Sampo-UI: A Full Stack JavaScript Framework for Developing Semantic Portal User Interfaces

Esko Ikkala [a,*], Eero Hyvönen [a,b], Heikki Rantala [a], and Mikko Koho [a,b]
[a] *Semantic Computing Research Group (SeCo), Aalto University, Department of Computer Science, Finland*
E-mail: firstname.lastname@aalto.fi
[b] *HELDIG – Helsinki Centre for Digital Humanities, University of Helsinki, Finland*
E-mail: firstname.lastname@aalto.fi

Editor: Tania Tudorache, Stanford University, USA
Solicited reviews: Peter Haase, Metaphacts, Germany; Five anonymous reviewers

Abstract. This paper presents a new software framework, SAMPO-UI, for developing user interfaces for semantic portals. The goal is to provide the end-user with multiple application perspectives to Linked Data knowledge graphs, and a two-step usage cycle based on faceted search combined with ready-to-use tooling for data analysis. For the software developer, the SAMPO-UI framework makes it possible to create highly customizable, user-friendly, and responsive user interfaces using current state-of-the-art JavaScript libraries and data from SPARQL endpoints, while saving substantial coding effort. SAMPO-UI is published on GitHub under the open MIT License and has been utilized in several internal and external projects. The framework has been used thus far in creating five published and six forth-coming portals, mostly related to the Cultural Heritage domain, that have had tens of thousands of end-users on the Web.

Keywords: Linked Data, Semantic portal, User interface, Web application, JavaScript, Software framework

1. Introduction

A fundamental underlying idea of the Semantic Web is to share Linked Data (LD) which is harmonized using ontologies. This approach has been proven useful in, for example, the Cultural Heritage (CH) domain in aggregating, harmonizing, and enriching data silos of different galleries, libraries, archives, and museums (GLAM) whose contents are typically heterogeneous and distributed, but often related semantically to each other [1].

Semantic Web content is published for machines via LD services and for human consumption using web-based LD applications, such as semantic portals. In general, semantic portals are information systems which aggregate information from multiple sources and publish them using Semantic Web technologies into user interfaces for solving information needs of end-users [2–6]. Such portals, based on knowledge graphs published in LD services, typically provide the end-user with intelligent services for data exploration [7] and analysis based on the well-defined semantics of the content. These services may include faceted, ontology-based, and entity-based search engines, semantic browsing based on semantic relations extracted and reasoned from the underlying knowledge graphs, and tooling for data-analysis, visualization, and serendipitous knowledge discovery [8].

To extend the notion of sharing and reusing data on the Web, this paper proposes that one should harmonize and share the way in which semantic portals, especially their user interfaces, are implemented and used,

*Corresponding author. E-mail: firstname.lastname@aalto.fi.

too. It is argued that in this way implementing semantic portals can be made easier for software developers, and from the end-user's perspective, portals based on similar functional logic are easier to learn to use.

An approach towards these goals is the *Sampo* model[1] that has been found useful in practise while developing a series of semantic CH portals [9]. This model advocates the idea of providing the end-user with multiple application perspectives to the contents. The application perspectives are used in two basic steps: Firstly, data of interest is filtered out using faceted search [10] based on ontologies. Secondly, data-analytics tools are applied to the filtered data.

As LD can be used to express virtually everything, a vital challenge is to be able to create user-friendly domain-centric semantic portals with minimal effort. Domain-centric means that the underlying knowledge graphs have been selected beforehand and the scope of the portal is limited to a specific domain. However, the availability of tools for building such custom applications on top of LD is limited [11].

To tackle these challenges, the key contribution of this paper is to introduce the SAMPO-UI framework[2] (hereafter SAMPO-UI refers to the SAMPO-UI framework). SAMPO-UI is a new tool for addressing the following research question: *"How can user interfaces for semantic portals based on Linked Data be built efficiently?"*. As a methodological basis, design science [12] is applied.

A prerequisite for using SAMPO-UI is that the underlying data is published as knowledge graph(s) in SPARQL end-point(s) using the principles of LD [13]. SAMPO-UI is not targeted for data curation, but for creating user interfaces for existing data.

An example of an application of SAMPO-UI, is the Mapping Manuscript Migrations (MMM) portal [14]. Figure 1 depicts one of its faceted search perspectives on over 200 000 medieval and renaissance manuscripts. Manuscripts are first filtered using the facets on the left. After this, the result set can be studied with data-analytic visualizations by choosing one of the following tabs: *Table, Production places, Last Known Locations, and Migrations*. In the figure the Migrations tab has been opened for analyzing the general trend of manuscript movement from their places of production into their last known locations. The MMM portal is an example of a domain-centric semantic portal, where aggregated and harmonized LD and modern web technologies enable studying manuscript provenance on a large scale.

This paper is structured as follows. Section 2 reviews existing tools and surveys related to creation of user interfaces for semantic portals. In Section 3 we present specific requirements for tools for building user interfaces for semantic portals. In addition, requirements regarding the portals themselves are explicated. The SAMPO-UI framework is presented in Section 4. Experiences of applying SAMPO-UI are discussed in Section 5 through a case study on how the framework was used for building a new user interface of a semantic portal about military history. Finally in Section 6 the contributions, impact, and limitations of SAMPO-UI are summarized, and the framework is compared with other related systems.

2. Related Work for Semantic Portal User Interfaces

User interfaces for LD applications can be created in various ways, and can employ a variety of user interaction paradigms [15]. This section provides some relevant references for designing and implementing user interfaces for semantic portals.

Khalili et al. [16] discuss the state of end-user application development for LD, and present the Linked Data Reactor (LD-R), which is an open-source software framework for building modern web-based LD user interfaces. LD-R is based on the idea of using existing solutions, such as the Flux pattern[3] and the React library[4]. The idea is to provide the software developer with a general starting base of a JavaScript web application, which can be configured to view, browse and edit LD. LD-R provides faceted search functionalities through FERASAT [17], which is a serendipity-fostering faceted browsing environment built on LD-R components and their configurations.

SPARQL Faceter [18] is a JavaScript library for building faceted search user interfaces, implemented

[1] Sampo is, according to the Finnish epic Kalevala, a mythical machine giving riches and fortune to its holder, a kind of ancient metaphor of technology.

[2] Information about SAMPO-UI can be found on the homepage https://seco.cs.aalto.fi/tools/sampo-ui; the framework is available under the open MIT License on GitHub: https://github.com/SemanticComputing/sampo-ui.

