The Implications of Using Linked Data when Connecting Heterogeneous User Information

Karl Lundfall
Faculty of Sciences
Vrije Universiteit
Amsterdam, The Netherlands
kalle.lundfall@gmail.com

Abstract—In the realm of database technologies, the reign of SQL is slowly coming to an end with the advent of many NoSQL (Not Only SQL) alternatives. Linked Data in the form of RDF is one of these, and is regarded to be highly effective when connecting datasets. We revised a real-world system for linking datasets based on a much more mainstream NoSQL technology, and by altering the approach to instead use Linked Data, we saw how we could improve on the current solution. The result was a more modular system living up to many of the promises of RDF. On the other hand, we also found that there for this use case are some obstacles in adopting Linked Data.

We saw indicators that more momentum needs to build up in order for RDF to gradually mature enough to be easily applied on use cases like this. The implementation we present and demonstrates a different flavor of Linked Data than the common scenario of publishing data for public reuse, and by applying the technology in business contexts we might be able to expand the possibilities of Linked Data.

1 Introduction

As data is growing both in quantity and complexity, increasingly many developers are looking for alternatives to the conventional relational database systems. The current recession of relational databases like SQL is often attributed to problems such as scalability issues [1] and an object-relational impedance mismatch [2]. The adoption of NoSQL (Not only SQL) techniques is thereby getting more widespread and is heavily accelerated thanks to high-performing services developed by business giants [3], such as Amazon's Dynamo [4] and Google's Bigtable [5].

In addition, the schema-less nature of many NoSQL databases also brings about an improved ability to handle loosely organized data with ease [6]. One of the most ambitious efforts in pursuing this is *Linked Data*, usually expressed by triples or quadruples adhering to the RDF (Resource Description Framework) format [7]. This semantic technology can be of great asset when facing challenges like integrating datasets, ameliorating reusability, and interpreting complicated data structures [8]. Linked Data is often well-suited for contributing to an open web of information in the spirit of the Linked Open Data Movement [9]. However, not all Linked Data is open, and the technique can also be of benefit when linking heterogeneous datasets internally [10], which was the incentive behind this project.

We used RDF to design a system that generalize and aggregate various data structures of user information. We looked into how the choice of database can affect the development, maintenance, and quality of a product by revising a solution for the social enterprise TTC Mobile. The system we examined elicits information originating from different sources and presents this to the user through an interface. The tool also serves as a central hub allowing different applications to be able to communicate and reuse data. The RDF-based solution was compared to the existing implementation that is using the document-oriented store MongoDB[1], which is one of the most well established NoSQL database as of today.

TTC Mobile[2] (Text to Change Mobile, hereafter referred to as *TTC*) is a non-governmental organization equipping customers in developing countries with high-quality information and important knowledge they could not acquire for themselves. TTC offers mobile-based solutions such as SMS and call services and focuses on projects implying a social change coherent with the values shared by the company.

[1]https://www.mongodb.com/
[2]http://ttcmobile.com/

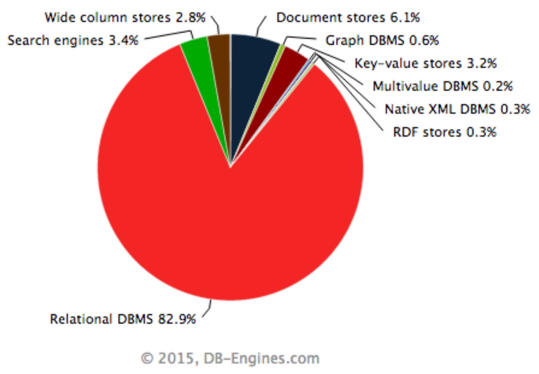


Fig. 1: The distribution of popularity measures for different database types as defined by `db-engines.com`. The figure was retrieved in June 2015.

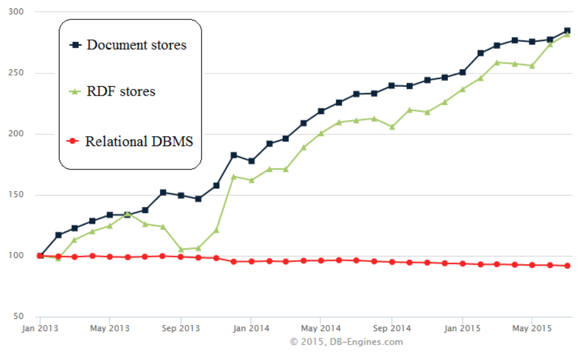


Fig. 2: The popularity change for database types as defined by `db-engines.com`. The y-axis depicts the percentage of scores in relation to the start of the time line in January 2013. The figure was retrieved in June 2015.

1.1 Problem Summary

By using semantic technology to connect the datasets of TTC, we obtained insights in possibilities of using Linked Data as opposed to a document store, which would be a more common solution to this problem. We did this by revising the original document-oriented solution already implemented by TTC, and looked into different possibilities of Linked Data improving on this.

2 Background

2.1 ICT4D and Linked Data usage

The process of utilizing ICT to aid developing countries is often referred to as *ICT4D* (Information and Communication Technology for Development), and has been practiced under different names since the 90s [11]. The intersection between semantic technology and ICT4D has also been touched upon. Initiatives in this direction are mainly oriented towards Linked Open Data, and has its roots in the fact that open ICT operations is thought to be an important catalyst in developmental aid [12]. A recent example of this is the initiative of converting documents from the International Aid Transparancy Initiaitive (IATI) to Linked Open Data [13]. In our project, the objective of publishing data was of secondary nature, but we still allowed for this as a future possibility when designing the system.

2.2 Linked Data popularity

Although the prevalence of Linked Data is relatively big within ICT4D, the overall interest in RDF technology is not very pronounced outside academic contexts. Figure 1 shows the current proportional popularity[3] among different database types, as defined by the ranking site `db-engines.com`. Relational databases appear to be the most popular method by far, followed by document stores. According to the measure, RDF stores constitute 0.3% of the total, giving it the impression of being one of the more obscure entries in this chart.

These statistics can partly be explained by a gap of maturity between SQL technologies that have been developed since the 70s, and the RDF standard which was published in 2004 [7]. But when looking at Figure 2, we see a hint that the tide could be turning. The popularity of relational databases is declining, while RDF stores are getting more attention. Given these figures, semantic technology could be considered to still be in its infancy, and we review some well-known problems of Linked Data later in this section. We also see that the document stores, being the second biggest database model, grows by a slightly higher rate than RDF, proportionally speaking. Statistics from `db-engines.com` also concur with the common view that MongoDB is the most popular document store as well as being the biggest NoSQL player overall.

We also note the share of graph databases (0.6%), of which RDF stores can be considered to be a more narrow subset. In this work we

[3]For information about calculation of this measure, please refer to http://db-engines.com/en/ranking_definition

choose to focus on RDF over other graph databases because of the standardized, lightweight, and non-proprietary format given by the Web Consortium (W3C) [7]. The effectiveness of this format is demonstrated in the evolution of the community-based collaboration cloud DBPedia [14], which successfully is being able to interlink a vast number of data sources.

2.3 Data representation on a conceptual level

Putting technical details aside, these presented data model trends might also be seen from a perspective of how digital representation accommodate to human conceptualization of knowledge. The field of *concept modeling* aims to describe information in an understandable manner. The idea of applying this domain on computer science has been around since the 80s [15]. There are different approaches on this, even from the philosophical domain [16], but a more pragmatic definition [17] states that a proper conceptual model should strive to:

1) Enhance an individual's understanding of the representative system.
2) Facilitate efficient conveyance of system details between stakeholders.
3) Provide a point of reference for system designers to extract system specifications.
4) Document the system for future reference and provide a means for collaboration.

If the gap between the conceptual model and the internal representation is bridged, the system will likely be more easy to develop and maintain, especially with regards to item 3. Moreover, it will enable personnel with a non-engineering background to gain a deeper understanding of the system. From this viewpoint a database representation that is easier to understand is preferred over one that is more cryptic.

Understandability is often thought of as a problem when using relational databases, where data has to be structured in predefined tables, apply cross referencing using foreign keys, and they usually have to be mapped from an intermediate stage of Entity Relationship Modeling (ERM). Thinking in tables might by a spontaneous choice when recording simple measurements (e.g. temperature over time), but when portraying more complex entities, the model often gets relatively complicated. An object-oriented view is often practical for programmers to embody concepts, but can not easily be transformed to a relational database. This discrepancy is usually referred to as the Object-Relational Impedance Mismatch [2].

