A Comparison of Mobile Rule Engines for Reasoning on Semantic Web Based Health Data

William Van Woensel, Newres Al Haider, Patrice C. Roy, Ahmad Marwan Ahmad and Syed SR Abidi
NICHE Research Group, Faculty of Computer Science, Dalhousie University
Halifax, Canada

***Abstract*—Semantic Web technology is used extensively in the health domain, due to its ability to specify expressive, domain-specific data, as well as its capacity to facilitate data integration between heterogeneous, health-related sources. In the health domain, mobile devices are an essential part of patient self-management approaches, where local clinical decision support is applied to ensure that patients receive timely clinical findings. Currently, increases in mobile device capabilities have enabled the deployment of Semantic Web technologies on mobile platforms, enabling the consumption of rich, semantically described health data. To make this semantic health data available to local decision support as well, Semantic Web reasoning should be deployed on mobile platforms. However, there is currently a lack of software solutions and performance analysis of mobile, Semantic Web reasoning engines. This paper presents and compares the mobile benchmarks of 4 reasoning engines, applied on a dataset and rule-set for patients with Atrial Fibrillation (AF). In particular, these benchmarks investigate the scalability of the mobile reasoning processes, and study reasoning performance for different process flows in decision support. For the purpose of these benchmarks, we extended a number of existing rule engines and RDF stores with Semantic Web reasoning capabilities.**

I. Introduction

Semantic Web technologies are extensively used in the health domain, as they provide a formal model to represent healthcare knowledge. In doing so, these technologies enable expressive reasoning over health data, allowing clinical decision support to be realized. In addition, the use of specialized medical ontologies facilitate data integration between heterogeneous data sources that characterize a typical health scenario; including disparate sources on drugs, patients and diseases [1]. An important aspect of Semantic Web frameworks is the provision of model-based reasoning capabilities, which allow new facts to be inferred based on known rules. In this regard, there exist a large number of reasoning engines, using an assortment of reasoning techniques, that are able to reason over knowledge represented as RDF triples. These reasoners range from description logic reasoners based on OWL semantics [2], to general-purpose reasoners using various generic rule languages such as SWRL [3] and SPIN [4]. In general, rule-based reasoning techniques, used for decision support applications, allow a clear separation between domain knowledge and application logic. As a result, domain knowledge can be easily edited, updated and extended without the need to disrupt the underlying decision support application.

Typically, knowledge-centric decision support systems, incorporating ontology-based knowledge representation and reasoning capabilities, have been implemented as desktop or server applications. With the emergence of mobile devices with adequate processing capabilities and memory storage, there is a case for developing mobile decision support systems, where the knowledge and application logic is contained within the mobile device. Reflecting these increased capabilities, we note that Semantic Web technology has already been deployed on mobile platforms, with a number of RDF data stores allowing the querying and manipulation of RDF data. These include RDF On the Go [5], AndroJena[1], i-MoCo [6], and systems such as MobiSem [7]. In order to develop mobile decision support systems, consuming semantically described health data, the next step is to deploy Semantic Web reasoning capabilities on mobile devices as well. In general, the ability to perform reasoning locally on the mobile device, as opposed to relying on remote services, is important for a number of reasons. Relevant to the health domain, mobile devices are now capable of collecting a range of health data from individuals (such as blood pressure, heart rate, etc). Combined with localized reasoning and decision support, this allows the generation of suitable and timely clinical alerts, action plans and recommendations, even in cases where connectivity is lacking. Secondly, given the myriad of data that can be collected about mobile users, privacy issues can play a role. A patient could (rightly) be uncomfortable with sharing certain medical data outside the mobile device. By deploying reasoning processes locally, no privacy-sensitive data needs to be wirelessly communicated. Patient self-management approaches in the health domain [8], [9], [10], as well as context-aware systems [11], [12], have already recognized the potential of deploying reasoning processes directly on a mobile device.

Regardless of their potential, work on mobile, general-purpose Semantic Web reasoning engines is currently lacking, both regarding freely available software solutions and performance analysis. We currently have knowledge of only one system suiting this description for the Android platform, namely AndroJena. At the same time however, a number of Javascript RDF stores and reasoning engines have become available, which can be deployed on mobile platforms using cross-platform development tools such as Phonegap[2]. Although promising, some of these RDF stores currently lack reasoning support, while others do not support Semantic Web data and rules. Furthermore, their performance on mobile devices and under a realistic, health domain scenario, needs to be

[1] http://code.google.com/p/androjena/
[2] http://phonegap.com/

investigated. Various factors, such as the type of reasoning and the data- and ruleset scale, can greatly influence reasoning efficiency. Adding to this is the fact that, although the capabilities of mobile devices have grown considerably in recent years, they are still very limited when compared to a server or even a contemporary desktop system.

