Architectural Concerns for Constrained Stream Reasoning

Lionel Médini, Alexandre Bento, Ghislain Atemezing, Kamal Singh, Frederique Laforest

Architectural Concerns for Constrained Stream Reasoning

Lionel Médini[1], Alexandre Bento[1], Ghislain Atemezing[2], Kamal Singh[3], Frédérique Laforest[1]

[1] Univ Lyon, CNRS, INSA Lyon, Université Claude Bernard Lyon 1, LIRIS UMR 5205, Villeurbanne, France
[2] Mondeca, Paris, France
[3] Université de Lyon, UJM, CNRS, LaHC UMR 5516, Saint-Etienne, France
{alexandre.bento,lionel.medini,frederique.laforest}@liris.cnrs.fr, ghislain.atemezing@mondeca.com, kamal.singh@univ-st-etienne.fr

Abstract. Semantic Web Reasoning on streaming data is a challenging issue, especially on constrained devices on the Web of Things. In this position paper, we propose some architectural clues to deploy reasoning on such devices in fog or edge paradigms, compliant with the W3C WoT specification. In particular, we discuss both an asynchronous architecture and a less resource-consuming synchronous one.

Keywords: IoT · WoT · semantic WoT · constrained objects · servient architecture · embedded reasoning · distributed reasoning.

1 Introduction

Internet of Things (IoT) applications often leverage various distributed devices, such as sensors, actuators and intermediate network nodes. Semantic Web technologies aim to bridge the interoperability gap among these devices, giving birth to the Semantic Web of Things (SWoT). SWoT servients[1] allow to describe, discover and use heterogeneous devices as well as to process application data at the semantic level. On edge and fog computing infrastructures, applications optimize the use of the more or less constrained devices computing capabilities [2] and reduce network load by processing data as close as possible to their sources and/or destinations, instead of transferring them back and forth to distant platforms [9].

Stream reasoning (SR) has ingredients from Data Stream Management Systems, Complex Event Processors (CEPs), Knowledge Representation (KR) and Semantic Web (incl. reasoning) [4,5]. SR usually relies on time windows and/or fades data as they get old. This is not the case for incremental reasoners. IMaRS [6] performs the incremental maintenance of window materializations over RDF streams. IMaRS assumes that RDF streams consist of timestamped triples; it processes triples using a time window. The approach extends the DRed algorithm [8] by adding an expiration timestamp to each fact. Cascade reasoning [3,1] uses a hierarchy of reasoners. At lower levels, high frequency data are

[1] https://www.w3.org/TR/wot-architecture/#sec-servient-implementation

filtered with very less reasoning in order to reduce the data frequency. At higher levels, more complex reasoning is performed on the filtered data stream.

In the Constrained Semantic Web of Things (CoSWoT) project[2], we aim at pushing incremental rule-based reasoning on to the nodes of edge or fog infrastructures. This paper addresses design questions for the modular CoSWoT servient architecture and for the reasoning components in servients.

2 Possible architectures for constrained stream reasoning

2.1 Servient architecture

In the W3C WoT Architecture specification, a servient is an abstract entity of a WoT platform, which comprises components dedicated to several concerns including behavior, interactions / API, security, networking, etc. CoSWoT relies on the deployment of servients on devices with a variety of computing, networking and energy capabilities.

All CoSWoT servients are semantic by nature and embed a local in-memory Knowledge Base (KB). Servients are implemented in an asynchronous manner that relies on a Semantic Service Bus (SSB) component. Other components can post RDF fragments to the SSB message queue and subscribe to the SSB fact pattern registry. Only the SSB is allowed to write RDF facts into the KB. Other components can be notified of KB updates (insert or delete) and are allowed to read it. According to the device architecture capabilities, asynchronism can be implemented using microservices, OS multitasking, or a basic on-purpose event loop. This way, CoSWoT applications can perform various non-blocking tasks with little overhead.

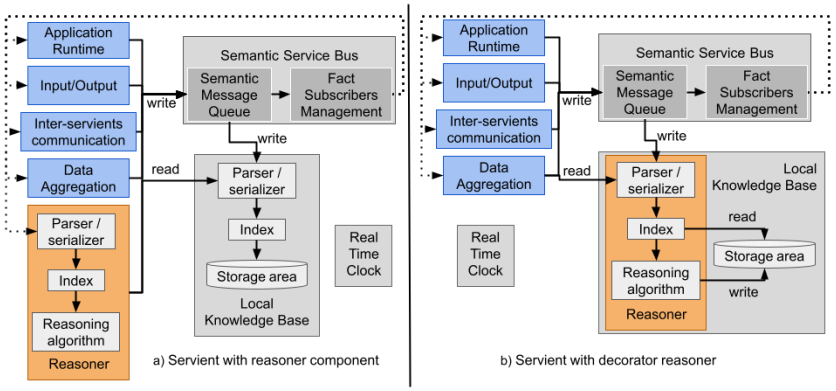

Fig. 1. Two possible CoSWoT servient architectures.

The other optional components are:

– *Application Runtime:* executes (part of) the application control flow, which consists in coordinating the other components behaviors. Decisions may be taken according to the reasoner outputs.