NoSQL databases often mitigate these problems by allowing less fixed structures. In document stores, information is structured hierarchically through keys and values, allowing for more convenient storage representations. One implied restriction in this is that a data item cannot easily be referenced in a way that breaks the hierarchy. By instead using graph databases, data can be represented in an even more natural way, as claimed by proponents of this view such as Webber et. al in the textbook *Graph Databases* [18]. For example, consider a social network where users are friends with other users. Using documents, the friends of a given user are usually listed as references to identifier values, which then have to be looked up in the collection in order to retrieve the names of these friends. This look-up typically also has negative performance impact, in addition for being a bit bulky.

The renowned data analysist Manuel Lima describes what he considers an ongoing paradigm shift: "*Tree structures are no longer capable to accommodate the inherent complexities of the modern world*"[4]. He states that as Information Systems are getting more intelligent and need to process more intricate data, network structures are the most effective way conceive information.

On the other hand, there are many contexts where tree structures fit the problem quite well. Consider as an example a system storing cookbook information. A probable situation is that the data then consists of three hierarchical levels: The cookbooks, their recipes, and their ingredients. Neo4J, the most popular graph database today[5], points out in an instructional document[6] that any common data model can be represented as a type of graph. While this is true, there is naturally no prerequisite enforcement of preserving the characteristics of those structures, e.g. preventing cycles, neither convenient manners of storage and retrieval. The need for accommodating different data abstractions has spawned the incentive of hybrid representations, such as OrientDB[7]. We also saw the need for such an approach in this project, which we elaborate on in later sections.

[4]This talk can be watched at http://www.wearecognitive.com/videos/rsa-animate-the-power-of-networks (retreived July, 2015)

[5]As stated by `db-engines.com`

[6]This document was retreived in July 2015 from http://neo4j.com/docs/stable/tutorial-comparing-models.html

[7]http://orientdb.com/

2.4 Issues with Semantic Technology

There has been a fair amount of criticism directed towards Linked Data, which is mainly concerning the way it is being applied in the wild, rather than problems inherent to the technology itself [19]. The main points center around poor data integrity and weak reliability of information. This is often caused by insufficiently documented provenance information and an inability to resolve conflicts between statements due to a limited use of powerful reasoning languages such as OWL. Even when using these expressive languages, problems can arise due to incorrect interpretations of these statements and misconceptions of official documentation [20]. Furthermore, most datasets published today use different methods of structuring their schemata, which complicates the process of linking different datasets, where ideally the linking of Linked Data should be easy to perform.

It is important to be aware of these concerns, although the circumstances here were different than the typical use case of publishing data for public reuse. The data models developed in this project are primarily intended for internal use within the organization, where company requirements are paramount. There is also a remote possibility of publishing data, but was regarded only as a secondary objective. This means we valued pragmatic use case driven solutions higher than rigorous usage of strict vocabulary. We therefore payed close attention to the intentions and desires behind the functionality of the system.

2.5 Business context

Being a Dutch company, TTC have their headquarters in The Netherlands. Their operations typically reach out for people in Africa or South America. Projects can roughly be divided into these three categories:

- **Data collecting**. Services carried out in order to investigate public knowledge on certain topics. An example of this is a collaboration on a research project with the World Bank, where surveys were carried out in order to obtain information on the downstream outcomes of intra-regional trade commitments in Africa[8].
- **Providing of information or education**. Campaigns oriented towards equipping participants with knowledge. Here we consider projects such as when citizens of Peru receive messages about their personal credit status of their savings[9]. The information can also be of a more educative nature, like a project in Bolivia where farmers get advice on how to improve their crops[10].
- **A combination of the two**. In many cases, participants can get feedback from their answers in surveys, thereby educating themselves on important subjects in an interactive way. The HIV/AIDS prevention campaign in Congo is a good example of this[11].

[8]http://ttcmobile.com/portfolio/worldbank/

TTC have a variety of tools at their disposal for different purposes. The software they use to realize a campaign depends on the medium used to transmit information. These are the communication channels used today:

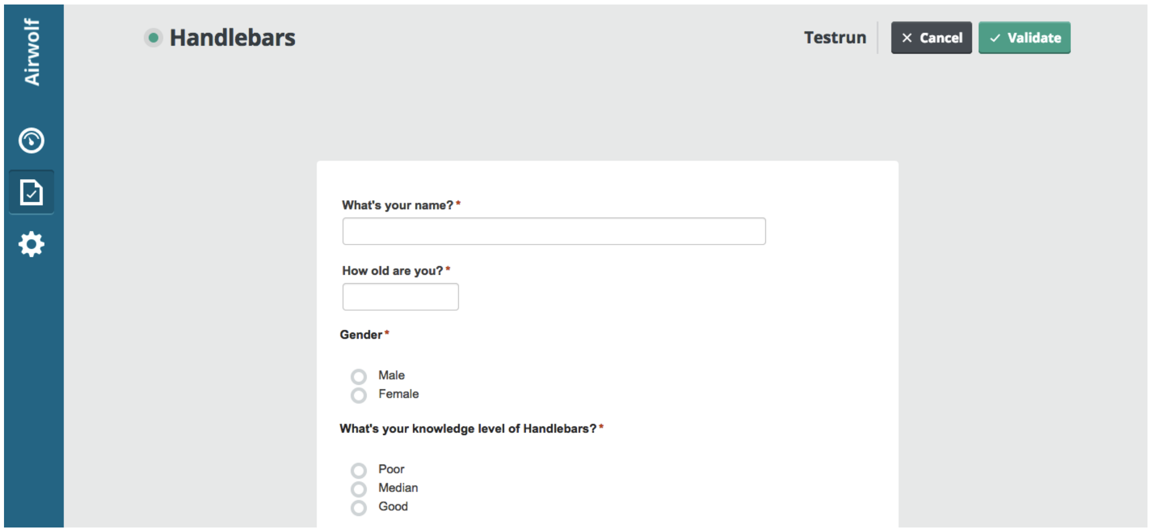
- **SMS** (Short Message Service).
- **USSD** (Unstructured Supplementary Service Data). A protocol for communicating through interactive menus or pop-ups. It is supported by most GSM cellular phones. In comparison to SMS, USSD has the advantage of being free of charge and a bit more flexible. The drawback is that the participant only see the current message and is unable to view any previous information exchange.
- **IVR** (Interactive Voice Response). Prerecorded voice-based systems controlled with the number pad of the participant. In other contexts, the IVR input can also be done by voice, but this is not supported by TTC due to the variety of languages used across different projects.
- **Call centers**. Outbound or inbound calls conducted by live personnel.

In the future TTC also expects possibilities of doing web based services to come about. With responsive web design these could be accessible by clients with smartphones, laptops, or desktop computers, as well as traditional feature phones.

The emphasis within this research was oriented around two tools used by TTC:

- **Vusion**. Creating SMS and USSD services with conditional logic in order to achieve

[9]http://ttcmobile.com/portfolio/financial-awareness-peru/
[10]https://vimeo.com/104697345
[11]http://ttcmobile.com/ttc-launches-large-scale-hivaids-prevention-campaign-in-congo-with-cordaid-vodacom/



What's your name? *

How old are you? *

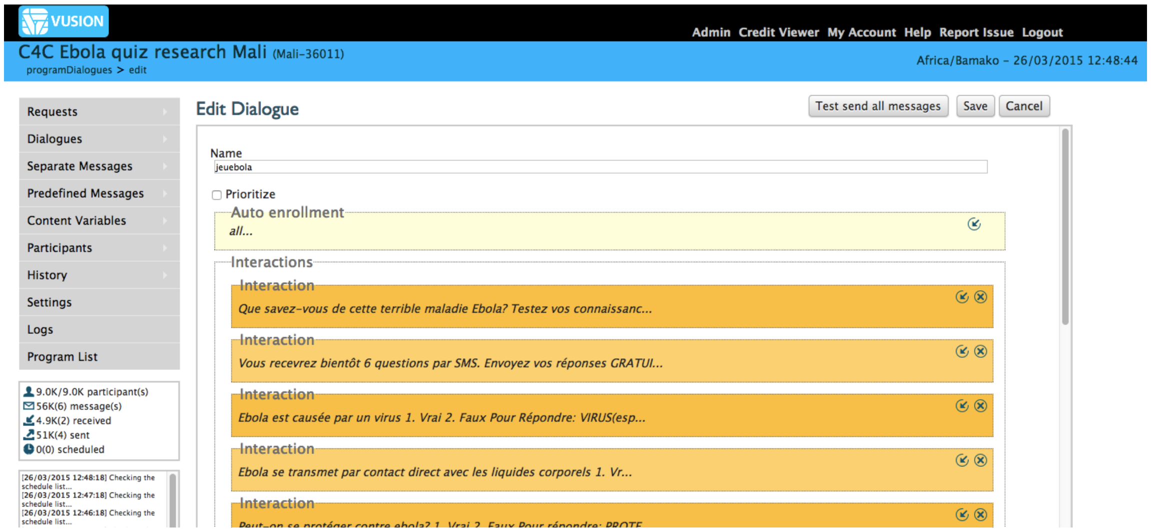
Gender *

Male
Female

What's your knowledge level of Handlebars? *

Poor
Median
Good

Fig. 3: *Airwolf*, an application for logging call center surveys.



