SoftwareX 27 (2024) 101779

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

SH: Service-oriented system for HTN planning in real-world domains

Ilche Georgievski *, Akash V. Palghadmal, Ebaa Alnazer, Marco Aiello

Service Computing Department, IAAS, University of Stuttgart Universitätsstraße 38, 70569 Stuttgart, Germany

ARTICLE INFO

Keywords:
HTN
AI planning
Real-world domains
AI engineering

ABSTRACT

SH is an open-source and domain-independent system for Hierarchical Task Network (HTN) planning, a branch of Artificial Intelligence Planning and Scheduling. SH adopts a modular architectural design that enables flexibility, extensibility, and maintainability. The planning system supports representing complex planning problems and offers various capabilities, including problem parsing, problem grounding, plan generation, and integration, making SH suitable for addressing real-world challenges. The combination of these features puts SH in a unique position, offering researchers and practitioners the power to adopt and tailor SH to meet their specific requirements coming from real-world applications.

Code metadata

Current code version	v2.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00133
Permanent link to Reproducible Capsule	NA
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Scala
Compilation requirements, operating environments & dependencies	https://github.com/PlanX-Universe/sh/wiki/Setting-Up-the-SH-Planning-System
If available Link to developer documentation/manual	https://github.com/PlanX-Universe/sh/wiki/
Support email for questions	planx.universe@gmail.com

1. Motivation and significance

Artificial Intelligence (AI) Planning is a vital branch of AI dedicated to addressing planning problems and encompasses several primary tasks, including the creation and representation of planning problems, the design and analysis of algorithms to solve these problems, and the engineering of AI planning systems. Among the AI planning approaches, Hierarchical Task Network (HTN) planning stands out as a popular one due to its support for representing rich domain knowledge, intuitive domain modelling, and the potential of finding plans quickly [1]. The central concept is a task network consisting of primitive and compound tasks. Primitive tasks are actions that can be directly executed in the world, while compound tasks encapsulate domain knowledge extending beyond basic world-altering actions. Compound tasks can be further decomposed into other tasks. A user's objective is modelled as an initial task network, which is automatically decomposed starting from an initial world state using the knowledge about primitive and compound tasks. HTN planning has found applications in diverse domains, from games [2] to smart buildings [3], autonomous vehicles [4], and cloud deployments [5].

The area of HTN planning has seen the emergence of several software tools, such as the pioneering planners NOAH [6], Nonlin [7], SIPE-2 [8], O-Plan2 [9], and UMCP [10]. While these planners made significant contributions to the area, their software distributions have become unavailable or obsolete. More contemporary HTN planners include SHOP2 [11], SIADAX [12], HyperTensioN [13], PANDA [14], PDDL4J [15], Lilotane [16], and pyHiPOP [17]. These planners have primarily been put to the test in benchmark settings, showcasing their effectiveness and efficiency, albeit in tackling only narrow, very specific problems.

We introduce SH, an open-source HTN planning system that marks a new milestone in the field of AI planning. It embodies a modular design

* Corresponding author.
E-mail addresses: ilche.georgievski@iaas.uni-stuttgart.de (Ilche Georgievski), akashpalghadmal2508@gmail.com (Akash V. Palghadmal), ebaa.alnazer@iaas.uni-stuttgart.de (Ebaa Alnazer), marco.aiello@iaas.uni-stuttgart.de (Marco Aiello).

https://doi.org/10.1016/j.softx.2024.101779
Received 1 March 2024; Received in revised form 10 May 2024; Accepted 27 May 2024
Available online 12 June 2024
2352-7110/© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

