Tiny-ME Wasm: Description Logics reasoning in your browser

Giuseppe Loseto[1][0000−0002−7995−8494], Ivano Bilenchi[2][0000−0001−8294−2445], Filippo Gramegna[2][0000−0003−4162−957X], Davide Loconte[2][0000−0002−0182−4672], Floriano Scioscia[2][0000−0002−7859−9602]*, and Michele Ruta[2][0000−0003−2125−327X]

[1] LUM University "Giuseppe Degennaro", Casamassima BA 70010, Italy
loseto@lum.it
[2] Polytechnic University of Bari, Bari 70125, Italy
{name.surname}@poliba.it

Abstract. The World Wide Web is increasingly adopted as an application development platform in desktop and mobile contexts. As a building block of the Semantic Web of Everything vision, reasoning engines should be able to run in the client side of Web applications without resorting to remote servers for inference tasks. In order to counter the current dearth of such systems, this paper proposes Tiny-ME Wasm, a WebAssembly port of the Tiny-ME Description Logics reasoning and matchmaking engine. Tiny-ME Wasm exposes standard and non-standard inference services through a JavaScript API on top of the Tiny-ME core compiled to WebAssembly. A case study highlights the usefulness of the proposed solution and a preliminary performance analysis demonstrates its usability on laptop and mobile devices.

Keywords: Semantic Web of EveryThing · Description Logics Reasoning · WebAssembly · Mobile Web

1 Introduction

World Wide Web (WWW) technologies are currently adopted as universal interoperability standards to allow sharing, aggregation, and composition of information and services in distributed computing environments comprising heterogeneous platforms. The Web has become a dominant deployment environment for new software systems and applications and the browser has become the default interface for many relevant applications [14]. The Web of Things (WoT) [8] aims to extend the reach of the WWW to physical objects and environments, countering the fragmentation of Internet of Things (IoT) technologies. The WoT enables access to the digital counterparts of physical entities, exposing data and/or services as addressable resources. Nevertheless, mobile and pervasive computing require more advanced approaches than traditional WWW solutions for information storage, discovery, management, and integration, to cope with device

* Corresponding author Tel.: +39-080-596-3054; fax: +39-080-596-3410

and resource volatility issues generated by location dependency, intermittent connectivity, performance limitations and energy consumption constraints.

The Semantic Web of Everything (SWoE) vision promotes the integration of Semantic Web languages and technologies for knowledge representation and automated inference on mobile, embedded and pervasive autonomous devices, enabling scalable interactions among people, things, environments and processes [10]. In order to fully include the Web as an application platform in this vision, reasoning engines should be able to run inside Web browsers at the client side of Web applications. Unfortunately, this has happened very rarely so far, due to the performance penalty of traditional JavaScript (JS) interpreters embedded in Web browsers. On the other hand, depending on remote servers for reasoning is not feasible in mobile and ubiquitous contexts due to the aforementioned issues, whereas exploiting local servers is forbidden by security policies of prevalent mobile and embedded operating systems.

This paper presents *Tiny-ME Wasm*, a port of the *Tiny Matchmaking Engine* [10] Description Logics (DL) reasoner, which can run in the *WebAssembly* (Wasm) [9] virtual machine embedded in major modern desktop and mobile Web browsers. It supports *standard* (satisfiability, subsumption, coherence, and classification) and *non-standard* (compatibility, abduction, contraction, bonus, difference, and covering) inference services; the latter are particularly useful for on-the-fly resource discovery and query processing in SWoE scenarios. Web developers can use these capabilities through a straightforward JS Application Programming Interfaces (API) mapping Tiny-ME's original C++ API. In order to demonstrate the usability and usefulness of the proposal, a mobile Web case study on a privacy-oriented local event finder is described. Since user experience in Web applications is particularly sensitive to latency, [5] a performance evaluation has been carried out on both standard and non-standard inferences to assess reasoning times on macOS, iOS and Android devices.

The remainder of the paper is as follows. Section 2 discusses relevant related work, while Section 3 outlines the architecture and implementation of the proposed system. The case study is in Section 4, and experiments in Section 5 precede conclusions.

