Semantic RESTful APIs for Dynamic Data Sources

Felix Leif Keppmann and Steffen Stadtmüller

Institute AIFB
Karlsruhe Institute of Technology (KIT)
{felix.leif.keppmann, steffen.stadtmueller}@kit.edu

Abstract. The amount of available highly dynamic data sources on the Web is constantly increasing. Such dynamic data sources usually require client applications to be capable to process continuous data streams. We present an approach that enables dynamic data sources to be exposed with RESTful APIs. The use of REST implies that individual resources are exposed, which enables clients to interact in a more fine-grained manner with the data source compared to the processing of a data stream. Furthermore the use of Linked Data resources allows for the direct integration with other data sources. We demonstrate the practical applicability of our approach based on a proof-of-concept implementation of a RESTful API that exposes video sensor data and evaluate the feasibility of a RESTful communication pattern.

1 Introduction

In the context of services on the Web, recent years have witnessed the proliferation of Web APIs, also referred to as RESTful services, when conforming to the Representational State Transfer (REST) design principles [8]. In fact, ProgrammableWeb[1], currently the most popular directory for Web APIs, reports 70% of the known Web APIs to be built following REST. This wide adoption demonstrates the advantages of the so implemented solutions in terms of flexibility and loose coupling, which are directly fostered by the REST architectural style [12].

At the same time many developments on the Web have been marked by the increased use and importance of Linked Data [6]. This trend lays the foundation for a rich data environment that enables the design and implementation of a variety of client applications, benefiting from the integration of distributed data available through different sources. Furthermore, this landscape of mostly static data is enriched though the availability of dynamic data sources. Such time-sensitive data can be generated by environmental measurement sensor networks, motion detectors, or smartphones, and provides, for instance, information about weather, traffic, user tracking or geospatial positioning [1]. Dynamic data sources are often perceived as streaming data sources.

Still, up to date, data streams have been seen as incompatible with REST-based application design, especially regarding the imposed stateless communication of resource state representations. Therefore, most approaches for handling dynamic data sources are limited to using RESTful APIs only to communicate meta information about the data sources or, alternatively, to perform isolated one-off lookups. It is commonly assumed

[1] http://www.programmableweb.com/

that applications that require the processing of dynamic data are based on a technology stack that already implements protocols for data streaming.

In this paper we introduce an approach to expose highly dynamic data sources via a semantic RESTful interface (Section 2.1). Our approach maintains the characteristics of a RESTful implementation and, at the same time, keeps the dynamic nature of the original source. Furthermore, the use of Linked Data enables the direct integration with other data sources.

As opposed to the direct processing of a data stream, the use of semantic RESTful APIs, as foreseen by our approach, provides advantages for processing dynamic data sources that satisfy the requirements imposed by real time applications:

Data Preparation: RESTful APIs expose individual resources, which can be accessed and integrated independently into client applications. Thus the client is relieved from identifying the resources contained in a stream.

Data Selection: Applications are enabled only to retrieve the required data, rather than accessing the complete stream data.

Interoperability: The employment of self-descriptive Linked Data resources enables the integration of static and dynamic cross-domain sources, e.g., based on different sensor devices.

Performance Decoupling: Clients can exhibit large differences for update rate requirements of different resources. The enabled decoupling of server-side updates and client request rates avoids unnecessary data transfers and performance bottlenecks.

The architectural design is verified with a first proof of concept system implementation, called *Natural Interaction via REST* (NIREST) (Section 2.2). In Section 3 we show the feasibility of our approach in a highly dynamic environment, based on a preliminary evaluation of the NIREST system.

1.1 Preliminaries

According to the Richardson maturity model [13] REST is identified as the interaction between a client and a server based on three principles:

- The use of URI-identified resources.
- The use of a constrained set of operations, i.e., the HTTP methods, to access and manipulate resource states.
- The application of hypermedia controls, i.e., the data representing a resource contains links to other resources.

