Model-based, client-side integration of heterogeneous data from REST services

Milan Zdravković*, Nikola Vitković*, Miroslav Trajanović*, Milan Trifunović*, Nikola Korunović*
* Faculty of Mechanical Engineering in Niš, University of Niš, Niš, Serbia
{milan.zdravkovic, nikola.vitkovic, miroslav.trajanovic, milan.trifunovic, nikola.korunovic}@masfak.ni.ac.rs

***Abstract*— With the increasing implementation of Internet-of-Things paradigm, the increase of data volume and diversity threatens to diminish the potentially valuable effects of their reuse in multiple circumstances. While increasing number of datasets, coming from devices or elsewhere is being exposed by using APIs, very few of them is based on the standard dictionaries, data structures and models. For that reason, it becomes very difficult to integrate data of similar or identical semantics, coming from different sources. In this paper, we propose the meta-model for client-side application which aggregates semantically identical or similar data from multiple REST APIs in a single Angular-based widget. Main concepts of the proposed meta-model are data objects, services with defined object and property mappings, use cases and user interface components. Model is implemented in the data integration platform and it is validated in the case of searching data about research projects from two sources. The search widget is implemented on the EURAXESS portals.**

I. INTRODUCTION

The world of computing has dramatically changed since the introduction of the new devices that continuously generate data, such as smart phones. According to IBM, 2.5 quintillion bytes of data are created every day [1]. However, only small portion of this data can be effectively used in multiple contexts. In fact, our digital environments are typically full of unwanted and unused data, which are not usable for the purposes other than the initial one, because of the use of proprietary, "closed", or simply unknown and unrecognizable (by machines) formats. However, value of this data is potentially significant, when considering their use in many different contexts. For example, the personal location data from smart phones is typically used to calculate health related information, such as distance walked or run, calories burned, etc; on the other side, this data can be also used to produce higher value data, for example, to calculate risks for viral infections spread in different regions [2].

On the positive side, there exists a trend of increasing openness of data, mostly through representational state transfer (REST) web interfaces. Although this trend does not contribute to unification of data structures, it does help to have data at least easily accessible.

When referring to the term of "structured data", we assume that such data can be used by machines. Thus, we assume that this data is based on standard dictionaries or agreements which are explicitly and formally described (with underlying ontology).

The main research question we want to address in the work behind this paper is: how to integrate data from multiple REST endpoints to build a complex data structure, based on the formal models. The problem above is defined in the specific circumstances. Namely, according to the project requirements, the integration needs to take place on the client side (in browser), based on the previously defined mappings. The motivation for such an approach is rapid and easy implementation of the corresponding solution on any web page.

The specific requirements make this research problem unique. To the best of our knowledge, there exists no framework or package for implementing its solution.

The solution is assumingly based on the application model, which is built in the implementation phase by the developer/knowledge engineer. Main goal of the application model is to establish formal relationships between the concepts of main data model and schema elements of the existing data sources and thus, to build the integrated data model. This data model needs to be managed from a single web widget. That widget is a generic client, which can be easily embedded in any web page. Client accesses the integration platform to get the application model and then makes the requests to the defined web services, performs integration and makes the instantiated model available for any kind of user manipulation.

In the remainder of this paper, first we present the data aggregator client as a solution to the above problem. Then, an application model is presented as the main scientific contribution of the paper. Finally, a short case study of using the client for searching multiple REST sources of project funding data is presented.

II. DATA AGGREGATOR CLIENT

Data aggregator client is a lightweight HTML client which interprets the application model to: 1) generate HTML UI elements for interacting (searching, adding) with data coming from more than one, different API sources; 2) enable integration of data coming from different sources, based on the mappings defined in app model; 3) display integrated data in a way, described by application model.

Data aggregator client can be embedded in any HTML page, by simply including its JavaScript source.