2 Related work

The wide majority of Description Logics inference engines has been designed and developed for specific computing platforms, which support the required programming languages and runtime libraries. Due to inferior performance optimization, the Web-oriented JavaScript language has been seldom chosen. For this reason, Web applications typically exploit remote services for DL reasoning. A large number of semantic-enabled Web applications for conventional desktop clients follows this approach, covering a wide range of domains; unfortunately, both technological and user interface design choices prevent their adaptation to mobile and ubiquitous contexts. Notable semantic-enabled mobile-oriented Web apps include the *DBpedia Mobile* [2] and *LOD4AR* [18] location-based dis-

covery clients for mobile devices. *OntoWiki Mobile* [4] is a mobile knowledge management applications exploiting HTML5 and jQuery Mobile to locally store knowledge annotated by users in the field, even without Internet connection, and synchronize it with a remote server when a connection becomes available. Overall, the survey in [20] on semantic-enabled mobile apps has found that only 4 out of 36 apps were developed with Web-based cross-platform technologies.

Efforts on using Web technologies for cross-platform reasoning engines are sparse. The *EYE* (Euler Yet another proof Engine) [17] reasoner, implementing forward and backward rule chaining over Euler paths, has been ported to JS-based environments like the Web and *Node.js*[3] by leveraging a JS implementation of the *SWI Prolog* [19] engine. The *MobiBench* mobile semantic rule engine evaluation framework [16] supports integration with rule engines written in JS for the Web and further platforms, by means of the *Apache Cordova*[4] Software Development Kit (SDK). *HyLAR+*[5] [15] is a hybrid OWL 2 EL reasoner: both the server side and the client side can execute reasoning tasks by running the same JS code with Node.js and *AngularJS*[6], respectively. It leverages the *JSW* semantic technology framework for JS[7], which includes the *BrandT* browser-hosted OWL 2 EL inference engine [13].

WebAssembly discloses a viable path to developing or porting reasoning engines to Web applications and stand-alone runtime environments like Node.js with minimal performance penalty. Proposals, however, are in very early stages. *EYE JS*[8] is an initial WebAssembly port of EYE. A prototypical port of the *Tiny-ME* DL reasoner and matchmaker [10] has been included in a Web application for semantic-based Quality of Experience (QoE) adaptation in Web multimedia [6].

3 Architecture and implementation

The design and development of the Tiny-ME OWL reasoner and matchmaking engine [10] have been guided by architectural and technological choices aimed at ensuring support for a wide range of target platforms, one of the key requirements of the SWoE. System architecture comprises a shared core, which efficiently implements data structures and inference procedures, along with a set of high-level APIs that facilitate development for supported platforms. The core is written in the C programming language, which ensures a high degree of portability, as most programming languages and platforms provide some form of interoperability with C code. Web browser environments are no exception, as C programs and libraries can be cross-compiled for the WebAssembly runtime

[3] Node.js: `https://nodejs.org/`
[4] Apache Cordova home: `https://cordova.apache.org/`
[5] HyLAR GitHub repository: `https://github.com/ucbl/HyLAR-Reasoner`
[6] AngularJS home: `https://angularjs.org/`
[7] JSW GitHub repository: `https://github.com/JS-WindowFramework/JSW`
[8] EYE JS GitHub repository: `https://github.com/eyereasoner/eye-js`

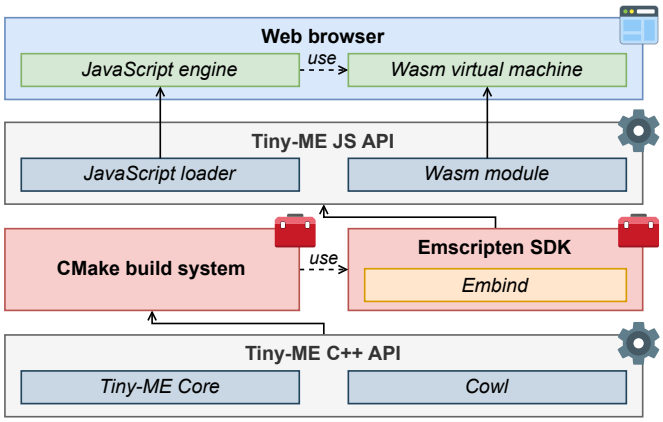


Fig. 1: Porting workflow and high-level architecture

through the *Emscripten*[9] toolchain, and then invoked from JS. While it is possible to directly compile C code and expose a procedural API to JS client-side code, better interoperability can be achieved by providing a thin object-oriented C++ wrapping layer and generating bindings to JS classes through *Embind*.[10]

