Integrated Planning and Execution on Read-Write Linked Data

Nico Aßfalg[a], Helen Schneider[a] and Tobias Käfer[a]

[a]*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

Abstract

We present (1) a virtual flamethrower factory as an abstraction of a real-world industrial setting that provides Read-Write Linked Data interfaces to devices, and (2) an agent that operates in this environment with the aim to produce flamethrowers. The interfaces allow for (1) obtaining information about state and capabilities of the devices, and (2) controlling the devices. The agent can (1) plan and (2) execute workflows. The agent is written in the Notation 3 language with ASM4LD semantics, can perform Hierarchical Task Network planning, and can execute workflows described in the WiLD ontology.

Video: https://nico-assfalg.de/semantic-web/HTN-WiLD_Demo.mp4
Code: https://github.com/nico1509/htn-wild

Introduction

Under the impression of the recently standardised Web of Things descriptions[1], which are a step towards read access to descriptions of (Internet of Things) device capabilities and read-write access to such devices' state and functionality using Linked Data and other web technologies, we want to provide a showcase on the "demand" side for such possibilities for access to devices. Also outside of the Internet of Things, read-write capabilities to state maintained in Linked Data is on the rise around the SoLiD project[2]. Our showcase is therefore based on Read-Write Linked Data access to devices. As the stack of Linked Data is all about interoperability, facilitated in part using reasoning, e. g. to flexibly swap components that speak different vocabularies, we want rule-based reasoning as part of our showcase. To make use of the functional descriptions and write access, we want in our showcase to add to this traditional setting (Linked Data and rule-based reasoning) an agent that plans a workflow (from the functional descriptions and a goal) and executes this workflow.

Technologically, we therefore built an agent that uses HTTP for communicating with devices, where messages are described in RDF. We use rules to express reasoning. To describe the agent's behaviour, we employ a workflow language that can be executed on Linked Data, specifically the WiLD ontology [1], whose operational semantics can also get expressed in a rule language,

All the Agents Challenge (ATAC 2021)
nico.assfalg@student.kit.edu (N. Aßfalg); tobias.kaefer@kit.edu (T. Käfer)
https://nico-assfalg.de/ (N. Aßfalg); https://www.aifb.kit.edu/web/Tobias_Käfer (T. Käfer)
0000-0003-0382-7655 (N. Aßfalg); 0000-0003-0576-7457 (T. Käfer)

[1]https://www.w3.org/TR/wot-thing-description/
[2]http://solidproject.org/

ASM4LD [2]. Lastly, our agent makes use of an ontology to express planning problems according to the Hierarchical Task Network (HTN) method [3], for which we formulated operational semantics in ASM4LD. Conveniently, all different rule aspects of this agent can be expressed in the same syntax, Notation3, and executed on the same interpreter, Linked Data-Fu [4].

As a setting, we chose manufacturing, where interoperability is a challenge and the increasing requirement for small lot sizes requires flexibility. To simplify the prototype development, we work in a virtual setting based on Factorio, a video game. Factorio[3] is all about automation and the efficient and flexible production of various goods. Tech-trees and resource distribution in the virtual environments require you to repeatedly adapt your strategy. One of the less complex products in the game is a flamethrower, using which (1) enemies can be fought who want to destroy the manufacturing line or (2) trees can get burned down to make room for extensions of the manufacturing line. To build a flamethrower in Factorio, you need a set of iron gearwheels and steel plates which themselves are made using raw iron ore. In the game, you could go and do all the crafting by hand but it is cumbersome and not time-efficient. Therefore, you build a factory with ovens and other assembly machines, connected by robotic arms and transport belts. We want to make use of this game scenario, have re-built parts in Virtual Reality with Linked Data interfaces, and for our showcase, control it using an agent that builds and executes a workflow that produces a flamethrower. We have a virtual flamethrower manufacturing line which we know to work, which allows us to validate if the planned workflow works by checking if after execution a flamethrower is produced.