[3] Flux pattern for building user interfaces: https://facebook.github.io/flux

[4] https://reactjs.org

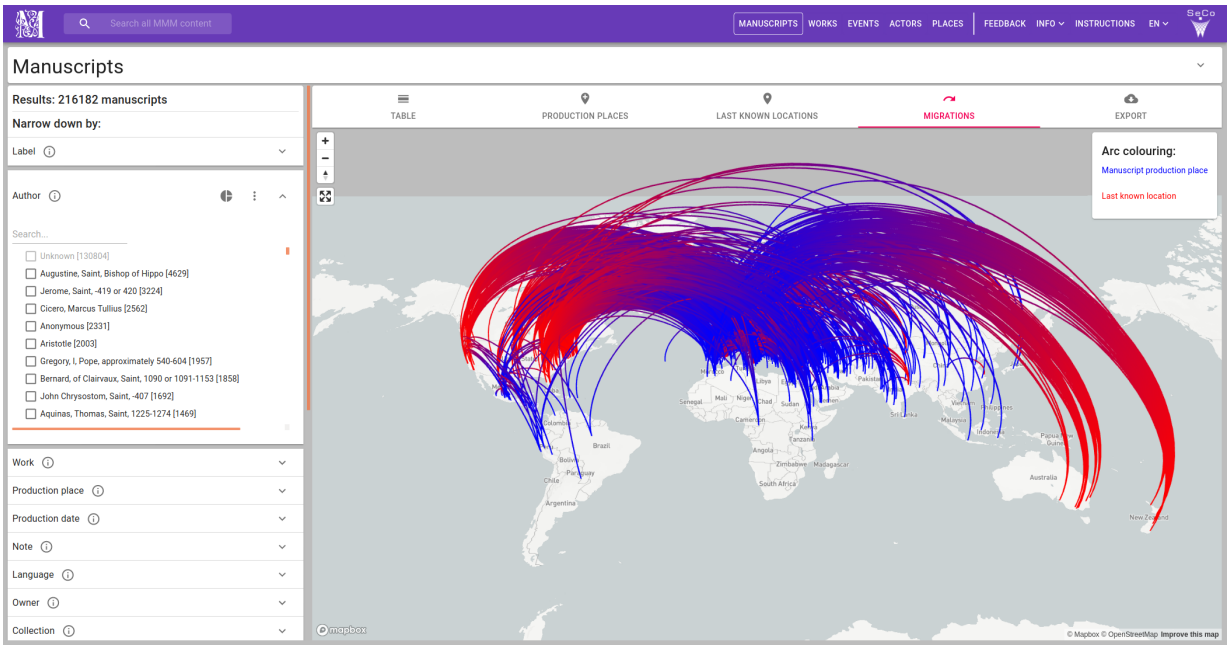

Figure 1. Visualization of the movement of pre-modern manuscripts as arcs from places of production to last known locations using the Mapping Manuscript Migrations Portal. Subsets of manuscripts can be filtered out for visualization using the facets on the left (Label, Author, Work, Production place etc.). The blue end of the arc shows the production place and the last known location is in red (this is not necessarily visible in black and white print). There are different options available for inspecting the filtered result set using tabs on top of the image: Table, Production places, Last knows locations, Migrations (selected in the image), and Export. The Export tab enables the user to export the SPARQL query, which was used for generating the Table result view, into the public SPARQL query service Yasgui. In Yasgui the current result set can be, e.g., downloaded in CSV format for external analysis.

using AngularJS[5] which is not anymore actively developed.

The metaphactory platform [11] is used for building semantic web applications for LD, with focus on both addressing management needs of large organizations and providing domain-centric intuitive end-user interfaces. The platform is deployed in several different use cases, including the ResearchSpace project using the British Museum collection as LD.

Bikakis and Sellis [19] have surveyed LD exploration and visualization systems, and presented some general features and requirements of such systems. Klímek et al. [20] have surveyed existing tooling for LD consumption for non-technical end-users and presented general requirements for end-user LD platforms, encompassing variety of topics, such as, dataset discovery and data manipulation.

Po et al. [21] present LD visualization techniques and tools, and evaluation of the tools in different use cases. Of the presented 16 use cases of LD visualization, five cases (5, 6, 8, 11, and 12) are relevant for semantic portals in general, and cover the basic functionalities of them. None of the evaluated visualization tools are able to handle all of these use cases.

For general tasks related to querying, parsing, and processing RDF[6] data in JavaScript applications, the W3C's RDF JavaScript Libraries Community Group is creating standards and a collection of libraries for using LD on the Web[7].

This section presented related work for creating user interfaces for semantic portals. The SAMPO-UI framework is compared with these related systems in Section 6.2.

3. Requirements for Tools for Building Semantic Portal User Interfaces

In this section, desired features of semantic portal user interfaces are outlined, together with require-

[5] https://angularjs.org

[6] https://www.w3.org/TR/rdf11-concepts

[7] http://rdf.js.org

ments for tools for building them. This is based on earlier research presented in Section 2, as well as the authors' own experience in developing such systems [9, 18].

From the related work regarding tools (e.g. software libraries and frameworks) for implementing LD applications, we found the following requirements (denoted by T) that tools for implementing semantic portal user interfaces should fulfill:

T1. Enable rapid creation of use case-specific applications with minimal effort [11, 18],
T2. Ability to query data directly from a SPARQL endpoint [18, 20],
T3. Support for hierarchical data exploration using ontologies [18, 21],
T4. Scalable techniques used, that can handle a large number of data objects over an exploration scenario, using a limited number of resources [19],
T5. Enable creating intuitive and user-friendly user interfaces for non-technical users [11, 17, 18, 20],
T6. Ability to produce visualizations that automatically adapt to available resources, especially screen resolution and size [21],
T7. Designed and implemented in a way that fosters sustainability [18, 20].

From the related work regarding LD applications, we gathered requirements (denoted by SP) that are relevant for the structure and functionalities of the user interfaces of semantic portals. Most of the semantic portals would need to fulfill these to solve the information needs of end-users:

SP1. The portal supports having multiple perspectives for accessing the knowledge graph(s) [9, 21],
SP2. The perspectives provide methods for browsing and exploring the knowledge graphs to find something useful and interesting without initially knowing exactly what to search for [19, 21],
SP3. The portal employs various search paradigms [19, 22],
SP4. Users can visualize instances that share one or more specific properties, e.g., all instances of a specific class with a common property value [21],
SP5. Users can visualize the properties and relations of a specific instance [21],
SP6. Users can explore and visualize the instances of a specific class [21],
SP7. Users can explore and follow links between various instances [21],
SP8. Users can visualize the data graphically, e.g., geographically on maps, temporally on a timeline, or as a graph diagram of connected entities [20, 21],
SP9. Users can export non-RDF output, e.g., a CSV file [20].

SAMPO-UI and the existing semantic portals using it are evaluated against these requirements in Section 6.1. The next subsections present more detailed observations and requirements related to search paradigms, result set visualizations, and structure of semantic portal user interfaces.

3.1. Search Paradigms and Result Set Visualizations for Semantic Portals

The following search paradigms are essential in LD user interfaces [11, 21] like semantic portals.

Free text search across the whole knowledge graph is the most widely recognized search paradigm. The search can use all textual metadata fields of all entities, or to be narrowed down to, e.g., only use labels, and/or only use entities of specific classes.

Faceted search [10, 23], known also as view-based search [24] and dynamic hierarchies [25], is based on indexing data items along orthogonal category hierarchies, i.e., facets (e.g., places, times, document types, etc.). In searching, the user selects in free order categories on facets, and the data items included in the selected categories are considered the search results. After each selection, a count is computed for each category showing the number of results, if the user next makes that selection, omitting categories with no hits. The idea of faceted search is especially useful on the Semantic Web where hierarchical ontologies used for data annotation provide a natural basis for facets, and reasoning can be used for mapping heterogeneous data to facets. Faceted search can be implemented with server-side solutions, such as Solr[8], Sphinx[9], and ElasticSearch[10], or in a LD setting using a set of configurable SPARQL queries, as in [17, 18].

In *geospatial search* the user can see resources on a map, and browse and explore them. It should be pos-

[8] http://lucene.apache.org/solr
[9] http://sphinxsearch.com/blog/2013/06/21/faceted-search-with-sphinx
[10] https://www.elastic.co

sible to filter the result set by making a selection on a map with e.g. drawing a bounding box.

Temporal search can be performed by constraining the result set based on a timeline component or a timespan selector. A timeline can be used to browse and explore the resources.

The result set that has been constrained using one or more search paradigms can be shown to the user in many different ways. The user can be provided with options of various result set display methods to choose from, that are considered relevant: *Table* is often an intuitive way of displaying the resulting resources as a simple 2-dimensional table with each resource as a separate row and their most important properties shown as table columns. Another useful tabular format is to use a grid to position and show each result inside a rectangle. *Geospatial visualizations* show the results visually on a map as, e.g., markers, polygons or heatmap layers. *Temporal visualizations* display the results on a timeline. *Spatio-temporal visualizations* display the results with interlinked geospatial and temporal components. *Statistical visualizations*, e.g., bar charts, histograms, line graphs, pie charts, and sankey diagrams, are useful if the result set is large, by providing summaries of the results instead of individual resources.

All of the previous display methods need to able to handle information overloading issues [21] related to large result sets with, e.g., pagination, infinite scrolling, or clustering of individual resources into larger groups.

3.2. Structuring the User Interface of a Semantic Portal

The structure of the user interface should support having multiple perspectives for data exploration and searching, each of which are built around entities of a specific class. This way distinct perspectives to the underlying data can be built iteratively and individually.

