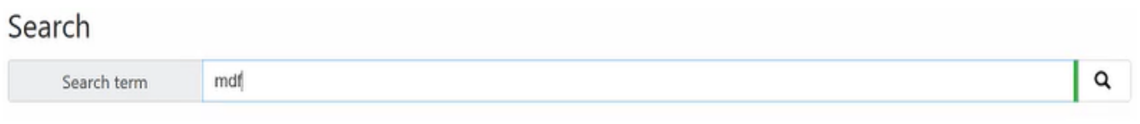
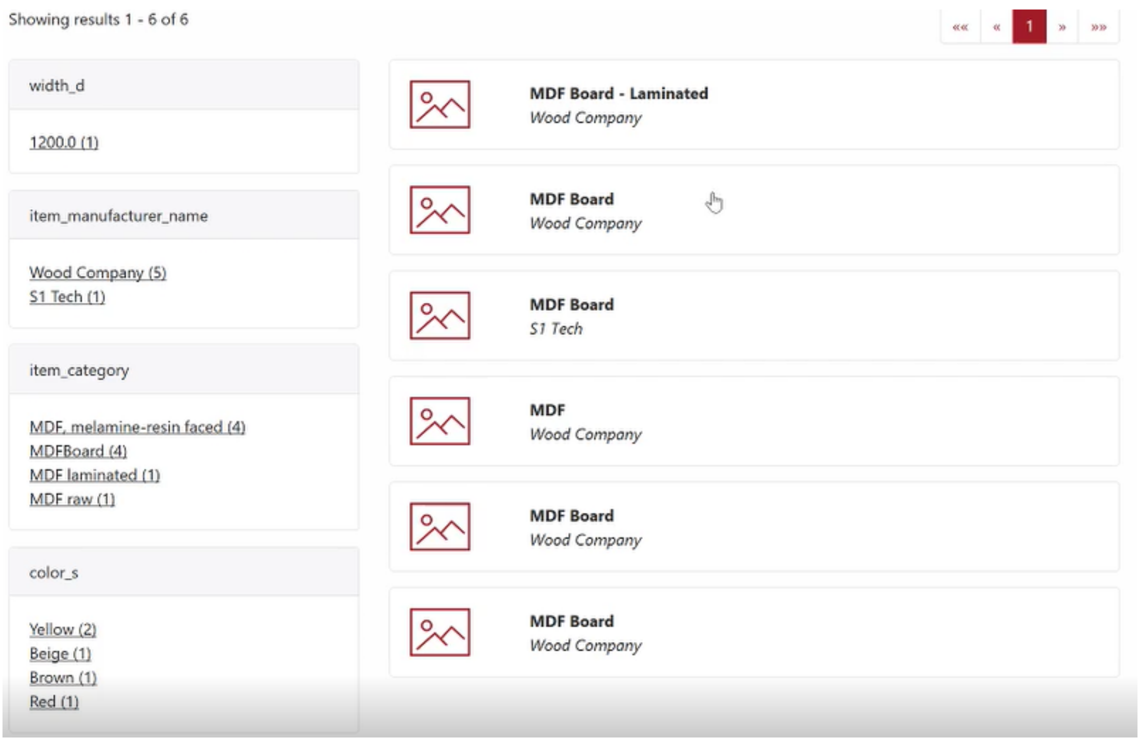


Figure 15 Initial Keyword Entry for the Keyword based Search

Figure 16 describes all relevant results that the user obtains when the keyword is requested.


Showing results 1 - 6 of 6

width_d
1200.0 (1)

item_manufacturer_name
Wood Company (5)
S1 Tech (1)

item_category
MDF, melamine-resin faced (4)
MDFBoard (4)
MDF laminated (1)
MDF raw (1)

color_s
Yellow (2)
Beige (1)
Brown (1)
Red (1)

MDF Board - Laminated
Wood Company

MDF Board
Wood Company

MDF Board
S1 Tech

MDF
Wood Company

MDF Board
Wood Company

MDF Board
Wood Company

Figure 16 Results for the Provided Keyword and Respective Filtering Attributes

2.4.2.2 Step 3: Define Property Values to Apply Filters

The left panel of Figure 16 displays the relevant filters for the keyword that are available for the user to filter the results into more appropriate and limited search results. Upon clicking the filters on the left panel, the results of the product are dynamically changed to provide the best result. For instance, the user can filter out boards which have particular colours or certain width.

2.4.2.3 Step 4: User Starts His Search

The chosen filter reduces the number of listed items on the right panel. In our example, the user finds the product MDF Board made by Wood Company which the user finds of interest. Figure 17 shows an example of a found item. For each relevant item, just the name and the value of the selected property is shown. To gain a detailed overview, additional actions are necessary.

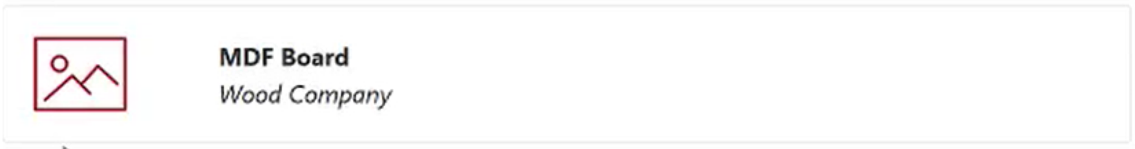
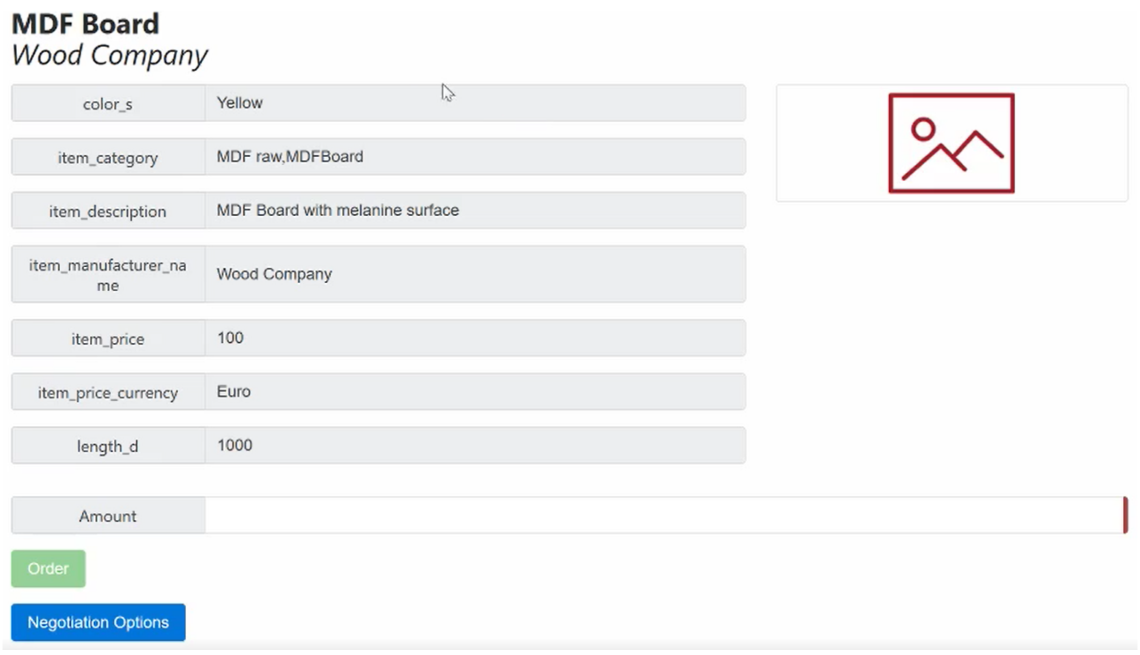


Figure 17 Selection of the Desired Product

2.4.2.4 Step 5 & 6: User Compares all Search Results and Starts Negotiation

Upon clicking the product under consideration, the user is rerouted to the product's details page, which provides all the relevant information about the product. Figure 18 gives an overview and provides the user with precise details before ordering the product. The *Amount* input box is where the user orders the quantity of the product. The user can also proceed with a negotiation process concerning the product. In this instance, the user wishes to go ahead with the order rather than start a negotiation.