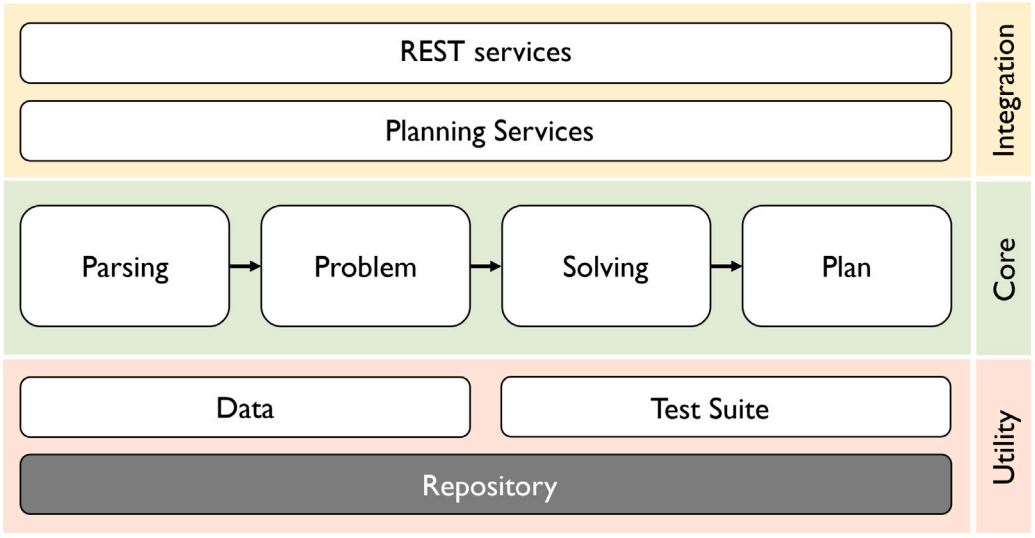


Fig. 1. Layered Modular Architecture of the SH planning system.

that prioritises expressiveness on the one hand, and flexibility, extendability, and maintainability, on the other hand. Its standout features are the support for modelling and reasoning about complex planning problems, robust performance, and provision of core functionalities as services, allowing SH to serve as a standalone tool or to be integrated into larger systems. SH grounds the planning problems during planning when it uses totally ordered task decomposition while following a depth-first search strategy.

2. Software description

SH is a domain-independent planning system based on state-based HTN planning [1]. It is designed to serve as an open-source system for addressing complex planning problems. Its key capabilities include problem parsing, grounding, plan generation, and integration, all tailored to facilitate real-world applications. SH features a modular architecture that adheres to the principles of Service-Oriented Computing (SOC) [18]. Furthermore, SH is entirely implemented in the Scala programming language [19]. The combination of SOC design principles and Scala enables not only modularity, flexibility, and maintainability of SH's components but also conciseness of the SH's implementation and robustness, efficiency, and scalability of the system.

2.1. Software architecture

SH's architecture is layered and composed of nine modules, as illustrated in Fig. 1. The bottom layer consists of the Repository, Data, and Test Suite modules, enabling reading, storing, and configuring planning data and testing the core planning functionalities. The Core layer comprises Parsing, Problem, Solving, and Plan modules. The Parsing module enables parsing planning problems, while the Problem module handles an HTN planning problem's representation at the programming level. The Solving module encompasses the functionalities for plan generation, space search, and grounding. The Plan module includes the programming construct representing the solution to a planning problem. The top layer features the Planning Services and REST Services modules, offering functionalities for system utilisation and integration. The subsequent section details the key functionalities and capabilities of the modules at the Core and Integration layers.

2.2. Software functionalities and capabilities

Specifying planning problems in HPDL. The adoption of a well-defined and understandable language for planning problem specification holds significance because it facilitates the research on planning system development, lowers the learning curve, and enhances the adoptability of planning systems. In this context, SH supports specifying HTN planning problems using the Hierarchical Planning Definition Language (HPDL) [20].[1] HPDL bears a resemblance to the planning language utilised by SIADEX. It retains the elements essential to the definition of state-based HTN planning while omitting modelling constructs exclusive to the SIADEX planner. HPDL mandates the specification of HTN planning problems through two .HPDL files: one holds a description of the domain model, while the other is a description of a specific problem instance. Designing an HPDL domain model and corresponding problem instances for use with the SH planning system requires comparable effort and expertise as modelling planning problems in other planning languages, such as the Planning Domain Definition Language (PDDL) [21]. It is important for potential users to recognise that the challenges of modelling planning problems, which is generally acknowledged as a difficult task [22], are inherent to the nature of planning problem formulation itself, rather than a specific burden imposed by our system. The SH planning system utilises the HPDL format to enable HTN planning, but it does not simplify or complicate the task of planning problem modelling. Users experienced in developing domain and problem files in any planning language should find the modelling process for SH familiar and directly comparable.