The porting workflow of Tiny-ME Wasm, illustrated in Figure 1, has involved the following steps:

- implementing an object-oriented C++ API by wrapping the Tiny-ME core component and the Cowl [3] library, which is used by the reasoner to access and parse Web Ontology Language 2 (OWL) ontologies;
- providing appropriate binding annotations, so allowing Embind to generate a JS interface that directly maps to the low-level C++ API;
- configuring the *CMake*[11] build system to use Emscripten and invoking it to cross-compile the system for the WebAssembly runtime.

Emscripten generates two output artifacts: a Wasm module containing bytecode for the WebAssembly virtual machine, and a JS file responsible for loading the module and setting up the runtime environment. The latter can be imported in any Web application by means of HTML `script` tags or *JS modules*,[12], allowing client-side code to use all the functionalities of the reasoner as a regular JS library.

Figure 2 summarizes the low-level system architecture. The object-oriented design of the C API of the reasoner [10] has resulted in a relatively straightforward mapping of C structures and related functions to C++ classes and methods.

[9] Emscripten home: `https://emscripten.org`
[10] Embind documentation: `https://emscripten.org/docs/porting/connecting_cpp_and_JS/embind.html`
[11] CMake home: `https://cmake.org`
[12] JS modules: `https://developer.mozilla.org/en-US/docs/Web/JS/Guide/Modules`

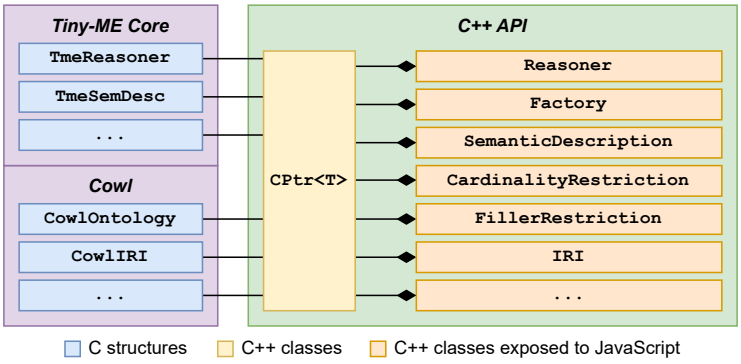


Fig. 2: Low-level architecture

One friction point was due to the significantly different memory management paradigms of the underlying libraries, as the Tiny-ME core follows the traditional `malloc`/`free` approach, while Cowl adopts *reference counting*. Care has been devoted to standardizing object lifecycles under the *Resource Acquisition is Initialization* (*RAII*) paradigm, by encapsulating C pointers in a `CPtr<T>` template type, that implements type-specific memory management logic. This approach simplifies the handling of dynamic memory by delegating it to the C++ runtime, and allows the resulting JS bindings to correctly dispose of unneeded allocations.

The library entry point is the `Reasoner` object, which provides access to all Tiny-ME's standard (satisfiability, subsumption, coherence, and classification) and non-standard (compatibility, abduction, contraction, bonus, difference, and covering) inference services, which are recalled in [10]. Non-standard inferences are particularly useful in SWoE scenarios, as they go beyond the boolean answers of standard ones by allowing approximated matches and semantic-based (dis)similarity measures, making them suitable to be composed into more sophisticated reasoning tasks. One such task is *semantic matchmaking*, where semantic-based annotations of domain resources are compared in order to find the best matches for a specific request, based on the aforementioned semantic similarity.

4 Case study: a privacy-oriented local event finder

In order to highlight the usage and usefulness of the proposed reasoning engine in mobile and ubiquitous Web scenarios, a case study is proposed about retrieving nearby local events and ranking them according to user preferences. Nowadays, Web platforms typically require users to expose and share their profile information in order to obtain personalized recommendations for events, products and other items. The proposed workflow, depicted in Figure 3, represents a different general-purpose approach, where a mobile Web application (1) retrieves available information from the Web, (2) annotates it according to an OWL domain ontology, (3) exploits it to identify resources best matching user's profile and