The idea behind REST is that applications, i.e., clients, interact with APIs by retrieving and manipulating exposed representations of resources. A resource can be a real world object or a data object on the web. The representation of a resource details the current state of the resource.

The characteristic of a dynamic data source is a very high update rate of the resource states. To enable clients to capture the resulting velocity of the data, dynamic data sources often expose a stream of data, which provides the information about the occurring state updates.

Linked Data unifies a standardised interaction model with the possibility to align vocabularies using RDF, RDFS and OWL. Following the motivation to look beyond the exposure of fixed datasets, the extension of Linked Data with RESTful technologies has been proposed [5, 17] and many existing approaches recognise the value of combining RESTful services and Linked Data [10, 14, 16, 15].

2 Approach

Our approach shows the feasibility of a RESTful Linked Data architecture in the context of real time applications. An architecture is developed which copes with the requirements of both the high dynamic nature of the resources and the demand for integration and scalability. In this section we detail our approach and illustrate the advantages with the example of a video sensor and describe a proof of concept implementation.

2.1 Architecture

Many exposed dynamic data sources expect an architecture where client applications consume a data stream that continuously delivers updates (cf. Figure 2.1a). Client applications in such an architecture have to process this data stream and identify the required information to perform an intended task, e.g., by means of event processing or stream reasoning.

We propose an architecture, where the server itself processes the data stream and exposes identified resources directly in a RESTful way (cf. Figure 2.1b). The resources are exposed in accordance with the Linked Data principles.

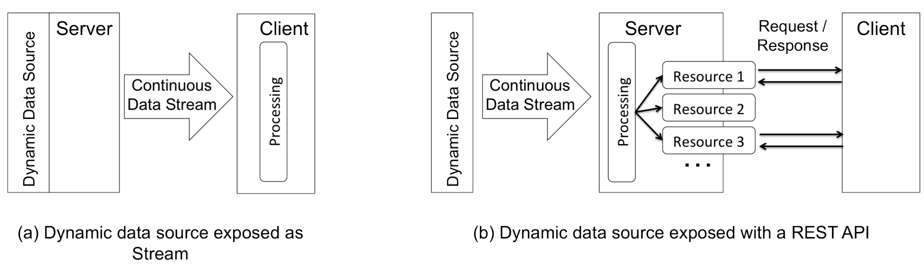


Fig. 1. Exposure of dynamic data sources

Data Preparation: The proposed architecture design reliefs the client from the challenge to process the data stream and identify the relevant pieces of information. Since the data is already encapsulated as individual resources, a client can focus on performing its actual tasks, thus reducing hardware requirements on the client side. A client might still further transform the retrieved data. However, such further processing is still expected to be less tedious, than the processing of a data stream.

Example: A video sensor exposes a data stream that contains the information of objects in front of the sensor. A RESTful API exposes (constantly changing) resource representations for each the recognized objects individually. Therefore a client must not analyze the stream to match beginning and end of information for individual objects.

Data Selection: A resource oriented architecture as fostered by REST additionally enables a reduction of the amount of communicated data: On the one hand clients can limit their requests to the information (i.e., resource states) required for their tasks. On the other hand irrelevant information can already be discarded during the pre-processing on the server-side. However, this implies some foresight as to the expected use-cases when designing the RESTful API. Note, that if a client requires information that was discarded, the client can still fall back to the complete stream.

Example: A video sensor can distinguish recognized persons and other objects in front of the sensor. If a client requires only the position of persons, the client does not need to retrieve information about the objects.

Interoperability: The use of Linked Data resources entails the advantage of easy data integration and ease the combination of different data sources. Resources can provide links to other relevant information from the same as well as third party sources, thus enabling clients to navigate to additional relevant information. The use of domain ontologies further reduces the required effort for the clients to leverage the provided information for their tasks.

Example: If a video sensor supports face recognition, the exposed resources could automatically include links to social network information (e.g., foaf files) of the recognized persons in front of the sensor.