C4C Ebola quiz research Mali (Mali-36011)
programDialogues > edit
Africa/Bamako - 26/03/2015 12:48:44

9.0K/9.0K participant(s)
56K(6) message(s)
4.9K(2) received
51K(4) sent
0(0) scheduled

```
[26/03/2015 12:48:18] Checking the schedule list...
[26/03/2015 12:47:18] Checking the schedule list...
[26/03/2015 12:46:18] Checking the schedule list...
```

Name: jeuebola

Prioritize

Auto enrollment: all...

Interactions

Interaction: Que savez-vous de cette terrible maladie Ebola? Testez vos connaissanc...

Interaction: Vous recevrez bientôt 6 questions par SMS. Envoyez vos réponses GRATUI...

Interaction: Ebola est causée par un virus 1. Vrai 2. Faux Pour Répondre: VIRUS(esp...

Interaction: Ebola se transmet par contact direct avec les liquides corporels 1. Vr...

Interaction: Peut-on se protéger contre ebola? 1. Vrai 2. Faux Pour répondre: PROTE

Fig. 4: *Vusion*, an application for creating SMS and USSD services

various goals. This is the most frequently used application within the company.

- **Airwolf**. This application acts as a logger for call center agents carrying out surveys. The surveys are defined by authorized moderators, where the agents then conduct these surveys by reading the questions to the participants. The answers of the participants are stored in the system.

The user interfaces of Airwolf and Vusion can be seen in Figure 3 and 4, respectively.

Aggregating information from internal tools like the ones presented serves two purposes. Firstly, applications can reuse data by performing look-ups against a central unit. Secondly, statistics can be extracted by humans through a user interface.

2.6 Case overview

Without an internal data broker, TTC are posed with the issue of maintaining multiple projects within different systems, not being able to connect any data from one context to the other. Different campaigns are ran on different platforms, causing interoperability problems which has lead to the implementation of the tool solving these issues. The transition from traditional phones (also known as *feature phones*) towards smart phones is slowly coming about among the target market, calling for even more additional services, which would likely be more intricate than the existing ones. In order to centralize all of this, the company made an internal API acting as a middle man between various applications; a portal being able to track diverse types of information and mediate between different sources. The system can be utilized by visualization techniques depicting comprehensible summaries of the immense quantity of information possessed by the company as a whole. The system supports a temporal dimension in the data, where modified fields are not overwritten but logged and preserved by the central system. In this way, the historical trends of user information are available. The system is called *Mash*.

The original version of Mash uses the document-oriented database MongoDB and was written in Node.js. Node.js is a run time environment for server-side applications in Javascript. Although Node.js is not widely used on an overall scale, it is more common among high-traffic web sites, and server-side Javascript usage is doubling every year [21]. The starting point for our project was from this version, and from there we explored what benefits RDF could bring to the table by modifying existing code.

MongoDB, the database used by Mash, stores documents in the BSON format [22], closely related to JSON. As long as a data entry conforms to this format and certain length restrictions, there are no taxonomic constraints on a given document. However, this flexibility comes with trade-offs in rigidity, and enforcing constraints can be tedious when done manually in application logic. Most applications will always handle entities containing the same fields on storage and retrieval, which often elevates an explicit schema definition to be a feature rather than a burden. This is why Mash, among many other applications, uses the schema-enforcing library Mongoose. Mongoose is very popular and download statistics suggests that about half of the developers using MongoDB also use Mongoose on top of it. These numbers come from the Node.js package manager *npm*, broadcasting current download counts [23][24].

Mash allows data exchange in the following manners:

- By using the graphical interface,
  - to make a CSV import or export,
  - or manually inputting/viewing participant information.
- Through GET requests (for applications communicating with the API).

3 Research question

What are the implications of using an RDF database when designing a hub for internally aggregating heterogeneous information?

We conducted a case-study by developing a proof of concept for TTC Mobile. This means creating an alternative branch in the development of the system *Mash*. We demonstrated the functionality of Linked Data by preserving the key requirements of the product as well as providing new features. In the process of executing this project we could investigate the following items:

– Appropriate data representations when aggregating datasets originating different applications.

– The technical and organizational obstacles explaining the limited popularity of RDF techniques.

– The tradeoffs in using an RDF database as opposed to a document store.

– Methods of versioning data allowing for viewing the history of previous states using RDF.

4 System Design

We saw a risk that handling triples through a SPARQL interface would result in a similar awkwardness other applications have when using plain MongoDB, with regards to consistency as discussed in Section 2.6. The network-based representation provides even more arbitrary structures than that of a document store, and while being of great benefit when linking heterogeneous data, it also poses challenges on maintaining internal consistency and ease of use. There was a need to be able to express schema definitions that could handle insertion, validation, and retrieval in a modular way similar to Mongoose.

We approached this problem by using the RDF store Cliopatria, to build up a higher level of abstraction than just triples. Cliopatria is based on SWI-Prolog [25], which allows for exchange of RDF data to be executed in a more powerful manner. The main incentive here is to be able to push data boilerplate transaction logic into the querying language. When doing this we can avoid an extensive mix of verbose queries and client code that often renders the software hard to maintain. This very reason was one of the main drivers behind Cliopatria as explicitly stated by the authors [26]. Cliopatria is also still compliant to the standards by providing a SPARQL interface, which could be used by any authorized application to query specific data.

4.1 Modeling

The model as stated in Mongoose compensates a need of storing information that can be useful for later compatibility with a specification of relevant elements. To this end, the Mongoose model utilizes a certain degree of non-schematic flexibility while still having a fixed characteristic. For the sake of completeness, the entire schema declaration is included in Figure 5. However, it is not necessary to go into details of the exact use of every field. Instead we focus on the flexibility aspect that comes into play with the very last attribute, `profile`. Within `profile` one can invent keys representing various aspects of the participant. Among these, three keys are already hard-coded to be handled by the user interface:

- `profile.dob`. The date of birth.
- `profile.location`. The current place of residence.
- `profile.gender`. The gender of the participant.

The values of `profile` are in turn objects by themselves containing two keys:

- `profile.<key>.value`. The value of the field.
- `profile.<key>.history`. An array of history entries, where each item is an object with the following attributes:
    - `value`. The value that was assigned.
    - `timestamp`. A specification of *when* this assignment took place.

With this solution, temporal provenance can be stored for every modification of a value in `profile`. However, a limitation of this method is the lacking possibility of expressing deeper hierarchies within a profile item. For example, it would not be easily achieved having a composite location attribute consisting of house number, city, and postal code. These fields would either have to be represented in a flat way, with entries like `location_postal_code`, `location_city` and `location_house_no`, or the data model has to be revised, and will likely get even more complicated. In the current implementation, every data input needs a restructuring to fit this format when communicating with Mash. Furthermore, there is no way of validating data types within `profile`, e.g. that `profile.dob` has to be a date.

We resolved this by being able to first store the data, and then define its model, rather than the other way around. By defining simple conversion procedures from various formats to RDF, we could preserve as much information as possible, and from there on decide what to make out of it. We thereby took advantage of the fact that every other data structure can be expressed as a graph. Drawing inspiration from the nomenclature used in the Linked Open Data Laundromat project [27], we made a distinction between *clean* and *dirty* data. Here, dirty data denote any piece of information converted to RDF using a simple set of rules (conversion strategies are elaborated on in Section 4.4). Still adopting the same terminology, we *wash* the dirty data into structures with fixed fields mimicking a document-like data format. By storing dirty data once, we can create custom washing heuristics in a versatile manner when making sense of the data. A bird's-eye view of the architecture illustrating this idea can be found in Figure 7.