Parsing and converting. SH manages HPDL files through a structured process encompassing three phases. When an HPDL file is read into the system as a string, SH starts with lexical analysis, which breaks down the string into a sequence of tokens based on regular expressions. Then, parsing determines whether the tokens conform to the formal grammar of HPDL. Finally, the third step is the conversion of the parsed results into Scala case classes, each representing HTN planning constructs. We implement these tasks utilising parser combinators. Within this framework, a parser is a function that transforms tokens into meaningful programming-level constructs, and a parser combinator is a combination of simple parsers into more powerful parsers. Leveraging the concept of parser combinators and Scala, we have a unique capability — pattern matching directly against the parsed results, ultimately resulting in the creation of Scala objects. Additionally, parser

[1] https://github.com/PlanX-Universe/sh/wiki/HPDL

combinators support the modularity, flexibility, and composability of our system, enabling the SH's parsing functionality to remain organised, transparent, maintainable, and extendable (e.g., as HPDL evolves).

Reasoning for real-world domains. Expressiveness encompasses the ability of a planning system to handle diverse domain characteristics. SH can excel in reasoning across a wide range of domains because it supports various expressiveness features: conjunction of formulas in a task's preconditions and predicates in action's effects; negation, disjunction of formulas and implication in a task's preconditions; universal quantification in both preconditions and effects; numeric constraints in preconditions and numerical effects; axioms; recursive tasks in the domain model; dynamic objects (i.e., creating and destroying objects during planning); and parameter typing. Explanations and formal definitions of these features are available elsewhere, e.g., [1,23]. SH employs a clear separation for implementing the logical and numerical expressions, utilising dedicated case classes for each expressiveness construct and a Scala object for numeric calculations.

Plan generation. SH's core functionality is plan generation. The plan generation algorithm takes as input an initial state, a list of compound tasks, a list of actions, the initial task network, and the number of plans to be generated. The algorithm combines ordered task decomposition [1], depth-first search [24], and on-the-fly grounding (see the next paragraph), ensuring effective and efficient planning for a diverse range of domains and scenarios.

Grounding. In contrast to many existing planning systems, our approach does not compute all possible groundings for a planning problem before the planning process initiation. Instead, we use lifted representation and perform predicate grounding and task instantiation during planning, addressing only the grounding needs at hand. We achieve dynamic grounding and efficient access to predicates in the state through the use of a predicate-property map. To ground a predicate, we retrieve the predicate's associated property map to identify predicates where a specific argument already has a binding. This method minimises the number of bindings that need to be checked when iterating over possible bindings. Task instantiation, which involves assigning values to variables representing a task's parameters, is accomplished similarly to predicate grounding. We implement variables using Scala Symbols, which are interned — Symbols with identical names are referred to the same Symbol object instance, making them reference equals. This ensures that all HTN constructs will reference the same object instance associated with the first match of the Symbol with the object instance.

Integration. We put a strong emphasis on modularity and separation of concerns in the design and implementation of the SH planning system. This principled engineering approach allows us to incorporate a collection of diverse services. SH currently provides 8 planning services for parsing planning problems, creating Scala objects for domain models and problem instances, storing domain models and problem instances, and solving planning problems. Using these services, SH can be employed directly in a standalone mode, integrated within larger systems, or deployed as an accessible Web service. The system can be packaged as a .jar file for direct deployment. In the context of Web services, we have implemented several SH's planning services as Representational State Transfer (REST) resources [25] and provide a lightweight, asynchronous HTTP server for hosting these REST services, all achieved using Akka HTTP.[2] Our design choice ensures that SH's architecture maintains its independence and adaptability while leveraging powerful technologies for service delivery. Detailed information and instructions on how to integrate the SH planning system are available in the GitHub repository.