Moreover, each entity of interest in the knowledge graph should have a landing page, which shows the metadata related to the single entity. Additionally, the landing pages provide both internal and external recommendation links to other related entities, e.g., for the landing page of a person, their family, relatives, and friends could be shown as links to the landing pages of the persons in question. Social network visualizations can be integrated to the landing pages of people, to provide a useful and intuitive way of seeing the person in their social and historical context. The URLs of the landing pages must be constructed in a in a systematic way to enable easy linking within the portal and from external applications.

4. SAMPO-UI Framework

This section presents the architecture, design principles, and the main user interface components of the SAMPO-UI framework. More specific documentation and instructions for software developers can be found from SAMPO-UI's GitHub repository[11]. A case study of applying SAMPO-UI is presented in Section 5.

4.1. Architecture

The SAMPO-UI framework provides software developers with a comprehensive set of reusable and extensible components, well-defined application state management, and a read-only API for SPARQL queries, which can be used for creating a modern and responsive user interface for a semantic portal with minimized coding effort. If the existing functionalities of SAMPO-UI are not enough, the architecture is designed to provide the software developer plenty of freedom to incorporate new functionalities by writing JavaScript code, utilizing the underlying libraries or adding new libraries. Much effort has been put in testing different open source libraries and choosing those with the best prospect for long-term sustainability as a basis for SAMPO-UI.

The framework is meant for creating full stack JavaScript web applications. The main parts are 1) a client based on the widely used and established React[12] and Redux[13] libraries, and 2) a Node.js[14] backend built with Express framework[15]. Angular[16] could have been another choice for a basis, but as it is a complete framework per se, choosing React as a basis for the client enforces a more modular structure for SAMPO-UI, in which individual libraries (e.g. for handling the application state or routing) can be replaced by new ones if they become obsolete. On the other hand, this modular structure demands more careful maintenance of the interplay of individual libraries.

[11] https://github.com/SemanticComputing/sampo-ui
[12] https://reactjs.org
[13] https://redux.js.org
[14] https://nodejs.org/en
[15] https://expressjs.com
[16] https://angular.io

Figure 2 depicts the overall architecture of SAMPO-UI. The general idea is that the focus of the client is on displaying data. The business logic of fetching the data from SPARQL endpoints using various search paradigms is placed on the backend. The client makes use of SPARQL end-point(s) by sending API requests to the Node.js backend. Based on the API requests and predifined configurations, the Node.js backend generates the SPARQL queries, sends them to the SPARQL endpoint(s), and maps and merges the raw results rows in the SPARQL 1.1 Query Results JSON Format[17] into a more developer friendly array of potentially nested JavaScript objects.

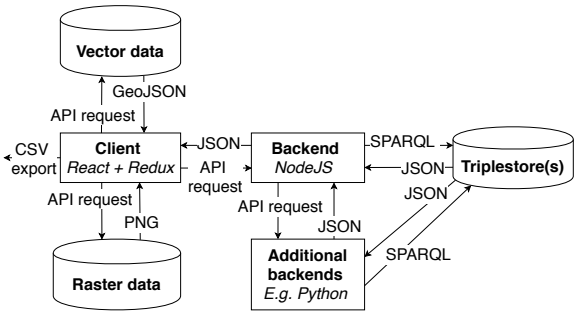

Figure 2. General architecture of SAMPO-UI.

The API endpoints provided by the Node.js backend are described using a document that conforms to the OpenAPI Specification[18]. The same document is used for both documenting the API, and validating the API requests and responses. The documentation is published with the API documentation tool Swagger UI[19]. The API documentation of an example portal[20] built with SAMPO-UI can be used for testing the API.

For handling the API requests related to faceted, full text, and spatial search, the Node.js backend includes a set of generalized templates[21] for SPARQL queries. In order to connect the Node.js backend to a specific SPARQL endpoint, the developer needs to provide configuration[22] for all desired facets and results sets. The placeholders in the SPARQL query templates are replaced with patterns from respective configuration objects by SAMPO-UI's core functions.

While the Node.js backend makes it possible to run any relevant JavaScript libraries on the server, the architecture can be extended by connecting the Node.js backend to additional backend services, as illustrated in Figure 2. This is useful, e.g., if Python modules are needed for dynamic processing of SPARQL query results. An example of such extension is the Sparql2GraphServer API[23] which integrates SPARQL queries and their results with the NetworkX[24] Python package for advanced network analysis tasks that are not currently available for JavaScript.

Figure 2 also shows how API requests to external vector and raster-based services for geographical data can be made directly from the client, as long as the respective API key does not need be hidden. If the API key need to be hidden, the API requests are routed through the Node.js backend. In SAMPO-UI it is also possible to export the results of the SPARQL queries in CSV format, a feature deemed useful by the end-users willing to analyse the data using additional software libraries and tools, such as spreadsheet programs and R. Additionally, the specific SPARQL query that was used for the search results can be given to the end-user if needed.

The modular architecture of the client is depicted in Figure 3, where boundaries between the main client-side libraries are marked with a dashed line. The state of the application (e.g., the user's facet selections and the search results) is maintained using the strict unidirectional data flow enforced by Redux[25]. To reduce the complexity of handling side effects, asynchronous data fetching is carried out uniformly by a Redux middleware named Redux Observable[26]. Redux Observable provides a base for SAMPO-UI's epics[27], which listen for asynchronous Redux actions, send API requests to the Node.js backend and other external services, implement debouncing when necessary, and handle the possible errors in the API responses.

[17] https://www.w3.org/TR/sparql11-results-json
[18] https://swagger.io/specification
[19] https://swagger.io/tools/swagger-ui
[20] https://sampo-ui.demo.seco.cs.aalto.fi/api-docs
[21] Generalized templates for SAMPO-UI's core SPARQL queries: https://github.com/SemanticComputing/sampo-ui/blob/master/src/server/sparql/SparqlQueriesGeneral.js
[22] See instructions for configuring the Node.js backend for SPARQL endpoints in SAMPO-UI's readme: https://github.com/SemanticComputing/sampo-ui#configuration-and-folder-structure
[23] https://github.com/SemanticComputing/Sparql2GraphServer
[24] https://networkx.github.io
[25] https://redux.js.org
[26] https://redux-observable.js.org
[27] In Redux Observable epics are functions which take a stream of actions as input and return a stream of actions.

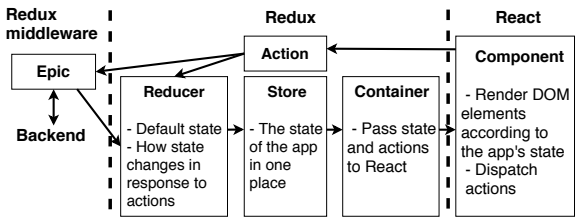

Figure 3. SAMPO-UI client architecture based on the strict unidirectional data flow of Redux integrated with React components.

4.2. Adaptable Server-Side or Client-Side Faceted Search

Nowadays the size of the knowledge graphs that a semantic portal needs to be able to process ranges from thousands to tens of millions triples. Furthermore, the data may be available from a single triplestore or from multiple triplestores. For handling these varying settings we have developed two main approaches for implementing faceted search in SAMPO-UI. We call the first approach "server-side faceted search" (ServerFS), where all queries relating to faceted search are sent to the triplestore. With this approach all limits regarding the size of the knowledge graph and the complexity of the queries are imposed by the triplestore and the server used for hosting it.

The most critical requirement for ServerFS is that all underlying data needs to be made available from a single SPARQL endpoint. ServerFS also causes considerable load on the triplestore when the knowledge graph is large or the portal has a usage spike[28], as a selection in one facet causes the recalculation of both the result set and the result counts of all of the active facets.

In some application settings, e.g. in NameSampo portal [27] that re-uses several existing SPARQL endpoints, it is not possible to aggregate all data into a single triplestore. For these settings we have developed a second approach named "client-side faceted search" (ClientFS). The main idea here is that the whole initial result set for faceted search is first fetched into the client as JavaScript objects, after which all faceted search functionalities are implemented using a set of Redux selectors[29].