This means that applications do not necessarily have to provide input using a preset data model. Rather, data can be transmitted in a native format, which is then handled by Mash accordingly. Data can thus be imported with raw database dumps directly from the application. This could be done either through API communication or through the user interface. As a proof of concept within this project, we only implemented the latter.

We present a Prolog way of modeling documents internally represented in RDF. Using this method clean data is expressed in fixed structures. The model definition can then be rewritten like in Figure 6.

These are key differences that distinguishes the new schema in Figure 6 from the previous schema in Figure 5:

- By means of the versioning solution later explained in Section 4.3, we can remove many fields indicating dates of creation and update. The `created` attribute of `participantOrigin` is kept, because

```
var participantEventsSchema = Schema({
  application: Schema.Types.ObjectId,
  created: Date,
  pid: Schema.Types.Mixed,
  pname: String,
  event: String
}, {_id: false});

var participantOriginSchema = Schema({
  application: Schema.Types.ObjectId,
  created: Date,
  provider : {
    /** Field name "type" means that the
        type has to be set explicitly **/
    type: {type: String},
    name: String,
    comment: String,
  },
});

var participantSchema = Schema({
  __schema_version: {type: Number,
    default: currentSchemaVersion},
  created: Date,
  updated: Date,
  status: String,
  author_app: Schema.Types.ObjectId,
  phone_number: {type: String, index: true,
    unique: true},
  country: String,
  origin : [participantOriginSchema],
  events: [participantEventsSchema],
  profile: {}
});
```

Fig. 5: The schema definition as stated in Node.js using Mongoose.

it represents the creation of the origin of the participant, rather than a creation of the data entry itself.

- We require every schema to define a version as a last argument of the `use_model/3` predicate, which later can be requested with the predicate `instance_version/2`. The version number ensures that a different declaration of the same model needs to increase this value, or else the database will throw an exception.
- The `profile` field is removed, since the additional data already is stored on beforehand. Instead, we place the fields `dob`, `location`, and `gender` in the root of the participant.

To illustrate one of the added values in simulating document storage in RDF, consider the query for fetching the full participant information in Mongoose for a certain phone number:

```
Participant.find({
        phone_number: "233242023017"
    })
```

```
use_model(_{
      application: resource("app"),
      pid: string,
      pname: string,
      event: string
   }, "participantEvents",1.0).

use_model(_{
      application: resource("app"),
      created: date,
      provider:_{
         type: string,
         name: string,
         comment: string
      }
   }, "participantOrigin",1.0).

use_model(_{
      status: string,
      author_app: resource("app"),
      phone_number: string|unique,
      country: string,
      origin: ["participantOrigin"],
      events: ["participantEvents"],
      dob: date,
      gender: string,
      location: string,
   }, "participant",1.0).
```

Fig. 6: The new schema definition as stated in Prolog using Cliopatria.

Now, imagine a SPARQL query of retrieving an entire participant structure, given the same original data model. Such a query would be quite verbose, since we explicitly have to mention every field and get into more complicated nesting when retrieving the arrays of `participantOrigin` and `participantEvents` with several `OPTIONAL` parameters, since there could be missing fields. This query would also return all permutations of array values, and in practice such a query would likely be split up into several smaller queries. A glimpse of how the SPARQL query would look like if put into one request can be found in Appendix A.

With the our Prolog library we can instead inquire for the specified participant using this syntax:

```
search_data_model(_{
      phone_number:"233242023017"
   },Data,"participant").
```

The participant information is then bound as a Prolog dict in `Data`. The underlying information is still stored as RDF, but the extra layer of abstraction helps a lot. Insertion of data is also done through a Prolog dict whose format is validated upon insertion. Schematic information compliant to the RDF schema vocabulary [28] is also stored, which is valueable if the data needs to be reused in another context. The generated schema definition for the participant has been attached in Appendix B.

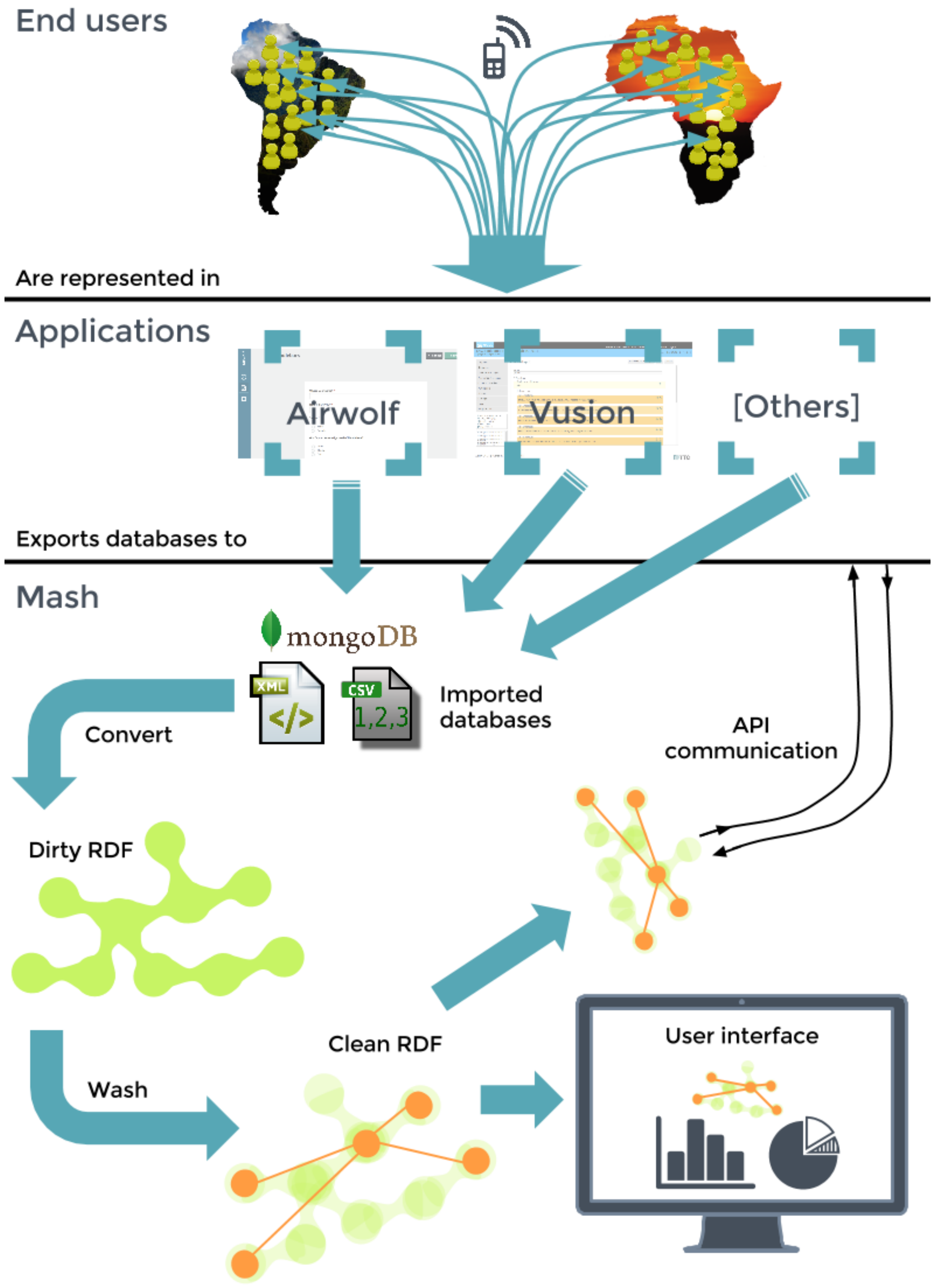


Fig. 7: An overview of our implementation of Mash and its interactions.