MDF Board
Wood Company

color_s	Yellow
item_category	MDF raw,MDFBoard
item_description	MDF Board with melanine surface
item_manufacturer_name	Wood Company
item_price	100
item_price_currency	Euro
length_d	1000
Amount	

Figure 18 Description of the Product in Consideration

In the *Amount* input box the user enters the quantity of the MDF Boards that need to be purchased. For instance, the user wishes to acquire 90 such boards. Upon entering 90 in the input box the user proceeds to click the Order button, which is shown in Figure 18.

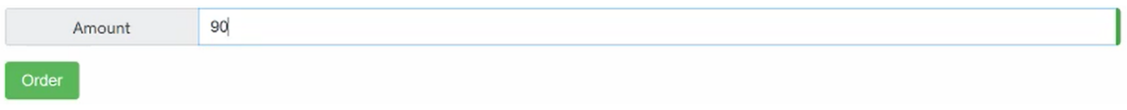

Amount 90

Figure 19 User Input for Ordering the Quantity of the Desired Product

Upon ordering the desired amount the user gets a small confirmation alert on the same page.

2.4.3 Semantic Query Patterns Based Search

The current state of this kind of search is on the conceptual level. The implementation is still ongoing. Therefore, the following figures are screenshots from the concept slides instead of the live demonstrator.

Assumption

This kind of search assumes that the user is fairly familiar with the terminology of the NIMBLE platform and the desired product. This precondition must be met, because the support by intensional queries is lower, compared with the graph based navigation.

Goal

The goal is to provide the user a command based search of the products and services. The user shall create a sentence that defines the search term including filters in an iterative way. In each iteration the sentence contains more commands and values and the search term becomes more precise.

In the following, this search uses also the Step 2 from the Graph based navigation to define the starting point of search. The chosen starting point is an ontology concept from which the user can add restrictions and filters.

After the user chooses a concept as starting point, the text area appears for the definition of the search term, which is shown in Figure 21.

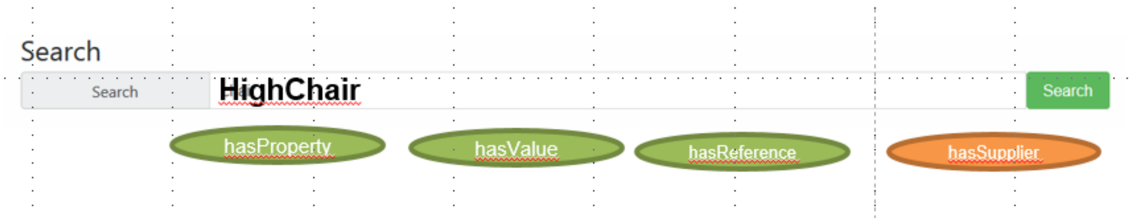


Figure 20 Starting Point for the Semantic Query Patterns based Navigation

2.4.3.1 Step 3: Define Property Values to Apply Filters

The goal of this kind of search is not to define the relevant set of properties as is done in the graph based navigation. Instead, we now want to define the filters that shall be included in the search term. There are two groups of relations available for this: the first group (*viz. hasProperty, hasValue, hasReference*) defines basic relations to choose properties and define values for them. This kind of relations uses the direct properties of the starting concept. The second group (*viz. hasSupplier* in Figure 21) are platform-specific relations where the semantics may differ from relation to relation. Each relation of the second group allows to define virtual properties which could refer to any transitive property. The virtual relations must be aligned with the underlying ontology (could also be part of the ontology as an object property) rather than the relations in the first group. For example, the object property *hasSupplier* is not connected directly with a product but is linked transitively to it. Moreover, relations could be generated which includes transitive links as well as conditions. A common condition could that the supplier belongs to a specific region.

The application of these relations follows the same procedure. First, the user selects a relations and it will be added to the search term. In Figure 21, a corresponding example is shown.

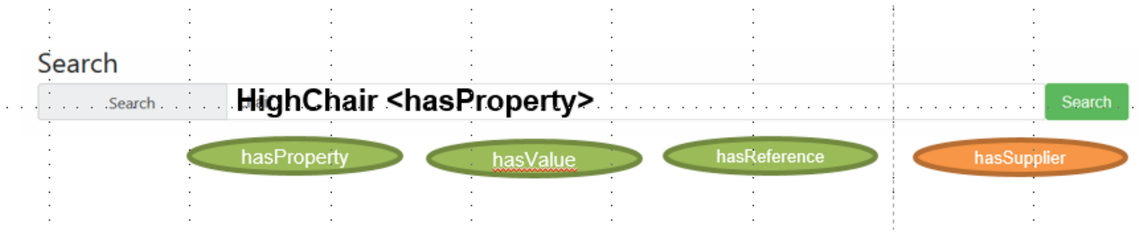


Figure 21 Selection of Relation for a Search Term

After the relation is selected the user shall restrict it. In our example, the user shall define which property shall be further restricted. For that purpose, the user clicks into the textbox and starts typing the name of the property and this triggers the auto-complete function. This supports the user to choose the right term without knowing the exact spelling of the property name. Then, the correct property is added to the search term, which is shown in Figure 22.

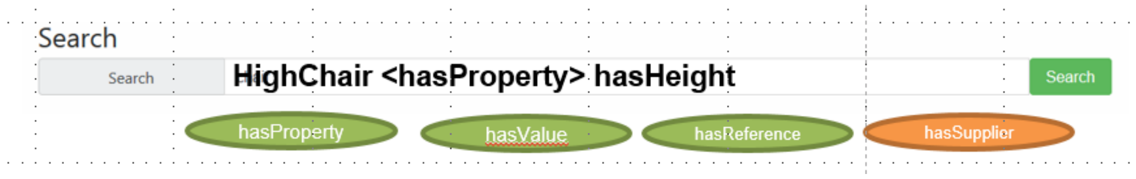


Figure 22 Auto-Completion of the Entered Property

After the user has defined the property he could also define the value for this property. In such a case, the user would select the *hasValue* button and would subsequently add a value.

Apart from the datatype properties, the user could also follow object properties to add filters to transitive properties. In such a case, the search space will be extended from the root concept to linked concept. Such capabilities allow the user for example to define that the manufacturer of a product must be certified according to some certification scheme or that the manufacturer produces his products in a specific country. To apply such kind of filters, the user must use the relation *hasReference*. After the user selects this relation, the relation will be added to the search term and the user must specify the concept that should be connected. In this example, the user enters Manufacturer with the help of the auto completion and could also specify further filters for Manufacturer. A corresponding search term is shown in Figure 23.