Performance Decoupling: A client might not require realtime updates for all available information. Due to hardware limitations clients might also not be capable to process high frequency updates. The RESTful exposure of the individual resources with a pull-based approach for retrieval enables clients to scale the retrieval intervals of different resources in a fine grained manner.

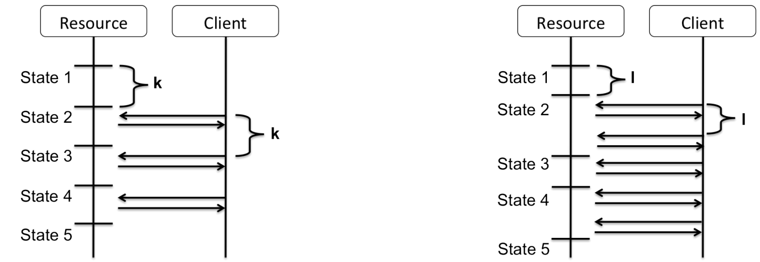
Example: The exposed resources that represent objects in front of a video sensor can link to the precise polygon information detailing the shape of the objects. A client that renders the objects in front of the video sensor must retrieve the precise polygon shape only once. Further requests of the client can be constrained to position information of the object.

The pull-based approach to retrieve the resource states is often perceived as a downside of a REST interface: To accurately capture all state changes a client has to request the resource states at least at the same rate as it is updated on the server (cf. Figure 2.1a). However, in case the resources are updated at non-regular intervals a client has to request the resources at an interval equivalent to the shortest update interval to capture all state updates. The resulting higher request rate implies that during periods with lower update rates on the server the client retrieves duplicates of a resource (cf. Figure 2.1b).

However, in many use-cases the overhead due to pull-based data retrieval is negligible and compensated by the described advantages achieved by leveraging a RESTful design pattern.

2.2 Proof of Concept

As proof of concept system we develop *Natural Interaction via REST* (NIREST) to show the feasibility of our approach. NIREST combines different technologies to provide access on sensor details, recognized persons, and skeleton coordinates, extracted



(a) Regular state update interval k => client has to retrieve the resource state at an interval of at least k to capture all states

(b) Non-regular state update, with shortest interval l => client has to retrieve the resource state at an interval of at least l to capture all states

Fig. 2. Resource update and request rates for dynamic data sources

on-the-fly from a video sensor. The system exposes a RESTful API with Linked Data resources according to domain-specific ontologies.

NIREST is implemented as application container, deployable on application servers, e.g. an Apache Tomcat[2] instance. For video analysis and low level access to a sensor (e.g. a Microsoft Kinect[3]) we employs the OpenNI[4] framework and the NiTE[5] middleware.

The OpenNI framework acts as a hardware abstraction layer between sensor and middleware libraries or applications, thus enabling unified access to, e.g., colour video, depth video, sensor metadata, or device events.

The NiTE middleware accesses the depth video provided by the OpenNI framework and implements body tracking algorithms. NiTE enables the recognition of objects in front of the sensor as persons, the coordinates of the derived skeleton joints as well as basic gestures.

We use Apache Jena[6] to serialise the extracted information in different RDF formats. The RDF representation of all relevant resources (e.g., a person in front of the sensor or the sensor itself) is continuously updated (including links between the resources). Every resource is accessible with individual URIs. To implement this RESTful interface we use the Jersey[7] framework.