4.2 Washing

The triple-based representation is still highly useful when dealing with incoming dirty data. We demonstrate a simple example of washing data in Figure 9. Several washing goals like this can be added in a central file, which are then carried out upon execution of the predicate `wash_data/0`, which generates clean information. In the presented example, data is processed from a CSV file with columns in the order of phone number, gender and date of birth. We describe the few simple steps that are done in the provided code snippet:

1) Ask for the identifier of the application *Vusion*.
2) Get the graph belonging to a CSV import from this application.
3) Specify triple patterns to query the raw data using the predicate `rdf/3`.
4) Save the participant specifying extracted data accompanied by the origin(s) and a string naming the washing heuristic (in this case “Version 1.0”).

```
washing_goal((
   forall((
      properties_id_model(_{name:"Vusion"},
         App_id,"app"),
      graph_from_origin(Origin, _{
         application: App_id,
         provider:_{ type:"csv-import" }
      }),
      rdf(Line,rdf:type,raw:line,Origin),
      rdf(Line,raw:'Col0',
         literal(Phone_number),Origin),
      rdf(Line,raw:'Col1',
         literal(Gender),Origin),
      rdf(Line,raw:'Col2',
         literal(Dob),Origin)
   ),
      save_clean_participant(_{
         phone_number:Phone_number,
         gender:Gender,
         dob:Dob,
         country:Country},
         [Origin],"Version 1.0")
   )
)).
```

Fig. 9: An example of washing dirty data.

Participants will then appear in the user interface and can be retreived by the outside world. With these 12 lines of code, Mash is compatible with a new type of data structure. This is a modular approach that allows any data import to be accounted for, as long as a conversion to RDF is possible in the first place.

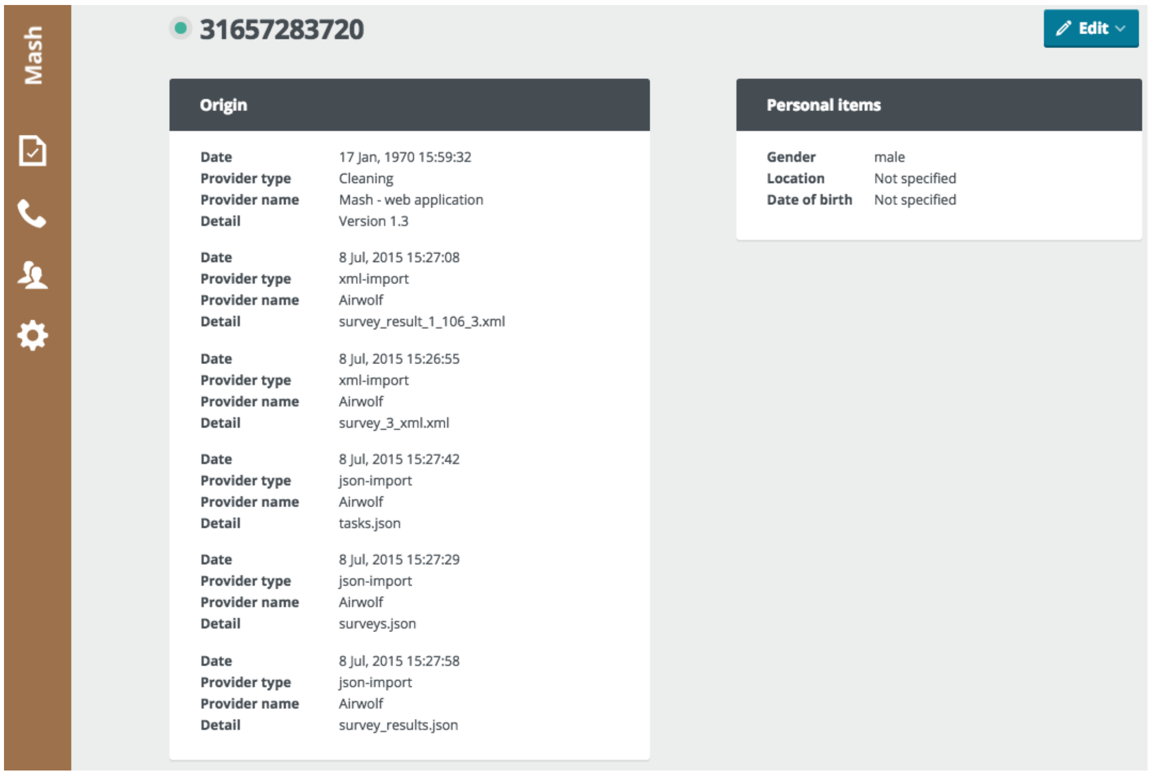
Of course, more complicated data structures will require more elaborate code. When importing data from the survey tool Airwolf, the record of one participant has to be extracted out of 5 different records. Ariwolf stores the survey definition as well as the result as XML files, which are pointed at through respective MongoDB collections. These 2 collections plus 2 XML files makes 4. Finally, there is another collection where call tasks are stored, containing the phone number. The washing of this data took about 100 lines of code. The look of a user resulting from this process can be seen in Figure 8. The user in the picture answered “male” on a question on his gender, which is recognized by the washing procedure.

4.3 Versioning

An important requirement stated that the change of an attribute should still preserve a record of the old value accompanied by an indication of a time span indicating validity of that value. The newest value is in most cases the only relevant item to the user, but sometimes there is also a need for going back in history to view changes. In RDF terms, we wanted every object to be unique for every subject-predicate pair. A new object for such a pair would be equivalent to an “overwrite” of an attribute of a given subject. The database needed to be optimized for retrieval of information that holds in the present. We could therefore afford making history retrieval more expensive since it is a less common operation.

With the advent of named graphs introduced in the RDF 1.1 standard [7], triples can be grouped together with a graph identifier of which statements then could be made about temporal validity. There is not a formal consensus nor official recommendation on exactly how to use a named graph [29], and the solution will be different depending on use cases and requirements. There has been different approaches in achieving versioning in RDF, especially from the domain of artificial intelligence. For example, a summary and performance comparison of 7 different methods are presented in [30], albeit none of them utilizing named graphs.

We made the assumption that named graphs should provide the effect of improved flexibility and performance that was the intention and rationale of including them in the standard. A straightforward example in doing this is to include every triple in a named graph with specified validity time stamps, such in an example on named graphs provided by the JSON-LD standard [31]. The problem with this approach is that retrieval of a given triple yielding multiple results needs to be resolved by comparing the time stamp of every resulting graph. Doing this for every triple inquiry can be a costly procedure. Instead, the solution of by Cassidy



31657283720

Origin

Date	17 Jan, 1970 15:59:32
Provider type	Cleaning
Provider name	Mash - web application
Detail	Version 1.3
Date	8 Jul, 2015 15:27:08
Provider type	xml-import
Provider name	Airwolf
Detail	survey_result_1_106_3.xml
Date	8 Jul, 2015 15:26:55
Provider type	xml-import
Provider name	Airwolf
Detail	survey_3_xml.xml
Date	8 Jul, 2015 15:27:42
Provider type	json-import
Provider name	Airwolf
Detail	tasks.json
Date	8 Jul, 2015 15:27:29
Provider type	json-import
Provider name	Airwolf
Detail	surveys.json
Date	8 Jul, 2015 15:27:58
Provider type	json-import
Provider name	Airwolf
Detail	survey_results.json

Personal items

Gender	male
Location	Not specified
Date of birth	Not specified

Fig. 8: The user interface of Mash when viewing a participant imported from different data sources.

and Ballantine [32] was preferred, where events of insertions or removals of triples are logged in named graphs with temporal properties. This solution was appealing since it does not degrade retrieval performance.

Cliopatria enabled us to superimpose Prolog predicates to operate on a higher level of abstraction. By starting from the predicate `rdf_assert/3`, we defined `rdf_assertv/3` to denote a versioned assert storing both the assertion and an assertion event. In the same way we defined `rdf_retractallv/3` on top of `rdf_retractall/3` to do a retraction and record this event. As part of the requirements, we let `rdf_assertv/3` replace the old object for a new object belonging to an already existing subject-predicate pair. To go back in history, the predicate `rdf_history/5` is used. A demonstration of this implementation is shown in Figure 10.

4.4 Conversion

Three conversion features were realized, in accordance to what was needed for the applications at hand. The encountered formats were XML,

```
?- rdf_assertv(mash:bob,mash:plays,
| mash:soccer).
true.

?- rdf_assertv(mash:bob,mash:plays,
| mash:tennis).
true.

?- rdf(mash:bob,mash:plays,What).
What = mash:tennis.

?- rdf_history(mash:bob,mash:plays,
| What,Start_stamp,End_stamp).
What = mash:soccer,
Start_stamp = 1435655448.993872,
End_stamp = 1435655461.178657;
What = mash:tennis,
Start_stamp = 1435655461.178657.
```

Fig. 10: A demonstration of versioning implemented using the solution by Cassidy and Ballantine [32] .