In this paper, we present a comparison of four currently available mobile reasoning engines, applied on Semantic Web data and rules from the health domain. We study a number of performance criteria, including data and rule loading times, reasoning times and memory consumption, during reasoning processes typical for mobile clinical decision support. We evaluate AndroJena, a native Android framework, and RDFQuery[3] [13], RDFStore-JS[4] [14] and Nools[5], which are Javascript systems. To suit our purposes of mobile, Semantic Web-based reasoning, we extended RDFStore-JS with (naïve) reasoning support, while a Semantic Web layer was built on top of the Nools engine. In order to determine the scalability of mobile Semantic Web reasoning, we performed this benchmark with increasing amounts of data. The benchmark dataset and inference rules are derived from the Integrated Management Program Advancing Community Treatment of Atrial Fibrillation (IMPACT-AF) project, for providing a Clinical Decision Support System (CDSS) for AF patients [15]. We note that this ruleset has not been optimized to suit the inner mechanisms of the different reasoning engines (e.g., RETE [16]), nor the particular structure and composition of the benchmark dataset. Systematically investigating the effect of the various existing optimization techniques, while a very interesting research question, is beyond the scope of this paper.

The rest of this paper is structured as follows : Section II briefly describes the area of Semantic Web reasoning, and summarizes the evaluated reasoning engines. Next in Section III, we describe our approach for our benchmark comparison, including the benchmark domain, deployment scenario and concrete setup. Afterwards, Section IV illustrates the benchmark results, while Section V discusses and reflects on these results. In Section VI, we shortly mention other related benchmarking approaches. Finally, we present conclusions and future work in Section VII.

II. General Background

A key component of the Semantic Web is the Resource Description Framework (RDF) [17], which represents knowledge as a set of facts in the form of subject, predicate, object (s, p, o) triples. The Web Ontology Language (OWL) [18], which has a formal grounding in Description Logics (DL), allows to define restrictions on RDF datasets depending on the underlying domain (e.g., healthcare); for instance, including subtype, subproperty and cardinality constraints. Given these restrictions, reasoners are able to make useful inferences on the RDF data. However, in many domains, including the clinical domain, more extensive and custom reasoning is generally required to operationalize all relevant knowledge [19]. In particular, knowledge often needs to be encoded as a set of general IF-THEN rules. Reflecting this need, various rule languages have been proposed to supplement the OWL semantics in the Semantic Web, including SWRL and SPIN.

As mentioned, freely and publicly available mobile Semantic Web reasoning engines, supporting rule-based reasoning, are currently lacking. Therefore, we elected to include Javascript RDF stores and reasoning engines in our comparison, which can be deployed in native mobile apps using cross-platform development tools such as Phonegap. Below, we elaborate on the four benchmarked systems.

A. Reasoning Engines

1) AndroJena: Apache Jena[6] is a well-known Java framework for working with Semantic Web data. AndroJena is a ported version of this framework to the Android platform. RDF data can be directly loaded from a local or remote source into an RDF store called a *Model*, supporting a wide range of RDF syntaxes (e.g., RDF/XML).

Regarding reasoning, AndroJena supplies an RDFS, OWL and rule-based reasoner. The latter provides both forward and backward chaining, respectively based on the standard RETE algorithm [16] and Logic Programming (LP). In addition, the reasoning engine supports a hybrid execution model, where both mechanisms are employed in conjunction[7]. Rules are specified using their Domain-Specific Language (DSL), which resembles a SPARQL-like rule syntax, and are parsed and passed to a reasoner object. By applying this reasoner on a populated Model, an *InfModel* is created, which supplies query access to the inferred RDF statements. Afterwards, new facts can be added to this InfModel; after calling the *rebind* method, the reasoning step will be re-applied.

2) RDFQuery: RDFQuery is an RDF plugin for the well-known jQuery[8] JavaScript library. RDFQuery makes an effort to bridge the gap between the Semantic Web and the regular Web, by allowing developers to directly query RDF (e.g., injected via RDFa [20]) gleaned from HTML pages. RDF datastores can also be populated directly with RDF triples.

In addition to querying, RDFQuery also supports rule-based reasoning. Conditions in these rules comprise triple patterns and general-purpose filters, implemented by JavaScript functions. Filter functions are called for each currently matching data item; based on their return value, items are kept or discarded. The reasoning algorithm is "naïve", meaning rules are executed in turn until no more new results occur[9].

3) RDFStore-JS: RDFStore-JS is a JavaScript RDF graph store, and can be either deployed in the browser or as a module in Node.js[10], a server-side JavaScript environment. Comparable to AndroJena (see Section II-A1), triples can be loaded into an RDF store from a local or remote data source, supporting multiple RDF syntaxes.

[3] https://code.google.com/p/rdfquery/wiki/RdfPlugin
[4] http://github.com/antoniogarrote/rdfstore-js
[5] https://github.com/C2FO/nools
[6] https://jena.apache.org/
[7] See http://jena.apache.org/documentation/inference/#rules
[8] http://jquery.com
[9] The engine had to be extended to resolve variables in the rule result.
[10] http://nodejs.org/

Regarding querying, RDFStore-JS supports SPARQL 1.0 together with parts of the SPARQL 1.1 specification. Since RDFStore-JS does not natively support rule-based reasoning, we extended the system with a reasoning mechanism that accepts rules as SPARQL 1.1 INSERT queries. In these queries, the WHERE clause represents the rule condition and the INSERT clause the rule result. This mechanism is naïve, executing each rule in turn until no more new results are inferred (cfr. RDFQuery).

4) Nools: Nools is a RETE-based rule engine, written in JavaScript. Analogous to RDFStore-JS, this system can be deployed both on Node.js as well as in the browser.

