HACID - Deliverable

Dashboards for use cases

This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101070588. UK Research and Innovation (UKRI) funds the Nesta and Met Office contributions to the HACID project.

Deliverable number:	D3.2
Due date:	28.02.2025
Nature[1]:	OTHER
Dissemination Level[2]:	PU
Work Package:	WP3
Lead Beneficiary:	CNR
Contributing Beneficiaries:	CNR

[1] The following codes are admitted:

○ R:	Document, report (excluding the periodic and final reports)
○ DEM:	Demonstrator, pilot, prototype, plan designs
○ DEC:	Websites, patents filing, press & media actions, videos, etc.
○ DATA:	Data sets, microdata, etc.
○ DMP:	Data management plan
○ ETHICS:	Deliverables related to ethics issues.
○ SECURITY:	Deliverables related to security issues
○ OTHER:	Software, technical diagram, algorithms, models, etc.

[2] The following codes are admitted:

- ○ PU – Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)
- ○ SEN – Sensitive, limited under the conditions of the Grant Agreement
- ○ Classified R-UE/EU-R – EU RESTRICTED under the Commission Decision No2015/444
- ○ Classified C-UE/EU-C – EU CONFIDENTIAL under the Commission Decision No2015/444
- ○ Classified S-UE/EU-S – EU SECRET under the Commission Decision No2015/444

Document History

Version	Date	Description	Author	Partner
V1	14/07/2025	First draft with document structure and introduction	Massimiliano Schembri	CNR
V2	17/07/2025	Section 2 and 3 drafted	Massimiliano Schembri	CNR
V3	23/07/2025	Section 2 restructured and completed with pictures	Massimiliano Schembri	CNR
V4	25/08/2025	Updated section 3	Massimiliano Schembri	CNR
V5	01/09/2025	Completed section 3, and first document revision	Massimiliano Schembri, Vito Trianni	CNR
V6	08/09/2025	Revision of the full document	Vito Trianni, Massimiliano Schembri	CNR

Table of content

1. Introduction

In this document, we describe the dashboard for case knowledge refinement developed for the climate service use case. While the initial project plan envisioned developing dashboards for the two use cases—starting with a basic version for the medical diagnostics case and later a full-featured dashboard for the climate service use case—the actual work focused solely on the climate service use case. Indeed, we quickly recognised that integrating a novel interaction method onto the Human Dx mobile app would have been too expensive and risky, jeopardizing the user experience that Human Dx customers are familiar with. Also, there was no real need to expose Human Dx users to the domain knowledge graph for medical diagnostics, as the autocompletion method already provides entity linking features.

The developed dashboard was designed for climate scientists and it took the form of a knowledge graph visualization and query system (see also D5.2 for a user research conducted to determine the best visualization and exploration approach). As we will explain in the next sections, it enables users to explore and interact with the Domain Knowledge Graph (DKG) and tag concepts and relations relevant to a given climate case. The dashboard is available as a standalone web application or as an integrated component within the HACID-DSS web application, where it supports climate scientists in the case solution process.

The working principle of the developed dashboard can be applied to any KG with minor intervention. Hence, it can also work as a dashboard for the medical diagnostics DKG, should such a necessity arise in the future.

Note: the version of the software described in this document is dated August 1, 2025. Updates versions of both the software and this manual are available on the GitHub repository at the following address: https://github.com/hacid-project/hacid-kg-visualisation

2. The Knowledge Graph Visualisation and Query System

Case knowledge exploration and refinement faces significant challenges due to the overwhelming size and complexity of modern knowledge bases. Even when well-structured, vast domain knowledge can lead to cognitive overload, confirmation bias, or oversimplification, making it difficult for individual or collective experts—and even AI systems—to efficiently identify relevant evidence. Traditional dashboards, often limited to fixed features and static visualizations, risk either being overwhelming for users or stripping away essential trade-offs, and they rarely support interactive knowledge exploration. Addressing open-ended domains where the scope of relevant information is unknown requires advanced tools that combine intuitive exploration interfaces with computational methods for prioritizing and pruning knowledge. One of the main goals of the HACID dashboard is to make knowledge graph data easier to explore and understand. A knowledge graph represents information as a network of "nodes" (which stand for concepts or entities) connected by "links" (which show relationships between them). By visualising this data as an interactive network/map, users can intuitively see how ideas are related, uncover hidden patterns, and follow connections in a way that feels natural—much like navigating a mind map. This approach transforms abstract data into a clear and engaging visual experience, making it more accessible to people without technical backgrounds.

At the beginning of the dashboard design process, we were uncertain which visualization system would be most suitable for our target users. The process began with a review of the existing knowledge graph exploration system (Bernasconi et al., 2023; Po et al., 2020), followed by the development of two functional prototypes. The first prototype was based on a custom node-link visualisation system while the second one was based on the Sparnatural visual query system [ref]. These two approaches were presented to a small group of users during the Knowledge Graph visualisation and Value Elicitation workshop (see D5.2). They found the node-link approach particularly effective for exploratory use, noting that it could be useful for identifying gaps in the knowledge graph. Moreover, compared to the visual query (VQ) system, the node-link interface was also perceived as more intuitive and easier to navigate. Based on this feedback, we decided to proceed with the further development and refinement of the node-link visualization system as the primary interface for the dashboard. To complement its exploratory strengths, we also chose to integrate visual query capabilities, providing users with a more structured and flexible way to search and filter information within the knowledge graph.

2.1 The main working area

The main working area of the HACID dashboard is primarily dedicated to the visualization of knowledge graph information. Figure 1 illustrates an example where several concepts are displayed along with their connecting relationships. A small toolbar is located on the left side of the screen, allowing users to select different types of exploration and filtering tools. The currently selected tool is highlighted with a gray background. In the following sections, we will provide a detailed description of the main functionalities of the working area.