MongoDB, and CSV. All converters were rather general in their realizations, which means that they operate on data of which we do not know the structure of at conversion time. This differs from other situations where data is expected to contain certain fields, and the conversion process can be tailored to the context. Some useful insights were handy in the different conversion processes:

- **XML to RDF**. Drawing inspiration from the IATI project [13], we assigned an underlying element only consisting of text or one attribute to be reduced to that value rather than being a separate entity by itself.
- **CSV to RDF**. We used concepts such as row and column naming, casting data types, and resource promotion, developed in previous research [33].
- **MongoDB to RDF**. We utilized the fact that RDF has several serialization formats. This made the conversion simple, thanks to similarities between the format used by MongoDB documents called BSON (Binary JSON [22]) and the RDF compatible JSON-LD (JSON for Linked Data [31]).

For future compatibility, new conversion procedures may have to be added to the system. We do not consider this to be difficult tasks per se, as long as the converted structure is accounted for in the subsequent inference (washing) of new participants. A likely extension is to be able to convert data coming from relational databases. Such conversions has been investigated on behalf of W3C, and are described in [34].

5 Evaluation

5.1 *Time measurements*

Performance of storage and retrieval was measured comparatively between the two versions of Mash. Our implementation consistently performed worse than MongoDB, although there were not really any noticeable waiting times when using the graphical interface, except when uploading files. Looking at the numbers of the clock, MongoDB clearly outperforms Cliopatria, but when retrieving 20 participants as in one page of the user interface, retrieval times still averaged around "instant" speeds of 30 ms using RDF. This was observed consistently when the total number of participants were of different orders of magnitude. MongoDB was similar in this sense; read times tended to be dependent on the number of extracted entries rather than on the collection size.

We measured loading times of a single page, filtering patricipants to be male, born in the 80s, and have a phone number that contained the digit 8. These settings are equivalent to the filter specifications in the user interface shown in Figure 11. In this test there was a total of 1.052 participants.

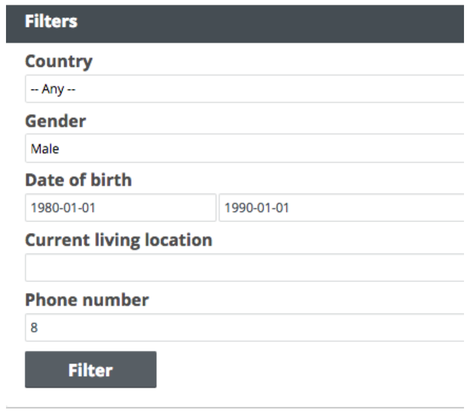

Filters
Country: -- Any --
Gender: Male
Date of birth: 1980-01-01 1990-01-01
Current living location:
Phone number: 8

Fig. 11: The filtering options used for time evaluation for "Detailed filter retrieve single page".

The waiting times were here quite short in both databases. In the second experiment we conducted a retrieval of all participants with a phone number containing an 8 followed by a 0 (hence searching for substring ”80”). In this trial the database contained 10.633 participants. The loading times now started to be noticeable in Cliopatria. Loading that all information from that many participants at the same time, however, is not really thought to be necessary.

The biggest difference was in writing data. One participant insertion costed on average 32.1 milliseconds in Cliopatria, while this took less than 1 millisecond in MongoDB. As described in Section 4.3, write operation speed was not prioritized since reads are anticipated to be more frequent than writes. Therefore the adopted versioning solution degrades write performance, while keeping the execution of reading at the same pace. Because of this we observed uploads of big files and producing clean data to be time extensive operations, and inferring 10.000 participants took about 10 minutes to complete. A summary of performance measures is to be found in Table 1.

Most likely, a triple-based storage of a documents is not the most ideal back-end representation, from a performance perspective. However, the authors believe that these speeds have a potential to be faster than current levels. Exploration of performance improvement was outside of the scope of this project, but a future extension would be to adopt results of previous work in optimizing Prolog queries [35].

Table 1: Measured times in milliseconds for two different technologies used in Mash. All times are averaged over 10 sequential runs. The exact commands used to test these tasks can be found in Appendix C

Database	Insert one participant item	Detailed filter retrieve single page	Basic filter retrieve all
MongoDB	<1	8.2	45.8
Cliopatria	32.1	29.8	427.7

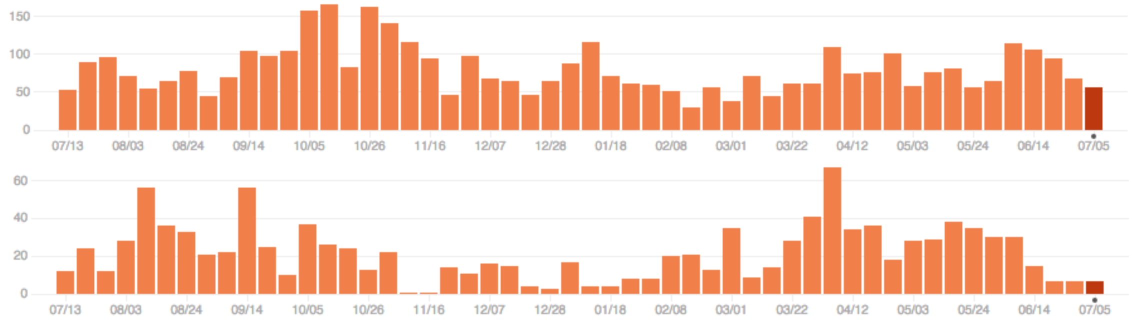


Fig. 12: The commit frequencies for MongoDB (above) and SWI-Prolog (below) between July 2013 and July 2015 (taken from `github.com`).

Database	# of contributors
MongoDB[12]	216
Sesame[13]	29
Blazegraph[14]	28
4Store[15]	20
Jena[16]	11
Cliopatria[17]	8
OpenLink Virtuoso[18]	7
Mulgara[19]	3

Table 3: The number of registered contributors to the repository of MongoDB compared to other known open source triple stores as of June 2015.

5.2 *Scalability in practice*

Just like other triple stores such as Jena [36], Hexastore [37], RDFox [38], and Bitmat [39], Cliopatria operates in main memory. This is a serious obstruction for a company like TTC hosting about 100 Gb of participant information. Such quantities would unlikely be able to fit into the RAM of any server. This means that the presented solutions would have to incorporate an element of cold storage, in order not to fill up memory. Such a feature is currently unavailable in Cliopatria. This would have to be solved either by extending Cliopatria or using another triple store.

[12]https://github.com/mongodb/mongo/tree/240596e01525374c6d6c0dabc44c107e2c0b900d

5.3 *Stability*

TTC prefers to work with non-proprietary open source projects, which applies both to Cliopatria, whose functionality heavily rests upon the open source project SWI-Prolog, as well as MongoDB. Quantifiable indicators of stability of different tools can be found by looking into statistics of repositories. The codebase of MongoDB has 216 listed contributors[12], whereas the repositories of SWI-Prolog[20] and Cliopatria[17] has 25 and 8 contributors, respectively. Although the number of contributors does not tell the whole story as to how stable a software is, the measure is relevant as to whether TTC can feel confident that they are using a well maintained technology stack. In terms of commit frequencies, MongoDB activity is about double than that of SWI-Prolog as can be seen in Figure 12.

[13]https://bitbucket.org/openrdf

[14]http://sourceforge.net/p/bigdata/git/ci/master/tree/

[15]https://github.com/garlik/4store/tree/1b3120295f03f5e2f459091ff62546acd3b19094

[16]https://github.com/apache/jena/tree/13855a6a384cda9f7b9bd2d78718a8c25cc0eb8e

[17]https://github.com/ClioPatria/ClioPatria/tree/ff02edb4ce969999946aba3e2a3a234e3e5bfa5f

[18]https://github.com/openlink/virtuoso-opensource/tree/5b95d916d4d6f38722f9518e1a71fdec0c384959

[19]https://github.com/quoll/mulgara/tree/82f8d32658075f8121ffa3fe224301a65754021c

[20]https://github.com/SWI-Prolog/swipl-devel/tree/f22780c532af35c19e05b3bdacc9781c5ba8d5ed

Table 2: Potential or existing features for different versions of Mash.

Database	Import data from applications	Export data for public reuse	Import external datasets
Cliopatria	Can be configured to handle new formats and the semantic meanings in these can be defined flexibly.	Can be published smoothly since it conforms to the RDF standard.	Can be implemented by using SPARQL queries against appropriate endpoint.
MongoDB	Can be configured to process different formats, but the storage model has a fixed characteristic.	Not easily achievable.	Not easily achievable.