As opposed to the other evaluated systems, Nools does not natively support RDF. Consequently, we developed a Semantic Web layer on top of Nools, which accepts RDF data (N-Triples format) and a SPARQL-like rule syntax (i.e., a simplified version of the SPIN syntax). In the explanation below, we indicate the extensions to Nools required to realize this layer.

In contrast to the two other evaluated JavaScript systems, Nools presents a fully-fledged reasoning engine, supporting a non-naïve reasoning algorithm (RETE). The engine is also used differently when performing reasoning. In case of Nools, a developer first supplies the rules in the form of a *flow*, formulating them using their DSL. Custom data types can be included in the rule definitions (e.g., data type *Message* when dealing with messages), which can then be instantiated in the incoming dataset and referenced in the defined rules. To realize our Semantic Web extension, we defined custom RDFStatement, RDFResource, RDFProperty and RDFLiteral data types. Incoming rules are automatically converted to rules in the Nools DSL referencing these data types.

A *session* is an instance of the flow, containing the RETE working memory in which new facts are asserted. After creating and compiling the rule *flow*, the dataset is asserted in the *session*, whereby the asserted data can be matched to the defined rules. In our Semantic Web layer, the RDF dataset is converted to instances of the aforementioned custom data types, and asserted as facts in the session.

5) Summary: Below, we classify each system with regards to our relevant criteria: mobility, reasoning support, and support for Semantic Web data and rules. In addition, we position the systems relative to our benchmark ruleset and dataset.

Mobility: AndroJena is the only engine specifically meant for mobile deployment. At the same time however, it is not clear to what extent reasoning was optimized for mobile devices, or simply ported from the desktop- (and server-) oriented Apache Jena. The other three engines were developed for use on either the server-side or in a desktop browser, and were deployed on mobile devices via a cross-platform development tool. As such, it is unclear for any of these systems how well they will perform on a mobile platform.

Reasoning support: Nools, AndroJena and RDFQuery have built-in reasoning support, whereby only AndroJena and Nools feature non-naïve reasoning mechanisms. We extended RDFStore-JS with naïve reasoning support for the purposes of our benchmark. We note that our real-world dataset and ruleset (described in Section III-A) does not require chaining or advanced conflict resolution, aspects of reasoning that would surely be better handled by the fully-fledged Nools and AndroJena reasoning engines.

Semantic Web support: Nools was the only reasoning engine without explicit Semantic Web support. To remedy this, we built a Semantic Web layer on top of the system, leveraging their supplied extension support (e.g., custom data types).

III. Approach

In this section we give a description of the approach followed in our benchmark, including the domain and deployment scenario as well as the practical setup.

A. Benchmark Domain

The benchmark data- and ruleset are based on ongoing work on the Integrated Management Program Advancing Community Treatment of Atrial Fibrillation (IMPACT-AF) project. IMPACT-AF aims to provide Web and mobile-based clinical decision support tools for primary care providers and patients, with the goal of better managing Atrial Fibrillation (AF).

A patient's AF dataset comprises health factors related to AF, including clinically relevant personal info (e.g., age, gender) and health measurements (e.g., blood pressure), for instance dynamically collected from wireless and wearable sensors [21]. In addition, it contains AF-specific symptoms and the International Normalized Ratio (INR).

In addition to measurements, the dataset also contains personalized upper and lower limits for the different kinds of clinical facts, indicating problematic situations for the particular patient. For every collected fact, the timestamp and value are recorded. In Code 1, we give an example RDF snippet encoding a clinical fact (namespaces omitted for brevity), accompanied by its patient-specific limits.

```
impact:Patient0 impact:hasINR impact:INR0 ;
    impact:hasLowerLimitTargetINR impact:LowerINR ;
    impact:hasUpperLimitTargetINR impact:UpperINR .
impact:INR0 impact:hasValue "4.34" .
impact:LowerINR impact:hasValue "2.5" .
impact:UpperINR impact:hasValue "3.5" .
```

Code 1: Example clinical fact from AF dataset.

The AF ruleset is derived from guidelines for the treatment of Atrial Fibrillation, given by the Canadian Cardiovascular Society [22] and European Society of Cardiology [23].

The IMPACT-AF Clinical Decision Support System (CDSS) comprises two reasoning components: a mobile, client-side component, which yields increased responsiveness and is utilized to execute time-critical, lightweight reasoning tasks; and a server-side component, performing heavyweight reasoning (involving more rules and data). The rule- and datasets used in this benchmark are deployed on the mobile CDSS component.

B. Deployment Scenario

In our experience, a mobile app can apply reasoning in two general process flows to realize clinical decision support:

Frequent Reasoning: In the first process flow, the mobile app stores collected health measurements and observations, collectively called clinical facts, in a data store. To infer new clinical conclusions, frequent reasoning is periodically applied to the datastore, comprising all collected data together with the patient's profile. Concretely, this entails loading a reasoning engine with the entire datastore each time a certain timespan has elapsed, and executing the relevant ruleset.

Incremental Reasoning: In the second process flow, the mobile app implements clinical decision support by applying reasoning each time a new clinical fact is entered. In this case, the reasoning engine is loaded with an initial baseline dataset, containing the patient's clinical profile and historical (e.g., previously entered) clinical facts. Afterwards, the engine is kept in memory, whereby new facts are dynamically added to the reasoning engine. Each time, reasoning is re-applied to infer new clinical conclusions[11].