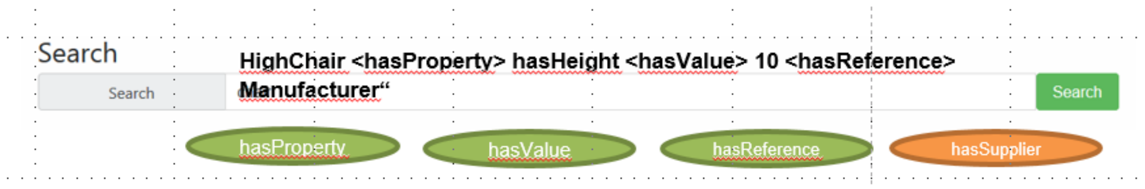

Search
Search
HighChair <hasProperty> hasHeight <hasValue> 10 <hasReference> Manufacturer“
Search
hasProperty
hasValue
hasReference
hasSupplier

Figure 23 Extending the Search Space by Following Object Properties

2.4.3.2 Step 4, 5 & 6 of the Search Process

The steps 4, 5 and 6 are the same as in the graph based navigation. That means, the search results are listed in a table and the user can request the details for each of them via the More button. Finally, the user can start the negotiation in the details view. The details view will be the same as the details view in the graph based navigation - explorative search (Figure 14).

3 Implementation of NIMBLE Search

The objective of T3.3 is to develop an intuitive user interface for search capabilities on the NIMBLE platform. That means, it shall give access to products, services via catalogues. Extensional queries for specific products or services and intensional queries for the structure and relationships between products and services shall be provided. The combination of these kinds of queries enables technology-independent access without the need to understand ontologies or query languages. The solution contains two software modules which are shown in Figure 24.

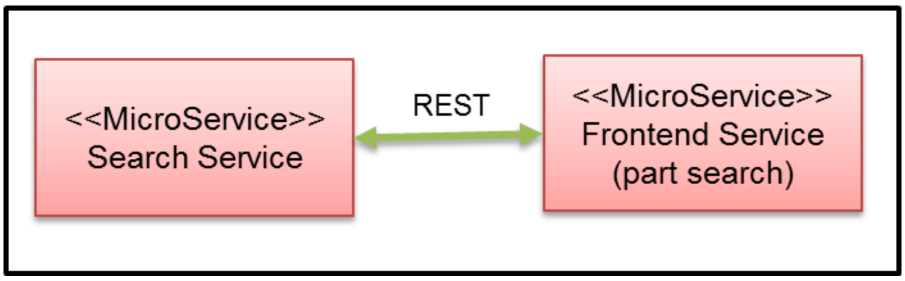


Figure 24 Overall Solution

The module “<<MicroService>> Search Service” is a Java Spring web service which acts as a micro service and provides search capabilities via an API. In the following, this service is called backend service. The module “<<MicroService>> Frontend Service” is part of the NIMBLE UI and invokes the backend service.

As shown earlier, two kinds of searches are established: a facetted search by using Apache SOLR and an explorative search which is developed from scratch. In the following, the implementation of the explorative search is presented in detail to show how research results are obtained.

3.1 Integration in the Overall Architecture

The outcome of task T3.3 will be added to the overall NIMBLE platform. The main outcome is shown Figure 25.

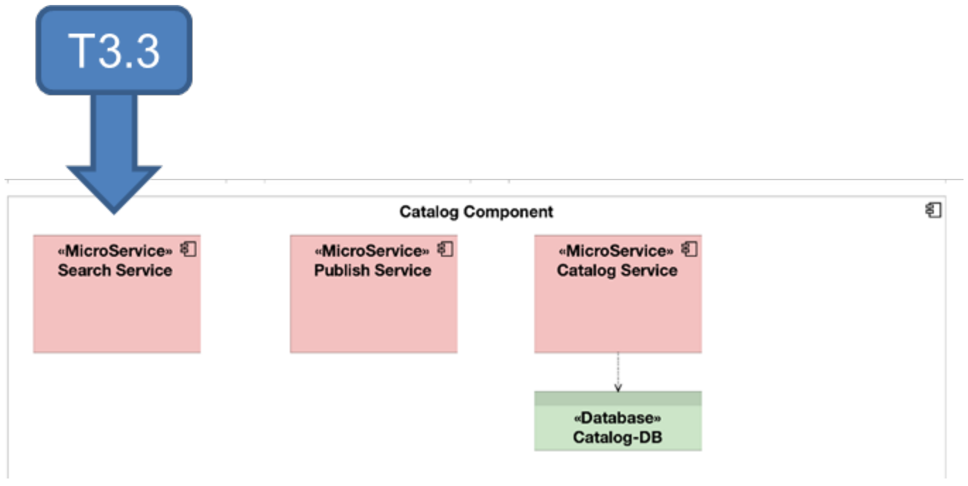

T3.3
Catalog Component
«MicroService» Search Service
«MicroService» Publish Service
«MicroService» Catalog Service
«Database» Catalog-DB

Figure 25 Integration in the Overall NIMBLE Architecture

Apart from the development of the module "<<MicroService>> Search Service", the module "<<MicroService>> Frontend Service" will be extended to the search capabilities.

3.2 Backend Service

This section provides the necessary insight for the explorative search features which are provided for the front-end service. The main goal of the search-catalogue service (alias <<MicroService>> Search Service) is to translate the user interactions into SPARQL queries as well as to return the resulting ontology to the front-end.

3.2.1 Micro Service Design

In order to obtain information regarding the search term, the front-end service has to make some API calls to the back-end Service. Each response will be sent in the form of a JSON (JavaScript Object Notation) structure. Hereby, each invocation carries all necessary information that is required to generate the result. Intermediate results or session related data are not stored. To obtain user related context information another set of NIMBLE micro services are consumed. In addition, the authentication will be handled by the overall platform.

3.2.2 Architecture & Core Modules

The architecture of the backend service includes two main classes and some POJOs (Plain Old Java Object) to define the input and output parameters of all methods. The backend service uses JSON to exchange data with any other service. To support JSON, the backend service uses a library to translate a POJO automatically in a corresponding JSON String. To keep the description of the backend service simple, only the two main classes are shown in the following class diagram.