Data aggregator client is a part of the data aggregator platform, which also includes server-side architecture for instantiating an app model. This instantiation involves (see Figure 1, below):

1. definition of relevant APIs and their endpoints (services);
2. definition of a formal data model (ontology)

3. definition of UI model, based on application model schema
4. semi-automatic mapping, based on the pre-defined formal data model (ontology);

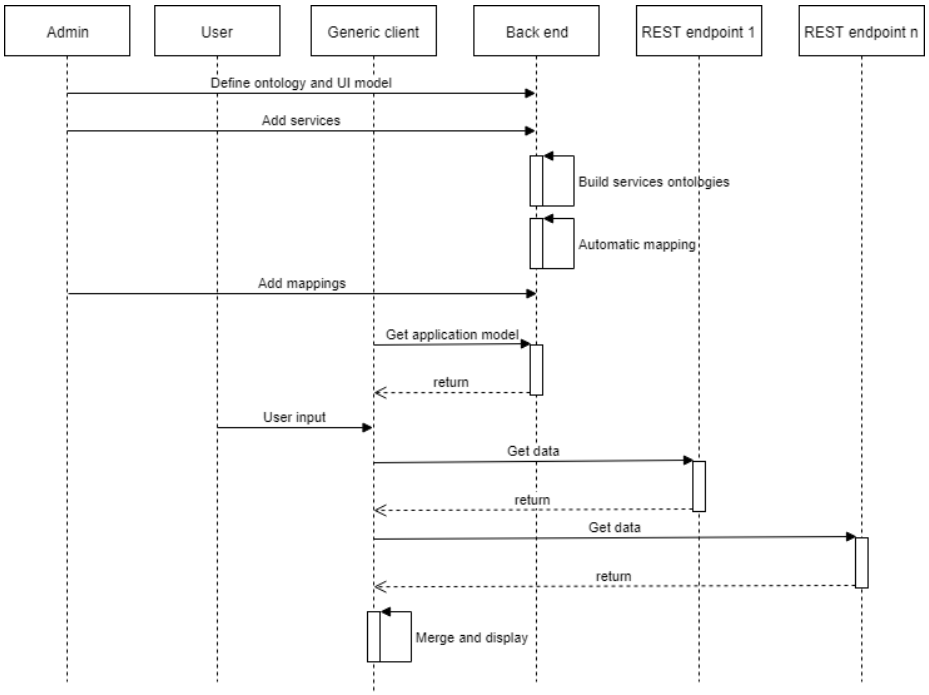

Figure 1. Workflow for instantiating client app model and running app

Admin user is using data aggregator back end to construct the application model. First, ontology representing main data model is defined (URI). The concepts are mapped to the UI model schema to instantiate the layout of the application. Data sources are defined, by indicating REST services endpoints and their arguments. Based on the test arguments, the back end reads data and builds corresponding RDF/RDFS models which are then semi-automatically mapped to the concepts of the main data model ontology.

The generic client is embedded in the web page which is accessed by the visitor. Upon access, the client makes the authenticated request to backend API, which returns the application model. The model is delivered in the XML format and it is a structured representation of the main data schema, application (client UI) layout, as well as the descriptions of the REST services which deliver actual data. Finally, the model includes defined mappings between the elements of the expected REST services responses and main data model objects.

Based on the application model, the generic client builds the user interface, which is then used by the web page visitor to enter data (for example, search keywords). Upon the user action, generic client makes the requests to the REST services in the model and collects responses which are then merged and integrated by using the main data model. Finally the data is shown, according to the UI model.

Overall architecture of the platform is illustrated on a Figure 2, below.

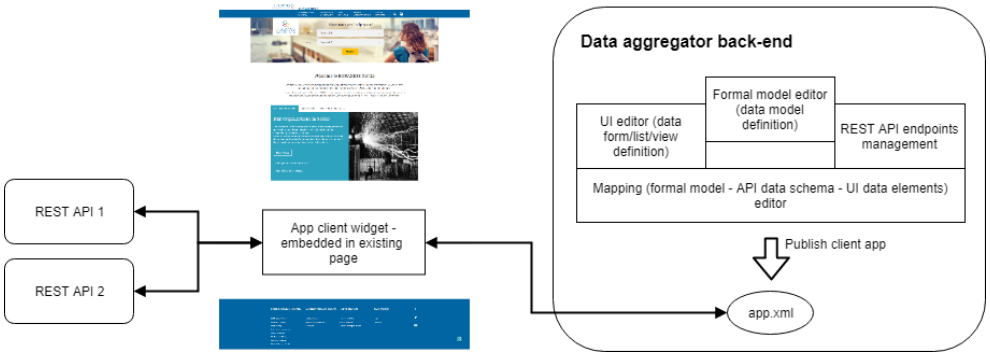

Figure 3. Workflow for instantiating client app model and running app

At this moment, only app client widget development is completed. It is functional and it is being used at the existing website, as described in the case study below. For the purpose of the case study, app.xml instantiation of the application model schema is manually coded. The development of data aggregator back-end is in progress.