Figure 1 shows Frequent Reasoning (FR) and Incremental Reasoning (IR) for RDFQuery, RDFStore-JS and AndroJena, where data is first loaded into the engine and rules are subsequently executed. Figure 2 shows the same process flows for Nools, where rules are first loaded into the engine followed by the dataset. For both diagrams, Frequent Reasoning entails going through the entire diagram each time a particular timespan has elapsed (time event), whereby *Dataset* stands for all relevant data. For Incremental reasoning, the system traverses the diagram from start to finish at startup time (whereby *Dataset* represents the baseline dataset). The system then proceeds from the indicated place (see receive-signal event) each time a new fact is received.

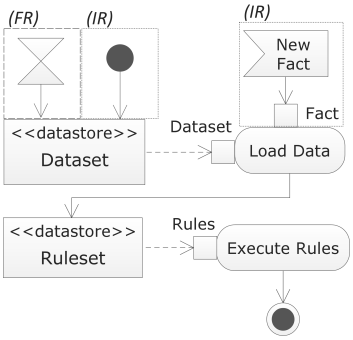

Fig. 1: Illustration of two CDSS process flows (RDFQuery, RDFStore-JS, AndroJena)

It is clear that the former flow reduces responsiveness to new clinical facts, while also incurring a larger performance overhead since the entire dataset needs to be continuously re-loaded. At the same, the latter process flow results in a larger consistent memory overhead, since the reasoning engine is continuously kept in memory. We note that other process flows are also possible, combining characteristics from the two described above; e.g., a hybrid scenario, whereby responsiveness is ensured by re-loading the dataset and executing the ruleset each time a new clinical fact is entered; sacrificing performance to reduce consistent memory load. Our goal in this paper is not to identify the most suitable process flow to realize mobile clinical decision support. Instead, we aim to evaluate (in our experience) two useful process flows, and compare the performance of the evaluated engines in each flow.

[11] An algorithm is proposed in [24] to optimize this kind of reasoning.

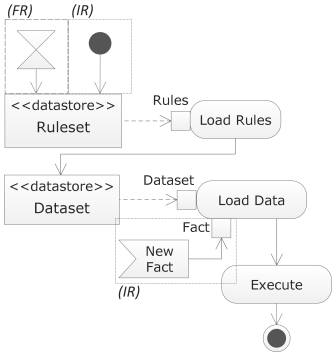

Fig. 2: Illustration of two CDSS process flows (Nools)

C. Benchmark Setup

Below, we elaborate on the practical setup of the benchmarks, including the concrete deployment of the reasoning engines, studied performance metrics, utilized dataset and ruleset, measurement methodology and benchmark hardware.

1) Concrete deployment: In order to deploy the JavaScript reasoning engines (see section II) to a mobile app, we used the Phonegap cross-platform development tool. Phonegap allows developers to run local, embedded Websites in native mobile apps, which are built using standardized Web technologies such as HTML, CSS and JavaScript. Importantly, Phonegap also allows local JavaScript code to access device features via plugins, such as the device camera, native storage, contacts and notifications. The AndroJena reasoning engine was directly deployed to a native Android app.

2) Performance metrics: In our benchmark, we study and compare the following performance metrics, due to their relevance to the considered process flows:

- *Data and rule loading times*: Time needed to load data (and rules, if necessary) into the reasoning engine.
- *Reasoning times*: Time needed to execute the rules on the dataset and infer new data.
- *Memory consumption*: Memory consumed by the reasoning engine.

3) Dataset and ruleset: For our benchmark, we use an Atrial Fibrillation (AF) decision support ruleset encompassing a total of 10 rules. For each rule, the rule head refers to the latest clinical fact of a certain type (e.g., blood pressure, INR) and checks whether its value is out of bounds, given the related patient-specific limit. If so, a clinical conclusion is inferred in the rule body, indicating the severity of the situation, type of conclusion, label and identifier of the triggered rule. No chaining occurs in the ruleset.

As mentioned in the introduction, we note the ruleset was not optimized to suit the employed reasoning mechanisms (e.g., RETE, Logic Programming) or dataset composition. A number of relevant techniques can be utilized for this purpose, for instance based on RETE [25] or borrowed from SPARQL query optimization [26], [27]. Investigating and measuring the effects of the various potential optimizations is beyond the scope of this paper, and will be considered in future work.

We generated benchmark datasets containing the clinical data described in Section III-A, whereby fact values were created based on ranges encompassing both clinically normal values as well as abnormal ones. With the goal of investigating the scalability of mobile reasoning, our benchmarks consider a sequence of datasets, each containing an increasing amount of data. These datasets contain personal clinical info (e.g., weight, height) and additionally comprise 1 (137 triples), 5 (393 triples), 10 (713 triples), 25 (1673 triples), 50 (3273 triples) and 75 (4873 triples) health measurements and observations, respectively. Each dataset triggers 40-50% of the rules. The dataset and ruleset can be found at https://niche.cs.dal.ca/wic.