Example: A resource representing a person contains information about the user position (centre of mass) and information about all the joint points of its skeleton. Listing 1 shows an excerpt of such a resource accessible under `http://[...]/device/0/user/1`, in Turtle format.

```
@prefix nirest: <http://vocab.arvida.de/2014/02/nirest/vocab#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

<>
```

[2] http://tomcat.apache.org
[3] http://www.microsoft.com/en-us/kinectforwindows/
[4] http://www.openni.org/
[5] http://www.openni.org/files/nite/
[6] http://jena.apache.org/
[7] http://jersey.java.net/

```
  a nirest:User ;
    nirest:centerOfMass [ a nirest:Coordinate3D ;
      nirest:x "-133.63106"^^xsd:float ;
      nirest:y "-27.113548"^^xsd:float ;
      nirest:z "2298.0164"^^xsd:float ] ;
    nirest:skeleton [ a nirest:Skeleton ;
      nirest:joint [ a nirest:RightHandJointPoint ;
        nirest:coordinate [ a nirest:Coordinate3D ;
          nirest:x "-2.150751"^^xsd:float ;
          nirest:y "-1.2886047"^^xsd:float ;
          nirest:z "2324.944"^^xsd:float ] ;
        nirest:orientationConfidence "1.0"^^xsd:float ;
        nirest:positionConvidence "1.0"^^xsd:float ] ;
...
```

Listing 1. User and Skeleton Tracking Data in Turtle Format

The NIREST ontology[8] used to annotate the extracted data has been created for the specific use case of the proof of concept application. In the past the domain of 3D sensors, tracking, virtual reality, augmented reality, and related fields has not been in the focus of ontology engineering. The custom ontology will be replaced by ontologies developed within the ARVIDA[9] project, as soon as these ontologies are stabilized. Focus of the ARVIDA project is the development of a reference architecture for virtual services and applications, which includes the development common ontologies for the domain, covering the needs of NIREST.

3 Evaluation

We present a preliminary evaluation on the performance of the implementation described in Section 2.2.

In the setup of the evaluation we use as server an Apache Tomcat with a deployed NIREST container and a connected Microsoft Kinect[10] as sensor. The Kinect has a nominal update frequency of 30 images per second[11]. In a sample of 200 updates we measured an average server side update frequency of the exposed RDF resources of 30.9 Hz with a standard deviation of 5.7 Hz (cf. Figure 3a).

As client we use a second machine that requests the representation of a person in front of the Kinect. Every request is performed with a complete HTTP request/response. We measured the number of retrieved distinct representations, which the client can acquire with different request frequencies between 5 Hz and 30 Hz (cf. Figure 3b). An retrieval rate of 30 Hz allows a theoretical optimal retrieval, where all server updates are captured without retrieving duplicates. Note, that representations of several resources could be done in parallel, and would not affect our measurements.

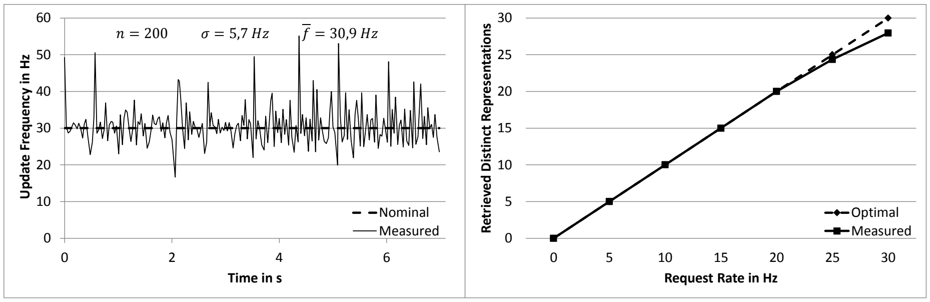
We see that even though the server exhibits noticeable fluctuations in its update rates, the client can retrieve on average the optimal amount of distinct representations up to to a request rate close to 25 Hz. Only for a request rate of 30 Hz the fluctuations

[8] http://vocab.arvida.de/2014/02/nirest/vocab

[9] http://www.arvida.de/en/

[10] In the context of this work, in particular the first generation of the Microsoft Kinect for Xbox 360/Windows is used for evaluation

[11] The RDF representations are extracted from the available 24 bit RGB video and 16 bit depth video with 640x480 pixels



(a) Server side update frequencies

(b) Distinct representations for different client request frequencies

Fig. 3. Client and Server Update Frequencies

of the server-side update frequency becomes noticeable and the client can only capture 28 distinct representations.