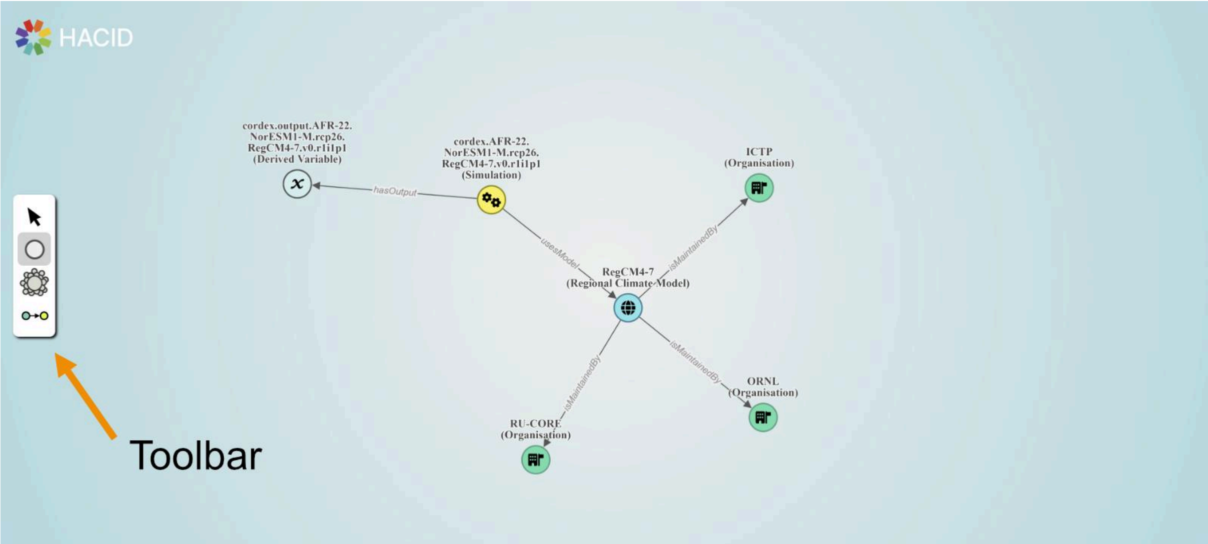


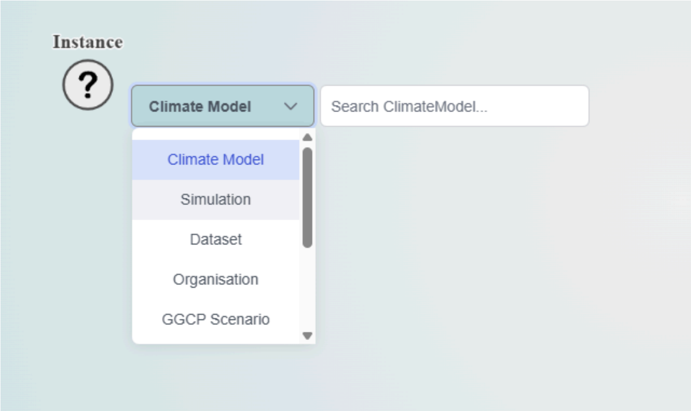
Fig. 1 - Working area of the dashboard

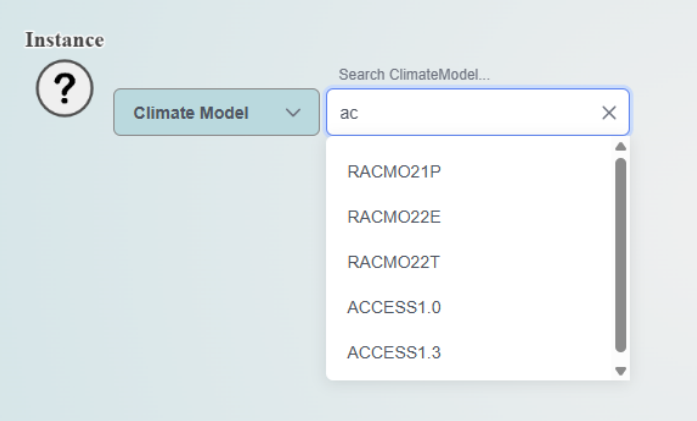
2.1 Navigating the workspace: pan and zoom

When dealing with large and complex knowledge graphs, a significant amount of information may be displayed simultaneously, making it challenging for users to interpret or focus on specific areas. The ability to zoom in and out, performed with the mouse wheel, allows users to control the level of detail they see. Zooming in helps examine individual nodes and their immediate relationships more closely, while zooming out provides an overview of the broader structure. Panning, which is done by right-clicking and dragging the mouse, complements the zooming functionality by enabling users to navigate across different parts of the graph without losing their zoom level. Together, these interactions offer a flexible and intuitive way to explore densely populated visualizations, helping users maintain context while examining specific areas of interest.

2.2 Exploring KG instances and their relations

One possible starting point for the visualisation is searching for a specific instance in the knowledge graph and starting the exploration from there. For this purpose we can use the instance tool (second from top in the toolbar). Once selected the tool, we can click on the working area in the place we want to position the new instance. A drop down menu and a search-bar will appear. We can choose the class type of the new instance and then we can use the search bar to look for an instance filtering by the label.





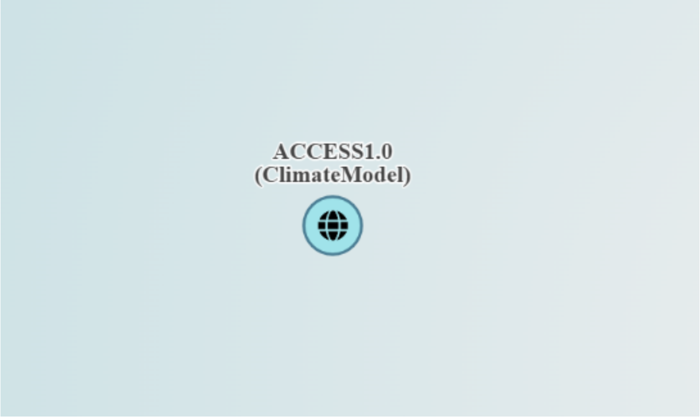


Fig. 2 - Selecting a KG instance

The selected concept will be instantiated as a colored circle containing an icon that indicates the class type it belongs to.