[28] If the triplestore is deployed using a Docker container or similar, it is possible to handle the usage spikes with autoscaling [26] carried out by a container-orchestration system.

[29] https://github.com/reduxjs/reselect

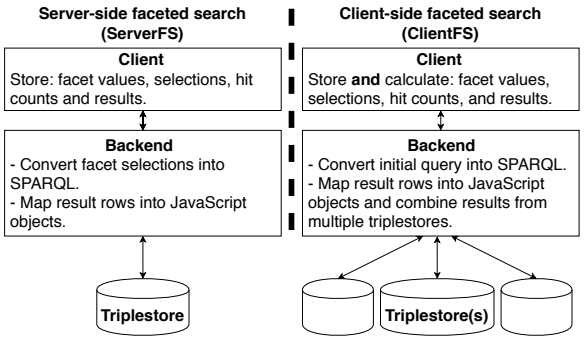

Figure 4. Faceted search architecture options of SAMPO-UI.

Based on our experiences, due to the memory limit set by the browser running the client, the initial result set in ClientFS has to have an upper limit of approximately 30 000 instances on modern mobile devices or desktops with more than 4 GBs of RAM. Even though this limitation rules out many large knowledge graphs, we have found ClientFS effective especially when the initial query is a full text query, which is sent to multiple triplestores simultaneously. Then the results from each triplestore are merged and used as the initial result set for ClientFS. This approach is in use, e.g., in the above-mentioned NameSampo portal.

Figure 4 illustrates how the different tasks related to faceted search are divided between the client and the backend, when using either ServerFS or ClientFS. The example configurations of SAMPO-UI include templates for both ServerFS and ClientFS approaches. It is also possible to use the ServerFS and ClientFS approaches simultaneously in a single semantic portal, provided that they are separated into distinct perspectives. This is demonstrated in the example portal[30], which combines perspectives from NameSampo and MMM portals.

4.3. Main Views of a Semantic Portal

Figure 5 illustrates the three main views of SAMPO-UI for building a complete user interface of a semantic portal.

[30] https://sampo-ui.demo.seco.cs.aalto.fi

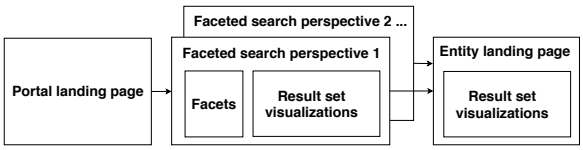

Figure 5. Main views of a semantic portal, using the default structure provided by the Sampo-UI framework. The arrows depict navigation links between the views.

The *portal landing page* welcomes the user with an introduction text and list of links to different faceted search perspectives for studying the underlying knowledge graph(s). To minimize page load time, no SPARQL queries are being executed on this view.

Each *faceted search perspective* typically contains a group of facets configured for a specific entity type in the knowledge graph. For viewing the search results and conducting data analysis tasks several result set visualization components can be added.

The *entity landing page* provides information related to a single entity. The faceted search perspective and the entity landing page share the same result set visualization components.

Component-based user interface development is enforced by many state-of-the-art libraries and frameworks for web application development, such as React[31], Angular[32], and Vue.js[33]. By adopting component-based design principles, SAMPO-UI provides the developer with a selection of approximately 120 ready-to-use user interface components[34], which act as the building blocks for the three main views presented in Figure 5.

Figure 6 demonstrates how a group of SAMPO-UI components can be used for building a faceted search perspective. The following subsections explain these components and also a number of other components for facets and result set visualizations in detail.

4.4. User Interface Components for Facets

The *Hierarchical checkbox facet* component has been developed for filtering with URIs, literal values, or geographical regions. The user can make selections with checkboxes or by drawing a bounding box on a map. There are several additional user-controllable options for faceted search, such as sorting of facet values (e.g., by hit counts), visualization of facet value distributions (e.g., by a pie chart), and selecting multiple disjunctive values in a single facet. For showing hierarchies of arbitrary depth efficiently, only concepts and their parent concepts need to be selected in the SPARQL query of the facet. The SAMPO-UI backend offers functions for converting the raw SPARQL results into a complete hierarchy, where all child concepts of each concept are explicitly available. This complete hierarchy is required for showing the hierarchical facet values and their hit counts to the end-user.

For free text search a *Text facet* component can be used. For filtering with date or integer ranges the developer can choose to use either the *Date facet* with date pickers, the *Range facet* with input fields, or the *Slider facet* component.

Specific user-controllable options for each facet are provided by the *Facet header* component. The currently active facet selections are shown to the user with the *Active filters* component. All facet components of SAMPO-UI are connected to the centralized state of the application, as presented in Figure 3. Having a centralized state offers a uniform way for listening for user input in any of the facet components, and updating the state of the other facets and result set visualizations accordingly.

4.5. User Interface Components for Result Set Visualizations

On the faceted search perspective shown in Figure 5, the *Perspective tabs* component provides the user the ability to switch between different result set visualizations, while persisting the state of the facets. The default component for rendering the results is *Paginated table*, illustrated in Figure 6. It has controls for pagination actions, changing the rows per page, and sorting the result set. The Paginated table component consists of a number of smaller components, which are used to handle for example expanding of rows and showing multiple values in a single table cell. Separate components are provided for displaying plain text, HTML, or images in the Paginated table component, or elsewhere in the application.

For rendering geographical data, SAMPO-UI provides two map components: *Leaflet map* and *Deck.gl map*. The Leaflet map component is based on the

[31] https://reactjs.org/docs/thinking-in-react.html
[32] https://angular.io/guide/architecture-components
[33] https://vuejs.org/v2/guide/components.html
[34] The user interface components of SAMPO-UI are documented at https://semanticcomputing.github.io/sampo-ui, where the documentation texts are generated dynamically from the comments in the source code using the popular open source development tool Storybook.

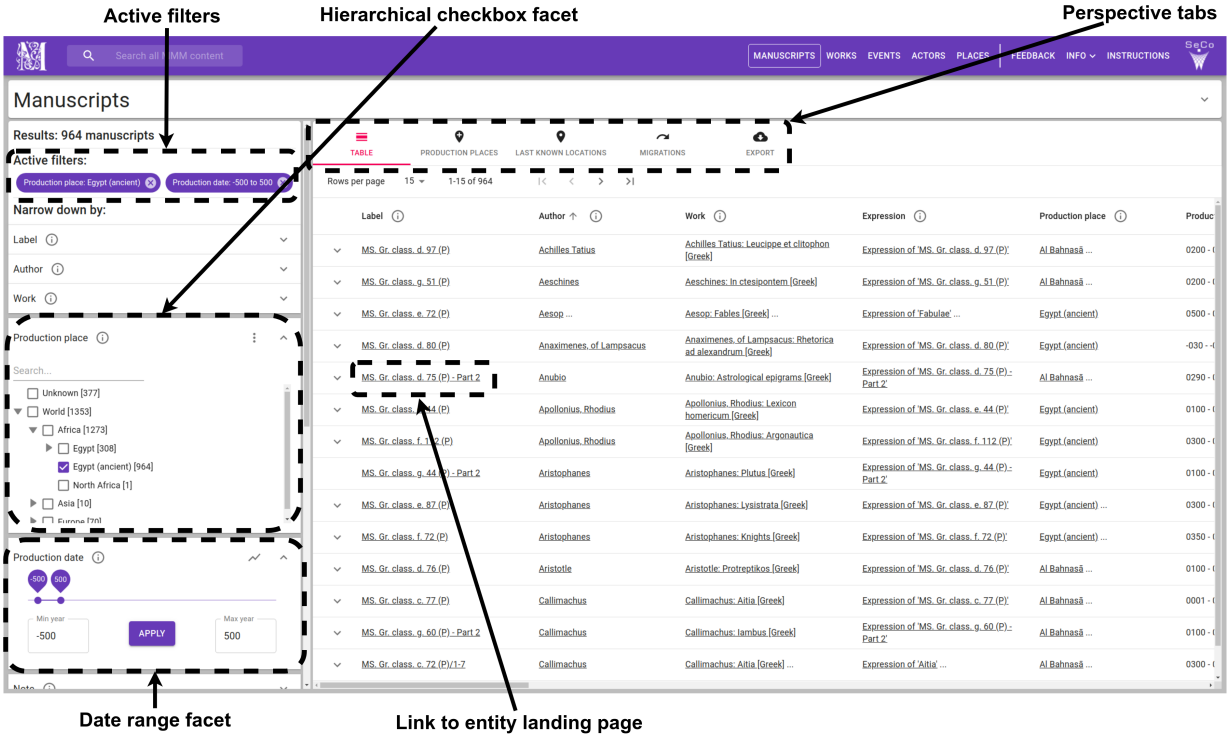

Figure 6. A selection of SAMPO-UI components for building a faceted search perspective of a semantic portal.