Related Work

Planning with semantic technologies has been investigated in different technology settings: SHOP2 [5] is an implementation of HTN, which has been used in [6] to compose Semantic Web Services (SWS). In SWS, function calls via POST requests are in the focus, whereas we build on state information provided via REST [7]. Similarly, [8] do planning on the semantic web, using proofs, and consider function calls, but as they never retract knowledge, they cannot work with scenarios in which function calls may contradict each other over time, whereas we determine the state afresh in each ASM step. [9] use an off-the-shelf AgentSpeak(L)-based [10] planner in a semantic manufacturing environment. Our approach however is an integrated planning and execution approach implemented on the web architecture.

ROSPlan [11] is also an integrated planning/execution approach, but made for the pub-sub ROS architecture, which is fundamentally different from REST [7]. ROSplan uses an ontology for the world state, which they update using ROS messages, whereas we assume semantic information provided by the environment itself. The integrated CX [12] is also built on a rule engine, but for a different environment. CX uses PDDL planning instead and has separate modules for world, planner, execution, and domain, whereas we combine all into one. Approaches based on Golog [13] such as [14] are also for a different assumption set: They are based on the history-based situation calculus, which is different from working with states [15].

[3]https://www.factorio.com/

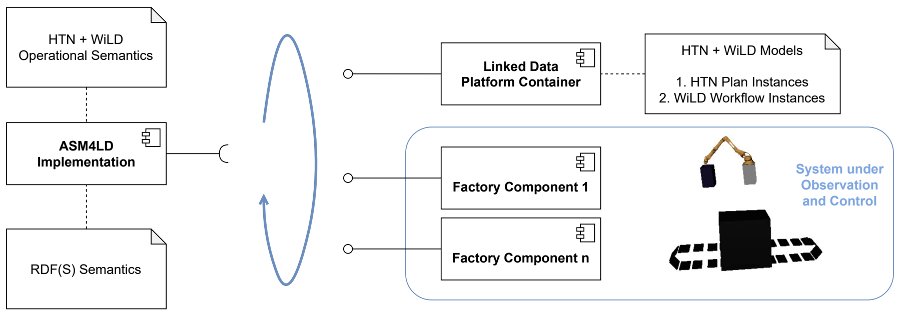
HTN + WiLD Operational Semantics
ASM4LD Implementation
RDF(S) Semantics
Linked Data Platform Container
HTN + WiLD Models
1. HTN Plan Instances
2. WiLD Workflow Instances
Factory Component 1
Factory Component n
System under Observation and Control

Figure 1: Our demo setup with the stateless interpreter (left) and components with Linked Data interface (right).

Virtual Manufacturing Environment

Our virtual factory shown in Figure 1 includes a set of robotic arms, transport belts and machines. All of them provide RESTful Linked Data interfaces and allow user agents to control them using unsafe HTTP requests. Moreover, we have descriptions of the device capabilities. The factory is built using the Java-based jMonkeyEngine[4] along with 3D models built and animated using Blender[5]. In the next section we will explain the details of our prototype.

HTN Planning for Linked Data

The theoretical foundation of our planner lies in Hierarchical Task Networks (HTN), first designed by Sacerdoti [3] and later formalized as UMCP by Erol et al. [16]. The key idea of HTN planning lies in the problem structure, where a goal is defined along with a set of methods that contain task networks, describing more specifically what needs to be done and which constraints apply. The tasks of the task networks may also need to be further decomposed using other methods until you have a set of primitive tasks in an order that can be executed.

However, our main goal was not to implement a full-blown HTN planner, but rather provide a set of rules that fulfil the core HTN decomposition principle, but with less technical complexity. There is no algorithm besides a processing engine that applies rules to a set of data, leading to easier implementation and acceptance through explainability.

