A Distributed POI Data Model based on the Entity-Component Approach

Arto Heikkinen, Ari Okkonen, Antti Karhu, Timo Koskela
Center for Internet Excellence
University of Oulu
Oulu, Finland
firstname.lastname@cie.fi

Abstract— **Point of interest (POI) information is commonly utilized in different location-based services, which are becoming more popular due to widespread adoption of GPS-enabled smartphones. This paper presents a distributed and modular data model for representing POIs. In addition, RESTful service architecture for accessing the POI data is described. The design of the POI data model is based on the principles of the entity-component model, which allows partitioning the POI data into separate components that are linked to a POI entity. The performance of the POI data model was evaluated in terms of data volume, client-side data parsing time and client-side memory consumption using a mobile web application. The POI data model was also compared with an existing POI implementation. Based on the measurements, the ability to select only relevant data components was found very important feature for achieving good client-side performance.**

Keywords— point of interest; location-based service; gis; service architecture; mobile web; xml3d; sensor data

I. INTRODUCTION

The aim of Location-based Services (LBS) is to deliver relevant and timely services based on the location of a user [1]. The potential user base for LBS is growing rapidly, since the majority of new mobile devices are today embedded with several sensors used for determining the location and the orientation of a user. The development of mobile devices have also enabled introduction of Mobile Augmented Reality (MAR) applications that can be treated as a special case of LBS [1]. In MAR applications, the data from embedded sensors is used collaboratively for augmenting the view of the physical real world with location-based information [2].

In the near future, 3D content is becoming more widely available due to the emergence of new technologies for 3D content creation [3][4] and new web standards such as WebGL that enable viewing 3D content in a cross-platform supported web browser [5]. At the same time, many cities are being instrumented with various sensors taking the first steps towards the vision of a smart city. In a smart city, technology, people and institutions are intelligently combined together to create new capacity for problem solving and innovation [6]. From the standpoint of LBS, both 3D content and sensor data provide new possibilities for application development.

In LBS, each geographic entity is typically described using a point of interest (POI) [7]. A POI is presented as data model containing information on a single geographic entity that can be permanent such as a building, semi-permanent such as a restaurant or temporal/periodic such as a festival. In order to avoid having a separate set of POIs for every single application, the POI data model should be general-purpose and dynamically extensible. In addition, the POI data model should allow distribution of its contents to enable distributed storage and administration as well as combination of POI data model extensions from multiple different sources. Although many POI data models already exist, e.g. [8][9], none of them are on their own, general-purpose, dynamically extensible, distributed and capable of incorporating 3D content and sensor data.

This paper presents a distributed POI data model derived from the principles of the entity-component (EC) model previously known to virtual worlds and online gaming [10]. According to the EC model, each POI is treated as an entity that can possess one or more components. The distributed POI data model provides a general-purpose core component of which information can be easily extended by adding more application-specific components. To demonstrate this, components for 3D content and sensor data were defined for MAR applications. Using a simple mobile web application for testing, the performance of the distributed POI data model was evaluated in terms of (1) the amount of data transmitted, (2) client-side data parsing time and (3) client-side memory consumption. Based on the evaluation results, the distributed POI data model provides an efficient and a universal approach for defining POIs for novel LBS.

The rest of the paper is organized as follows: Section 2 presents the related work on POI data models; Section 3 introduces the distributed POI data model; Section 4 presents a distributed service architecture for providing access to the POI data; Section 5 describes the experimental setup for evaluating the performance of the distributed POI data model; Section 6 presents the results of the experimentation, and finally, Section 7 concludes the paper and provides some ideas for future work.

II. RELATED WORK

Several different POI data models are currently in use. Many navigator manufacturers, for instance, have their own proprietary data models and data serialization formats. In addition, some organizations and companies have published POI data models and APIs for general use. Next, the most relevant publicly available POI data models are introduced.

OpenPOIs [11] is a public database by Open Geospatial Consortium (OGC). It provides POI data in an earlier POI data model defined by World Wide Web Consortium POI Working Group (W3C POI WG) [9]. This data model is quite complex and requires substantial effort to apply. Moreover, the model is difficult to extend. One prominent feature of OpenPOIs data is the large amount of metadata associated to each data field. It allows compiling the POI data from multiple sources while each data item can be traced back to its origin. However, this significantly increases the volume of the transferred data.