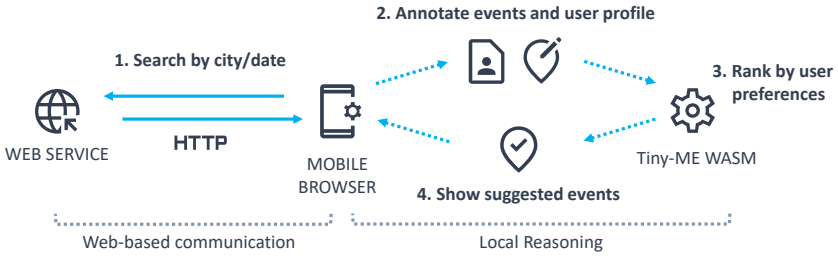


Fig. 3: Framework architecture - Reference workflow

preferences, and (4) displays results in the user interface for selection and/or query refinement. Notably, all processing occurs locally on the user's mobile device: this enables a privacy-oriented information management and avoids the latency of interactions with a remote reasoning engine.

In order to better explain and underline the benefits of the proposed approach, let us consider the following mobile-oriented scenario. *Martina, while on vacation in San Francisco, is interested in purchasing a ticket for a musical event, spending no more than $80. She is a fan of alternative rock bands and would like to attend a tour date rather than a one-time concert.*

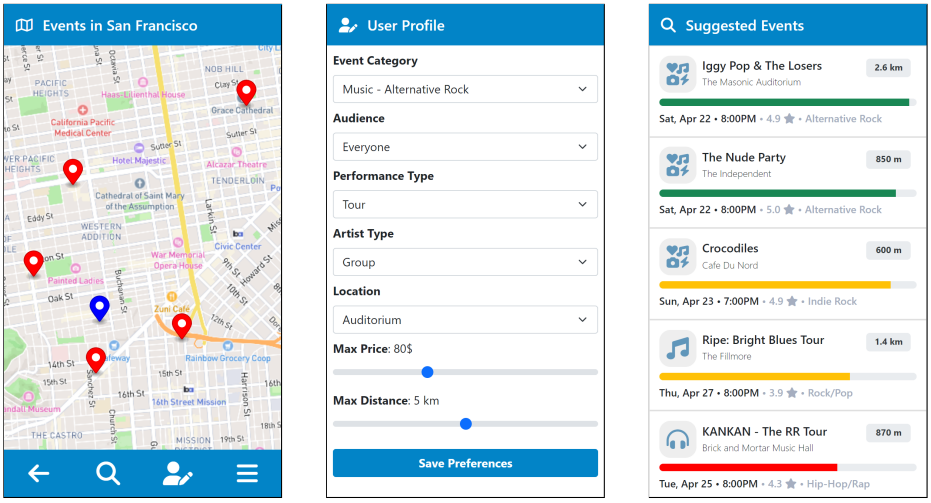
As shown in Figure 3, the user can access information on local events by means of a Web application in her mobile browser. Data for the case study are provided by *Ticketmaster*[13], through a public RESTful API[14]. In the first step (Figure 4a), the user retrieves all events nearby according to basic parameters, such as city and date range. This represents generic information, not personalized on her preferences. However, she can select several features and preferences to define her private personal profile (Figure 4b) and refine the initial query directly on the local device, without sending further requests to the server. The user profile is mapped to a semantic-based annotation by means of a prototypical domain ontology: Figure 5 depicts its taxonomy of concepts, which maps the categories provided by the aforementioned service API. The detail of the selected preferences is shown below in *Manchester syntax*:

```
User_Profile: Event and (hasAudience only EveryOne) and (hasPrice max 80)
and (hasStyle only Tour) and (hasCategory only Alternative_Rock) and
(hasType only Group) and (hasCategory min 1) and (hasStyle min 1)
and (hasAudience min 1)
```

After a pre-filtering step based on a maximum distance of 5 km from the mobile device location, the user profile is compared with the semantically annotated events descriptions, in order to obtain the list of resources best fulfilling both user preferences and compatibility constraints. All the events, reported hereafter in alphabetical order, are tagged as individuals using the same ontology:

[13] https://www.ticketmaster.com

[14] https://developer.ticketmaster.com/products-and-docs/apis/getting-started/



(a) Available events (b) User profile (c) Suggested events

Fig. 4: Screenshots of the Web application

```
Crocodiles: Event and (hasAudience only EveryOne) and (hasPrice min 163)
and (hasPrice max 180) and (hasStyle only Tour) and (hasType only Band)
and (hasCategory only Indie_Rock)

Iggy_Pop_and_The_Losers: Event and (hasAudience only EveryOne) and
(hasPrice min 50) and (hasPrice max 100) and (hasStyle only Tour) and
(hasType only Band) and (hasCategory only Alternative_Rock)

KANKAN-The_RR_Tour: Event and (hasAudience only EveryOne) and (hasPrice
min 25) and (hasPrice max 105) and (hasStyle only Concert) and (hasType
only Performer) and (hasCategory only Hip_Hop_Rap)

Ripe-Bright_Blues_Tour: Event and (hasAudience only EveryOne) and
(hasPrice exactly 27) and (hasStyle only Tour) and (hasType only Band)
and (hasCategory only Pop_Rock)

The_Nude_Party: Event and (hasAudience only EveryOne) and (hasPrice
exactly 20) and (hasStyle only Concert) and (hasType only Band) and
(hasCategory only Alternative_Rock)
```

The returned list of events, shown in Figure 4c, is ranked according to the following *utility function*:

$$u(R,C) = 100[1 - \frac{s_penalty(R,C)}{s_penalty(R,\top)}(1 + \frac{distance(R,C)}{max_distance})]$$

where $s_penalty(R,C)$ is the result of the *semantic matchmaking* task [10] and represents the semantic distance between user profile R and event annotation C, normalized dividing by $s_penalty(R,\top)$. This value is the semantic distance between R and the *universal concept* $\top$ (a.k.a. *Top* or *Thing*) and depends only on the ontology structure. The $distance(R,C)$ context parameter is included to

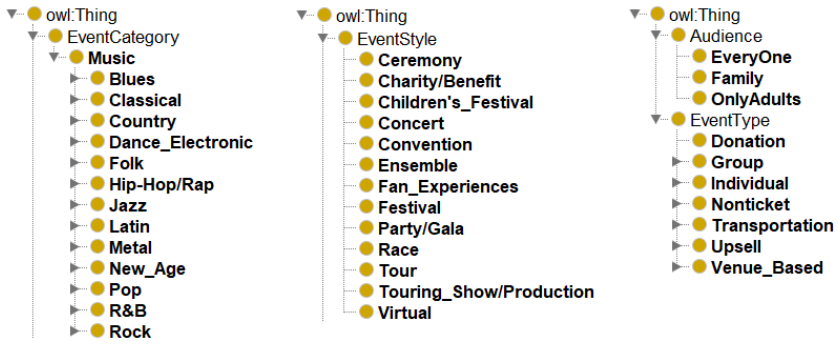


Fig. 5: Reference ontology

Table 1: Musical events ranked by the utility function

Musical events	$s_penalty(R,C)$	$distance(R,C)$	$u(R,C)$
Iggy Pop & The Losers	0.20	2.60 km	97.97 %
The Nude Party	1.00	0.85 km	93.22 %
Crocodiles	1.51	0.60 km	89.81 %
Ripe Bright Blues Tour	4.00	1.40 km	72.59 %
KANKAN - The RR Tour	6.24	0.87 km	57.68 %

exploit the event geographical distance as weighting factor. The utility function also converts the semantic distance value into a more intuitive percentage score, as shown in Table 1. It can be observed that if $s_penalty(R,C) = 0$ then the final score is 100%, regardless of the geographic distance. Based on user preferences and event descriptions, the most suitable event, as listed in Figure 4c, is the individual `Iggy_Pop_and_The_Losers`, even if its physical distance is greater than the other available resources.

5 Experimental evaluation

For any compute-intensive system, performance analysis is fundamental to assess whether it can provide outputs in a time frame that is compatible with application and use case requirements. This is particularly true when embedding reasoning capabilities into the client side of Web applications, as nowadays most Web content is consumed through battery-powered mobile devices [12] and any delay may significantly impact the perceived quality of experience. [5]

An experimental campaign has been conducted in order to evaluate the performance of the proposed Wasm reasoner in terms of turnaround time of standard and non-standard inference services. For each test, the following metrics have been measured:

– **Fetching**: the time spent by the client to request and download the target ontology. This value does not depend on the reasoner, but only on the device and the network link between the client and the server.