2.2.1 Context Radial Menu

Right-clicking on the instance of a concept will open a radial context menu, allowing one to choose from a range of functions that apply to that specific instance. The **delete** function will remove the instance from the visualisation (trashbin icon). The **information** function (indicated by the italic '*i*') will show some details of the instance (URI, label, comments, datatypes etc) and allows linking directly to the web page describing the RDF resource. The **tag** tool allows tagging as described in the “Tagging functionality” section of this document (see Section 2.6).

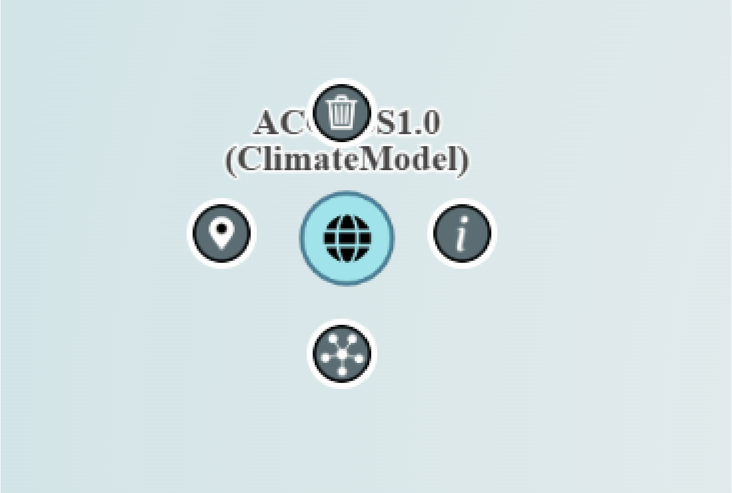
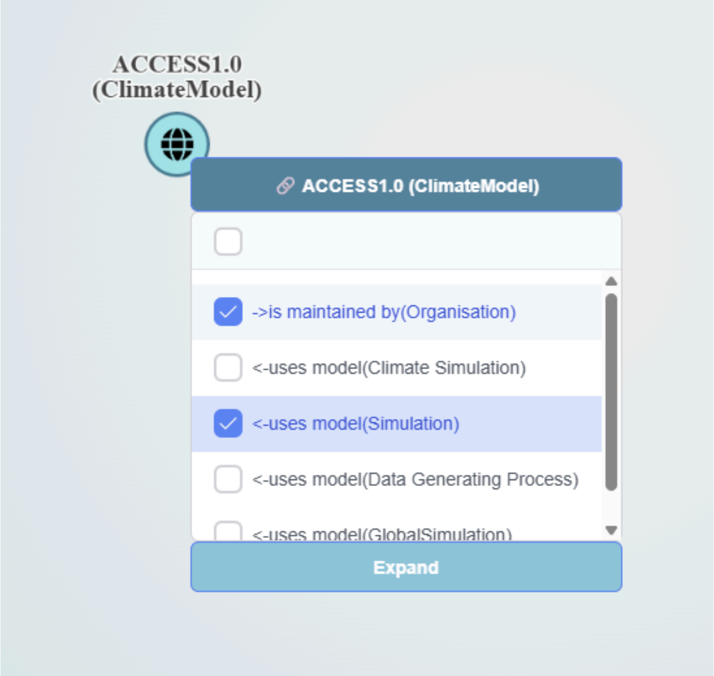


Fig. 3 - The radial context menu

The relation function is indicated by an icon representing some nodes linked together. It allows one to explore all the triples associated with the selected instance. Clicking on it will show a panel with a list of relations and associated classes.



☐

☑ ->is maintained by(Organisation)

☐ <-uses model(Climate Simulation)

☑ <-uses model(Simulation)

☐ <-uses model(Data Generating Process)

☐ <-uses model(GlobalSimulation)

Fig. 4 - Exploring instance relations

Each row contains a checkbox, an arrow indicating the direction of the relation, the name of the relation and the associated ontology class. We can select one or multiple relations (or all of them) and click the “expand” button to instantiate them. In the example shown in fig. 4 we are expanding two types of relations, the “is maintained by” and “uses model” relations that link a Climate Model respectively to Organisation and Simulation instances. We can see the results in fig 5.

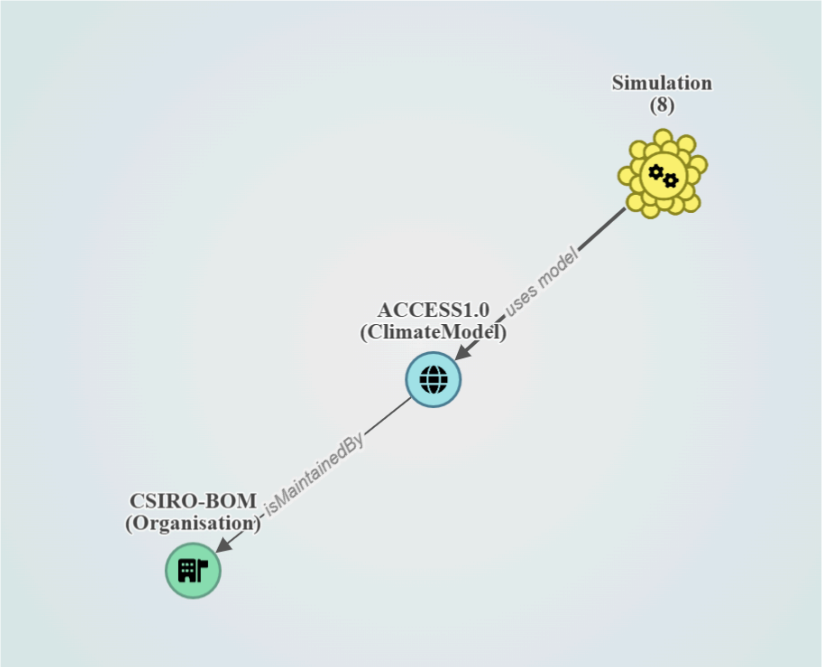


Fig. 5 - Expanded relations

As we can see the ACCESS1.0 Climate model is linked to the CSIRO-BOM organisation with a “is maintained by” relations, and there are 8 simulations that are connected to it with the uses model relation. To avoid cluttering the visualization with too many nodes, multiple nodes connecting to the same instance are represented by a group node'—a single node that stands for a collection of nodes (eight in this case). Group nodes, as we will see in the next section, are a powerful tool to implement some basic functionalities of visual querying and filtering.