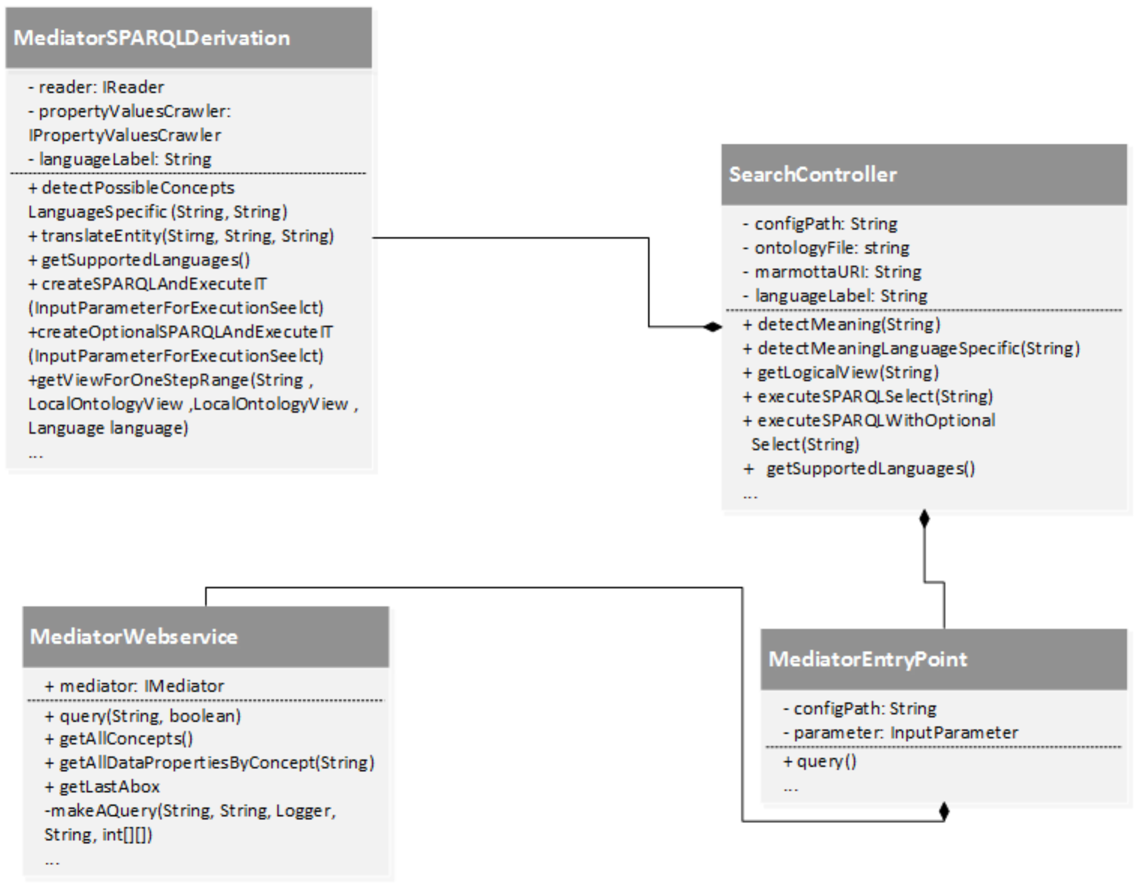

MediatorSPARQLDerivation
- reader: IReader
- propertyValuesCrawler: IPropertyValuesCrawler
- languageLabel: String
+ detectPossibleConcepts LanguageSpecific (String, String)
+ translateEntity(Stirng, String, String)
+ getSupportedLanguages()
+ createSPARQLAndExecuteIT (InputParameterForExecutionSeelct)
+createOptionalSPARQLAndExecuteIT (InputParameterForExecutionSeelct)
+getViewForOneStepRange(String , LocalOntologyView ,LocalOntologyView , Language language)
...
SearchController
- configPath: String
- ontologyFile: string
- marmottaURI: String
- languageLabel: String
+ detectMeaning(String)
+ detectMeaningLanguageSpecific(String)
+ getLogicalView(String)
+ executeSPARQLSelect(String)
+ executeSPARQLWithOptional Select(String)
+ getSupportedLanguages()
...
MediatorWebservice
+ mediator: IMediator
+ query(String, boolean)
+ getAllConcepts()
+ getAllDataPropertiesByConcept(String)
+ getLastAbox
-makeAQuery(String, String, Logger, String, int[][])
...
MediatorEntryPoint
- configPath: String
- parameter: InputParameter
+ query()
...

Figure 26 Simplified Architecture of Backend Service

The entry point for each request is the *SearchController*. The *SearchController* offers some web service methods. Each of them realizes a specific functionality. Table 3 summarizes the functionality.

Table 3 Web Service Methods for Search Capabilities

Web service endpoint	**Description**
/detectMeaningLanguageSpecific	This method determines which concepts could be relevant to a string
/getLogicalView	This method determines to a given concept all properties with a predefined visibility range
/executeSPARQLSelect	This method is derived on the basis of a user selection, which is in turn used as a SPARQL query and return the result
/executeSPARQLOptionalSelect	This method returns on the basis of an UUID of an instance, all available information to this instance.

All provided public functions require the class *MediatorSPARQLDerivation*. The main objective of this class is to transform the user interactions into the knowledge which ontology parts are requested and to trigger corresponding library functions. The library **de.biba.triple.store.access** is used as an abstraction layer between high level functions and SPARQL queries. There is one exception. The method *getLogicalView* is designed to create on basis of a root concept and a visibility range, a logical view including the root concepts, the object properties, the ranges of object properties and datatype properties. This visibility range determines the maximum distance from the root concept to a final datatype property. If the visibility range is greater than one, corresponding object properties will be traversed and resolved. The recursive functionality is not implemented by the library and therefore, the *MediatorSPARQLDerivation* includes SPARQL derivation functionality.

3.3 Frontend Service

This section provides the necessary insight for the explorative search feature created using the Angular Framework. The subsections provide information of each facet used to develop the UI for the Explorative Search.

3.3.1 Micro Service Design

In order to obtain further information concerning the current search term, the frontend has to make some API calls to the backend. The querying parameters are sent accordingly, to the backend, where they are processed and where a response will be generated in the JSON format. This response is then processed by different Angular Components in order to provide information that the user can understand and interact with, in order to obtain actionable results.

These API Calls are in fact, methods within a class, which are then available as Injectable Service provided by the Angular Framework. The service is available throughout all the components of the application and can be used to send and/or receive information from the backend service.

The complete architecture and the modules are discussed in the next section. The current section discusses which components make particular API calls to obtain the relevant information. Figure 27 illustrates this in some detail.

The *ExplorativeSearchComponent* comprises of three sub-components namely *ExplorativeSearchFormComponent*, *ExplorativeSearchDetailsComponent*, *ExplorativeSearchFilterComponent*. Each component makes API calls to the Catalog-Search-Service micro service (this is a NIMBLE core service). Some calls are made automatically in order to provide the information before the user actually needs it. An example is the language selection for the Explorative Search. Other calls are made via user interaction viz. a button click or interaction with the diagram. Information obtained in a parent component can be passed to a child component via Property Binding. In Figure 27, data that is available in the *ExplorativeSearchFormComponent* is passed on to the *ExplorativeSearchDetailsComponent*. When a child component wants to pass information to the parent, it does it via the Event Binding viz. *ExplorativeSearchFilterComponent* sends selected filter data from the user to the *ExplorativeSearchDetailsComponent*.