Other open-source RDF databases are also unable to match the same level involvement as that of MongoDB. Table 3 shows an overview of contributor counts in various projects. All numbers were retrieved in July 2015.

5.4 Maintainability

A concern of TTC is the need to educate personnel in Prolog and Linked Data. Adopting the presented system would mean to abandon a technology that the team has years of experience in handling and instead stepping into unknown territory. Doing this will steepen the learning curve of maintaining the system.

On the other hand, once developers at TTC have these technologies under their belt, they should in principle be able to save significant amouts of time when connecting new applications to Mash. In our presented solution, the connection of a new tool would mean adding a few new washing procedures as described in Section 4.2. This means that Mash, the central unit, is easily able to be adjusted to new applications, rather than that these application would need to adapt their data representations to fit Mash, which would be the case in the former implementation. This is extra relevant if TTC would acquire a new tool of which they do not own the source code. In that case a database dump could be translated into usable data, which would be much harder with the current system.

5.5 Features and extensibility

Two unimplemented but potential features for the RDF version of Mash is to be able connect with published Linked Data, as well as taking part in the Linked Open Data Movement by contributing with new data. In addition, the storage technique explained in Section 4.1 allows us to save data and process it in two separate steps, rather than forcing every incoming piece of information to fit a predefined model. These capability differences are summarized in Table 2

6 Conclusion

Using semantic technology we constructed a system that seems promising and has added value in comparison to the document-based equivalent. Using RDF, our system is able to import and handle data whose characteristics are unknown at storage time. We could here utilize the lightweight triple format of RDF to store data in adhoc structures, which then could be organized into fixed schemata by flexibly being able to specify the semantic meaning of imported data. On top of this, we developed an RDF technique allowing for applications to query data with the same compact fluency as that of document-oriented databases.

Although our system was slower in terms of storage and retrieval times, some optimization work is expected to be able to lessen this performance gap. However, we saw that this use case poses quite substantial challenges in using Linked Data. The team at TTC Mobile is very cautious towards working with a representation and querying language they have no previous experience in using, and educating personnel is considered a costly procedure. Furthermore, this wariness is strengthened by the fact that open source triple stores are being maintained to a lesser extent than conventional databases, flagging for instability issues. Lastly, Cliopatria, which we based our solution on turned out to have limitations in terms of data quantity, putting a cap on scalability.

A modest interest in an open-source product generally degrades the frequency of maintenance, whilst an unsatisfactory maintained product also discourages practitioners from using it, creating a vicious catch 22 effect. SWI-Prolog does unfortunately not seem to be an exception to this phenomena. Due to a known bug[21] of a predicate, the search for an instance in the database took unnecessary time where the search process essentially needed to be done twice for every successful result.

We suspect that if RDF stores were given more public attention, the quality and performance

[21]This bug is documented at https://github.com/lamby/pkg-swi-prolog/blob/5d03a59447eb853e10578a1a6ccb4cd776101abe/library/aggregate.pl#L443

of these would increase to levels closer to the true potential of semantic technology, which by many is enthusiastically thought of as the *Web 3.0*. During the development of this application, an unknown bug in Clioaptria was discovered and fixed, and we had to develop a Node.js module for Prolog communication with Cliopatria. This module is now available for public download[22], and might illustrate that new applications using RDF could contribute in creating a snowball effect of improved quality in RDF-powered applications attracting even more practitioners.

[22]The module is available here: https://www.npmjs.com/package/prolog-db

Appendix A
A SPARQL Query for Retrieval of all Participant Data

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mash: <http://mash.texttochange.org/>
PREFIX versioning: <http://versioning.texttochange.org/>

SELECT ?participant_updated ?participant_created ?version ?status ?author_app
    ?country ?origin_application ?origin_created ?origin_provider_type
    ?origin_provider_name ?origin_provider_comment ?event_application
    ?event_created ?event_pid ?event_pname ?event_event ?profile_key
    ?profile_value ?profile_value
    WHERE {
      ?participant rdf:type mash:participant;
        mash:phone_number "233242023017".
    OPTIONAL { ?participant versioning:updated ?participant_updated }
    OPTIONAL { ?participant versioning:created ?participant_created }
    OPTIONAL { ?participant mash:version ?version. }
    OPTIONAL { ?participant mash:status ?status. }
    OPTIONAL { ?participant mash:author_app ?author_app. }
    OPTIONAL { ?participant mash:country ?country. }
    OPTIONAL {
        ?participant mash:origin ?origins.
        ?origins rdf:rest*/rdf:first ?origin.
        OPTIONAL {?origin mash:application ?origin_application.}
        OPTIONAL {?origin mash:created ?origin_created.}
        OPTIONAL {?origin mash:provider ?provider.
            OPTIONAL {?provider mash:type ?origin_proider_type.}
            OPTIONAL {?provider mash:name ?origin_provider_name.}
            OPTIONAL {?provider mash:comment ?origin_provider_comment;}
        }
      }
OPTIONAL { ?participant mash:events ?events.
      ?events rdf:rest*/rdf:first ?event.
    OPTIONAL {?event mash:application ?event_application.}
    OPTIONAL {?event mash:created ?event_created.}
    OPTIONAL {?event mash:pid ?event_pid.}
    OPTIONAL {?event mash:pname ?event_pname.}
    OPTIONAL {?event mash:event ?event_event.}
  }
  OPTIONAL {?participant mash:profile ?participant_profile.
            ?participant_profile ?profile_key ?profile_item.
            ?profile_item ?history ?profile_date_stamp;
            ?profile_item mash:value ?profile_value}
}
```

Appendix B
The Exported Participant Model in Turtle

```
mash:application
 rdfs:domain mash:participantEvents ,
     mash:participantOrigin ;
 rdfs:range mash:app .

mash:author_app
 rdfs:domain mash:participant ;
 rdfs:range mash:app .

mash:comment
 rdfs:domain mash:schema_1_ ;
 rdfs:range rdf:PlainLiteral .

mash:country
 rdfs:domain mash:participant ;
 rdfs:range rdf:PlainLiteral .

mash:created
 rdfs:domain mash:participantOrigin ;
 rdfs:range xsd:dateTime .

mash:dob
```

```
 rdfs:domain mash:participant ;
 rdfs:range xsd:dateTime .

mash:event
 rdfs:domain mash:participantEvents ;
 rdfs:range rdf:PlainLiteral .

mash:events
 rdfs:domain mash:participant ;
 rdfs:range mash:list_participantEvents .

mash:file_type
 rdfs:domain mash:participantOrigin ;
 rdfs:range rdf:PlainLiteral .

mash:gender
 rdfs:domain mash:participant ;
 rdfs:range rdf:PlainLiteral .

mash:list_participantEvents
 rdfs:subClassOf rdf:List .

mash:list_participantEvents_member
 rdfs:domain mash:list_participantEvents ;
 rdfs:range mash:participantEvents ;
 rdfs:subPropertyOf rdfs:ContainerMembershipProperty .

mash:list_participantOrigin
 rdfs:subClassOf rdf:List .

mash:list_participantOrigin_member
 rdfs:domain mash:list_participantOrigin ;
 rdfs:range mash:participantOrigin ;
 rdfs:subPropertyOf rdfs:ContainerMembershipProperty .

mash:location
 rdfs:domain mash:participant ;
 rdfs:range rdf:PlainLiteral .

mash:name
 rdfs:domain mash:schema_1_ ;
 rdfs:range rdf:PlainLiteral .

mash:origin
 rdfs:domain mash:participant ;
 rdfs:range mash:list_participantOrigin .

mash:participant
 doap:Version 1.0 ;
 rdf:type rdfs:Class.

mash:participantEvents
 doap:Version 1.0 .

mash:participantOrigin
 doap:Version 1.0 ;
 rdf:type rdfs:Class.

mash:phone_number
 rdfs:domain mash:participant ;
 rdfs:range rdf:PlainLiteral .

mash:pid
 rdfs:domain mash:participantEvents ;
 rdfs:range rdf:PlainLiteral .

mash:pname
 rdfs:domain mash:participantEvents ;
 rdfs:range rdf:PlainLiteral .
```

```
mash:provider
 rdfs:domain mash:participantOrigin ;
 rdfs:range mash:schema_1_ .

mash:raw
 rdfs:domain mash:participantOrigin ;
 rdfs:range rdf:Resouce .