A. Application model schema

Main classes of the application model schema are Dataobject, Service and Objectmapping.

Dataobject represents a data entity that can be described by basic (Dataproperty) or complex (Objectproperty)

properties. In the backend, Dataobject is represented by the individual concept of the existing ontology, while its properties are represented by the concept's object and data properties, as defined in RDF/RDFS/OWL file with a given URI.

Service object is a representation of the REST API endpoints. Each service can be invoked with a modelled set of parameters (Parameter).

Mapping between the output of the REST API call and existing data object is represented by Objectmapping element, which establishes relation between a given data object and element of the JSON or XML structure, returned by the REST API call. That element is defined by XPath expression in Objectmapping element.

1) User interface model schema

Main element of the user interface model schema is user interface, represented by Ui object. One client application has one Ui object and it is only a placeholder for its parts. Main constitutive parts of the client application or Ui object are data UI components, represented by Datauicomponent objects. Those objects are implemented at the user interface as individual tabs.

Each of the components uses one or more services and this relationship is established by Usecase object. One client application can have multiple use cases and each Usecase object defines services which are used in it.

Tabs show one or more data panels of different types, namely a form (represented by Dataform and child Dataformelement schema elements), data list view (represented by Datalist and child Datalistitem elements) and individual data view (represented by Dataview and its child Dataviewsection and Dataviewitem elements). Each of the items in the data panels (namely, Dataformelement, Datalistitem and Dataviewitem) can be bound to the existing data properties, as defined in the main data model.

Figure 3 below shows detailed view to application model schema.