Leaflet[35] library and it's highly efficient marker cluster plugin[36] for rendering up to 50 000 map markers at a time. Furthermore, the component includes functionality for showing external map layers from Web Map Tile Service (WMTS) and Web Feature Service (WFS) APIs.

Unlike the Leaflet map component, the Deck.gl[37] map component is based on WebGL technology. WebGL enables a three-dimensional map view as well as new ways for the analysis of large geographical result sets. The result sets can be for example rendered as a heatmap layer for visualizing geographical distribution, or as an arc layer for visualizing the movement of entities.

To visualize spatio-temporal data, the Deck.gl component is used as a basis for the *Temporal map* component. The Temporal map component renders any geographical data with timestamps as an interactive animation.

For showing statistics, SAMPO-UI has ready-to-use *Pie chart*, *Line chart*, and *Bar chart* components based on ApexCharts[38] and Google Charts[39]. Each of these components have their own data processing and configuration functions for reducing the trouble of deploying them in new use cases.

Moreover, as LD provides a good basis for graph structures, SAMPO-UI includes a *Network* component for visualizing result sets as networks. The component is based on the Cytoscape.js[40] graph theory library. For advanced network analysis tasks that are not supported in the JavaScript ecosystem, the component utilizes the Sparql2GraphServer Python API, as explained in Section 4.1.

Because the underlying visualization libraries consume data in different formats, specific result mapper functions are provided in the backend of SAMPO-UI for converting the raw SPARQL results into desired formats for all the different libaries in use.

[35] https://leafletjs.com
[36] https://github.com/Leaflet/Leaflet.markercluster
[37] https://deck.gl
[38] https://apexcharts.com
[39] https://developers.google.com/chart
[40] https://js.cytoscape.org

Thanks to the component-based design and the centralized application state, the developer can easily add new components and result mapper functions for searching and for visualizing results sets as needed. The responsiveness and accessibility of the majority of SAMPO-UI's components is achieved using the Material-UI library[41], which is a React implementation of the established Material Design language[42] developed and backed by Google.

5. Case Study: Implementing the User Interface of a Semantic Portal

In this section experiences of applying the SAMPO-UI framework to create user interfaces are presented and reflected using the development work for the user interface of the WarVictimSampo 1914–1922 portal[43] [28] as a case study.

5.1. Background for Developing the Portal

The WarVictimSampo portal was created to replace an older web application that was used to provide access to the War Victims 1914–1922 database[44] maintained by the National Archives of Finland. The database contains detailed information especially about the victims of the Finnish Civil War in 1918[45], including all known death records from that time due to the wars. The data has been used both by researchers of history and by the general public interested in finding out information about their deceased ancestors.

Simultaneously with the development of the new user interface, the database was updated with new victims and converted into a LD knowledge graph. The new portal needed to work with the knowledge graph and enable exploring, visualising, and studying the results in ways that are useful for both researchers and the general public. The project had relatively little resources and time for creating the user interface of the portal, so it was was preferable to use some existing application or framework as a basis for the user interface. SAMPO-UI was selected because it is based on modern currently maintained technologies, and offers a comprehensive starting base of a complete full stack JavaScript web application.

The portal was created with two faceted search perspectives to the data with various options for search and visualization: the war victims perspective, and a perspective for battles of the Finnish Civil War. More perspectives are planned to be added later. The WarVictimSampo portal was opened to the public in November 2019 and gathered nearly 20 000 unique users in the first month.

5.2. Creating the User Interface

The development started by forking the SAMPO-UI GitHub repository. SAMPO-UI provides a preconfigured environment for full stack JavaScript development. Babel[46] is used for converting the latest features of JavaScript, such as arrow functions and the async/await syntax, into a backwards compatible version of the language for current and older browsers. Webpack[47] handles the automatically restarting development server for the client, and bundling all source code and dependencies into static assets. The Node.js backend is run concurrently with the client, and is automatically restarted using Nodemon[48] when the source code is changed. Uniform coding style is enforced by using the JavaScript Standard Style[49] package.

The War Victims of Finland 1914–1922 database has very detailed information about the victims and a piece of information can relate to a person in more than 150 different ways. In the portal this data can be filtered with multiple different types of facets to best fit each type of information. These facets were created using the facet components of SAMPO-UI presented in Section 4.4. Most of the facets, like birth place and occupation facets, were created with the Hierarchical checkbox facet component. The birth dates and death dates can be filtered with facets created with the Date facet component. Free text search for the victims name was created with the Text facet component, and a facet for number of children was created with the Slider facet component.

[41] https://material-ui.com/
[42] https://material.io/
[43] The portal is published at https://sotasurmat.narc.fi/en. See the homepage https://seco.cs.aalto.fi/projects/sotasurmat-1914-1922/en for more information about the project.
[44] http://vesta.narc.fi/cgi-bin/db2www/sotasurmaetusivu/main?lang=en
[45] The data contains also Finnish victims of the First World War and the Kindred Nations Wars in 1914–1922.
[46] https://babeljs.io
[47] https://webpack.js.org
[48] https://nodemon.io
[49] https://standardjs.com

After filtering out a result set in the portal, the results can be analyzed with various result set visualizations based on the components presented in Section 4.5. The default visualization of the results is a table created with the Paginated table component. There is also map for visualizing the death places and the battle places. The map was created with the Leaflet map component.

For statistical analysis the portal includes pie chart and a line chart result set visualization. The pie chart and line chart visualizations were created based on existing components of SAMPO-UI with some customization to better cater for this specific use case. For example, it was deemed useful to have the line chart visualization automatically calculate the average and median values of age for the war victims while also presenting the distribution as a chart.

The Temporal map component of SAMPO-UI was used for representing the course of events in the Finnish Civil War. The component, presented in Figure 7, is used for the animation result tab of the Battles perspective, where the temporal and spatial metadata of 1182 battles of the Finnish Civil War is rendered as an animation, with playback functionalities at the bottom of the map.

The entity landing pages for war victims are a core functionality of the portal. To best express things specific to this case, such as sources of different pieces of information, a custom component was created by extending the general entity landing page component of SAMPO-UI.

Finally the CSV export functionality of SAMPO-UI was deployed for creating a button that enables the downloading of the current result set as a CSV file where one row corresponds to a one victim. This feature was considered important and was requested by history researchers.

5.3. *Reflections of Using* SAMPO-UI

Faceted search, combined with visualizations for data analysis, was easy to implement with the SAMPO-UI framework. This kind of user interface can be a powerful tool when researching data, and can help finding serendipitous phenomena for further analysis by close reading.

For example, Figure 8 shows two different line charts generated by the line chart view of the WarVictimSampo for comparing the ages of people from one side of the Civil War whose official place of residence was in two different provinces of Finland. In both charts, the party "Red"[50] is selected from the "Party" facet. In the upper chart, Province of Turku and Pori is selected from the "Registered province" facet and in the lower one the Province of Viipuri. There is a big difference in the shape of the charts, and the median age at death for people from the Province of Viipuri is five years greater. Therefore, for some reason, members of the Red party from the Province of Viipuri died a lot older than those from the Province of Turku and Pori. This is apparently previously unknown phenomena in the data, but could be easily found using the combination of faceted search and result set visualizations for data analysis.