2.3 Visual querying and filtering

In the previous section, we discussed instance exploration, a type of exploration that begins with the content of the knowledge graph. In this section, we demonstrate how the visualization tool supports an alternative approach that starts from class types, enabling exploration of the knowledge graph's schema. While introducing relation functions, we presented the group node, a special type of node that represents a collection of instances belonging to the same class. Group nodes can also be created directly using the **group node tool** (the third tool from the top in the toolbar). By selecting this tool and clicking on the workspace, users can choose the class type they wish to explore (see Fig. 6). Once a class is selected, a new group node appears, labeled with the class name and a number indicating the total instances of that class present in the knowledge graph.
Starting from a group node and using the radial context menu (right click on the node), we can explore the relations of the class in a similar way to how we did with an instance.

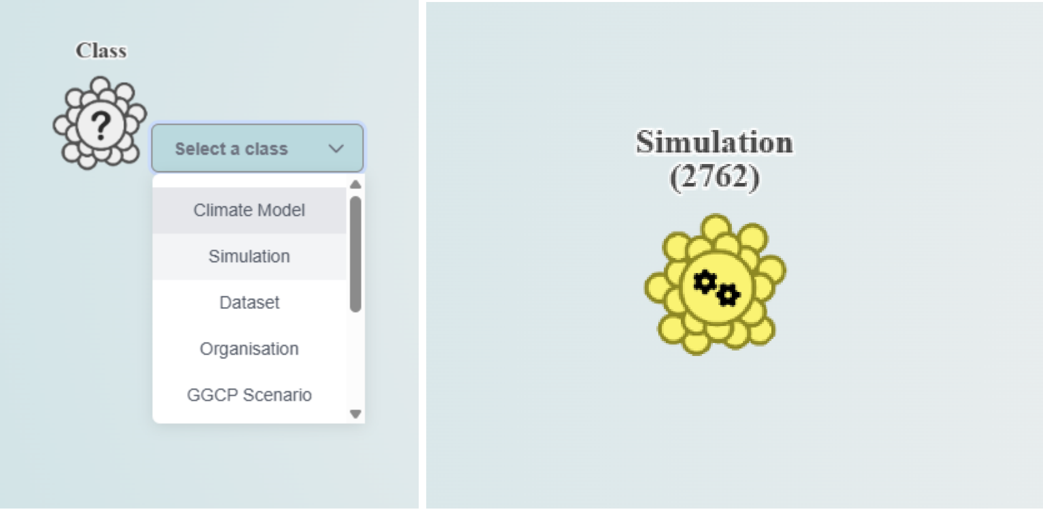


Fig 6 - group node tool. Selecting the class type (on the left). After selecting the class type (on the right).

The icons of the radial menu are the same as for the instance node. A relation panel appears showing all relations that the selected class has with other classes. As for the instance node, each row shows the direction of the relation (inward, outward), the name of the relation and the subject class.

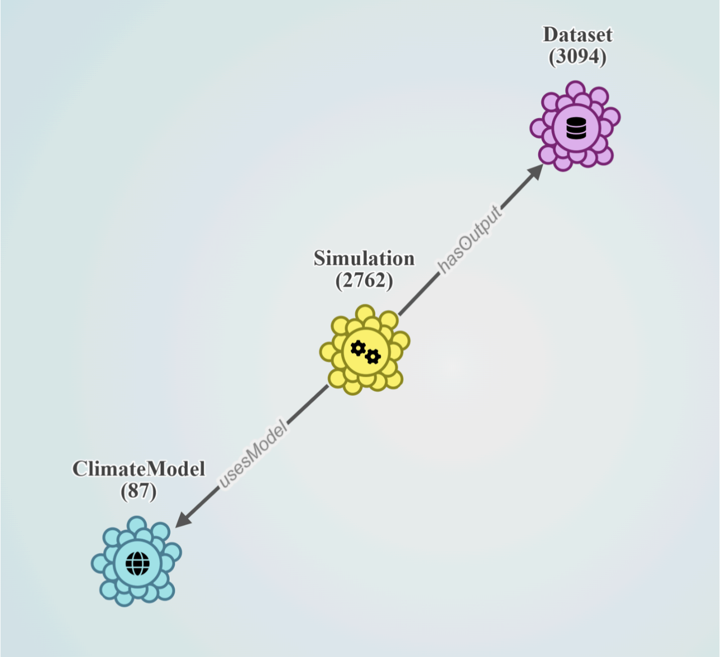


Fig 7 - Simulation node after two relations have been expanded

Multiple relations can be selected and expanded simultaneously. Figure 7 shows an example where the “uses model” and “has output” relations have been expanded. The numbers in parentheses represent the number of instances that are linked with other instances of the other classes.

Both the instantiation of a group node and the exploration of its relationships are carried out through interaction with the HACID visual interface. Under the hood, however, a SPARQL query is dynamically constructed during this process starting from the connected group nodes. Together with the filtering functions that we will describe later, this effectively characterises the system as a visual SPARQL query engine. A set of connected group nodes is referred to as a **query group**. This term will be used throughout the document to denote operations involving such structures.

2.3.1 Filtering functions

Instances of group nodes can be filtered in two different ways. One is by using the relation tool available in the tool box (last from the top). This tool can be used to connect a group node to the instance of a concept present in the workspace. The effect of this operation is to filter all concepts in the group node that have at least one relation with the connected concept. Let’s examine a concrete example. Suppose we have instantiated a simulation group node, which represents all the 2762 concepts of simulations present in the knowledge graph (as shown in Fig 6). Let’s now create an instance of a climate model concept. We select the instance tool, select Climate Model as class type and then we look for a specific climate model named “CanRCM4”. An instance of this concept will appear in the workspace. Now if we select the relation tool and then drag from the Simulation group node to the climate model instance we can connect them as shown in Fig 8.