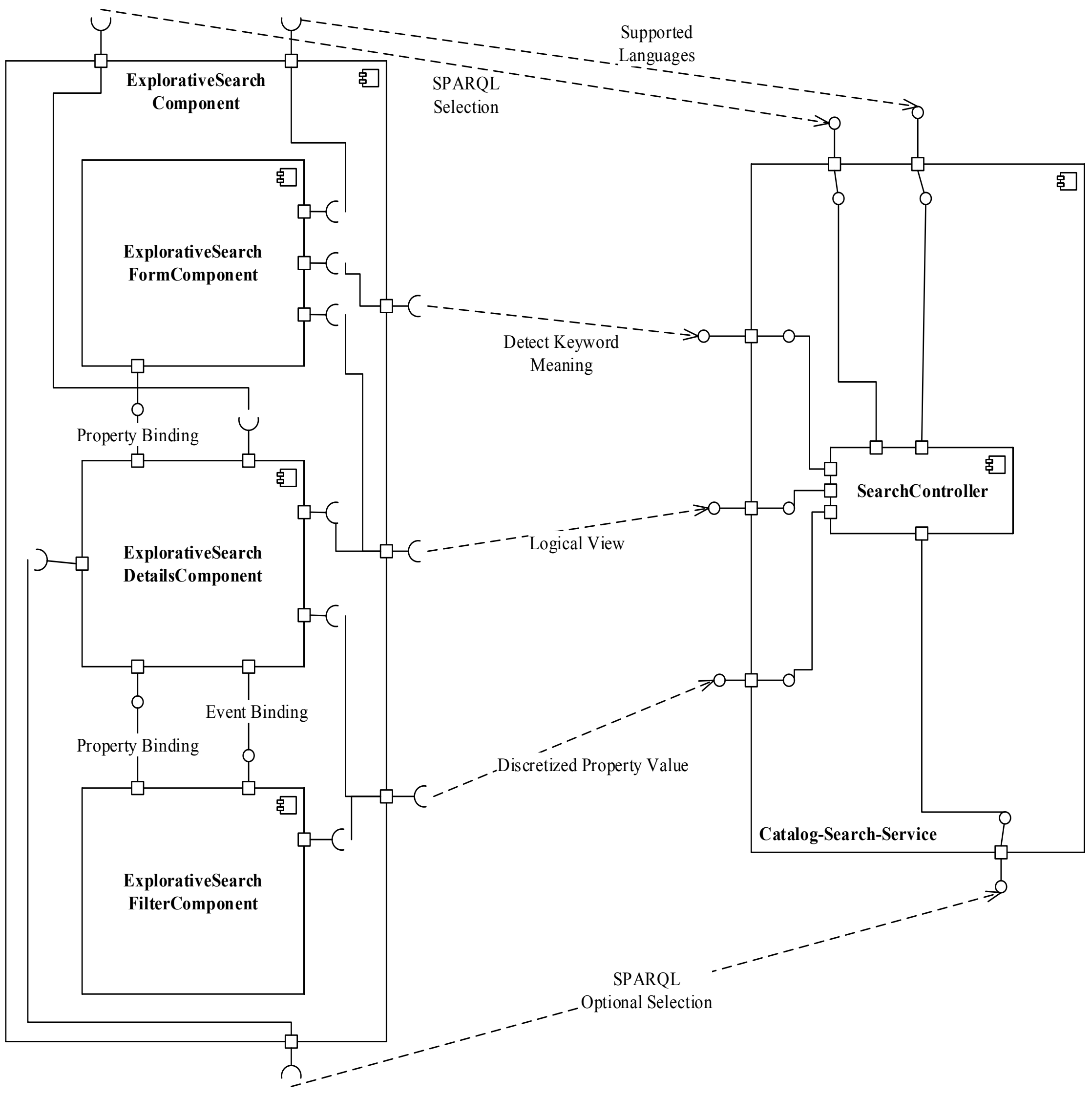

ExplorativeSearch Component
ExplorativeSearch FormComponent
Property Binding
ExplorativeSearch DetailsComponent
Property Binding
Event Binding
ExplorativeSearch FilterComponent
Supported Languages
SPARQL Selection
Detect Keyword Meaning
Logical View
Discretized Property Value
SearchController
Catalog-Search-Service
SPARQL Optional Selection

Figure 27 Overview of Micro Service Design for the Explorative Search Feature

ExplorativeSearchComponent is the search form where the user enters keywords for the product/service. The component also provides the language selection for the search. The information that needs to be obtained when the user clicks on the resultant keyword buttons as illustrated in Figure 3 is done via the Injectable Service to the backend. The Supported Languages, Detect Keyword Meaning, Logical View are the calls made to obtain the languages, the keyword query response and the graphical diagram for the keyword response respectively.

Whenever the user interacts with the visualization for a product/service, an API call Discretized Property Value is made in order to obtain filters for the data properties or when the diagram needs to be reloaded with new information for object properties API call Logical View is made. When the user clicks the Search button after selecting properties, the SPARQL Selection API call is made in order to obtain the best combination for the table results. The final tabular result in Figure 13 is obtained via the SPARQL Optional Selection.

3.3.2 Architecture & Core Modules

The Angular Framework uses the concept of Hierarchy for every component used to create a web application. An easy understanding for this can be devised using the concept of Parent-Child hierarchy between all components. The components are written in Typescript and generally the variables used within the component can be used in the component's respective Template (HTML) file. Within the same Template file, the successive child component can be declared. In order to communicate with the child component, information in the form of metadata can be interchanged with the help of property binding. If the child needs to communicate with the Parent component, then Event binding is used, where the child generates some metadata and the parent then acquires it. A visual representation of the hierarchy concept in Angular is illustrated in the figure below.

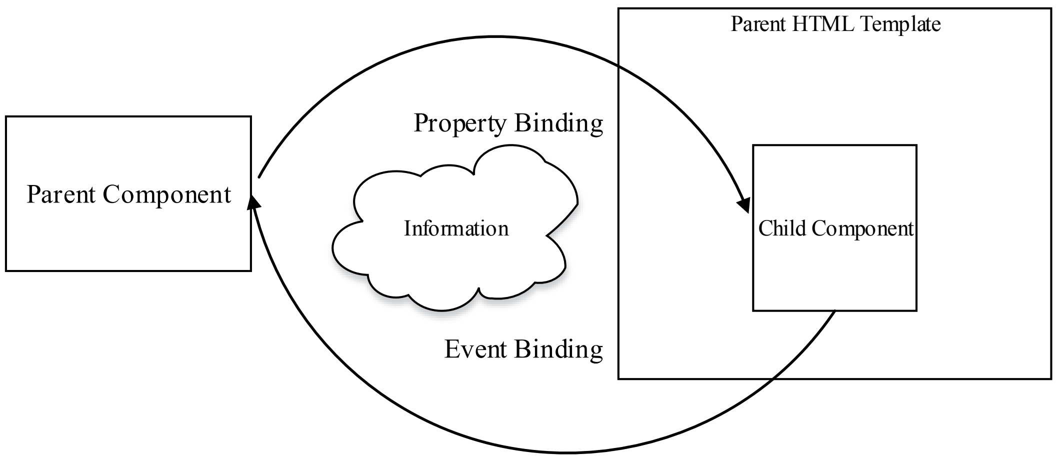


Figure 28 Information Exchange between Hierarchical Components in Angular Framework

This concept of hierarchical component design is used for the Explorative Search as follows:

- The complete application for the Frontend Service consists of an Angular Component called *ExplorativeSearchComponent*
- Within the *ExplorativeSearchComponent* there exists a child component called *ExplorativeSearchFormComponent*
- *ExplorativeSearchFormComponent* is responsible for handling the Search Bar where the user input is given and also the available buttons after a successful keyword search, as well as the languages preferences
- The graphical visualization for each product search is developed within a child component of the above-mentioned *ExplorativeSearchFormComponent*. This component is called *ExplorativeSearchDetailsComponent*
- *ExplorativeSearchDetailsComponent* takes care of all the user interaction within the graphical diagram as well as visualizing Filters for properties and tabular results. The user interaction with the Filters is taken care of within a child component of *ExplorativeSearchDetailsComponent*, namely *ExplorativeSearchFilterComponent*
- *ExplorativeSearchFilterComponent* takes care of the Filter boxes of the clicked properties within the diagram. As a child component, it needs to send back some selected filter metadata back the *DetailsComponent*. This is done using the concept of Event Binding between the *Filter* and *DetailsComponent*.
- Within the complete *ExplorativeSearchComponent* component there are explicit calls to the backend Service for acquiring information for the search. This is taken care of by the Injectable Service provided within the Angular Framework. The service is used within each of the above-mentioned components to make appropriate backend calls for information exchange. The Injectable Service is called *ExplorativeSearchService*

Figure 29 gives a graphical illustration of the components mentioned above and the inter-component communication within along with the Injectable Service for backend Communication within each respective component.