mash:schema_1_
 doap:Version 1.0;
 rdf:type rdfs:Class.

mash:status
 rdfs:domain mash:participant ;
 rdfs:range rdf:PlainLiteral .

mash:type
 rdfs:domain mash:schema_1_ ;
 rdfs:range rdf:PlainLiteral .
```

APPENDIX C
QUERIES USED FOR TIME MEASUREMENTS

```
> db.setProfilingLevel(2)
> db.participants.insert({
    phone_number:"311218278123",
    status:"active",
    author_app:"553e407f97c8074950a3ef33",
    country:"NLD",
    origin: [{
        application:"553e407f97c8074950a3ef33",
        provider:
            {type:"manual",
            name:"Mash - web client"}
        }],
        __schema_version : 1,
        __v : 0
})
> db.system.profile.find({op: "insert"})
```

Fig. 13: The query for “Insert one participant item” – MongoDB

```
> db.participants.find({
    phone_number: { '$regex': '.*8.*' },
    'profile.gender.value': 'male',    '
    profile.dob.value': {
        '$gte': '1980-01-01',
        '$lte': '1990-01-01'
    }
}).limit(20).explain()
```

Fig. 14: The query for “Detailed filter retrieve single page” – MongoDB

```
> db.participants.find({
    phone_number: { '$regex': '.*80.*' }
}).explain()
```

Fig. 15: The query for “Basic filter retrieve all” – MongoDB

```
-? time(
   assert_new_instance(New_id,_{
      phone_number:"312312312132",
      status:"active",
      author_app:"553e407f97c8074950a3ef33",
      country:"NLD",
      origin:[_{
         application:"553e407f97c8074950a3ef33",
         provider: _{
            type:"manual",
            name: "Mash - web client"
         }
      }],
      events:[],
      gender:"male",
      dob:"1990-07-20",
      location:""
   },"participant")
).
```

Fig. 16: The query for “Insert one participant item” – Cliopatria

```
-? time(
   findall(
      Data,
      limit(20, (
            Gender="male",
            freeze(Dob,Dob > 315532800000),
            freeze(Dob,Dob < 631152000000),
            freeze(Phone_number,(
               sub_string(Phone_number,_,_,_,"8"), !)),
            search_id_data_model(_{
               gender:Gender,
               dob:Dob,
               phone_number:Phone_number
            },
            _,Data,"participant",[offset=0])
         )
      ),
      All_data
   )
).
```

Fig. 17: The query for “Detailed filter retreive one page of information” – Cliopatria

```
-? time(
   findall(
      Data,(
         freeze(
            Phone_number,(sub_string(Phone_number,_,_,_,"80"), !)
         ),
         search_id_data_model(_{
            phone_number:Phone_number
         },
         _,
         Data,"participant",[offset=0])
      ),
   All_data)
).
```

Fig. 18: The query for “Basic filter retrieve all” – Cliopatria

References

[1] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[2] C. Ireland, D. Bowers, M. Newton, and K. Waugh, "A classification of object-relational impedance mismatch," in *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA'09. First International Conference on*. IEEE, 2009, pp. 36–43.

[3] A. Nayak, A. Poriya, and D. Poojary, "Type of nosql databases and its comparison with relational databases," *International Journal of Applied Information Systems*, vol. 5, no. 4, pp. 16–19, 2013.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[6] I. Mapanga and P. Kadebu, "Database management systems: A nosql analysis," *Interna-tional Journal of Modern Communication Technologies & Research (IJMCTR)*, vol. 1, pp. 12–18, 2013.

[7] D. Beckett. (2014) Rdf/xml syntax specification (revised) - w3c recommendation. [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar/

[8] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data: Evolving the web into a global data space," *Synthesis Lectures on the Semantic Web: Theory and Technology*, 2011.

[9] C. Bizer, R. Cyganiak, and T. Heath. (2008) How to publish linked data on the web. [Online]. Available: http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/

[10] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[11] R. Heeks, "Ict4d 2.0: The next phase of applying ict for international development," *IEEE Computer*, vol. 41, no. 6, pp. 26–33, 2008.

[12] T. Davies and D. Edwards, "Emerging implications of open and linked data for knowledge sharing in development," *IDS Bulletin*, vol. 43, no. 5, pp. 117–127, 2012.

[13] K. S. Brandt and V. de Boer, "Linked data for the international aid transparency initiative," *Journal on Data Semantics, ISSN*, pp. 1861–2032, 2014.

[14] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[15] J. Mylopoulos, "Conceptual modelling and telos 1," 2008.

[16] R. Hirschheim, H. K. Klein, and K. Lyytinen, *Information systems development and data modeling: conceptual and philosophical foundations*. Cambridge University Press, 1995, vol. 9.

[17] A. S. C.H. Kung, "Activity modeling and behavior modeling," in *Proceedings of the IFIP WG 8.1 working conference on comparative review of information systems design methodologies: improving the practice*, North-Holland, Amsterdam, 1986, pp. 145–171.

[18] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Inc., 2013.

[19] P. Jain, P. Hitzler, and P. Yeh, "Linked data is merely more data," *Linked AI: AAAI Spring Symposium "Linked Data Meets Artificial Intelligence"*, 2010.

[20] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson, "When owl:sameas isn't the same: An analysis of identity in linked data," 2010.

[21] W3Techs - Web Technology Surveys. (30 Jun 2015) Usage of javascript libraries for websites. [Online]. Available: http://w3techs.com/technologies/details/pl-js/all/all

[22] Bson – binary json. [Online]. Available: http://bsonspec.org/spec.html

[23] P. Vorbach. (30 Jun 2015) Download statistics for package "mongoose". [Online]. Available: http://npm-stat.com/charts.html?package=mongoose&author=&from=2013-06-30&to=2015-06-30

[24] ——. (30 Jun 2015) Download statistics for package "mongodb". [Online]. Available: http://npm-stat.com/charts.html?package=mongodb&author=&from=2013-06-30&to=2015-06-30

[25] J. Wielemaker, G. Schreiber, and B. Wielinga, "Prolog-based infrastructure for rdf: scalability and performance," in *The Semantic Web-ISWC 2003*. Springer, 2003, pp. 644–658.

[26] J. Wielemaker, W. Beek, M. Hildebrand, and J. van Ossenbruggen, "Cliopatria: A logical programming infrastructure for the semantic web."

[27] W. Beek, L. Rietveld, H. R. Bazoobandi, J. Wielemaker, and S. Schlobach, "Lod laundromat: a uniform way of publishing other people's dirty data," in *The Semantic Web–ISWC 2014*. Springer, 2014, pp. 213–228.

[28] D. Beckett. (2004) Rdf/xml syntax specification (revised) - w3c recommendation. [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar/

[29] A. Zimmermann. (2014) Rdf/xml syntax specification (revised) - w3c recommendation. [Online]. Available: http://www.w3.org/TR/rdf11-datasets/#named-graphs-have-no-meaning

[30] H.-U. Krieger, "A detailed comparison of seven approaches for the annotation of time-dependent factual knowledge in rdf and owl," in *Proceedings 10th Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation*, 2014, p. 1.

[31] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström. (2014) Json-ld 1.0. [Online]. Available: http://www.w3.org/TR/json-ld/

[32] S. Cassidy and J. Ballantine, "Version control for rdf triple stores." *ICSOFT (ISDM/EHST/DC)*, vol. 7, pp. 5–12, 2007.

[33] T. Lebo and G. T. Williams, "Converting governmental datasets into linked data," in *Proceedings of the 6th International Conference on Semantic Systems*, New York, USA, 2010, pp. 38:1–38:3.

[34] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat, "A survey of current approaches for mapping of relational databases to rdf," *W3C RDB2RDF Incubator Group Report*, pp. 113–130, 2009.

[35] J. Wielemaker, "An optimised semantic web query language implementation in prolog," in *Logic Programming*. Springer, 2005, pp. 128–142.

[36] B. McBride, “Jena: A semantic web toolkit,” *IEEE Internet Computing journal*, no. 6, p. 19, 2002.

[37] “Hexastore: sextuple indexing for semantic web data management,” *Proceedings of the VLDB Endowment*, 2008.

[38] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu, “Parallel materialisation of datalog programs in centralised, main-memory rdf systems,” in *Proc. AAAI*, 2014, pp. 129–137.

[39] M. Atre, J. Srinivasan, and J. A. Hendler, “Bitmat: A main memory rdf triple store,” *Proceedings of the VLDB Endowment*, 2009.