The results imply that a highly dynamic RESTful communication pattern is achievable. As main insight we observe that the largest potential for optimisation is to reduce the jitter in the update frequency of the server, rather than an increased request frequency of the client. A more homogenous update rate would allow a communication pattern in which a client can fully capture all information without increasing the the retrieval overhead.

4 Related Work

Many existing approaches (e.g., [2–4, 7, 11]) to deal with semantically enabled dynamic data sources focus on the processing of the stream. Our work should not be understood as competing with these approaches, since the decision for the communication pattern is highly dependant on the individual use-case. Furthermore, such approaches can be employed for the server-side processing and therefore also help to enable RESTful APIs.

Harth et al. [9] highlight the integration capabilities of RESTful Linked Data APIs. Here an integration scenario is described in which dynamic and static geospatial data sources are combined. Our work expands on this idea by pinpointing the specific advantages of REST for dynamic data sources.

STREST[12] is a protocol aimed at the combination of REST and data streams, by enabling clients to switch between both communication patterns. Such a combination is complementary to our work as it allows clients to fall back to a stream, when necessary.

5 Conclusion

We have outlined the advantages of RESTful APIs in combination with Linked Data to expose dynamic data sources, specifically the possibilities for fine-grained interaction

[12] `https://github.com/trendrr/strest-server/wiki`

with individual resources and for integration with other data sources. Our preliminary results indicate the feasibility of such a resource-based design pattern for scenarios that require high update rates. In future work we will continue to investigate how RESTful design pattern in highly dynamic environments can be employed. Specifically we target more regular server-side update rates with a precision in a low millisecond area to minimise the overhead of client requests. Further we aim for higher update and request frequencies in the low hectohertz area.

This work was partially supported by the German Ministry of Education and Research (BMBF) within the projects ARVIDA (FKZ 01IM13001G) and Software-Campus (FKZ 01IS12051) and by the PlanetData NoE (FP7:ICT-2009.3.4, #257641).

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. ComNet 38(4), 393–422 (2002)
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Ep-sparql: A unified language for event processing and stream reasoning. In: Conference on World Wide Web (2011)
3. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: Semantic foundations and query execution. VLDB 15(2), 121–142 (2006)
4. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-sparql: Sparql for continuous querying. In: Conference on World Wide Web (2009)
5. Berners-Lee, T.: Read-Write Linked Data (2009), avaiable at http://www.w3.org/DesignIssues/ReadWriteLinkedData.html, accessed 26th November 2013
6. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. IJSWIS 5(3), 1–22 (2009)
7. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. IJSWIS 8(1), 43–63 (2012)
8. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
9. Harth, A., Knoblock, C., Stadtmüller, S., Studer, R., Szekely, P.: On-the-fly integration of static and dynamic sources. In: Consuming Linked Data (2013)
10. Krummenacher, R., Norton, B., Marte, A.: Towards Linked Open Services. In: Future Internet Symposium (2010)
11. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: International Semantic Web Conference (2011)
12. Pautasso, C., Wilde, E.: Why is the web loosely coupled?: A multi-faceted metric for service design. In: Conference on World Wide Web (2009)
13. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2007)
14. Speiser, S., Harth, A.: Integrating Linked Data and services with Linked Data Services. In: Extended Semantic Web Conference. pp. 170–184 (2011)
15. Stadtmüller, S., Speiser, S., Harth, A., Studer, R.: Data-fu: A language and an interpreter for interaction with read/write linked data. In: Conference on World Wide Web (2013)
16. Verborgh, R., Steiner, T., Deursen, D.V., de Walle, R.V., Valls, J.G.: Efficient runtime service discovery and consumption with hyperlinked RESTdesc. In: Conference on Next Generation Web Services Practices (2011)
17. Wilde, E.: REST and RDF granularity (2009), available at http://dret.typepad.com/dretblog/2009/05/rest-and-rdf-granularity.html, accessed 26th November 2013