OGC POI Standards Working Group is continuing the work of W3C POI WG and has published a draft of Points of Interest Conceptual Model [12]. The draft is a complex monolithic specification having the same drawbacks as the earlier W3C format used by OpenPOIs.

OpenStreetMap is an open spatial information provider, which provides map data in an XML format called OSM XML [13]. The basic building block in OSM XML is a "node" that has a location and additional information in form of "tags". A tag is a key-value pair. The set of keys is quite unstructured and open, which makes it complex to classify the map data items. In addition, OpenStreetMap does not make any clear distinction between POIs and other map data, which makes POI data retrieval quite difficult.

Google Earth provides Keyhole Markup Language (KML), which is a data model for representing both POI and generic map data. KML is standardized by OGC [14]. KML is focused on visualization of geographic data. Therefore, it is not a generic-purpose POI data model. In addition, it does not provide support for systematic extension of the data model.

Google Maps provides POI data via Google Maps JavaScript Places Library [15]. The POI data structure provided by the library is quite compact and easy to use. However, it is clearly tailored to be used only with Google Maps, which dramatically limits its applicability.

According to our best knowledge, only KML provides direct support for 3D content. However, none of the presented POI data models support sensor data.

III. Distributed POI Data Model

The objective of the proposed POI data model is to enable implementing a variety of applications utilizing POI data for all kinds of networked devices. The main goal of the design has been to provide a POI data model that is generic, easy to distribute, extensible, simple and light-weight to process. To achieve these goals, some of the design principles of the EC model have been adopted.

A single POI is an entity, which is identified by a universally unique identifier (UUID). The data related to an entity is stored in different data components, which are linked to the entity using the UUID. These data components are independent of each other and every POI entity can have a different set of components. Every POI must, however, have a 'fw_core' component that contains the very minimal core data describing the POI. The distributed POI data model can be seen in Fig. 1.

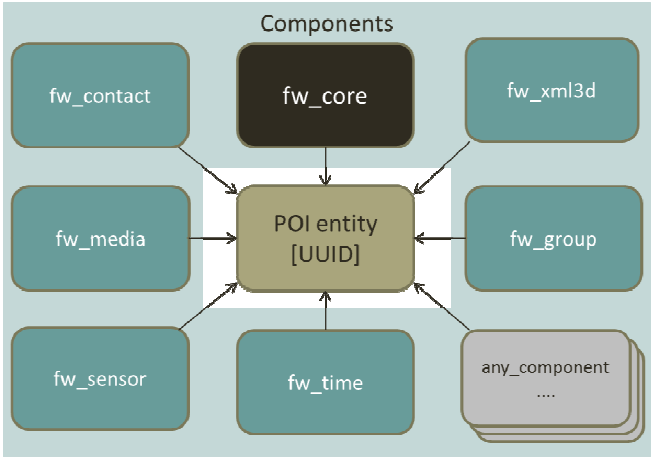

Fig. 1 Distributed POI Data Model.

The modular structure of the POI data model and loose coupling between the entities and the components has several benefits. The different data components can easily be distributed to different databases that reside on different servers. The amount of network traffic and the client-side processing load can also be dramatically reduced, as the client needs to fetch only the data components that are relevant to the application. The data model can be easily extended by adding new data components, as the data components are independent.

Currently, there are seven different data components defined for the POI data model.

- **fw_core** contains the essential information about a POI, such as the name, geographical location and category. This information is used for indexing the POI database thus enabling search functionality.
- **fw_contact** contains contact information for a POI, such as the street address, telephone number and e-mail address.
- **fw_time** stores time related information about a POI, such as the opening hours for a restaurant or dates for an event.
- **fw_media** can be used for including media items, such as photographs or videos, to a POI. The component also contains attributes for describing the media items, such as a title and a caption.
- **fw_group** is used for creating POI groups, such as all the main tourist attractions in a certain town. The group is expressed simply as a list of POI UUIDs. The group can be described in more detail using e.g. the fw_core component.
- **fw_sensor** has information about different sensor and actuator devices located within a POI. This information also contains the values for the sensors and statuses for the actuators. A meeting room could, for example, have a temperature sensor and an actuator for controlling the climate control.
- **fw_xml3d** contains the required information for linking a 3D model stored in an XML3D format to a POI. XML3D is an extension to HTML for representing 3D content in a declarative manner [16].