- **Parsing**: the time required by the reasoner to deserialize the ontology into its internal data structures.
- **Reasoning**: the time elapsed to carry out the requested inference service.

Tests have been executed on a 2021 Macbook Pro[15], an iPhone 12 Pro[16], and a OnePlus 7T[17], to achieve a sufficiently fair and diverse representation of Web content fruition devices, both hardware- and software-wise.

The reference dataset is inherited from [10]. Non-standard inferences have been carried out on four knowledge bases representing different SWoE case studies. For standard reasoning services, the dataset is a subset of the *ORE2014* corpus, [1] considering ontologies having at most $\mathcal{ALN}$ expressiveness, and whose file size does not exceed 1 MB in OWL 2 functional-style syntax, [7] for a total of 1140 ontologies.

The experimental methods consist in visiting a Web page containing client-side JS code responsible for performing all test operations. For each ontology in the dataset, the script retrieves the ontology document, then invokes the Tiny-ME Wasm reasoner to carry out the target inference. Results are saved to a CSV file, which is then fed to the *evOWLuator* [11] framework to generate visualizations. The test Web pages and ontologies are served by an *Nginx*[18] instance hosted on a desktop computer[19], configured to serve all requests without caching. All devices are located in the same 802.11n WLAN, hosted by a TP-Link TN-WR841N router.

Times are collected using the `performance.now()`[20] JS API call. By default, most browsers limit the resolution of the returned timestamp to 1-2 milliseconds as a mitigation for timing-based attacks and fingerprinting. This is enough for fetching and parsing, but it is often too coarse for inferences. Therefore, reasoning times have been computed by running multiple consecutive iterations of each task (10 for the two mobile devices, and 50 for the laptop), subtracting the cumulative fetching and parsing times from the total execution time, and averaging over all iterations.

5.1 Ontology classification

Figure 6 illustrates the results of the *ontology classification* standard inference service. The bar graph in Figure 6a shows the total time required for fetch-

[15] Apple M1 Max SoC, 8 performance cores @3.2GHz and 2 efficiency cores @2.0GHz, 64 GB UM RAM, 1 TB SSD, macOS Ventura 13.3.1, Safari 16.4 browser.

[16] Apple A14 Bionic, 2 high-power cores @3.1 GHz and 4 low-power cores @1.8 GHz, 6 GB LPDDR4X RAM at 4266 MT/s, 128 GB NVMe SSD, iOS 15.1, Safari Mobile 15.1 browser.

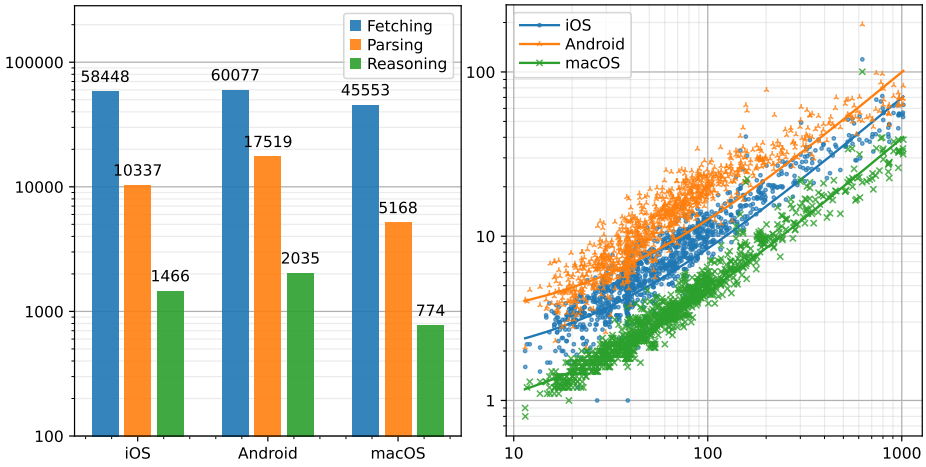
[17] Qualcomm Snapdragon 855+, 1 core @2.96 GHz, 3 cores @2.42 GHz and 4 cores @1.8 GHz, 8 GB LPDDR4X RAM, 128 GB UFS 3.0 Memory, Android 12, Chrome 112.0.5615.47 browser

[18] Nginx: `https://www.nginx.com/`

[19] Intel Core i7-3770k CPU, 4 cores @3.5GHz, 12 GB DDR3 RAM at 1600 MT/s, 2 TB SATA SSD, Windows 10.