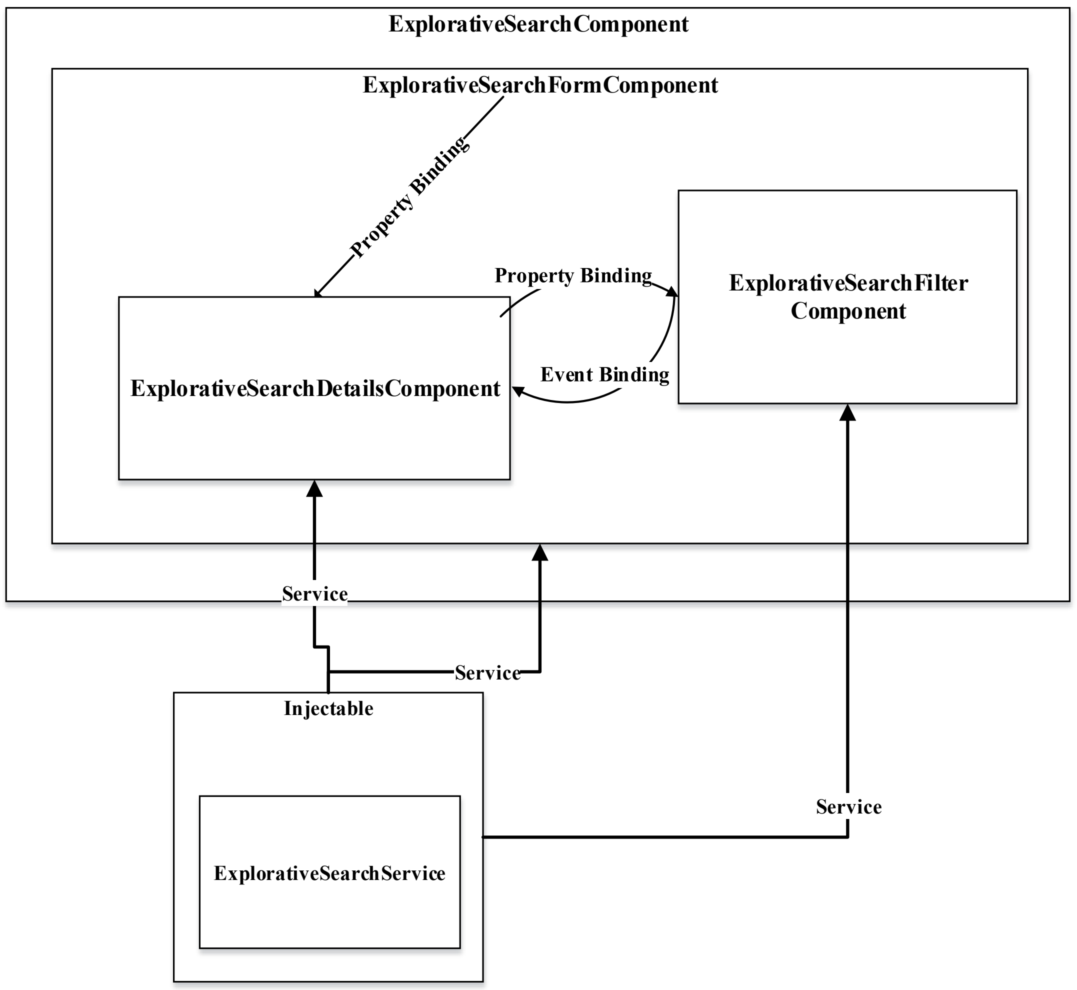

ExplorativeSearchComponent
ExplorativeSearchFormComponent
Property Binding
Property Binding
ExplorativeSearchFilter Component
Event Binding
ExplorativeSearchDetailsComponent
Service
Service
Injectable
Service
ExplorativeSearchService

Figure 29 Explorative Search Angular Components and Inter-Component Communication

Figure 30 describes the Explorative Search's Angular Component via a UML Class diagram. Each component is a class in Typescript. Angular's Life Cycle Hooks are interface realizations within the respective components and take care of how the DOM elements in the HTML template will respond upon initialization or upon change from the parent component.