IV. DISTRIBUTED RESTFUL SERVICE ARCHITECTURE

A. *Overview of the Service Architecture*

A RESTful service architecture is defined for hosting and provisioning the POI data. The services provide different functionality for querying, accessing and modifying the POI data. The services are invoked via RESTful APIs using the standard HTTP protocol. As the POI data can be easily distributed due to its modular structure, the natural outcome is that the service architecture can be distributed as well. Different POI service backends can host a different set of POI data components.

In addition to the obvious technical benefits, such as server and network load distribution and scalability, the distributed service architecture enables also different businesses to host their own application specific data. As an example, a hotel chain might want to develop an application for finding their hotels and booking rooms. The generic information about the hotels could be presented using some of the predefined data components hosted on a public POI service. The 'fw_core' component could be used for placing the hotels on a map and displaying the basic information. 'fw_xml3d' could be used for showing 3D models of the hotels. As these data components are hosted on a public server and can be seen also by other applications, the visibility of the hotels increases. In addition, an application specific component designed specifically for the hotel chain's needs for viewing the hotel's booking status and booking rooms could be utilized and hosted by the company's own servers.

The distributed service architecture also enables implementing service composition, i.e. service backends that do not necessarily host any POI data themselves, but instead fetch the POI data from a set of other backends. An example deployment of the distributed POI service architecture can be seen in Fig. 2.

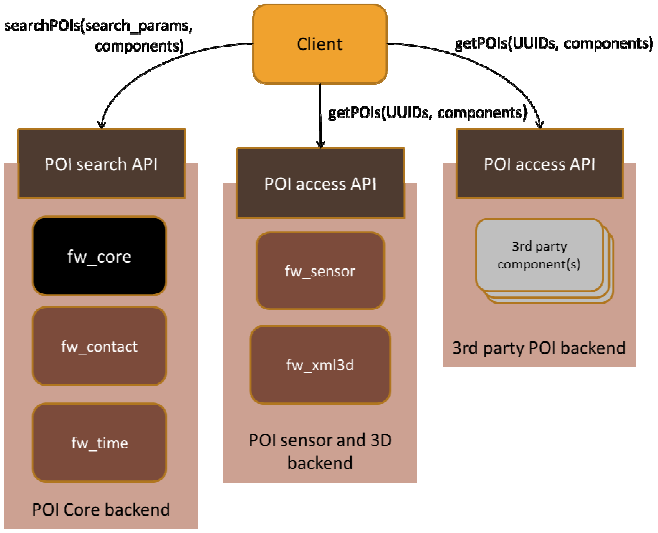

Fig. 2 An Example Deployment of the Distributed POI Service Architecture.

B. *Restful Web Service APIs*

The POI data can be searched and accessed via RESTful web service APIs. The POI access API enables accessing specific data components for POIs based on the UUIDs of the POI entities. The POI search API provides spatial search functions for a client to find relevant POIs in a certain location.

The POI access API provides functionality for accessing and modifying POI data. It contains the following functions:

- **get_pois** is used for retrieving the POI data. It takes as arguments a list of POI UUIDs and a list of data components.
- **get_components** returns a list of data components supported by the service backend.
- **add_poi** is used for adding a new POI entity into a database. It generates a UUID for the new POI and returns it to the client. The client can include different data components to the new POI by sending them along with the request.
- **update_poi** is used for updating data of an existing POI entity. Existing data components can be modified or new ones can be added. Each data component contains a 'last modified' timestamp in order to prevent concurrency issues.
- **delete_poi** deletes a POI entity from a database based on the given UUID.

The POI search API includes all the functionality of the POI access API. In addition, it provides spatial search functionality:

- **radial_search** can be used for searching POIs around a certain geographical point within a given radius, i.e. inside a circular area.
- **bbox_search** search enables searching POIs inside a given bounding box, i.e. a rectangular geographical area.

Both search functions share a set of common parameters. The search can be narrowed using the category parameter, which defines the categories of POIs to be returned by the search, such as 'restaurant' or 'café'. The searches also support defining temporal constraints, which can be used, for instance, to find restaurants that are currently open, or open during a given timespan. The number of search results can be limited using the 'max_results' parameter.