[2] `https://coswot.gitlab.io/`

- *Input/Output:* interacts with sensors by either querying data or handling data streams and with actuators by calling their APIs.
- *Inter-Servients Communication:* handles the distributed aspects of the application, including exchanging semantic data with persistence systems and communicating with other servients to collaboratively perform application and reasoning tasks.
- *Data Aggregation:* generates new explicit facts to be added to the KB (and removes old ones) by performing timely operations on raw measurement data such as sums or averages on sliding temporal windows [5,12]. This component could be directly derived from existing work in CEP such as [10].
- *Reasoner:* the reasoner is detailed hereafter.
- *Real Time clock:* to timestamp RDF fragments, perform aggregations, add timeouts to processing tasks and synchronize with other servients, we assume that each device possesses its own Real-Time clock. All components can access this clock, even if this is not represented on the figures for the sake of clarity.

2.2 Reasoner architecture

We identified two ways to include the reasoner in a CoSWoT servient.

The reasoner as an autonomous component (Fig. 1a): in this configuration, the reasoner is asynchronous and decoupled from the KB, like other components. It is an optional component that one can easily place or not on a given object. This configuration requires a publish-subscribe API mechanism as well as the duplication of some submodules, namely parser/serializer and index. From the SR point of view, asynchronism can prevent the reasoner from blocking the servient, and can also be used to skip reasoning tasks when overloaded. However, this can hamper application execution if it depends on implicit facts that are not synchronized with the explicit ones in the KB.

The reasoner as a decorator (Fig. 1b): the reasoner is part of the KB component and exposes the same API [7]. This approach ensures consistency with the KB, as both explicit and implicit facts are updated simultaneously. It also avoids duplication of parsing/serializing and indexing mechanisms. It also makes filtering easier within the reasoning task. However, this solution is synchronous and requires all reasoning tasks to be performed before writing facts in the KB. Hence it can block the servient when the frequency and complexity of updates increases.

3 Discussion

Given the two possible architectures described above, the component-based architecture requires the duplication of the parser/serializer, the index and the addition of a message handling function. It implies more resource consumption

than the decorator-based architecture regarding code size, messages management, etc. Nevertheless, this section examines whether it can bring interesting trade-offs in terms of optimization and ways to handle data overflow.

When reasoning tasks become too time-consuming, the event loop can be optimized with specific low-cost strategies such as task prioritization, turn-taking or event timeouts. We are also currently working on how to split the reasoning algorithm itself into several sub-tasks that could be separately pushed into the loop message queue.

The next question is how to handle sensor data overflow through an asynchronous event loop. In case of reasoning overflow, explicit and implicit facts stored in the KB may not correspond: an implicit fact may be inserted after the deletion of one of its explicit causes. As other components are supposed to base their decisions on implicit facts, such problems may cause latency issues, causing the servient to react to events after they are outdated. One possible solution to mitigate this issue is to avoid recomputing incremental maintenance in reoccurring situations, as done in [11]. This can for example be done in conjunction with the aggregation component, which decides which explicit facts to insert and delete in the KB.

When data stream velocity becomes even more intense, there is not even enough time to synchronize the local KB with all explicit facts. In such cases, some clues include a feedback loop to the I/O manager if the sensors can handle it, or even to switch off some other components. Another possible adaptation removes data processing tasks from a too constrained node and sends it to another more powerful servient.

4 Conclusion

In this paper, we analyse two possible servient architectures to deploy reasoning capabilities onto constrained objects. One is totally asynchronous and favors ease of deployment and error-proneness, while the other spares device resources and favors coherence between the reasoner and the KB. We are currently developing both prototypes so as to evaluate their behavior in different conditions of data stream overflow. Along with these architectural concerns, we are also working on optimizations that fit incremental reasoners embedded in constrained objects, as well as on the distribution of reasoning tasks among collaborating servients.

Acknowledgment

This work has been funded by the French National Agency for Research (ANR) under reference grant ANR-19-CE23-0012.

References

1. Bonte, P., Tommasini, R., Della Valle, E., De Turck, F., Ongenae, F.: Streaming massif: cascading reasoning for efficient processing of iot data streams. Sensors **18**(11), 3832 (2018)

2. Bormann, C., Ersue, M., Keranen, A.: Terminology for constrained-node networks. Internet Engineering Task Force (IETF): Fremont, CA, USA pp. 2070–1721 (2014)
3. De Brouwer, M., Ongenae, F., Bonte, P., De Turck, F.: Towards a cascading reasoning framework to support responsive ambient-intelligent healthcare interventions. Sensors **18**(10) (2018), https://www.mdpi.com/1424-8220/18/10/3514
4. Della Valle, E., Dell'Aglio, D., Margara, A.: Taming velocity and variety simultaneously in big data with stream reasoning: tutorial. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems. pp. 394–401 (2016)
5. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science **1**(1-2), 59–83 (2017)
6. Dell'Aglio, D., Della Valle, E.: Imars: Incremental materialization for rdf streams. Tutorial on Stream Reasoning for Linked Data at ISWC 2014 (2014)
7. Gamma, E.: Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Mass. : Addison-Wesley (1995)
8. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. ACM SIGMOD Record **22**(2), 157–166 (1993)
9. Maarala, A.I., Su, X., Riekki, J.: Semantic reasoning for context-aware internet of things applications. IEEE Internet of Things Journal **4**(2), 461–473 (2016)
10. Ren, H., Anicic, D., Runkler, T.: The synergy of complex event processing and tiny machine learning in industrial iot. arXiv preprint arXiv:2105.03371 (2021)
11. 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)
12. Tommasini, R., Della Valle, E.: Yasper 1.0: Towards an rsp-ql engine. In: International Semantic Web Conference (Posters, Demos & Industry Tracks) (2017)