[20] `https://developer.mozilla.org/en-US/docs/Web/API/Performance/now`

ing, parsing, and reasoning across all ontologies. The scatter plot in Figure 6b depicts the relationship between the size of the ontology and the total time required for parsing and reasoning, with the worst case requiring about 200 ms on the least capable device. Outcomes demonstrate satisfactory performance of inference services oriented to ontology management, which is essential for supporting interactive semantic-enabled Web applications in mobile environments. Although fetching time is the most prominent component, the domain ontology can usually be fetched just once and cached to improve application responsiveness.



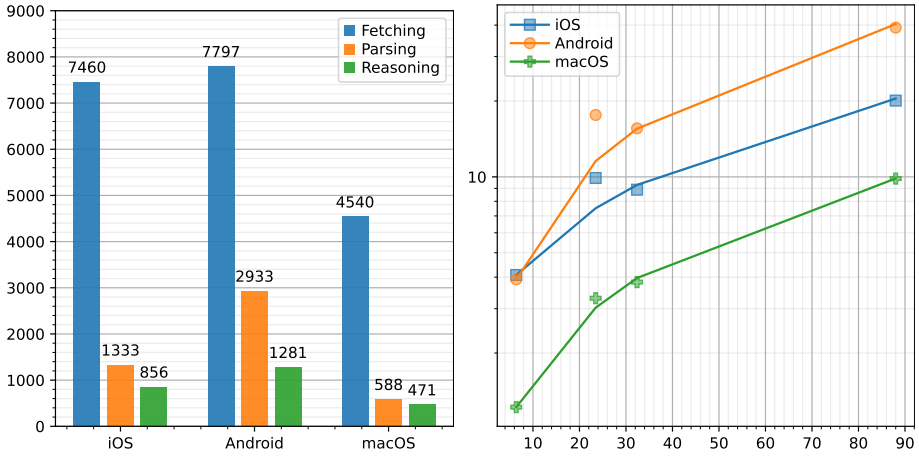
(a) Cumulative Reasoning Time (ms) (b) Reasoning Time (ms) by ontology size (kB)

Fig. 6: Tiny-ME Wasm ontology classification results

5.2 Semantic matchmaking

The *matchmaking* non-standard inference task involves iterating over all individuals in each of the 4 reference ontologies in order to find the best match for an individual acting as a *request* description. As explained in [10], matchmaking starts with a *compatibility check* between a $\langle request, resource \rangle$ pair of individuals: if their conjunction is satisfiable then *concept abduction* is performed, else *concept contraction* is executed, followed by abduction on the contracted version of the request. As results in Figure 7 show, the Wasm reasoner can carry out this inference in a few milliseconds, with 40 ms being the highest time measured for the largest ontology of the corpus.

These outcomes align with the ones in Section 5.1, but they are particularly relevant, as non-standard inferences are often more useful in SWoE scenarios, which Tiny-ME Wasm explicitly targets. The overall evaluation confirms that the proposed reasoner can be effectively integrated into interactive Web applications,



(a) Cumulative Reasoning Time (ms) (b) Reasoning Time (ms) by ontology size (kB)

Fig. 7: Tiny-ME Wasm semantic matchmaking results

providing both standard and non-standard inference services without negative impact on performance and user experience.

6 Conclusion

This paper has introduced Tiny-ME Wasm, a complete WebAssembly port of the Tiny-ME reasoning and matchmaking engine. The proposal represents one of the first Description Logics reasoners running on the client side of Web applications in modern browsers. It exposes all Tiny-ME's standard and non-standard inference services via a JavaScript API on top of the reasoner C core compiled to WebAssembly. Early performance tests of reasoning tasks demonstrate acceptable latency on PC and mobile platforms. A case study on a privacy-oriented Web application for location-based event discovery shows the benefits of the approach to SWoE applications leveraging well-known Web technologies.

Future work includes full integration into the Tiny-ME codebase and build automation framework, in order to keep updates always aligned with the master development branch. Further tests on a larger set of operating system and browser combinations will refine the compatibility and performance evaluation, suggesting possible optimizations. Release to the community is planned to enable the creation of a wide range of Web apps and gather feedback from developers.