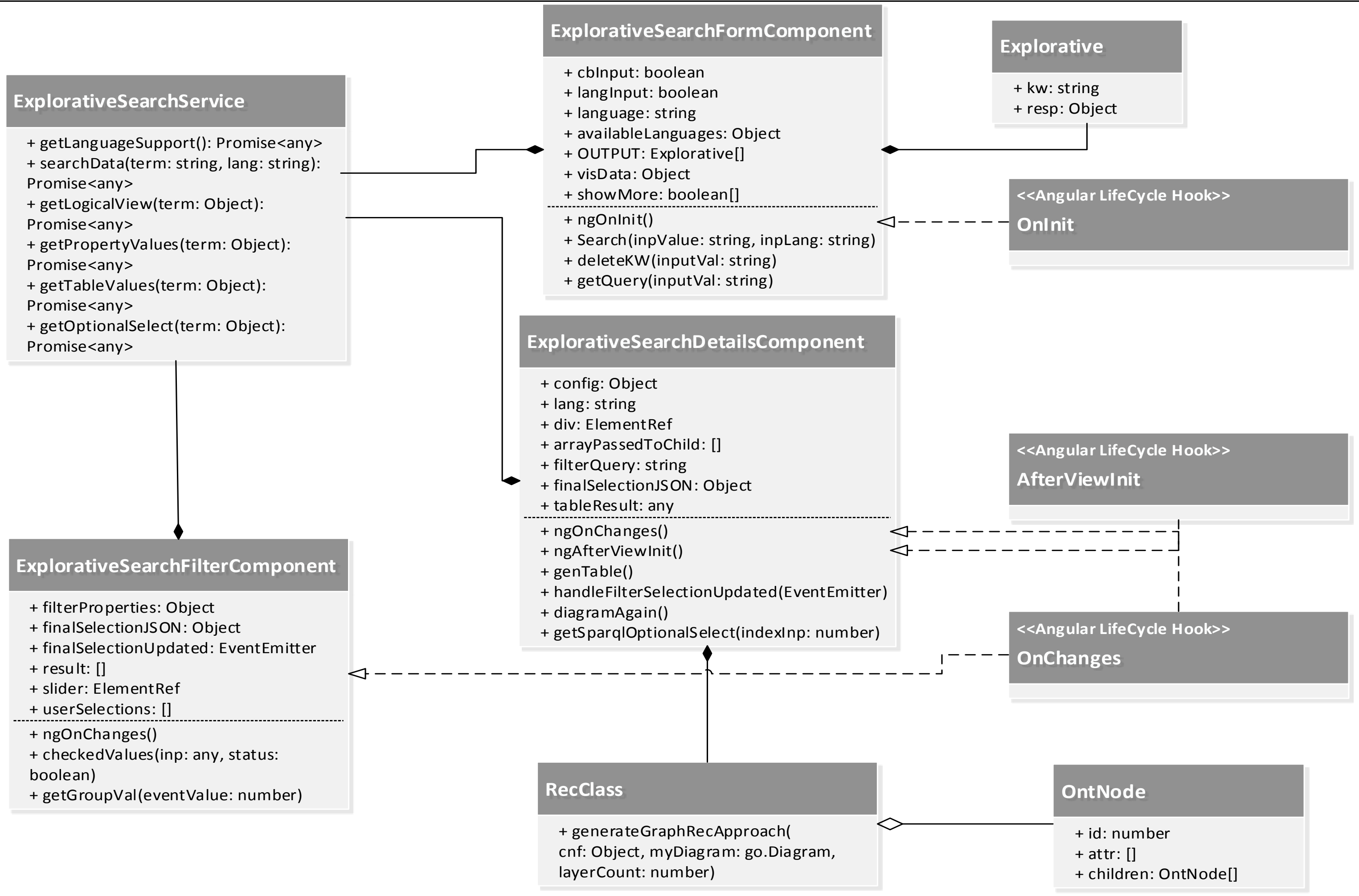

ExplorativeSearchFormComponent
+ cbInput: boolean
+ langInput: boolean
+ language: string
+ availableLanguages: Object
+ OUTPUT: Explorative[]
+ visData: Object
+ showMore: boolean[]
+ ngOnInit()
+ Search(inpValue: string, inpLang: string)
+ deleteKW(inputVal: string)
+ getQuery(inputVal: string)
Explorative
+ kw: string
+ resp: Object
ExplorativeSearchService
+ getLanguageSupport(): Promise<any>
+ searchData(term: string, lang: string): Promise<any>
+ getLogicalView(term: Object): Promise<any>
+ getPropertyValues(term: Object): Promise<any>
+ getTableValues(term: Object): Promise<any>
+ getOptionalSelect(term: Object): Promise<any>
<<Angular LifeCycle Hook>>
OnInit
ExplorativeSearchDetailsComponent
+ config: Object
+ lang: string
+ div: ElementRef
+ arrayPassedToChild: []
+ filterQuery: string
+ finalSelectionJSON: Object
+ tableResult: any
+ ngOnChanges()
+ ngAfterViewInit()
+ genTable()
+ handleFilterSelectionUpdated(EventEmitter)
+ diagramAgain()
+ getSparqlOptionalSelect(indexInp: number)
<<Angular LifeCycle Hook>>
AfterViewInit
ExplorativeSearchFilterComponent
+ filterProperties: Object
+ finalSelectionJSON: Object
+ finalSelectionUpdated: EventEmitter
+ result: []
+ slider: ElementRef
+ userSelections: []
+ ngOnChanges()
+ checkedValues(inp: any, status: boolean)
+ getGroupVal(eventValue: number)
<<Angular LifeCycle Hook>>
OnChanges
RecClass
+ generateGraphRecApproach(cnf: Object, myDiagram: go.Diagram, layerCount: number)
OntNode
+ id: number
+ attr: []
+ children: OntNode[]

Figure 30 UML Class Diagram for Explorative Search's Angular Component

3.4 Deployment

The deployment of the backend and frontend micro services relies on Docker containers. All created and maintained Docker containers have been uploaded to the Docker hub https://hub.docker.com/r/nimbleplatform/. To update the current version, the process of Figure 31 must be followed.

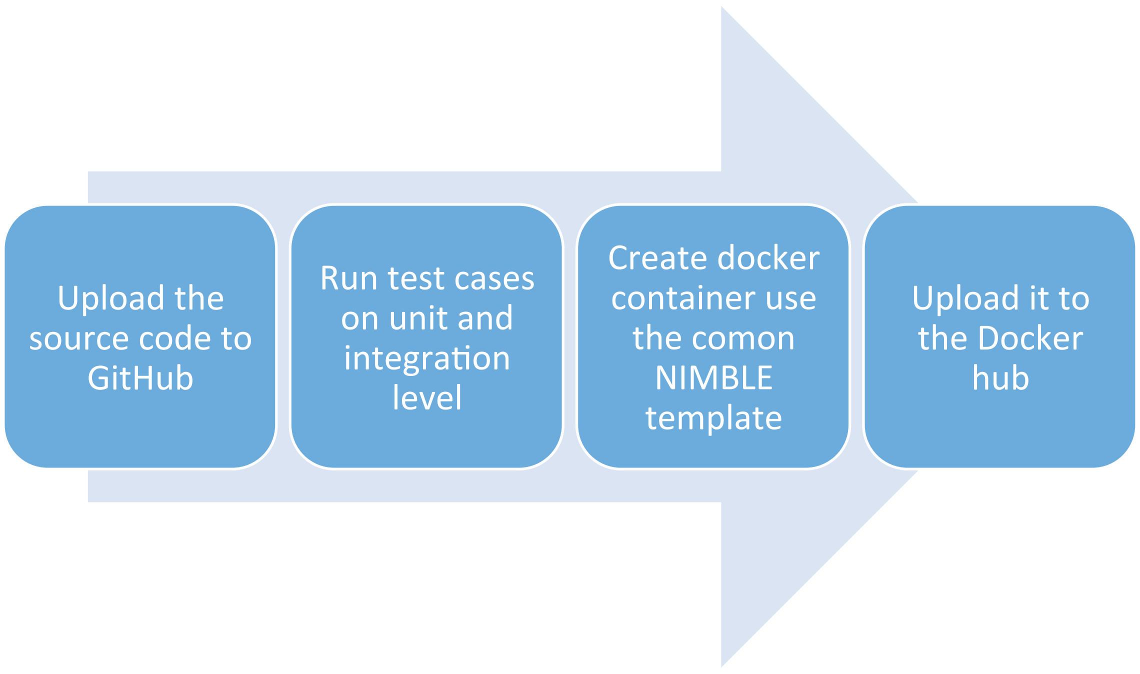

Upload the source code to GitHub
Run test cases on unit and integration level
Create docker container use the comon NIMBLE template
Upload it to the Docker hub

Figure 31 Deployment Process for the Search Modules of NIMBLE

The deployment process starts at the source code level. Here, the first step is to commit the changes on the common GitHub (https://github.com/nimble-platform) master branch. Then, Apache Maven is executed by a continuous integration tool like Jenkins or manually to compile and to execute the test cases. Test cases should cover both unit and integration test levels. Subsequently, Apache Maven is used to create a Docker container. Finally, the Docker container will be uploaded to the Docker hub. In the current case, the continuous integration tool Jenkins is used. Moreover, each NIMBLE micro service contains a Kubernetes configuration file which enables the direct inclusion and orchestration inside a Kubernetes environment.

4 First Step of Evaluation

The prototype of the NIMBLE search was presented at the NIMBLE review in July 2017. The prototype demonstrated the keyword driven search on basis of the generic NIMBLE ontology as well as the graph-based search on basis of the furniture ontology of the MICUNA case. After the review, the missing multi lingual support was incorporated into the graph based search.

The extended prototype of the search was deployed on a server including the last furniture ontology. A first evaluation web meeting was held on 2. August 2017. During this meeting, the search functionality was presented to AIDIMME. As an outcome, AIDIMME pointed out the need for additional search capabilities in the keyword driven part of the graph based search. A potential extension should make it possible to define additional property values to a given concept directly. The example “green HighChair” was mentioned. A further request made by the coordinator was to offer predefined semantic query patterns that cover typical business situations, e.g. finding a supplier of a particular product and adding some constraints such as location of the manufacturing plant. For this, the concept of the “Sematic Query Patterns based Search” was specified and will be implemented in the remaining time of the task.