4) Measurement methodology: To minimize the impact of background OS processes on results, we ran each performance benchmark 20 times and calculated the average execution times. Memory usage had to be measured differently for the JavaScript engines and the native Android engine. For AndroJena, the Android API was used to obtain a heap dump, which was later analyzed using the Eclipse Memory Analyzer[12] (MAT). For the JavaScript engines, the Chrome DevTools remote debugging support[13] was utilized to record and analyze heap allocations during loading and reasoning steps. In order to use this remote debugging support, we had to separately run our local Website, normally deployed in the Phonegap mobile app (see Section III-C1), in the mobile Chrome browser. Since both the WebView created by the Phonegap app and mobile Chrome utilize the same mobile WebKit engine, both deployment scenarios should result in the same memory allocations[14]. To evaluate the AndroJena reasoning engine, we relied on the default configuration settings, which uses the hybrid execution model (see Section II-A1).

5) Hardware: The benchmarks were performed on a Samsung Galaxy SIII (model number GT-I9300), with a 1.4GHz quad-core processor, 1GB RAM and 16GB persistent storage. The installed Android OS was version 4.3 (Jelly Bean) with API level 18. This model is currently two generations old at the time of writing (with the new Galaxy S5 model soon to be released). As such, this better reflects a real-world scenario, where users typically do not possess the latest device model.

IV. Results

In the two sections below, we show the benchmark results for each studied process flow (see Section III-B).

[12]https://www.eclipse.org/mat/

[13]https://developers.google.com/chrome-developer-tools/docs/remote-debugging

[14]Due to errors when running Nools in mobile Chrome, memory measurements for that engine were obtained via the desktop Chrome. However, these allocations are equivalent to the ones made in mobile Chrome.

A. Process flow 1: Frequent Reasoning

In this process flow, the reasoning engine is loaded with the entire dataset, and executed with the given ruleset. Table I summarizes the average times of the loading step for each engine. We note that for Nools, these loading times also include loading the rules into the engine, in order to build the internal RETE network. On average, this rule-loading time amounts to 7010 ms. In the table, we separately indicate the data loading time between parenthesis for Nools.

# triples	RDFQuery	RDFStore-JS	Nools	AndroJena
137	55	197	7187 (250)	99
393	197	528	7583 (877)	129
713	292	878	9103 (1762)	358
1673	512	1795	12174 (5198)	541
3273	814	3687	21020 (14539)	1189
4873	1453	5008	49465 (41845)	1560

TABLE I: Loading times for increasing dataset sizes (ms)

These results are visualized in Figure 3. Since the exceedingly high loading times for Nools skew the graph, we show the loading times for only RDFQuery, RDFStore-JS and AndroJena in Figure 4.

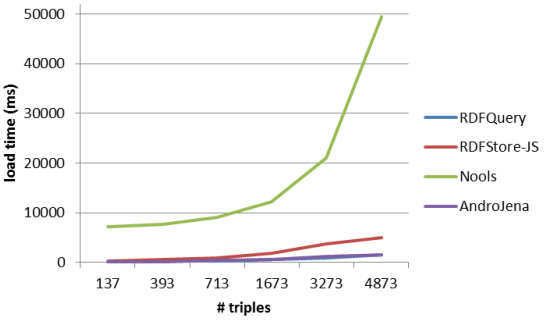

Fig. 3: Loading times for increasing dataset sizes

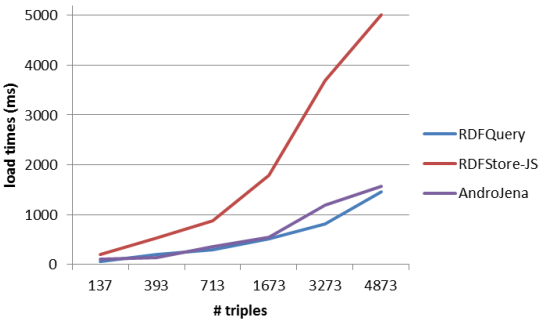

Fig. 4: Loading times for increasing dataset sizes (excluding Nools)

Table II shows the average performance overhead of the rule execution step. We note that for some reasoning engines, this step includes first creating rule objects; since this rule creation step turned out to be trivial (never exceeding 50 ms), these times were added to the displayed results.

# triples	RDFQuery	RDFStore-JS	Nools	AndroJena
137	156	1090	30	92
393	530	1290	37	83
713	1090	1197	47	393
1673	6029	1486	42	3302
3273	35405	1931	34	24537
4873	106722	2306	358	78549

TABLE II: Reasoning times for increasing dataset sizes (ms)

Figure 5 visualizes these results. Since the reasoning times for AndroJena and RDFQuery become exceedingly high for large datasets, we separately show the reasoning times for the RDFStore-JS and AndroJena engines in Figure 6.