C. *Reference implementation*

The reference implementation of the RESTful web service architecture is implemented with PHP and it runs on top of Apache web server. The POI data is transmitted over the network using HTTP protocol in a JSON format, as it is widely used as well as easy and lightweight for a client to parse [17]. The data components are defined using JSON Schema, which can be used e.g. for validating the data. Two different database engines are used for storing the POI data. The 'fw_core' data is stored in a PostgreSQL database utilizing PostGIS database extension for enabling efficient spatial searches. All other data components are stored in a document oriented MongoDB database. MongoDB stores the documents in a JSON document structure, which enables easy access to and modification of the data. It also enables the reference implementation to support extending the data model with new components. An example of the POI JSON data can be seen in Fig. 3.

```
{
 "pois": {
   "30ddf703-59f5-4448-8918-0f625a7e1122": {
      "fw_core": {
        "category": "cafe",
        "name": "SomeCafe",
        "location": {
            "wgs84": {
              "latitude": 12.345,
              "longitude": 56.789
            }
        }
      },
      "fw_xml3d": {
        "model_id": "coffee1",
        "model": "<xml content>"
      },
      "any_component" {
       … component content …
      }
    },
    "6be4752b-fe6f-4c3a-98c1-13e5ccf01721": {
    … another POI …
    }
  }
}
```

Fig. 3 An Example of POI Data in a JSON Format.

V. Experimental Setup

The feasibility of our distributed POI data model and service architecture was evaluated by measuring different aspects related to client-side performance. The client software was a simple web application written in JavaScript and running in a web browser. It fetches the POI data from a server in a JSON format and parses the data into an object structure. The following measurements were performed:

- volume of payload data transferred over network
- JSON data parsing time
- memory consumption of the parsed JSON data object

The tests were conducted using three different mobile devices: a tablet and two smartphones. The tablet was ASUS Transformer Pad TF300TG with a 1.2 GHz NVIDIA Tegra 3 4-Plus-1 Quad Core CPU running Android 4.2.1. The first smartphone was Motorola Droid Razr i with a 2 GHz Intel Atom Z2460 CPU running Android 4.1.2. The second smartphone was Samsung Galaxy S II with a 1.2 GHz dual-core ARM Cortex-A9 CPU running Android 4.0.4. The web browser used in all devices was Firefox version 27.

Three test cases were defined representing application scenarios of different complexity. The first test case represents a very simple LBS application that requires only the essential data of a POI, e.g. the 'fw_core' component. The second test case represents a more complex application scenario, such as a tourist guide application. It includes a total of four data components: 'fw_core', 'fw_contact', 'fw_time' and 'fw_media'. The third test case represents a full-blown MAR application and it includes all seven data components.

Three test datasets were created, one for each test case. Each dataset contains a total of 1000 POIs selected from a certain geographical area. Dataset 1 contains the real information about the POIs in the 'fw_core' component: name, category and location. Dataset 2 and Dataset 3 were generated by injecting replicated hand-written example data components into the POIs.

Fourth dataset, obtained from the OpenPOIs Database, was used as a comparison. It contains the same 1000 POIs as our own datasets in a JSON format. OpenPOIs was selected for the evaluation as it is basically the only available free POI service that provides global POI data via an open service API. The OpenPOIs data contains mainly three attributes for each POI: name, category and geographical location. These same attributes are stored for each POI in Dataset 1. In addition to these core attributes, some POIs in the OpenPOIs dataset also contain some additional properties, such as street address (13.9%), website (7.5%) and telephone number (4.3%).

VI. Results

A. Data volume

The volume of the POI data is especially interesting from the client's point of view, as it affects several things related to the application performance: data transfer time, data processing time and memory consumption. The data volume for each dataset was determined simply by measuring the size of the JSON file. The data volumes for each dataset can be seen in Fig. 4. The sizes of the datasets were the following: Dataset 1 - 350 kB, Dataset 2 - 2258 kB, Dataset 3 - 5080kB and OpenPOIs - 5339 kB.

The benefits of our modular approach, especially the client's ability to select only relevant data components can be clearly seen by observing the data volumes between datasets 1-3. Compared to Dataset 1, Dataset 2 was 6.4 times larger and Dataset 3 14.5 times larger. The OpenPOIs dataset was 15.3 times larger compared to Dataset 1, which essentially contains almost the same information.

The data volume of the OpenPOIs dataset is even slightly larger than our largest dataset, Dataset 3. In essence, however, Dataset 3 contains much more information compared to the OpenPOIs dataset. The main reason why our model consumes less space is the fact that our JSON data format is more compact. The OpenPOIs JSON data format contains for example many attributes with "null" value, which in most cases serve no purpose and only increase the data volume.