We built an HTN vocabulary along with an operational semantics in the Notation3 language with ASM4LD semantics [17]. Using this approach, we can plan workflows in the WiLD [18] ontology for specifying workflows that can be executed on Linked Data. The operational semantics for WiLD can also be given in Notation3/ASM4LD. In our setup, see Fig. 1, an agent orchestrates the virtual devices along with a Linked Data Platform Container [19] that stores the vocabularies along with workflow models and instances. The agent runs the operational semantics for HTN planning and WiLD workflow execution using the Linked Data-Fu[6] interpreter. The code, including an HTN problem description, along with setup instructions is available

[4]https://jmonkeyengine.org/
[5]https://www.blender.org/
[6]https://linked-data-fu.github.io/

online, see the links on Page 1. In the following we want to give you an overview of how HTN works along with WiLD [18] and where it provides flexibility.

Notation

First, we need to introduce basic technologies from our environment Read-Write Linked Data.

We assume the data transferred in the message body of HTTP requests to be given using the Resource Description Framework (RDF) [20]. RDF is a graph-based data model, where the data is encoded in $(subject, predicate, object)$ triples. With $\mathscr{U}$ as the set of all URIs, $\mathscr{B}$ the set of all blank nodes, and $\mathscr{L}$ the set of all literals (e. g. strings, numerical values), triples are restricted as follows: $(subject, predicate, object) \in \mathscr{U} \cup \mathscr{B} \times \mathscr{U} \times \mathscr{U} \cup \mathscr{B} \cup \mathscr{L}$. In this paper, we use a logical notation for RDF. As a triple encodes a binary predicate, we use $predicate(subject, object)$ to talk about a triple. As triples in RDF are valid conjunctively, we use the logical and ($\wedge$) to connect triples to a graph. We use Uniform Resource Identifiers (URIs) [21] to denote things, abbreviated as CURIEs[7] and follow the abbreviations from prefix.cc[8]. We thus write a triple as follows:

$$\textit{rdf:type}(\textit{ex:produce-flamethrower-method}, \textit{:Method})$$

As *rdf:type* assigns things to a class, we use the unary predicate for a class assignment:

$$\textit{:Method}(\textit{ex:produce-flamethrower-method})$$

HTN Planning of our Manufacturing Workflow

Before we add modifications, we need to (simplifying) present how the planning works for our manufacturing workflow.

We can manufacture a flamethrower by executing tasks, which we identify using URIs. The tasks need to be in a specific order (workflow). The tasks can be executed as HTTP requests to a manufacturing device, which we can describe in RDF. We thus need to embed the requests in a structure that tells us to execute the requests (we call it primitive task).

$$\begin{gathered}\textit{:PrimitiveTask}(\textit{ex:assemble-plate})\\ \wedge\ \textit{:hasHttpRequest}(\textit{ex:assemble-plate}, \textit{ex:assemble-plate-request})\\ \wedge\ \textit{:PrimitiveTask}(\textit{ex:assemble-gearwheel})\\ \wedge\ \textit{:hasHttpRequest}(\textit{ex:assemble-gearwheel}, \textit{ex:assemble-gearwheel-request})\\ \wedge\ \textit{:PrimitiveTask}(\textit{ex:produce-flamethrower})\\ \wedge\ \textit{:hasHttpRequest}(\textit{ex:produce-flamethrower}, \textit{ex:produce-flamethrower-request}) \wedge \ldots\end{gathered}$$

Next, we need a another structure to determine the tasks' order. We use a network of tasks based on the (linked) RDF list and some enclosing RDF for that purpose:

$$\textit{:TaskNetwork}(\textit{ex:task-network})$$

[7]https://www.w3.org/TR/curie/

[8]http://prefix.cc/. We use the empty prefix to denote https://purl.org/uberq/htn/vocab# and the prefix ex: to denote http://example.org/#. An underscore in prefix position indicates a blank node.