References

1. Bail, S., Glimm, B., Jimenez-Ruiz, E., Matentzoglu, N., Parsia, B., Steigmiller, A.: ORE 2014 OWL Reasoner Evaluation Live Competition. `http://dl.kr.org/ore2014`, accessed: 2023-03-20

2. Becker, C., Bizer, C.: Exploring the Geospatial Semantic Web with DBpedia mobile. Journal of Web Semantics **7**(4), 278–286 (2009)
3. Bilenchi, I., Scioscia, F., Ruta, M.: Cowl: A Lightweight OWL Library for the Semantic Web of Everything. In: Agapito *et al.* (ed.) Current Trends in Web Engineering. ICWE 2022. pp. 100–112. Springer, Cham (2023)
4. Ermilov, T., Heino, N., Auer, S.: Ontowiki mobile: knowledge management in your pocket. In: Proceedings of the 20th International Conference Companion on World Wide Web. pp. 33–34 (2011)
5. Inan Nur, A., B. Santoso, H., O. Hadi Putra, P.: The method and metric of user experience evaluation: a systematic literature review. In: 2021 10th International Conference on Software and Computer Applications. pp. 307–317 (2021)
6. Loseto, G., Scioscia, F., Ruta, M., Gramegna, F., Bilenchi, I.: Semantic-based adaptation of quality of experience in web multimedia streams. In: 38th ACM/SIGAPP Symposium On Applied Computing (SAC 2023). pp. 1821–1830 (2023)
7. Parsia, B., Motik, B., Patel-Schneider, P.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C recommendation, W3C (Dec 2012), `http://www.w3.org/TR/owl2-syntax/`
8. Raggett, D.: The Web of Things: Challenges and Opportunities. Computer **48**(5), 26–32 (2015)
9. Rossberg, A.e.: WebAssembly Specification Release 2.0 (Draft 2023-04-08). `https://webassembly.github.io/spec/core/`, accessed: 2023-04-14
10. Ruta, M., Scioscia, F., Bilenchi, I., Gramegna, F., Loseto, G., Ieva, S., Pinto, A.: A multiplatform reasoning engine for the Semantic Web of Everything. Journal of Web Semantics **73**, 100709 (2022)
11. Scioscia, F., Bilenchi, I., Ruta, M., Gramegna, F., Loconte, D.: A multiplatform energy-aware OWL reasoner benchmarking framework. Journal of Web Semantics **72**, 100694 (2022)
12. StatCounter: Desktop vs Mobile vs Tablet Market Share Worldwide Jan - Dec 2022. `https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/2022`, accessed: 2023-04-20
13. Stepanovs, V.: BrandT: Browser Hosted OWL Reasoner. Tech. rep., University of Mancherster (May 2011)
14. Taivalsaari, A., Mikkonen, T.: The Web as a Software Platform: Ten Years Later. In: 13th International Conference on Web Systems and Technologies (WEBIST'17). pp. 41–50 (2017)
15. Terdjimi, M., Médini, L., Mrissa, M.: HyLAR+ improving hybrid location-agnostic reasoning with incremental rule-based update. In: Proceedings of the 25th International Conference Companion on World Wide Web. pp. 259–262 (2016)
16. Van Woensel, W., Abidi, S.S.R.: Benchmarking semantic reasoning on mobile platforms: Towards optimization using OWL2 RL. Semantic Web **10**(4), 637–663 (2019)
17. Verborgh, R., De Roo, J.: Drawing conclusions from Linked Data on the Web: the EYE reasoner. IEEE Software **32**(3), 23–27 (2015)
18. Vert, S., Dragulescu, B., Vasiu, R.: LOD4AR: Exploring Linked Open Data with a Mobile Augmented Reality Web Application. In: ISWC (Posters & Demos). pp. 185–188 (2014)
19. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming **12**(1-2), 67–96 (2012)
20. Yus, R., Pappachan, P.: Are Apps Going Semantic? A Systematic Review of Semantic Mobile Applications. In: 1st International Workshop on Mobile Deployment of Semantic Technologies. CEUR Workshop Proceedings, vol. 1506, pp. 2–13 (2015)