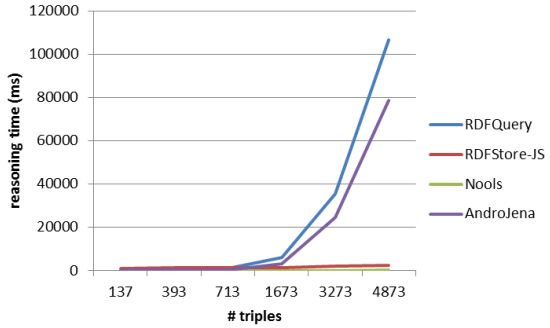

Fig. 5: Reasoning times for increasing dataset sizes

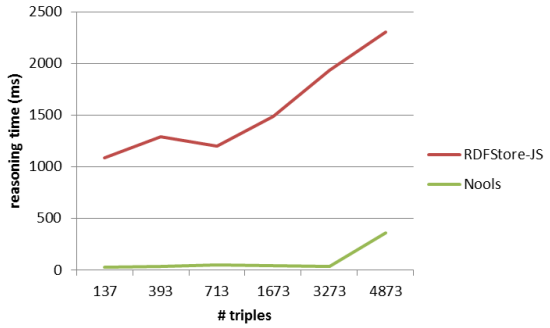

Fig. 6: Reasoning times for increasing dataset sizes (excluding AndroJena)

We note that, in this process flow, the loading and reasoning steps are always performed together and in sequence. We show the total times for the loading and reasoning steps in Table III.

# triples	RDFQuery	RDFStore-JS	Nools	AndroJena
137	211	1287	7217	191
393	727	1818	7620	212
713	1382	2075	9150	751
1673	6541	3281	12216	3843
3273	36219	5618	21054	25726
4873	108175	7314	49823	80109

TABLE III: Total times for increasing dataset sizes (ms)

Table IV shows the memory usage for each reasoning engine. We note these measurements were taken right after the reasoning step was performed; and before any cleanup has occurred (e.g., releasing resources occupied by the engine).

# triples	RDFQuery	RDFStore-JS	Nools	AndroJena
137	407	146	625	163
393	544	235	1186	201
713	695	347	1376	260
1673	1186	687	3576	422
3273	1944	1255	4536	759
4873	2707	1785	5556	1048

TABLE IV: Memory usage for increasing dataset sizes (Kb)

These results are visualized in Figure 7.

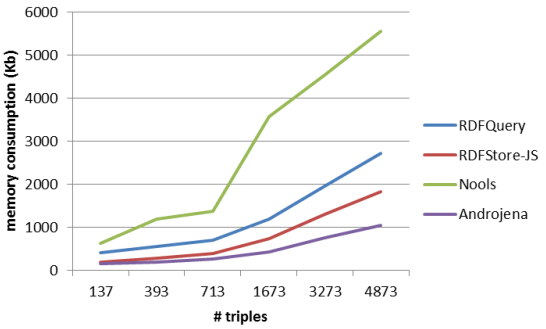

Fig. 7: Memory usage for increasing dataset sizes

B. *Process flow 2: Incremental Reasoning*

In this process flow, the reasoning engine is initially loaded with a baseline dataset, after which new facts are dynamically loaded into the engine as they are entered. As baseline dataset, we employed the dataset containing 25 clinical facts (1673 triples), whereby we additionally loaded a single clinical fact (5 triples) to the engine. Afterwards, the reasoning step is performed. In Table V, we show the average times for this single loading step and reasoning step, and the total time adding both averages together. For the loading times of the baseline dataset, we refer to Table I.

	RDFQuery	RDFStore-JS	Nools	AndroJena
loading	38	8	15	18
reasoning	6097	1440	12	3403
total	6135	1448	27	3421

TABLE V: Loading times for a single clinical fact (ms)

V. Discussion

Below, we discuss the obtained results.

A. Process flow 1: Frequent Reasoning

Figure 3 shows the average loading times for increasing dataset sizes. This graph shows the Nools loading time is problematic, especially for large datasets (> 1673). However, we note this loading time also includes a constant time for loading the rules (ca. 7 s). For smaller sources (≤ 713 triples), the data loading time (shown between brackets in Table I) is reasonable, although still higher than for the other engines. We also note that rule loading time will be lower for smaller rulesets[15]. Figure 4 zooms in on the loading times for the other reasoning engines. The figure shows that RDFQuery and AndroJena have good loading performance, even for the largest dataset (4873 triples); while the RDFStore-JS loading times increases significantly with the dataset size.

On the other hand, Figure 5 illustrates that RDFQuery and AndroJena, while still performing well for smaller datasets (≤ 713 triples), have very problematic reasoning performance for larger datasets (> 1673 triples). Table II shows that, although initially high, the RDFStore-JS reasoning time rises comparably much slower as the dataset size increases. We observe that Nools has by far the best overall reasoning performance, never exceeding 100 ms (except for the largest dataset). Table III shows the total times for each engine, including loading and reasoning. Finally, regarding memory usage, Nools consumes the most memory during reasoning, while the memory consumption for the other three engines are relatively close together. Overall, AndroJena has the lowest memory overhead, barely exceeding 1Mb for the largest dataset.

Overall, we conclude that **RDFStore-JS** presents the most *scalable* solution in this process flow, performing best for larger datasets (≥ 1673 triples); since its reasoning times stay relatively stable for increasing dataset sizes. **AndroJena** is the best solution for *smaller datasets* (< 1673 triples), with its overall low loading and reasoning times for small sources.

B. Process flow 2: Incremental Reasoning

Figure V shows the average times for loading a single clinical fact (5 triples) into the reasoning engine, given an initial baseline dataset of 25 clinical facts. As was to be expected from our discussion above, Nools performs extremely well for this reasoning step, with almost negligible reasoning times. In contrast, reasoning times for the other three engines is comparable to their reasoning performance for this dataset size in the first process flow.