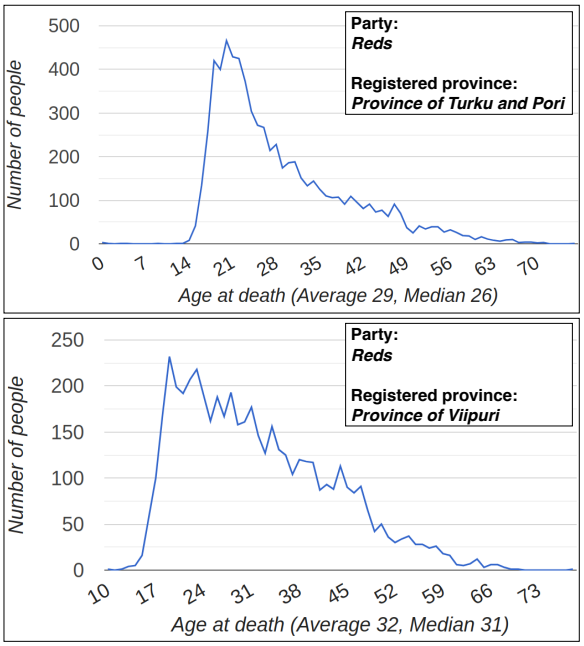

Figure 8. Two line charts generated with the WarVictimSampo portal depicting number of people of certain age at death in the result set. The upper chart depicts people of the Red party from the Province of Turku and Pori and the lower chart depicts people of the Red party from the Province of Viipuri.

Using SAMPO-UI as the basis for a new user interface for a semantic portal saved resources and time, and demanded considerably less programming skill than would have otherwise been necessary. The SAMPO-UI's default structure and responsive layout for a user interface could be adopted with minor configuration.

[50]The Civil War was fought between socialist Reds ("punainen" in Finnish) and conservative Whites ("valkoinen").

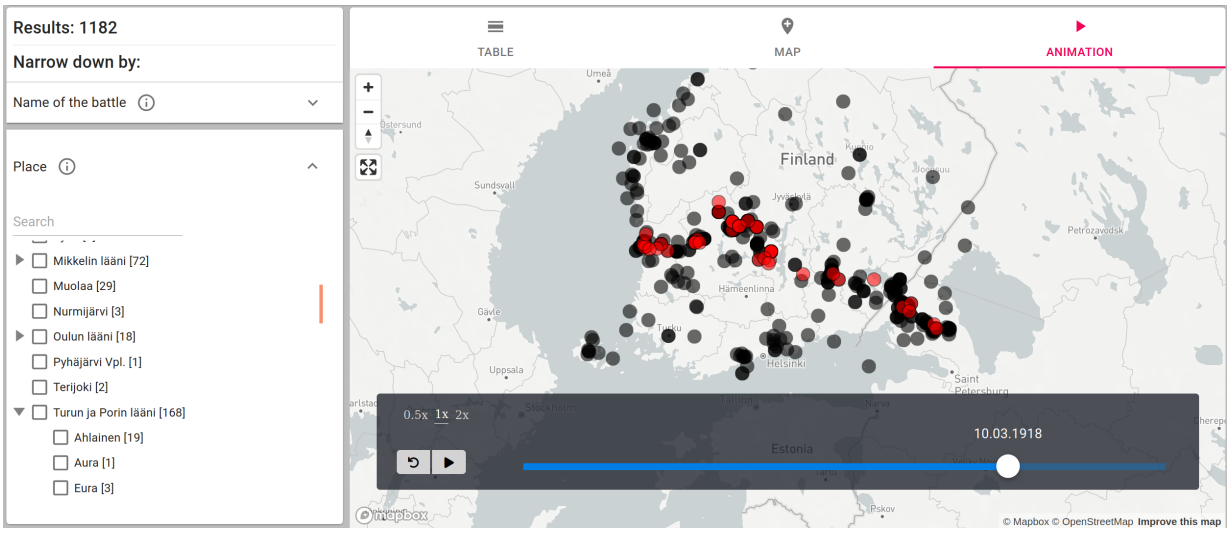

Figure 7. Animation result set visualization of the battles of the Finnish Civil War. As time goes by on the time slider at the bottom, new battles emerge as red spots and turn later into black spots. In the image, the main front line of the Civil War from east to west can be seen. The battles can filtered using the facets on the left.

The ability to easily customize the existing components, and add new ones, was very useful here. This made it possible to better express the characteristics of the specific rich and complicated data. The customization requires some programming skill, but the core functionality provided by the framework includes robust patterns and tools for processing the data and delivering it to the components in a predictable and uniform way. Because the framework is integrated with a plethora of open source JavaScript visualizations libraries, even the creation of new result set visualization components is relatively easy.

6. Discussion

This section describes the contributions, availability, sustainability, and limitations of SAMPO-UI.

6.1. Contributions

This paper presented general features and requirements for user interfaces of semantic portals and the new SAMPO-UI JavaScript framework for developing such user interfaces. SAMPO-UI is our proposed solution to the research question *"How can user interfaces for semantic portals based on Linked Data be built efficiently?"*. Experiences on applying Sampo-UI for creating a new user interface was presented through a case study.

Table 1 presents the already published and forthcoming semantic portals that have been implemented using SAMPO-UI. The portals categorized as *internal* in Table 1 have been developed by the Semantic Computing Research Group (SeCo)[51], while *external* portals, such as the portal for the Norwegian placename registry Norske Stadnamn[52], are examples of external adoption of SAMPO-UI.[53]

The impact of the framework has been demonstrated through the case study presented in Section 5, the portals already using it, and the several new semantic portal user interfaces that are being built with it as shown in Table 1.

The following list of SAMPO-UI's properties reflects the framework against the requirements for tools presented in Section 3:

T1. Ready-to-use basis for a user interface of a semantic portal, which needs only minor modifications for deploying as a modern web application into production,

T2. Well documented API and query templates for using single and multiple SPARQL endpoints,

[51] https://seco.cs.aalto.fi

[52] https://toponymi.spraksamlingane.no

[53] More information about the portals and related publications can be found on the homepage: https://seco.cs.aalto.fi/applications/sampo.

Table 1

Semantic portals using the SAMPO-UI framework. Five of these portals have already been published, and six forthcoming portals are underway. Some information regarding the forthcoming portals is marked as "to be announced" (TBA). The two different faceted search approaches are explained in section 4.2. An extended version of this table including links to the portals and their homepages can be found at https://seco.cs.aalto.fi/applications/sampo.

Portal	**Year**	**Domain**	**Unique users**	**Knowledge graph size in triples**	**Faceted search approach**	**Primary data owner**
			Internal portals			
NameSampo	2019	Place names	35 000	241 068 456	ClientFS	Institute for the Languages of Finland, National Land Survey of Finland, and the J. Paul Getty Trust
WarVictimSampo 1914–1922	2019	Military history	21 000	9 815 265	ServerFS	National Archives of Finland
Mapping Manuscript Migrations	2020	Pre-modern manuscripts	2200	22 472 633	ServerFS	Schoenberg Institute for Manuscript Studies, Bodleian Libraries, and Institut de recherche et d'histoire des textes
LetterSampo	TBA	Epistolary data	TBA	TBA	ServerFS	The Circulation of Knowledge project
LawSampo	TBA	Law	TBA	TBA	ServerFS	Ministry of Justice, Finland
AcademySampo	TBA	Finnish Academic People 1640–1899	TBA	TBA	ServerFS	University of Helsinki, Finland
HistorySampo	TBA	Historical events	TBA	TBA	ServerFS	Agricola Network of Historians
FindSampo	TBA	Archaeology and citizen science	TBA	TBA	ServerFS	Finnish Heritage Agency
ParliamentSampo	TBA	Parliamentary data	TBA	TBA	ServerFS	Parliament of Finland
			External portals			
Norske stadnamn	2019	Place names	Unknown	217 353 538	ClientFS	Norwegian Mapping Authority and the J. Paul Getty Trust
Staff portal	2019	Human resources	Unknown	Unknown	ClientFS	Lingsoft Ltd.

including password protected SPARQL endpoints,

T3. Hierarchical checkbox facet component for querying and visualizing ontological hierarchies of arbitrary depth efficiently,

T4. Two scalable faceted search approaches, ServerFS and ClientFS, for different use scenarios. Result set visualization components use pagination and clustering,

T5. Facilitate creating user-friendly, intuitive, and accessible user interfaces by enforcing the Material Design guidelines as a basis,

T6. User interface components and visualization libraries support responsive web design,

T7. Sustainability fostered by providing open source code, extensive documentation, and support for software development through preconfigured development environments.