$$\begin{gathered}
\wedge\ \mathit{:hasTaskList}(\mathit{ex:task\text{-}network}, \mathit{\_:li1}) \\
\wedge\ \mathit{rdf:first}(\mathit{\_:li1}, \mathit{ex:assemble\text{-}plate}) \wedge\ \mathit{rdf:rest}(\mathit{\_:li1}, \mathit{\_:li2}) \\
\wedge\ \mathit{rdf:first}(\mathit{\_:li2}, \mathit{ex:assemble\text{-}gearwheel}) \wedge \mathit{rdf:rest}(\mathit{\_:li2}, \mathit{\_:li3}) \\
\wedge\ \mathit{rdf:first}(\mathit{\_:li3}, \mathit{ex:produce\text{-}flamethrower}) \wedge \mathit{rdf:rest}(\mathit{\_:li3}, \mathit{rdf:nil})
\end{gathered}$$

The question a planner has to answer is to come up with such an order, i. e. workflow.

To determine such an order, a planner needs descriptions of the outcome of each task. We model the outcome as a postcondition (in HTN, this is called Literal) in the form of a SPARQL ASK query [22] in SPIN notation[9]. Such literals can be regarded as the goals the tasks fulfil.

$$\mathit{sp:Ask}(\mathit{ex:literal\text{-}flamethrower}) \wedge\ \mathit{sp:where}(\mathit{ex:literal\text{-}flamethrower}, \dots) \wedge \dots$$

The task network can now be regarded as fulfilling the goal of manufacturing the flamethrower.

$$\mathit{:Goal}(\mathit{ex:goal\text{-}flamethrower}) \wedge\ \mathit{:hasLiteral}(\mathit{ex:goal\text{-}flamethrower}, \mathit{ex:literal\text{-}flamethrower})$$

The method to achieve the goal now holds the task network, to be decomposed by the planner:

$$\begin{gathered}
\mathit{:Method}(\mathit{ex:flamethrower\text{-}method}) \\
\wedge\ \mathit{:forGoalTask}(\mathit{ex:flamethrower\text{-}method}, \mathit{ex:goal\text{-}flamethrower}) \\
\wedge\ \mathit{:hasTaskNetwork}(\mathit{ex:flamethrower\text{-}method}, \mathit{ex:task\text{-}network})
\end{gathered}$$

Modification

Now imagine we also want to manufacture a more precise flamethrower, by attaching mid-range optics to it. We could define a completely new task network. Instead, we leverage the hierarchy of HTN and reuse our goal along with the method. We define additional goal tasks and primitive tasks, e. g. for a precise flamethrower. Now we can define a method to create the precise flamethrower, which re-uses *ex:goal-flamethrower* from earlier.

$$\begin{gathered}
\mathit{:Method}(\mathit{ex:precise\text{-}flamethrower\text{-}method}) \\
\wedge\ \mathit{:forGoalTask}(\mathit{ex:precise\text{-}flamethrower\text{-}method}, \mathit{ex:goal\text{-}precise\text{-}flamethrower}) \\
\wedge\ \mathit{:hasTaskNetwork}(\mathit{ex:precise\text{-}flamethrower\text{-}method}, \mathit{\_:tn}) \\
\wedge\ \mathit{:hasTaskList}(\mathit{\_:tn}, \mathit{\_:tl1}) \\
\wedge\ \mathit{rdf:first}(\mathit{\_:tl1}, \mathit{ex:goal\text{-}flamethrower}) \wedge \mathit{rdf:rest}(\mathit{\_:tl1}, \mathit{\_:tl2}) \\
\wedge\ \mathit{rdf:first}(\mathit{\_:tl2}, \mathit{ex:attach\text{-}optics}) \wedge \mathit{rdf:rest}(\mathit{\_:tl2}, \mathit{rdf:nil})
\end{gathered}$$

Goal *ex:precise-flamethrower-goal* can then get decomposed into *ex:goal-flamethrower* and the primitive task *ex:attach-optics*. Further decomposition gives us the new workflow.

Conclusion

We presented an agent that does integrated planning and execution of workflows in Linked Data. We showcased the agent in a virtual manufacturing setting and outlined how the set of technologies can facilitate flexibility, next to the flexibility gained by using semantic reasoning.