We conclude that, once the initial data and rule loading is out of the way, Nools has by far the best reasoning performance when incrementally adding facts in this process flow. As noted in the previous section, Nools data loading times for small datasets are still acceptable (while the rule loading time will also decrease with the ruleset size). Therefore, **Nools** is the best option for this flow in case of *small datasets* (≤ 713 triples) *and rulesets*, since the low reasoning time makes up for the increase initialization time. In case *scalability* is required, **RDFStore-JS** is still the best option, with its scalable loading times (compared to Nools) and reasoning times (compared to RDFQuery and AndroJena).

[15]This was observed during other experiments.

VI. Related Work

There are a number of works that aim to facilitate benchmarking RDF(S)/OWL repositories and reasoning engines.

The Lehigh University Benchmark [28] supplies a set of test queries and a data generator to generate datasets, both referencing a university ontology. In addition, a test module is provided for carrying out data loading and query testing. Similarly, the Berlin SPARQL benchmark [29] supplies test queries, a data generator and test module for an e-commerce scenario. We note that these two works consider different application domains, do not support benchmarks for rule-based reasoning, and only focus on desktop and server setups.

OpenRuleBench [30] is a collection of benchmarks for comparing and analyzing the performance of a supported set of rule engines. It comprises a test suite for benchmarking internal reasoning engine aspects such as large joins and datalog recursion. The supported rule engines are meant for deployment on desktop or server UNIX environments, and are thus not meant for mobile deployment.

VII. Conclusion and Future Work

We note that, aside from AndroJena, none of the evaluated reasoning engines were actually meant for mobile deployment. Despite this, the benchmark results illustrate that, given a real-world clinical dataset and ruleset, and situated within a mobile clinical decision support scenario for patient self-management, these reasoning engines are usable on mobile platforms. Regarding the first decision support process flow, the most efficient total times remain below 1s for small sources (≤ 713 triples). Nools already presents an efficient solution for the second process flow, given the initial dataset and ruleset are small (≤ 713 triples).

However, we also observe that scalability is certainly an issue, with most efficient total times for the first process flow rising to ca. 7s for large datasets (4873 triples). When collecting increasing amounts of clinical data, or when working in different domains (e.g., context-awareness), datasets will likely exceed this size. As such, more work is needed on optimizing general-purpose, Semantic Web rule-based reasoning engines for mobile deployment. Automatically fine-tuning the rules to suit the particular reasoning mechanism and/or dataset (see Section III-C3) has the potential to yield large performance gains, and would be a very good step in this direction.

We also note that distributed CDSS systems (e.g., as implemented in the IMPACT-AF project and [9], [10]) can cope with this scalability issue by delegating lightweight, time-critical reasoning tasks to the client-side, and performing more heavy-weight reasoning on the server. In doing so, such approaches exploit the advantages of mobile, client-side reasoning while still guaranteeing efficient performance. Additionally, we note the benchmark device is currently two generations old; while this better reflects a real-world usage scenario, more advanced devices will clearly yield better reasoning performance.

Future work consists of extending our current benchmark setup into an extensible framework, which enables developers to use any rule- and dataset in benchmarks, and allows new reasoning engines to be plugged in as well. In doing so, we

aim to aid mobile developers in choosing the most suitable reasoning engine for their purpose. Finally, future benchmarks will be performed with datasets and rulesets requiring more advanced reasoning techniques (e.g., chaining, conflict resolution), while different optimization techniques will be applied as well to systematically evaluate their impact on performance.

ACKNOWLEDGMENT

This research project is funded by a research grant from Bayer Healthcare.

REFERENCES

[1] K.-H. Cheung, E. Prud'hommeaux, Y. Wang, and S. Stephens, "Semantic web for health care and life sciences: a review of the state of the art," *Briefings in bioinformatics*, vol. 10, no. 2, pp. 111–113, 2009.

[2] V. Haarslev and R. Möller, "Description of the racer system and its applications," in *Description Logics*, ser. CEUR Workshop Proceedings, C. A. Goble, D. L. McGuinness, R. Möller, and P. F. Patel-Schneider, Eds., vol. 49, 2001.

[3] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean *et al.*, "Swrl: A semantic web rule language combining owl and ruleml," *W3C Member submission*, vol. 21, p. 79, 2004.

[4] H. Knublauch, J. A. Hendler, and K. Idehen, "Spin-overview and motivation," *Member Submission, W3C (February 2011)*, 2011.

[5] D. L. Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth, "RDF On the Go: RDF Storage and Query Processor for Mobile Devices." in *ISWC Posters&Demos*, ser. CEUR Workshop Proceedings, A. Polleres and H. Chen, Eds., vol. 658. CEUR-WS.org, 2010. [Online]. Available: http://dblp.uni-trier.de/db/conf/semweb/pd2010.html\#PhuocPRH10

[6] C. Weiss, A. Bernstein, and S. Boccuzzo, "i-MoCo: Mobile Conference Guide Storing and querying huge amounts of Semantic Web data on the iPhone-iPod Touch," in *Semantic Web Challenge 2008*, 2008.