The semantic portals created with SAMPO-UI contain features which are here reflected against the semantic portal requirements presented in Section 3:

SP1. The portals provide multiple perspectives for accessing the knowledge graph(s),

SP2. Each perspective includes a user interface based on faceted search with different result set visualizations to support data analysis,

SP3. Faceted, text, temporal, and geographical search paradigms are supported,

SP4. Faceted search is efficient for visualizing instances that share one or more specific properties,

SP5. Customised entity landing pages provide various ways to study individual entities using several visualizations,

SP6. Each perspective is customized for a specific entity type,

SP7. The perspectives are interlinked with each other and they contain links to the entity landing pages, which are in turn interlinked with each other,

SP8. The data can be visualized using interactive maps, statistics, networks, and animations,

SP9. Ready-to-use functionalities for CSV export.

Considering the requirements for tools for creating user interfaces for semantic portals, as well as the requirements for the portals themselves, SAMPO-UI fullfills each requirement. As many of the requirements are somewhat subjective, limitations regarding the framework are discussed in Section 6.3.

6.2. Comparison with Existing Linked Data User Interface Libraries and Frameworks

The design philosophy behind SAMPO-UI differs from existing frameworks for LD user interfaces presented in Section 2. Instead of focusing on providing the software developer with configuration files and options as in, e.g., LD-R and SPARQL Faceter, in SAMPO-UI the center of attention is keeping the additional layer between LD specifics and underlying JavaScript libraries as thin as possible. This provides the developer with full control of writing and extending the actual client-side or server-side JavaScript code when necessary.

These choices are based on the experiences of developing SAMPO-UI in conjunction with multiple LD based projects in different domains, where it has become evident that implementing user interfaces for semantic portals requires a remarkably wide range of flexibility to be able to adapt to the particular data models, schemas, search indices, and external APIs in use. Furthermore, as the data models get more complex and domain specific, it becomes virtually impossible to create universally applicable user interface solutions that could be taken into use by employing solely configuration files.

SPARQL Faceter's ideas of using only SPARQL for data retrieval is also adopted in SAMPO-UI, and the other requirements found in the study have guided our work. SPARQL Faceter is a library, which means that there has to be an existing application where it can be imported for creating a fully functional semantic portal. SAMPO-UI takes the setting further by providing the developer a comprehensive ready-to-use basis for a complete JavaScript application with a client and a backend. This basis needs only minor modifications for deploying it in production as a polished user interface for a semantic portal.

Instead of using only client-side code, introducing a Node.js backend to the architecture adds some overall complexity, but the benefits are plentiful. A significant part of the logic related to various search paradigms and the processing of search results can be carried out in the backend using pure JavaScript, instead of having to integrate the functionality inside client-side frameworks or libraries, which are known to become deprecated considerably sooner than pure JavaScript code. Additionally, using the backend for SPARQL queries makes it possible to run many computationally expensive operations related to data processing on the server, instead of relying on the varying computational resources of the client. Also querying password protected APIs or SPARQL endpoints using only client-side code is impossible without exposing the API keys or passwords to the users.

A central task in developing user interfaces for LD is to be able to convert raw SPARQL result rows into arrays of JavaScript objects. With simple SPARQL queries this is trivial, provided that one result row corresponds to one entity of interest. However, in real world settings it is typical that a high number of SPARQL result rows need to be merged into arrays of deeply nested JavaScript objects. This kind of functionality was not supported in any of the existing libraries we have surveyed, so a comprehensive set of result handling functions were developed in SAMPO-UI.

6.3. Limitations and Future Work

When designing user interfaces for semantic portals, one has to select which search paradigms are supported. A key challenge here is that the search paradigm selection affects the data creation phase and publishing phase, so that ontologies and metadata annotations support the planned user interfaces, and that the needed APIs and search indices are deployed and properly configured. Hence the functionalities provided by the SAMPO-UI components are dependent on the nature of the knowledge graph at hand, and the performance and configuration of the triplestore used for deploying it.

Since a user interface built with the SAMPO-UI framework is a full stack application with a client and a backend, SAMPO-UI cannot be included as an independent library, such as Leaflet, into an existing JavaScript application. Instead, the framework is meant to be used as a basis for a complete user interface of a semantic portal, including a predefined development environment, centralized state management, routing, and other relevant features. This in turn enables building a complete web application from scratch by forking the SAMPO-UI repository, followed by minor adaptations for the knowledge graph(s) in question.

The backend of Sampo-UI could also be used by another client via the well defined API. In order to further improve the reusability of SAMPO-UI, in the future the client and the backend could be refactored and separated into respective packages, and published in the Node package manager Registry[54].

As the API provided by SAMPO-UI's backend is read-only, functionalities related to editing and maintaining knowledge graphs are intentionally left out from the scope of the framework to avoid the complexity and security issues related to user management. Furthermore, supporting editing of LD while maintaining the integrity of knowledge graphs with wildly differing and complex data models, such as the CIDOC CRM[55] and its derivatives, is very challenging, as many of the data models for knowledge graphs are designed solely for presenting the data, not for editing.

SAMPO-UI is built on a group of external dependencies, which are listed in the standard *package.json* file. For keeping the framework up to date, these dependencies need to be updated periodically, while maintaining their mutual compatibility.

Several new features are being planned for SAMPO-UI, based on the requirements of different projects using it. One requested extension for faceted search is "faceted search within a facet", where, for example, instead of filtering the result set of books by authors, one could filter by author's birth date. This would require refactoring the core application state and components of SAMPO-UI so that each facet could have it's own filters for all relevant properties of the facet values. Another development trend is creating new visualizations of result sets. In contrast to modifying the facet components, they are easier to implement individually without touching the core parts of the application's state. For example, visualizing imprecise temporal information on zoomable timelines, combined with proper clustering algorithms for large result sets would certainly bring out new insights from already existing knowledge graphs.

6.4. Availability and Sustainability

The SAMPO-UI framework is being developed in a GitHub repository[56], which contains example structure, configurations, and documentation for building a complete user interface for a semantic portal from scratch. User interfaces built with SAMPO-UI are being developed in separate repositories, which are created by forking the SAMPO-UI repository. When new functionalities are added to the SAMPO-UI core repository, they can be merged into other respective repositories when needed, to facilitate code re-use. Dependency management and backward compatibility of SAMPO-UI is ensured by using Semantic Versioning[57]. The first stable version of the framework was released on April 24, 2020.

Sustainability of SAMPO-UI is supported by releasing the framework with the open MIT License, and indeed the software has already been used by external users with some contributions. Several new portals are being developed using SAMPO-UI, which means that the framework will be supported and developed further by its developer, the Semantic Computing Research Group (SeCo) at the Aalto University and University of Helsinki (HELDIG).

Acknowledgments

The SAMPO-UI framework have been developed gradually during several projects in 2018–2020. Thanks to the Academy of Finland and University of Helsinki Future Fund for funding. The authors wish to acknowledge CSC – IT Center for Science, Finland, for computational resources.

References

[1] E. Hyvönen, *Publishing and Using Cultural Heritage Linked Data on the Semantic Web*, Synthesis Lectures on the Semantic Web: Theory and Technology, Vol. 2, Morgan & Claypool Publishers, San Rafael, CA, USA, 2012, pp. 1–159. doi:10.2200/S00452ED1V01Y201210WBE003.