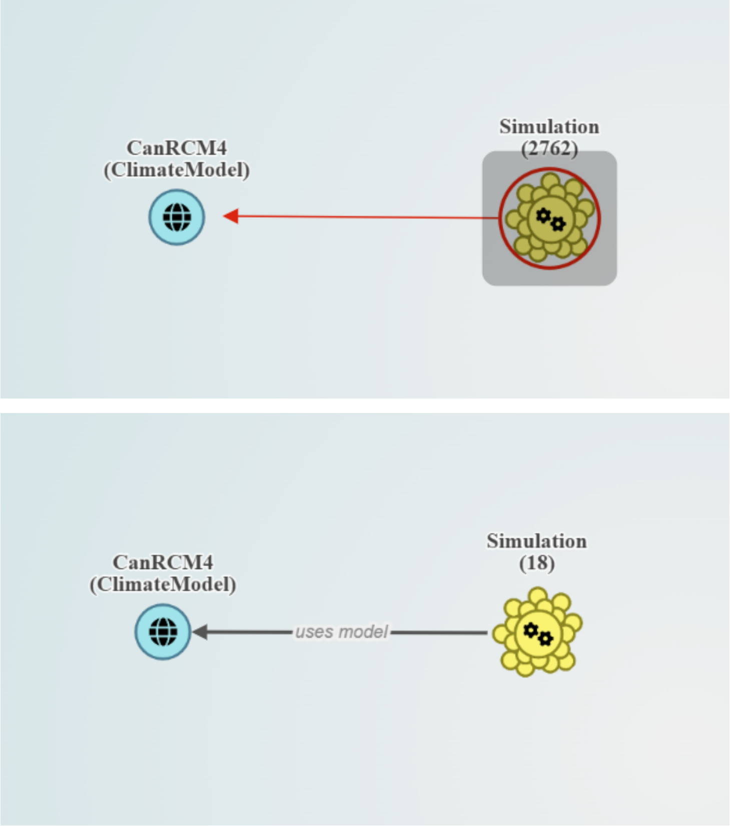
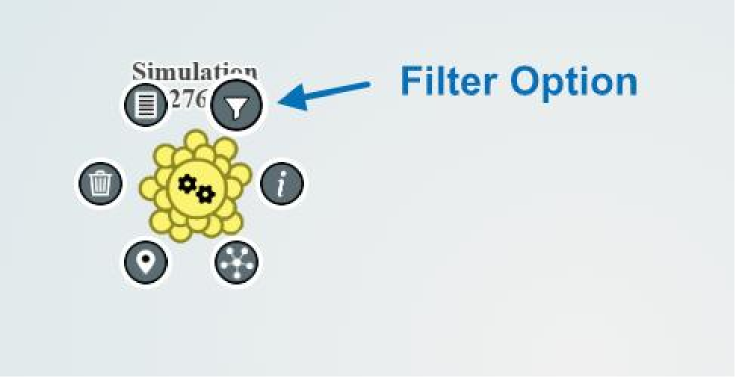


Fig 8 - Dragging and connecting the Simulation group node with climate model instance

We can notice that now the simulation group node comprises only 18 concepts, as this is the number of simulations present in the knowledge graph that use the specified model. That is because the connection with the CanRCM4 model acts as a filter showing only simulations that have a relation with that type of climate model.

Another type of filtering can be applied directly to the group node and is accessible through the corresponding option in the radial menu. For example, we can start again with the simulation group node as shown earlier. By right-clicking on it, we can locate the filter function, indicated by the funnel icon. If we click on it we are presented with the filters available for that type of group node as shown in Figure 9. In this case, the only available filter is the "Label contains" filter. If we choose it we can filter all the concepts whose label contains some text that we can specify in a text box. For instance, we can obtain all simulations from the Cordex project by looking for simulations that contains the word "cordex" in their label.



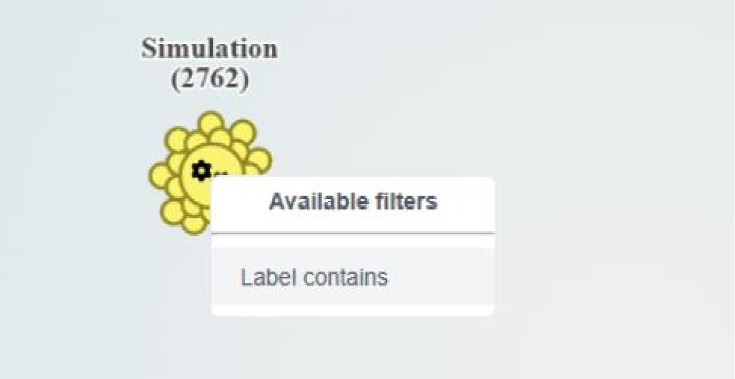


Fig. 9 - group node filter options

After performing the filtering operation the result is shown in Figure 10. The applied filter is visualised as an external funnel component that “acts” on the group node. As a result of applying the filter, the group node now shows 2121 out of a total of 2762 available simulations.

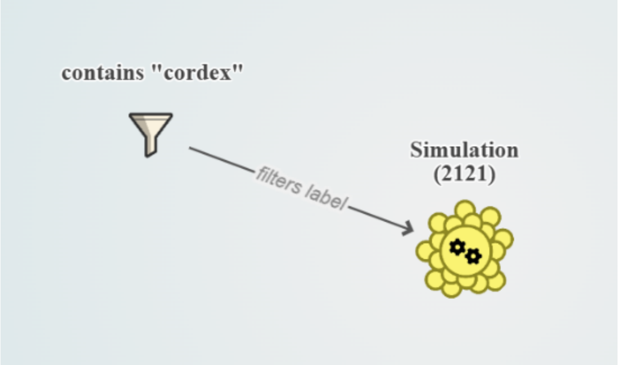


Fig. 10 - The “Label contains” filter applied to the Simulation group node

2.4 Example use case

The querying and filtering functionalities described previously will now be demonstrated through a practical use case. Starting from a specific organization, we will identify a dataset that meets defined criteria.
First, we will instantiate a node of type “Organisation” and we will look for an instance called “MOHC” which stands for Met Office Hadley Centre. From this instance, using the relation option we will expand all Climate Models maintained by this organisation. The results of these operations are displayed in Figure 11.