[9]http://spinrdf.org/

References

[1] T. Käfer, A. Harth, Specifying, monitoring, and executing workflows in linked data environments, in: Proceedings of the 17th International Semantic Web Conference (ISWC), 2018b, pp. 424–440.

[2] T. Käfer, A. Harth, Rule-based programming of user agents for linked data, in: Proceedings of the 11th International Workshop on Linked Data on the Web (LDOW) at the Web Conference (27th WWW), 2018a.

[3] E. D. Sacerdoti, Planning in a hierarchy of abstraction spaces, Artificial intelligence 5 (1974) 115–135.

[4] S. Stadtmüller, S. Speiser, A. Harth, R. Studer, Data-fu: a language and an interpreter for interaction with read/write linked data, in: Proceedings of the 22nd International World Wide Web Conference (WWW), 2013, pp. 1225–1236.

[5] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, S. Mitchell, Total-order planning with partially ordered subtasks, in: IJCAI, volume 1, 2001, pp. 425–430.

[6] D. Wu, E. Sirin, J. Hendler, D. Nau, B. Parsia, Automatic web services composition using SHOP2, Technical Report, University of Maryland, USA, 2006.

[7] R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. thesis, University of California, Irvine, USA, 2000.

[8] R. Verborgh, D. Arndt, S. van Hoecke, J. de Roo, G. Mels, T. Steiner, J. Gabarro, The pragmatic proof: Hypermedia api composition and execution, Theory and Practice of Logic Programming 17 (2017) 1–48. doi:10.1017/S1471068416000016.

[9] A. Ciortea, S. Mayer, F. Michahelles, Repurposing manufacturing lines on the fly with multi-agent systems for the web of things, in: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), 2018, pp. 813–822.

[10] A. S. Rao, Agentspeak (l): Bdi agents speak out in a logical computable language, in: European workshop on modelling autonomous agents in a multi-agent world, Springer, 1996, pp. 42–55.

[11] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtós, M. Carreras, Rosplan: Planning in the robot operating system, in: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2015, pp. 333–341. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10619.

[12] T. Niemueller, T. Hofmann, G. Lakemeyer, Goal reasoning in the CLIPS executive for integrated planning and execution, in: Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2019, pp. 754–763. URL: https://aaai.org/ojs/index.php/ICAPS/article/view/3544.

[13] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. B. Scherl, GOLOG: A logic programming language for dynamic domains, J. Log. Program. 31 (1997) 59–83. URL: https://doi.org/10.1016/S0743-1066(96)00121-5. doi:10.1016/S0743-1066(96)00121-5.

[14] T. Hofmann, T. Niemueller, J. Claßen, G. Lakemeyer, Continual planning in golog, in: Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI Press, 2016, pp. 3346–3353. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12435.

[15] R. Reiter, The situation calculus ontology, Electronic News Journal on Reasoning about

Actions and Change 2 (1998). URL: http://www.ep.liu.se/ej/enrac/1997/.

[16] K. Erol, J. A. Hendler, D. S. Nau, Umcp: A sound and complete procedure for hierarchical task-network planning., in: AIPS, volume 94, 1994, pp. 249–254.

[17] T. Käfer, A. Harth, Rule-based programming of user agents for linked data, in: LDOW@ WWW, 2018.

[18] T. Käfer, A. Harth, Specifying, monitoring, and executing workflows in linked data environments, in: International Semantic Web Conference, Springer, 2018, pp. 424–440.

[19] S. Speicher, J. Arwe, A. Malhotra, Linked data platform 1.0, 2015. URL: https://www.w3.org/TR/ldp, w3C Recommendation.

[20] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, Recommendation, W3C, 2014. http://www.w3.org/TR/rdf11-concepts/.

[21] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, RFC 3986, IETF, 2005. URL: http://www.ietf.org/rfc/rfc3986.txt, updated by RFCs 6874, 7320.

[22] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, Recommendation, W3C, 2013. http://www.w3.org/TR/sparql11-query/.