[2] O. Suominen, Methods for Building Semantic Portals, PhD thesis, Aalto University, Finland, 2013. http://urn.fi/URN:ISBN:978-952-60-5254-0?locale-attribute=en.

[3] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer and Y. Sure, Semantic community Web portals, *Computer Networks* **33** (2000), 473–491. doi:10.1016/S1389-1286(00)00039-6.

[4] N. Stojanovic, A. Maedche, S. Staab, R. Studer and Y. Sure, SEAL: a framework for developing SEmantic PortALs, in: *Proceedings of the 1st international conference on Knowledge capture*, K-CAP '01, Association for Computing Machinery, New York, NY, USA, 2001, pp. 155–162. doi:10.1145/500737.500762.

[54] https://docs.npmjs.com/using-npm/registry.html
[55] http://www.cidoc-crm.org
[56] https://github.com/SemanticComputing/sampo-ui
[57] https://semver.org

[5] E. Hyvönen, Semantic Portals for Cultural Heritage, in: *Handbook on Ontologies*, 2nd edn, S. Staab and R. Studer, eds, Springer, Berlin, Heidelberg, 2009, pp. 757–778. doi:10.1007/978-3-540-92673-3_34.

[6] H. Lausen, Y. Ding, M. Stollberg, D. Fensel, R. Lara Hernández and S.-K. Han, Semantic web portals: state-of-the-art survey, *Journal of Knowledge Management* **9**(5) (2005), 40–49. doi:10.1108/13673270510622447.

[7] S. Idreos, O. Papaemmanouil and S. Chaudhuri, Overview of Data Exploration Techniques, in: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 277–281. doi:10.1145/2723372.2731084.

[8] E. Hyvönen, Using the Semantic Web in Digital Humanities: Shift from Data Publishing to Data-analysis and Serendipitous Knowledge Discovery, *Semantic Web – Interoperability, Usability, Applicability* **11**(1) (2020), 187–193. doi:10.3233/SW-190386.

[9] E. Hyönen, "Sampo" Model and Semantic Portals for Digital Humanities on the Semantic Web, in: *DHN 2020 Digital Humanities in the Nordic Countries. Proceedings of the Digital Humanities in the Nordic Countries 5th Conference*, CEUR Workshop Proceedings, vol. 2612, 2020, pp. 373–378.

[10] D. Tunkelang, *Faceted search*, Synthesis Lectures on Information Concepts, Retrieval, and Services, Vol. 1, Morgan & Claypool Publishers, San Rafael, CA, USA, 2009, pp. 1–80. doi:10.2200/S00190ED1V01Y200904ICR005.

[11] P. Haase, D.M. Herzig, A. Kozlov, A. Nikolov and J. Trame, metaphactory: A Platform for Knowledge Graph Management, *Semantic Web – Interoperability, Usability, Applicability* **10**(6) (2019), 1109–1125. doi:10.3233/SW-190360.

[12] A.R. Hevner, S.T. March, J. Park and S. Ram, Design Science in Information Systems Research, *MIS Quarterly* **28**(1) (2004), 75–105. doi:10.2307/25148625.

[13] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, San Rafael, CA, USA, 2011, pp. 1–136. doi:10.2200/S00334ED1V01Y201102WBE001.

[14] E. Hyvönen, E. Ikkala, J. Tuominen, M. Koho, T. Burrows, L. Ransom and H. Wijsman, A Linked Open Data Service and Portal for Pre-modern Manuscript Research, in: *Proceedings of the Digital Humanities in the Nordic Countries 4th Conference (DHN 2019)*, CEUR Workshop Proceedings, Vol-2364, 2019, pp. 220–229.

[15] C. Bizer, T. Heath and T. Berners-Lee, Linked Data - The Story So Far, *International Journal on Semantic Web and Information Systems* **5**(3) (2009), 1–22. doi:10.4018/jswis.2009081901.

[16] A. Khalili, A. Loizou and F. van Harmelen, Adaptive Linked Data-driven Web Components: Building Flexible and Reusable Semantic Web Interfaces, in: *The Semantic Web. Latest Advances and New Domains. ESWC 2016.*, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S.P. Ponzetto and C. Lange, eds, Lecture Notes in Computer Science, Vol. 9678, Springer, Cham, 2016, pp. 677–692. doi:10.1007/978-3-319-34129-3_41.

[17] A. Khalili, P. Van den Besselaar and K.A. de Graaf, FERASAT: A Serendipity-Fostering Faceted Browser for Linked Data, in: *The Semantic Web. ESWC 2018*, A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai and M. Alam, eds, Lecture Notes in Computer Science, Vol. 10843, Springer, Cham, 2018, pp. 351–366. doi:10.1007/978-3-319-93417-4_23.

[18] M. Koho, E. Heino and E. Hyvönen, SPARQL Faceter – Client-side Faceted Search Based on SPARQL, in: *Joint Proceedings of the 4th International Workshop on Linked Media and the 3rd Developers Hackshop*, CEUR Workshop Proceedings, Vol 1615, 2016.

[19] N. Bikakis and T. Sellis, Exploration and Visualization in the Web of Big Linked Data: A Survey of the State of the Art, in: *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference*, CEUR Workshop Proceedings, Vol 1558, 2016.

[20] J. Klímek, P. Škoda and M. Nečaskỳ, Survey of tools for Linked Data consumption, *Semantic Web – Interoperability, Usability, Applicability* **10**(4) (2019), 665–720. doi:10.3233/SW-180316.

[21] L. Po, N. Bikakis, F. Desimoni and G. Papastefanatos, *Linked Data Visualization: Techniques, Tools and Big Data*, Synthesis Lectures on Data, Semantics and Knowledge, Vol. 10, Morgan & Claypool Publishers, San Rafael, CA, USA, 2020, pp. 1–157. doi:10.2200/S00967ED1V01Y201911WBE019.

[22] M.A. Hearst, *Search User Interfaces*, Cambridge University Press, Cambridge, UK, 2009. doi:10.1017/CBO9781139644082.

[23] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen and K.-P. Lee, Finding the flow in web site search, *Communications of the ACM* **45**(9) (2002), 42–49. doi:10.1145/567498.567525.

[24] A.S. Pollitt, The key role of classification and indexing in view-based searching, Technical Report, University of Huddersfield, UK, 1998.

[25] G.M. Sacco, Guided Interactive Diagnostic Assistance, in: *Encyclopedia of Healthcare Information Systems*, N. Wickramasinghe and E. Geisler, eds, IGI Global, Hershey, Pennsylvania, USA, 2008, pp. 631–635.

[26] E. Casalicchio and V. Perciballi, Auto-scaling of Containers: the Impact of Relative and Absolute Metrics, in: *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, Institute of Electrical and Electronics Engineers, New York, New York, USA, 2017, pp. 207–214. doi:10.1109/FAS-W.2017.149.

[27] E. Ikkala, J. Tuominen, J. Raunamaa, T. Aalto, T. Ainiala, H. Uusitalo and E. Hyvönen, NameSampo: A Linked Open Data Infrastructure and Workbench for Toponomastic Research, in: *Proceedings of the 2nd ACM SIGSPATIAL Workshop on Geospatial Humanities*, Association for Computing Machinery, New York, NY, USA, 2018, pp. 2:1–2:9. doi:10.1145/3282933.3282936.

[28] H. Rantala, E. Ikkala, I. Jokipii, M. Koho, J. Tuominen and E. Hyvönen, WarVictimSampo 1914–1922: A Semantic Portal and Linked Data Service for Digital Humanities Research on War History, in: *The Semantic Web: ESWC 2020 Satellite Events*, A. Harth, V. Presutti, R. Troncy, M. Acosta, A. Polleres, J.D. Fernández, J. Xavier Parreira, O. Hartig, K. Hose and M. Cochez, eds, Lecture Notes in Computer Science, Vol. 12124, Springer, Cham, 2020, pp. 191–196. doi:10.1007/978-3-030-62327-2_33.