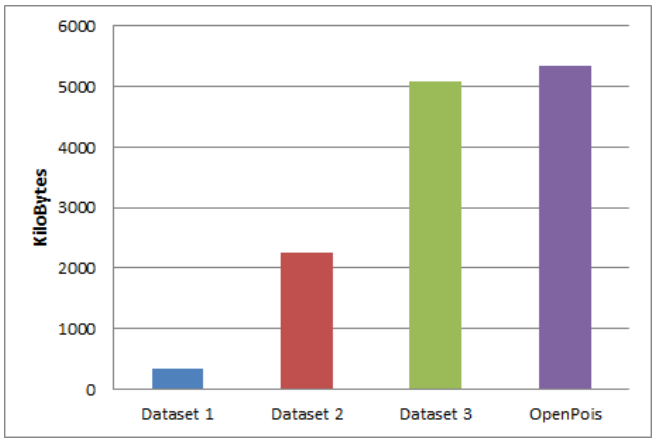

Fig. 4 Data volumes of different test datasets.

As most modern web browsers and web servers support compression, the HTTP payload data is usually gzip compressed. Therefore, we also measured the compressed sizes for Dataset 1 and OpenPOIs, which can be seen in Fig. 5. Dataset 2 and Dataset 3 were left out of this measurement as they contain a lot of replicated values, and therefore, the compression algorithm can compress them much more efficiently than the real-world values. Even when compressed, the OpenPOIs dataset is still larger than the uncompressed version of Dataset 1.

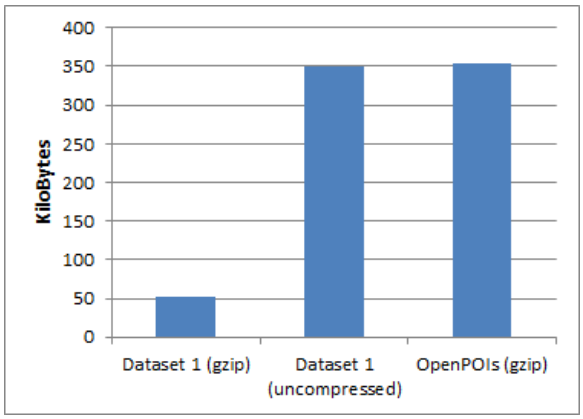

Fig. 5 Data volumes with compression.

The impact of the data volume for the data transfer time was also analyzed. Based on the field-studies conducted in live mobile networks [18][19], the data transfer times for each dataset were calculated as seen in Table 1. It is quite clear that the data transfer times become an important issue, especially with the slower connections and uncompressed data. However, even with the slowest WCDMA connection, uncompressed Dataset 1 can be transferred in within eight seconds, after which an LBS application is able to show the basic information about the POIs to the user.

TABLE 1. DATA TRANSFER TIMES FOR EACH DATASET IN DIFFERENT MOBILE NETWORKS (MEASURED IN SECONDS).

	WCDMA (370 kb/s) [18]	HSPA (2.8 Mb/s) [18]	Poor LTE (3.5 Mb/s) [19]	Good LTE (12 Mb/s) [19]
Dataset 1	7.57	1.00	0.80	0.23
Dataset 2	48.83	6.45	5.16	1.51
Dataset 3	109.84	14.52	11.61	3.39
OpenPOIs	115.44	15.25	12.20	3.56
Dataset 1 (gzip)	1.13	0.15	0.12	0.03
OpenPOIs (gzip)	7.67	1.01	0.81	0.24

B. *Data parsing time*

The time elapsed for the web browser to parse the JSON data into a JavaScript object structure was measured. The native JSON support in Firefox web browser was utilized, namely the JSON.parse(jsonString) function. The JSON data parsing time significantly affects the performance of the web application. This is due to the fact that the JSON parsing is a blocking function call, i.e. the whole application remains unresponsive until the call is completed. The time elapsed in the parsing was measured by recording the current time before and after the call to JSON.parse in the JavaScript code. The parsing times for the different datasets and different mobile devices can be seen in Fig. 6.