[7] S. Zander and B. Schandl, "A framework for context-driven RDF data replication on mobile devices," in *Proceedings of the 6th International Conference on Semantic Systems*, ser. I-SEMANTICS '10. New York, NY, USA: ACM, 2010, pp. 22:1—-22:5. [Online]. Available: http://doi.acm.org/10.1145/1839707.1839735

[8] S. R. Abidi, S. S. R. Abidi, and A. Abusharek, "A Semantic Web Based Mobile Framework for Designing Personalized Patient Self-Management Interventions," in *Proceedings of the 1st Conference on Mobile and Information Technologies in Medicine*, Prague, Czech Republic, 2013.

[9] N. Ambroise, S. Boussonnie, and A. Eckmann, "A Smartphone Application for Chronic Disease Self-Management," in *Proceedings of the 1st Conference on Mobile and Information Technologies in Medicine*, Prague, Czech Republic, 2013.

[10] A. Hommersom, P. Lucas, M. Velikova, G. Dal, J. Bastos, J. Rodriguez, M. Germs, and H. Schwietert, "Moshca - my mobile and smart health care assistant," in *e-Health Networking, Applications Services (Healthcom), 2013 IEEE 15th International Conference on*, Oct 2013, pp. 188–192.

[11] G. J. Nalepa and S. Bobek, "Rule-based solution for context-aware reasoning on mobile devices," *Computer Science and Information Systems*, no. 00, pp. 2–2, 2014.

[12] B. Motik, I. Horrocks, and S. M. Kim, "Delta-reasoner: A Semantic Web Reasoner for an Intelligent Mobile Platform," in *Proceedings of the 21st International Conference Companion on World Wide Web*, ser. WWW '12 Companion. New York, NY, USA: ACM, 2012, pp. 63–72. [Online]. Available: http://doi.acm.org/10.1145/2187980.2187988

[13] R. Styles, N. Shabir, and J. Tennison, "A pattern for domain specific editing interfaces using embedded RDFa and HTML manipulation tools," in *Proc. of Scripting and Development for the Semantic Web Workshop at the ESWC, Heraklion, Greece, May 31, 2009*, ser. CEUR Workshop Proceedings ISSN 1613-0073, S. Auer, C. Bizer, and G. A. Grimnes, Eds., vol. 449, June 2009, pp. 1–11.

[14] A. G. Hernández and M. Moreno García, "RESTful triple space management of cloud architectures," in *7th International Conference on Knowledge Management in Organizations: Service and Cloud Computing*, ser. Advances in Intelligent Systems and Computing, L. Uden, F. Herrera, J. Bajo Pérez, and J. M. Corchado Rodríguez, Eds. Springer Berlin Heidelberg, 2013, vol. 172, pp. 571–579. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30867-3_51

[15] Integrated Management Program Advancing Community Treatment of Atrial Fibrillation, "Impact AF," http://impact-af.ca/.

[16] C. L. Forgy, "Rete: A fast algorithm for the many patterns/many objects match problem," *Artif. Intell.*, vol. 19, no. 1, pp. 17–37, 1982.

[17] G. Klyne, J. Carroll, and B. McBride, "Resource description framework (RDF): Concepts and abstract syntax," *Changes*, 2004.

[18] W3C, "Owl 2 web ontology language profiles (second edition)," December 20120.

[19] E. S. Berner and T. J. La Lande, "Overview of clinical decision support systems," in *Clinical decision support systems*. Springer, 2007, pp. 3–22.

[20] I. Herman, B. Adida, M. Sporny, and M. Birbeck, "RDFa 1.1 Primer - Second Edition," 2013. [Online]. Available: http://www.w3.org/TR/xhtml-rdfa-primer/

[21] A. Pantelopoulos and N. G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 40, no. 1, pp. 1–12, 2010.

[22] Canadian Cardiovascular Society, "Atrial Fibrillation Guidelines," http://www.ccsguidelineprograms.ca.

[23] European Society of Cardiology, "Atrial Fibrillation Guidelines," http://www.escardio.org/guidelines-surveys/esc-guidelines/guidelinesdocuments/guidelines-afib-ft.pdf.

[24] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93. New York, NY, USA: ACM, 1993, pp. 157–166. [Online]. Available: http://doi.acm.org/10.1145/170035.170066

[25] C. Matheus, K. Baclawski, and M. Kokar, "Basevisor: A triples-based inference engine outfitted to process ruleml and r-entailment rules," in *Rules and Rule Markup Languages for the Semantic Web, Second International Conference on*, Nov 2006, pp. 67–74.

[26] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "Sparql basic graph pattern optimization using selectivity estimation," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 595–604. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367578

[27] M. Schmidt, M. Meier, and G. Lausen, "Foundations of sparql query optimization," in *Proceedings of the 13th International Conference on Database Theory*, ser. ICDT '10. New York, NY, USA: ACM, 2010, pp. 4–33. [Online]. Available: http://doi.acm.org/10.1145/1804669.1804675

[28] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2, pp. 158–182, 2005.

[29] C. Bizer and A. Schultz, "The berlin sparql benchmark," *International Journal On Semantic Web and Information Systems-Special Issue on Scalability and Performance of Semantic Web Systems*, 2009.

[30] S. Liang, P. Fodor, H. Wan, and M. Kifer, "Openrulebench: An analysis of the performance of rule engines," in *Proceedings of the 18th International Conference on World Wide Web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 601–610. [Online]. Available: http://doi.acm.org/10.1145/1526709.1526790