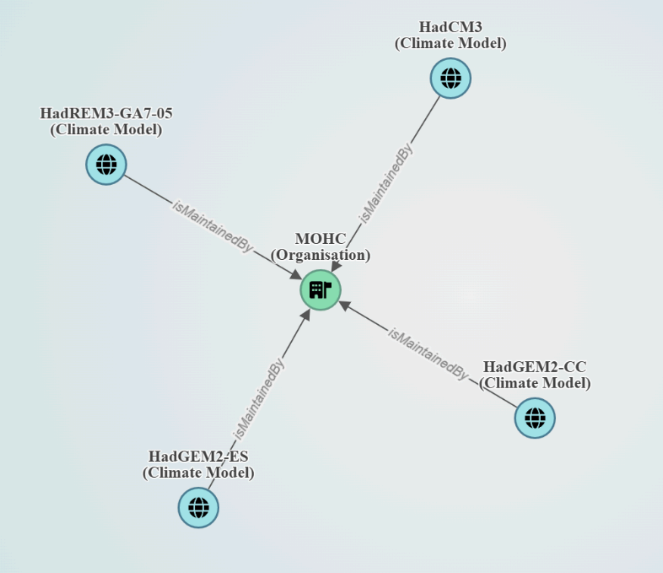


Fig. 11 - Looking for all climate models maintained by the MOHC Organisation

As shown, the MOHC organization maintains four climate models. Our focus is on simulations produced using the HadGEM2-ES model. To explore this, we navigate to the corresponding climate model instance and use the radial menu to expand the “uses model” relationship. This reveals a group node representing 25 associated simulations. To refine the results, we apply a *label filter* to identify simulations categorized as historical. By searching for the keyword historical in the labels, we narrow the results down to five simulations, as illustrated in Figure 12.

To view a detailed list of these simulations, we use the radial menu and select the List Detail option, represented by a document sheet icon. This opens a panel displaying the labels of the items within the group node (see Figure 13). From this list, one or more items can be selected and further expanded. In this context, the "expand" button has a different meaning with respect to the relation option. It refers to the fact that single items can be instantiated as an individual entity within the workspace, allowing it to serve as a new starting point for a new knowledge graph exploration.

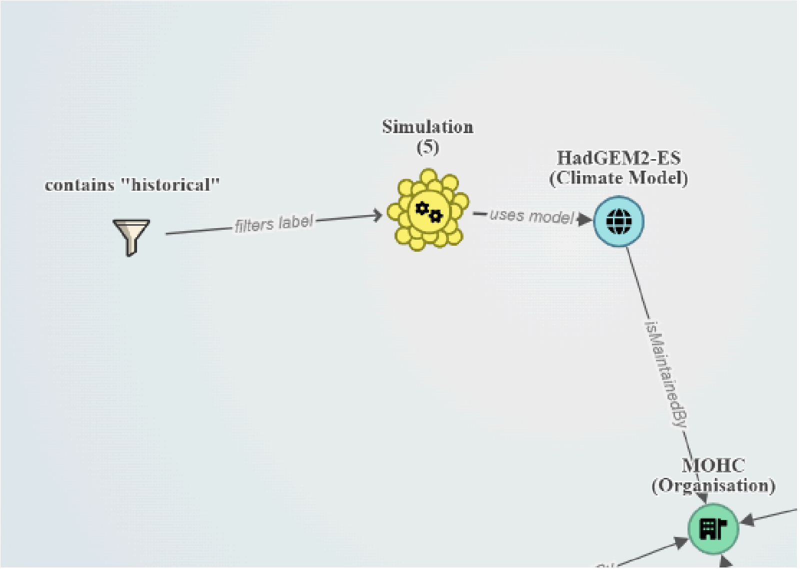


Fig. 12 - Finding the 5 historical simulation generated by the HadGEM2-ES model

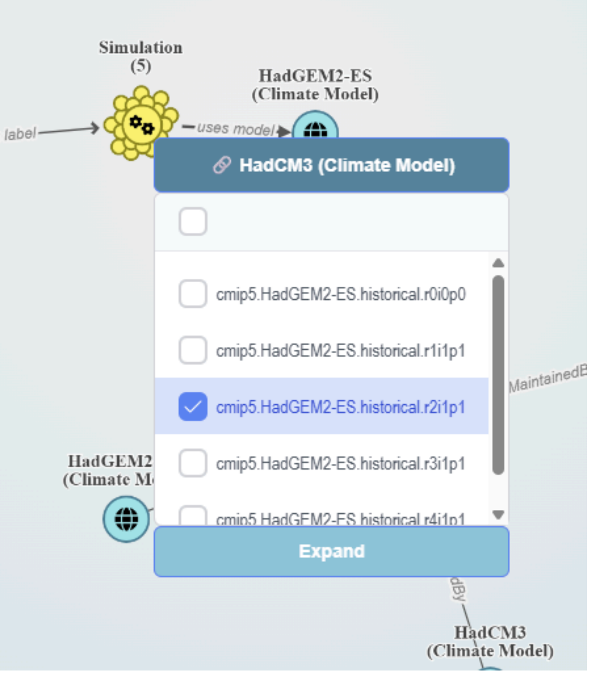

Simulation (5)
HadGEM2-ES (Climate Model)
label
uses model
HadCM3 (Climate Model)
□
□ cmip5.HadGEM2-ES.historical.r0i0p0
□ cmip5.HadGEM2-ES.historical.r1i1p1
☑ cmip5.HadGEM2-ES.historical.r2i1p1
□ cmip5.HadGEM2-ES.historical.r3i1p1
□ cmip5.HadGEM2-ES.historical.r4i1p1
Expand
MaintainedB
HadGEM2 (Climate M
dBy
HadCM3 (Climate Model)

Fig. 13 - Visualising the details of a group node items, in this case the 5 simulations found

2.5 The integrated YASGUI interface

YASGUI (Yet Another SPARQL GUI) is a web-based interface for querying RDF data using SPARQL. YASGUI offers features such as syntax highlighting, autocomplete, query history, and result visualization, making it easier to interact with SPARQL endpoints without the need for external tools.