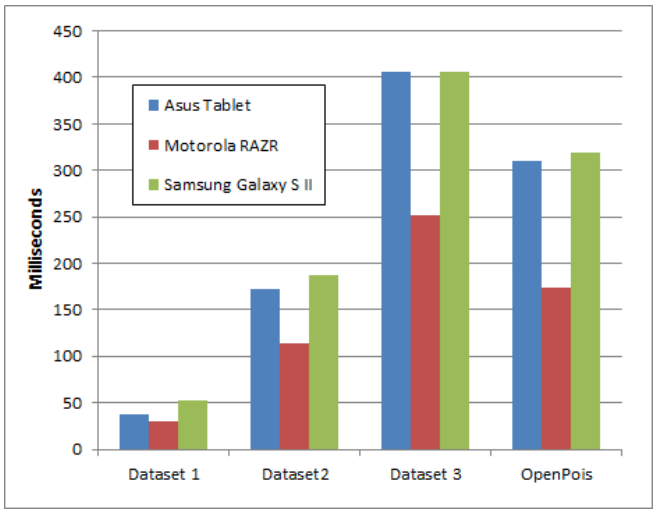

Fig. 6 Data parsing times for different mobile devices.

The parsing times are in quite close relation to the data volumes of the datasets. Dataset 1 can be parsed in under 50 milliseconds with all devices. The differences in processing power between the different devices become more evident with the larger datasets. Motorola RAZR is able to parse Dataset 3 about 150 milliseconds faster than Samsung Galaxy S II. Dataset 1 can be parsed roughly in 50 milliseconds even with the slowest device, while parsing the OpenPOIs dataset takes over 300 milliseconds, which is over 6 times longer. Even though the OpenPOIs dataset is the largest in data volume, its parsing time is slightly faster than for Dataset 3. This is because the data structures in the OpenPOIs dataset do not contain as much levels of hierarchy and information content as in Dataset 3.

C. *Memory consumption*

The memory consumption of the POI data was measured after the JSON data was parsed into a JavaScript object structure. The measurement was carried out using the "about:memory" memory measurement tool found in Firefox web browser. The memory consumption of the browser tab, where the test web application was running, was measured both before and after the JSON data parsing. This measurement was done only using the Asus Tablet device, as the memory consumption with all the mobile devices, and even a Windows PC, was found nearly identical. The memory consumption for the different datasets can be seen in Fig. 7.

The memory consumption between the different datasets seems to change in similar proportion as data parsing times for any single device. The JavaScript object structure for the OpenPOIs data is a bit smaller than for Dataset 3, even though

the JSON data for OpenPOIs is larger. As the Dataset 3 contains more complex data structures and more information content, the JavaScript object structure generated during the parsing is also more complex and therefore consumes more memory. However, the memory consumption of Dataset 1 is only about 7% of that of the OpenPOIs dataset. Especially with a large number of POIs, the memory consumption can become a key factor for LBS applications. Extensive memory consumption can lead to reduced performance and instability especially with web applications.

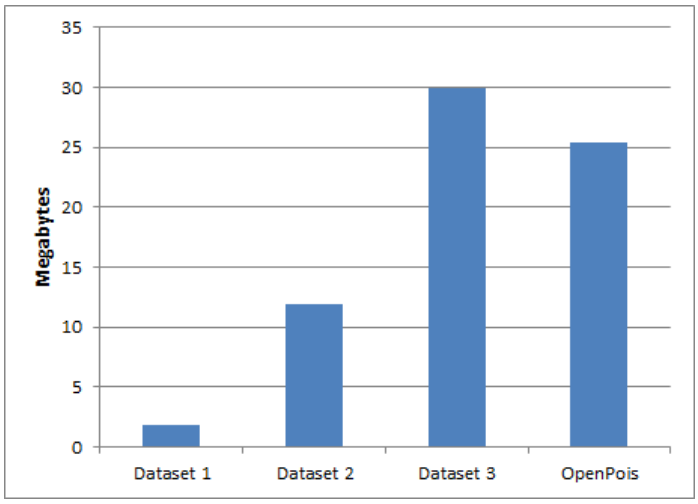

Fig. 7 Memory consumption for different datasets after JSON data parsing.

VII. Discussion and Conlusions

The distributed POI data model presented in this paper serves as a generic enabler for building a variety of LBS applications. Due to its modular design, the data model allows easy distribution and extension of the POI data. The RESTful service architecture provides APIs for efficient access to the POI data. It also enables different businesses and organizations to host their own POI data components serving specific application requirements. In terms of client-side performance, the proposed POI data model was shown to outperform an existing POI implementation in a mobile web context. Enabling the client to select only the data components relevant to the application was shown to be an effective mechanism for improving performance. As future work, the server-side performance of the distributed service architecture could also be evaluated and the POI data model could be compared against other existing POI implementations. The work presented in this paper will be used as a part of the European Commission FI-WARE open cloud-based infrastructure for building Future Internet applications.