5 Summary and Conclusion

This document explains in detail the search capabilities of the proposed NIMBLE platform including main aspects driving it and the way the proposed design will enable achieving the ambitious goals set out in the proposal. This document is issued to complement the software prototype D3.3.

The underlying technology for representing NIMBLE catalogues is an OWL/RDF encoded ontology. The ontology is applied to store the data structure as well as the specific catalogues items. An item could be a product, a service or a company. The huge amount of data within the catalogues requires the application of intensional and extensional queries to find items in an efficient way. In addition, specific filter are necessary to find the right item. The overall search has been designed to meet these challenges.

As an outcome, a search approach with reduced search space and indexed data and a search approach enabling the full exploration of data using a graph based navigation has been developed and will be evaluated in detail in the first validation phase of the project. New requirements such as support for multilingualism (resulting from the first NIMBLE review) have already been implemented and allow a smooth switch between languages. A first evaluation with the data of MICUNA showed that a user can find his favourite product, open the details and could start the negotiation for it. The graph-based search was helpful in cases in which the used terminology as well as the available information of a product/service are not well known. In these cases, the intensional queries were helpful to concretize the search requests.

The prototype of both search approaches has been deployed and will be improved continuously. The software deployment process of NIMBLE enables the automatic deployment of the prototype if changes are uploaded to GitHub.

6 Outlook

The on-going development will focus on the "Sematic Query Patterns based Search" as well as on the optimization of the usability to handle search requests in varying product catalogues. The variety usually impacts the structure and the amount of products to be explored. The usability must be ensured also in cases in which thousands of products and product categories are available.

Apart from the usability, the inclusion of the user context within search capabilities is essential. The user context shall be used to derive assumptions. The derived assumptions shall be used to reduce the search space and therefore, generate better search results. For example, the nationality, the company information and the current business processes should be processed in order to define the context.

Finally, the deployment and evaluation of the search for all business cases will close the work in this task.

7 Appendix

7.1 Additional Information for Users

This subsection provide additional information on the usage of the Explorative Search User Interface provided within the NIMBLE Platform. We will refer to figures illustrated in the aforementioned sections of the Deliverable in order to maintain brevity and avoid redundancy.

The information provided for the Explorative Search's User Interface is based on the implementation available in the GitHub Repository[3] of the Frontend-Service and is liable to change with future updates of the feature itself.

Pre-requisite: we assume that the user is already logged into the platform and can access all available features provided by the platform.

[3] NIMBLE Frontend-Service https://github.com/nimble-platform/frontend-service

Navigation Explorative Search's User Interface

The user can access the search by clicking on the **Explorative** option in the **Search** button situated in the Navigation Bar of the platform (Figure 32)

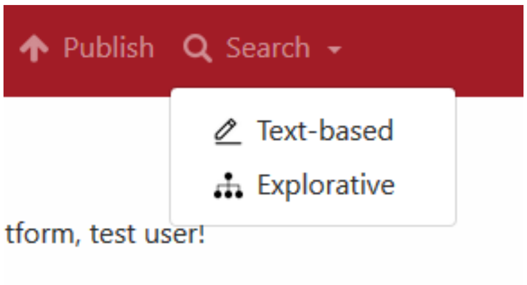


Figure 32 Navigating to the Explorative Search in the Platform

Search for Products

The steps are the same as mentioned in Section 2.4.1 where a user enters keywords in the search bar. The outcome provides results similar to Figure 4. Upon selecting a desired product button, a visualization of the product along with its properties are displayed similar to Figure 7.

Selection of Product Properties

The user can interact with the visualization by Mouse-clicks and using the CTRL key. The CTRL key is used when the user wishes to obtain a detailed information of the product w.r.t. multiple properties. In order to do so, the user needs to keep the CTRL key pressed when clicking the property nodes of the visualization.

The implementation of the single property and multiple properties are considered different use case scenarios hence if a User interacts with the product visualization without the CTRL key pressed, the user interface will provide a warning modal as shown below in Figure 33.

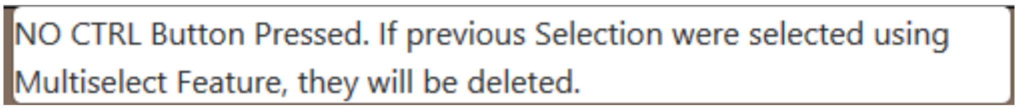

NO CTRL Button Pressed. If previous Selection were selected using Multiselect Feature, they will be deleted.

Figure 33 Modal Warning for User when using Single Selection of Product's Property

As the modal suggest if the previous interactions with the visualization were made with multiple selections they will be deleted and only the currently selected property will be displayed.

The user can click anywhere in the background to remove the modal warning.

Figure 8 and Figure 10 describe the outcome of using single and multiple selection of product's properties respectively.

Removal of Selected Properties / Resetting the Selection

If the user wishes to reset the selection of the visualization, it is possible to do so by clicking anywhere in the vicinity of the visualization itself. This will remove all the selected property nodes, their respective filters and the table below the figure.

If the user clicks on a property node and wishes to deselect that particular property; it can be obtained by clicking the node again with the CTRL key pressed. This method could be used for both: Single or Multiple Selection of properties. An illustration is seen in Figure 34.

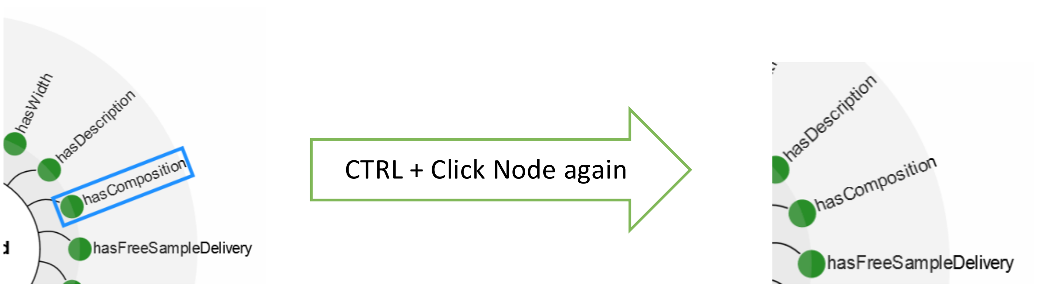


Figure 34 Deselection of a Product's Property

Removing Property Nodes from Visualization

If in some cases, if a user wishes to remove a node due to readability issues; it is possible to remove a node from the visualization by firstly selecting the property node and using the DELETE key.

However, we advise caution in cases when multiple selected properties are involved. If previous selections already exist, deleting the present node would delete all the selected nodes too. It is preferable to first remove the particular nodes without using the CTRL button *i.e.* using the single selection and then proceeding further to multiple selection of the properties.

Dragging Property Nodes within the Visualization

Property nodes can also be dragged to a different location as opposed to deletion for more clarity. In order to drag a node, simply keep the left mouse key pressed while interacting with the node of interest.

Filtering / Searching for Detailed Information

The outcome of selected product properties are similar to Figure 13 and Figure 14 when the user clicks on the red Search Button. Upon clicking the "More" button within the table a complete information of the product is displayed similar to Figure 18.