In the HACID dashboard, YASGUI has been integrated to provide users with an embedded query editor directly within our application. To see how it works let us examine the SPARQL query that retrieves the five simulations identified in the example use case. To do that, we have to go over the Simulation group node, right click and choose the info option in the radial menu. An info panel will pop out from the right side of the screen. Inside the panel there is a button with the SPARQL icon on it. This button allows you to edit the query associated with the node.

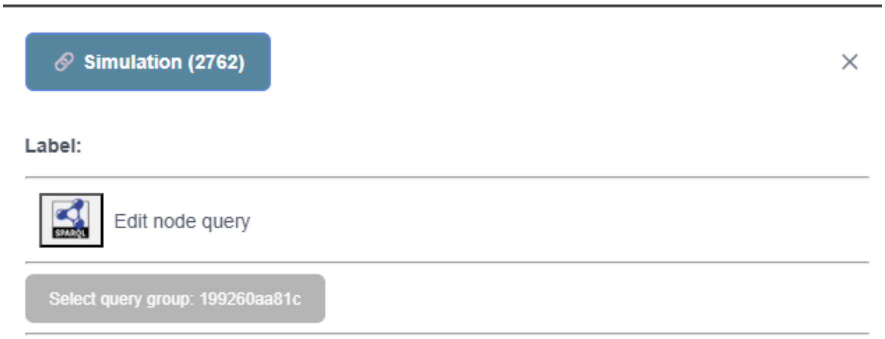
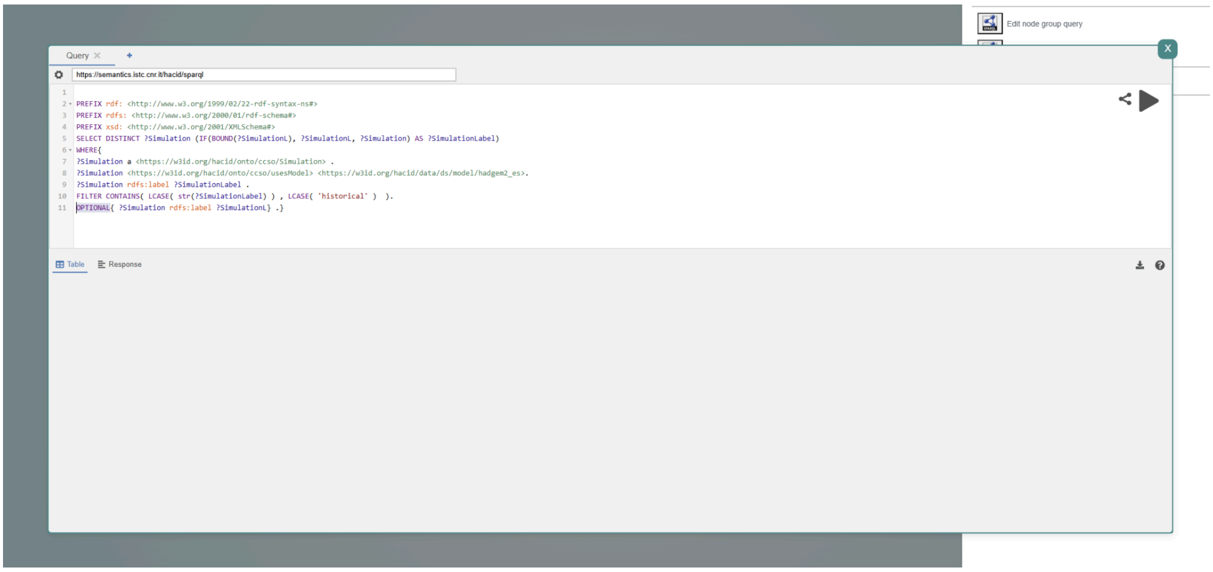


Fig. 14 - accessing the YASGUI interface

By clicking on it, the associated SPARQL query is displayed. Doing so will bring up a window with the classical YASGUI interface. It consists of an intuitive, browser-based editor designed for composing and executing SPARQL queries. It features a main text editor area with syntax highlighting and autocomplete to assist in writing queries. Below the editor, query results are displayed in a tabular format by default, with support for downloading results in various formats (e.g., JSON, CSV, XML). Additional features include query history, multiple tabs for managing different queries, and an endpoint-specific configuration panel, offering a user-friendly environment for interacting with RDF data.



```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?Simulation (IF(BOUND(?SimulationL), ?SimulationL, ?Simulation) AS ?SimulationLabel)
WHERE{
?Simulation a <https://w3id.org/hacid/onto/ccso/Simulation> .
?Simulation <https://w3id.org/hacid/onto/ccso/usesModel> <https://w3id.org/hacid/data/ds/model/hadgem2_es>.
?Simulation rdfs:label ?SimulationLabel .
FILTER CONTAINS( LCASE( str(?SimulationLabel) ) , LCASE( 'historical' )  ).
OPTIONAL{ ?Simulation rdfs:label ?SimulationL} .}
```

Fig. 15 - The integrated YASGUI interface, which enables inspection and modification of visually generated SPARQL queries.

On the left side of the editor there is an "Execute" button (play icon) used to run the query. If we click on it, we will execute the query and see the result below. Users that are familiar with the SPARQL query language can proceed with refining the query and make the modification they want. Please note that any changes made here will not affect the original query stored in the group node.

2.6 Tagging functionality

The user interface provides functionality for tagging nodes. The tagging functionality is accessible through the radial menu and allows users to tag nodes that they deem relevant.

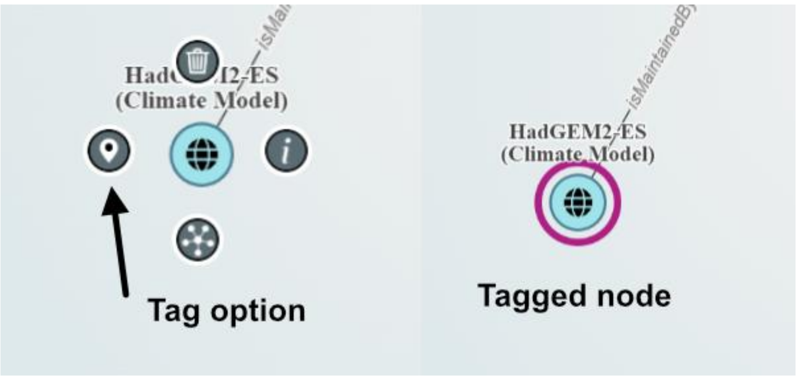


Fig. 16 - Node tagging

Once tagged a node will be highlighted by a coloured circle, and this action triggers a judgement pattern implementation within the underlying knowledge graph structure. This pattern, comprehensively documented in Deliverable D2.1, represents a fundamental mechanism for capturing user assessments and annotations within the semantic framework.