VIII. Aknowledgements

This research has been supported by the EU FI-PPP FI-WARE project and the CADIST3D project funded by the Academy of Finland.

References

[1] O. Ozdikis, F. Orhan, and F. Danismaz, "Ontology-based Recommendation for Points of Interest Retrieved from Multiple Data Sources," In Proc. International Workshop on Semantic Web Information Management (SWIM'11), ACM Press, Jun. 2011, Article No. 1, doi:10.1145/1999299.1999300

[2] T. Olsson, T. Kärkkäinen, E. Lagerstam, and L. Ventä-Olkkonen, "User evaluation of mobile augmented reality scenarios," Journal of Ambient Intelligence and Smart Environments, vol. 4, pp. 29-47, 2012.

[3] M. Santala, J. Hyvärinen, S. Hickey, and J. Vatjus-Anttila, "3D Object Reconstruction Processing Chain for Extensible Virtual Spaces," In Proc. Academic MindTrek Conference, ACM Press, Oct. 2011, pp. 223-224, doi: 10.1145/2393132.2393178

[4] M.D. Symes, P.J. Kitson, J. Yan, C.J. Richmon, G.J.T. Cooper, R.W. Bowman, T. Vilbrandt, and L. Cronin, "Integrated 3D-printed reactionware for chemical synthesis and analysis," Nature Chemistry, vol. 4, pp. 349-354, 2012.

[5] S. Ortiz, "Is 3D finally ready for the Web?" IEEE Computer, vol. 43, iss. 1, pp. 14-16, 2010.

[6] T. Nam, and T.A. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," In Proc. 12th International Digital Government Research Conference: Digital Government Innovation in Challenging Times, ACM Press, 2011, pp. 282-291, doi: 10.1145/2037556.2037602

[7] A. Rae, V. Murdoch, A. Popescu, and H. Bouchard, "Mining the web for points of interest," In Proc. 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press, 2012, pp. 711-720, doi: 10.1145/2348283.2348379

[8] M. Ruta, F. Scioscia, S. Ieva, G. Loseto, and E.D. Sciascio, "Annotation of OpenStreetMap Points of Interest for Mobile Discovery and Navigation," In Proc. IEEE International Conference on Mobile Services, IEEE Press, June 2012, pp. 33-39, doi: 10.1109/MobServ.2012.17

[9] W3C Points of Interest Working Group, "Points of Interest Core", W3C Editor's Draft, http://www.w3.org/2010/POI/documents /Core/core-20111216.html

[10] T. Alatalo, "An Entity-Component Model for Extensible Virtual Worlds," IEEE Internet Computing, vol. 15, iss. 5, pp. 30-37, 2011.

[11] "OpenPOIs Database", http://openpois.net/

[12] Open Geospatial Consortium, "Points of Interest Conceptual Model", https://github.com/opengeospatial/poi/blob/master/spec /poi.adoc

[13] "OSM XML – OpenStreetMap Wiki", http://wiki.openstreetmap.org/wiki/OSM_XML

[14] Open Geospatial Consortium, "KML", OGC Standards, http://www.opengeospatial.org/standards/kml/

[15] "Places Library – Google Maps JavaScript API", https://developers.google.com/maps/documentation/javascript/places

[16] K. Sons, F. Klein, D. Rubinstein, S. Byelozyorov, and P. Slusallek, "XML3D: Interactive 3D Graphics for the Web," In Proc. 15th International Conference on Web 3D Technology, ACM Press, 2010, pp. 175-184, doi:10.1145/1836049.1836076.

[17] B. Gil, and P. Trezentos, "Impacts of data interchange formats on energy consumption and performance in smartphones," In Proc. Workshop on Open Source and Design of Communication, ACM Press, July 2011, pp. 1-6, doi:10.1145/2016716.2016718

[18] J. Prokkola, P. Perälä, M. Hanski, E. Piri, "3G/HSPA Performance in Live Networks from the End User Perspective," In Proc. IEEE International Conference on Communications, IEEE Press, June 2009, pp. 1-6, doi: 10.1109/ICC.2009.5198575

[19] M. Ahmed, "Performance test of 4G (LTE) networks in Saudi Arabia," In Proc. International Conference on Technological Advances in Electrical, Electronics and Computer Engineering, IEEE Press, May 2013, pp. 28-33, doi: 10.1109/TAEECE.2013.6557190