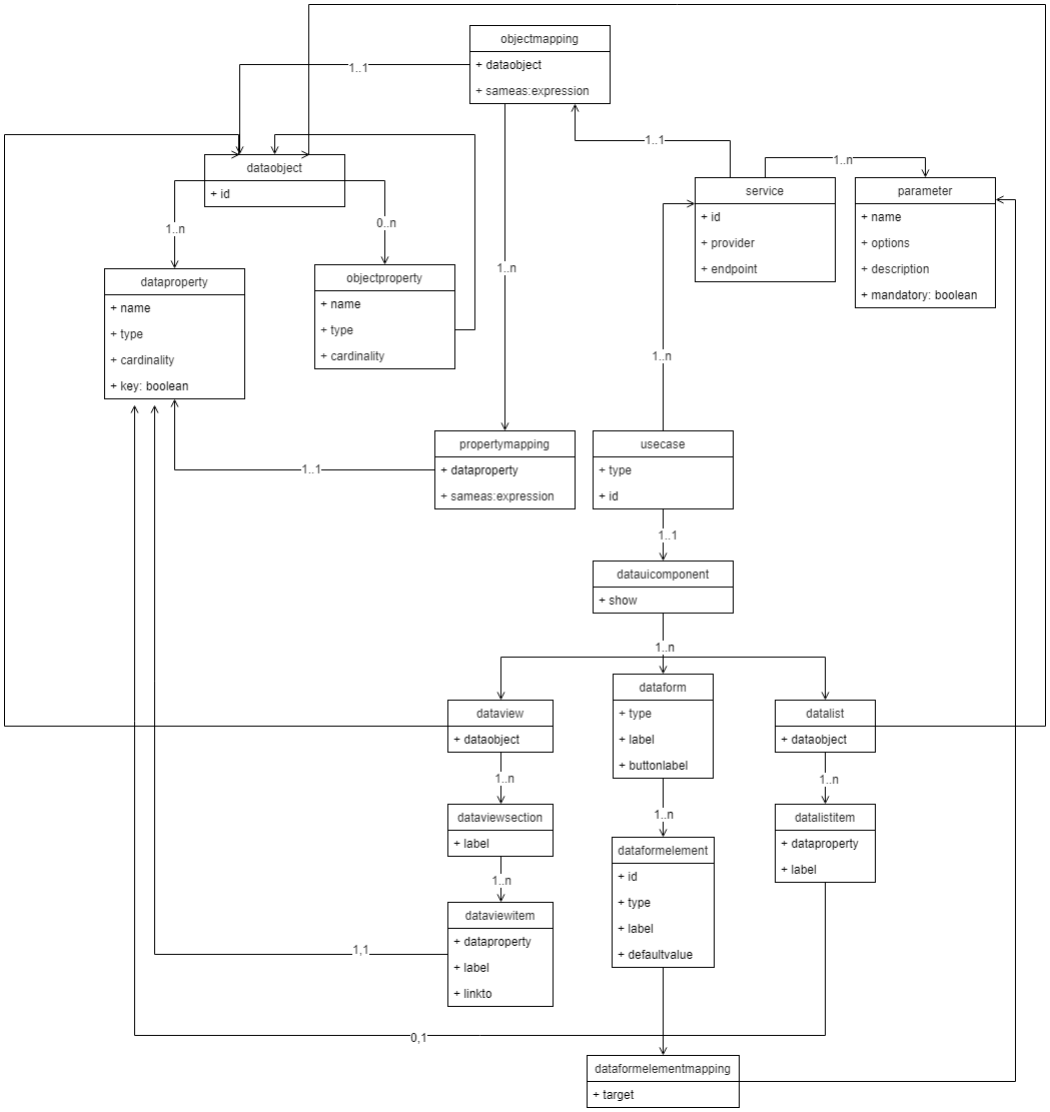

Figure 4. Workflow for instantiating client app model and running app

III. CASE STUDY IMPLEMENTATION

EURAXESS - Researchers in Motion is a unique pan-European initiative delivering information and support services to professional researchers. Backed by the European Union and its Member States, it supports researcher mobility and career development, while

enhancing scientific collaboration between Europe and the world.

EURAXESS portal is a website which publishes information and tools for mobile (incoming and outgoing) researchers or resident researchers who are interested in information and opportunities for career development. The core online service provided by the portal is database of jobs and grants for researchers and scientists, published by thousands of R&D organizations in Europe.

The EURAXESS portal is Drupal based platform, hosted by Amazon web services. It has national entry points, named EURAXESS national portals. Each EURAXESS national portal publishes content which is managed by the national EURAXESS offices. This content is static HTML. The problem is that often, national portals have a need to publish structured information from a database or from online service hosted elsewhere. Given very strict security and performance requirements by the web platforms maintained by European Commission, it is not possible to implement a back-end tool which would facilitate such integration. This was exactly the motivation for developing data aggregator platform, which can be hosted anywhere but integrate, merge and deliver relevant data to application client widget, easily embedded in existing web page.

A. Widget for searching data on EU-funded projects

The specific case of implementation involved integrated search of data about EU-funded projects, coming from two different sources:

- Datasets with information about FP5, FP6, FP7 and Horizon 2020 projects, funded by the European Commission. Datasets are imported in a database which data was then exposed by the developed API (endpoint: http://euraxess.eventiotic.com/euprojects/api/searchProjects)
- Data about research projects, published by OpenAIRE API (endpoint: http://api.openaire.eu/search/projects)

From the user point of view, the widget is used to search existing data on EU-funded projects, by keyword. Two consecutive REST API requests are made with a given keyword to the service endpoints above. Data is collected and merged, by using the approach described above.

IV. CONCLUSION

The paper presents the unique solution to integration of data from REST sources, implemented at the client side. Client-side implementation makes it easy to use on web, even in cases where administration access is not available. The solution is illustrated by a case study of integrating data from two different sources of project funding information.

Future research and development involves development of the backend platform framework and its components for semantic matching.

ACKNOWLEDGMENT

Part of the research behind this paper was funded by the European Commission, under project H2020-SEAC-2014-1.665934, Making European research careers more attractive by developing new services and enhancing the current services of the EURAXESS network – EURAXESS TOPIII.

Part of the research was funded by the Ministry of education, science and technological development, under project no III41017.

REFERENCES

[1] Jacobson, R. (2013) 2.5 quintillion bytes of data created every day. How does CPG & Retail manage it? IBM Consumer Products Industry Blog. https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/

[2] Chunara, R., Smolinski, M.S., Brownstein, J.S. (2013) Why We Need Crowdsourced Data in Infectious Disease Surveillance. Current Infectious Disease Reports. 15(4) 316-319