When a user initiates the tagging operation on a specific node through the interface, the system automatically instantiates the corresponding judgement pattern as defined in the knowledge graph ontology. The technical implementation involves the creation of a new instance belonging to the jdg:Judgement class, which serves as the central entity for representing the user's tagging action. This judgement instance acts as a hub that connects multiple related entities through well-defined semantic relationships, ensuring that the user's input is properly contextualized within the broader knowledge graph structure.

Two primary relational connections are established during this process. First, the system creates a `jdg:isJudgementOn` relation that explicitly links the judgement instance to the target node that the user has selected for tagging. This relationship clearly identifies the subject matter or entity that is being evaluated, ensuring that the judgement can be traced back to its specific target within the knowledge graph. Second, a `jdg:hasJudge` relation is established between the judgement instance and the user instance node, thereby capturing the provenance of the judgement and enabling the system to track which user made which assessments.

This relational structure not only preserves the immediate tagging information but also enables sophisticated querying capabilities, allowing the system to retrieve judgements by user, by target entity, or by various combinations of these criteria. The pattern supports the broader goals of maintaining data integrity, enabling collaborative annotation, and facilitating advanced analytical operations on the accumulated user-generated metadata.

For a comprehensive technical specification of the judgement pattern, including detailed class definitions, property descriptions, and usage examples, we suggest to consult the complete documentation provided in Deliverable D2.1, which offers an exhaustive treatment of this foundational semantic modeling approach.

3. Technical Implementation Overview

Modern web development relies on a rich ecosystem of technologies designed to create responsive, scalable, and interactive user experiences. At the core of this ecosystem is HTML (Hypertext Markup Language), which defines the structure of web content, and CSS (Cascading Style Sheets), which manages presentation and design. Together, these provide a standardized foundation for rendering interfaces across browsers and devices. On top of this foundation, JavaScript has evolved as the primary programming language for building interactive features on the web. Its superset, TypeScript, introduces static typing and advanced tooling, allowing developers to write more robust, maintainable, and scalable applications.

Modern web applications often adopt a Single-Page Application (SPA) architecture, where a single HTML page dynamically updates its content in response to user interactions rather than reloading the entire page. This approach enables smoother user experiences, faster navigation, and seamless integration with back-end services through APIs. Popular frameworks such as Angular, React, and Vue further streamline SPA development by offering modular, component-based architectures. In this paradigm, interfaces are built as reusable components that encapsulate functionality, design, and logic, promoting code reusability and flexibility.

Within this context, the HACID visualization system has been implemented as a self-contained Angular component. This component provides a powerful interface for navigating and refining complex knowledge graphs, leveraging Angular's modular design to ensure both scalability and adaptability. The visualization system can operate as a standalone web application, allowing direct deployment in a browser environment, or be embedded into other applications to extend their capabilities with HACID's advanced knowledge exploration features. This dual functionality makes the component suitable for a variety of deployment scenarios, from dedicated analytical platforms to integration with larger software ecosystems, while maintaining a consistent user experience.

The angular component is composed of two layers, one for the knowledge graph visualisation and one for the gui elements. These are described in the following sections.

3.1 The knowledge graph visualisation layer

The visualisation is implemented using the Cytoscape.js library (https://js.cytoscape.org/), an open-source JavaScript library specifically designed for graph theory and network visualization. It provides a robust and flexible framework for displaying complex, interactive networks on the web. For the custom visualization system developed within this project, Cytoscape.js was selected due to its powerful rendering capabilities, extensive customization options, and strong community support. Its robust feature set includes support for various graph types (directed, undirected, multigraphs, compound nodes), a comprehensive API for graph manipulation (adding, removing, modifying elements), built-in gesture support for intuitive user interaction (panning, zooming, dragging), and a powerful CSS-like styling system that allows for extensive visual customization. Its use has enabled the creation of a highly interactive and intuitive interface, allowing users to explore and analyze the intricate relationships within our data effectively.

3.2 The GUI overlay layer

All supplementary user interface elements, including the radial context menu, relation expansion panel, and toolbox, are implemented using the PrimeNG component library and rendered as overlay elements positioned above the Cytoscape visualization layer. PrimeNG (https://primeng.org/) is a comprehensive Angular UI component library that provides a rich collection of pre-built, professionally designed components optimized for enterprise applications.

The strategic decision to implement the GUI layer using PrimeNG provides numerous architectural and development benefits. PrimeNG offers a mature, well-tested component ecosystem that accelerates development while ensuring consistent design patterns and user experience across the application. The library's components are built specifically for Angular, seamlessly integrating with Angular's modular architecture, declarative templating system, bidirectional data binding capabilities, and sophisticated dependency injection framework.

The system architecture leverages PrimeNG's extensive component catalog to create sophisticated interface elements with minimal custom implementation. Each PrimeNG component encapsulates complex functionality, accessibility features, and responsive design patterns, while maintaining the flexibility for customization to meet specific application requirements. This approach significantly reduces development time and ensures adherence to established UI/UX best practices.

3.3 GitHub repository

The code and an updated version of the current document is available on a GitHub repository at the following address:
https://github.com/hacid-project/hacid-kg-visualisation

References

Bernasconi, E., Ceriani, M., Di Pierro, D. D., Ferilli, S., & Redavid, D. (2023). Linked Data Interfaces: A Survey. *Information*, *14*(9), 483. https://doi.org/10.3390/info14090483

Po, L., Bikakis, N., Desimoni, F., & Papastefanatos, G. (2020). *Linked Data Visualization: Techniques, Tools, and Big Data*. Springer International Publishing. https://doi.org/10.1007/978-3-031-79490-2