3. Illustrative examples

The SH planning system is a domain-independent platform, with all its components designed to be generic and not tailored to any application domain. As such, SH is versatile and can be applied across a broad spectrum of use cases. While not exhaustive, the following categories represent a comprehensive list of usage areas: (1) problem repository management, (2) domain model verification, (3) problem-solving, (4) performance comparison, (5) system integration, and (6) education. We delve into two specific examples that illustrate the third and fourth use cases. Instruction on how to reproduce these examples together with the domain models and problem instances are available in the GitHub repository.[3] All tests demonstrated next were run on a Windows 11 machine using an Intel processor running at 2.3 GHz coupled with 16 GB of RAM.

Solving cloud deployment problems. As an example of the first use case, we show the utilisation of the SH planning system for solving cloud deployment problems. SH includes a domain model called *deployment* in its repository dedicated to solving deployment problems for cloud applications [5,26]. Consider, for example, a simple application consisting of a dashboard component that requires the functionalities of an Apache2 server and a Cassandra database. The problem instance representing the cloud deployment request for this dashboard application, given the *deployment* domain model, is shown in the left-hand part of Fig. 2. If we run SH's service PLANWITHGIVENDOMAINANDPROBLEMNAMEPRINTPLANS with the problem instance as input string, SH generates the output shown in the right-hand part of Fig. 2. The output contains the plan for deploying the dashboard application and some run statistics.

Performance comparison on the benchmark rover domain. To exemplify the other use case, we choose to demonstrate and compare the performance of SH on the *rover* domain. This domain, centred around planetary exploration via rovers, serves as a benchmark for characterising and comparing the performance of planners [27]. Our illustrative example consists of a set of evaluation tests on a single planning problem from this domain, requesting SH to solve it three times and averaging the results for plan generation time and memory usage. We instructed SH to generate different numbers of plans for the problem to characterise a system's scalability. Fig. 3 displays SH's performance, revealing its consistent robust performance, as evidenced by its ability to find 100 plans in under half a second. For performance comparison, we selected JSHOP2, a Java version of the state-of-the-art SHOP2 planner, also designed for state-based HTN planning [28]. We ran JSHOP2 under the same conditions and collected relevant data. Fig. 3 shows that SH exhibits higher planning times than JSHOP2 in scenarios where it is tasked with finding up to 10 plans. However, SH shows remarkable performance when the challenge is to generate larger quantities of plans, effectively outperforming JSHOP2. We hypothesise that these differences in planning time between SH and JSHOP2 can be attributed to their planning strategies and domain model configurations [29]. Specifically, JSHOP2 considers tasks during planning in the order they are provided in the domain model file. This strategy might optimise its performance for simpler planning problems where the modelled list of tasks aligns well with the planning process, potentially reducing the search space and leading to quicker solutions. In contrast, SH employs a non-deterministic strategy of selecting tasks, which does not rely on their ordering in the domain model. This strategy can be advantageous for more complex planning problems where task arrangements made during planning may reduce the need for extensive backtracking. Concerning memory usage, SH demonstrates its efficiency by consuming fewer resources compared to JSHOP2.

[2] https://doc.akka.io/docs/akka-http/current/index.html

[3] https://github.com/PlanX-Universe/sh/wiki/Illustrative-Examples

```
(define (problem p-dashboard)
 (:domain deployment)
 (:requirements :strips :equality
   :typing :fluents
  :negative-preconditions
   :universal-preconditions)
 (:objects httpd cass cass-up - port
  dashboard cassandra apache2 -
   component)
 (:init
  (installed-require dashboard httpd)
  (running-require dashboard httpd)
  (running-require dashboard cass-up)
  (installed-provide apache2 httpd)
  (installed-provide cassandra cass)
  (running-provide cassandra cass-up)
  (= (instance-number) 0))
 (:goal-tasks
  (sequence (run-configuration
   dashboard)))
)
```

```
Plan 1:
    1) createInstance(dashboard)
    2) createInstance(apach2)
    3) start(a1)
    4) bind(httpd,d0,a1)
    5) start(d0)
    6) createInstance(cassandra)
    7) start(c2)
    8) run(c2)
    9) bind(cass-up,d0,c2)
   10) run(d0)

--* Statistics *--
Parsing time: 968.0 ms
Plan-generation time: 63.0 ms
Memory usage: 133.169152 MB
Number of evaluated states: 34
Number of decomposed tasks: 24
```

Fig. 2. Simple cloud deployment problem and its plan computed by SH.

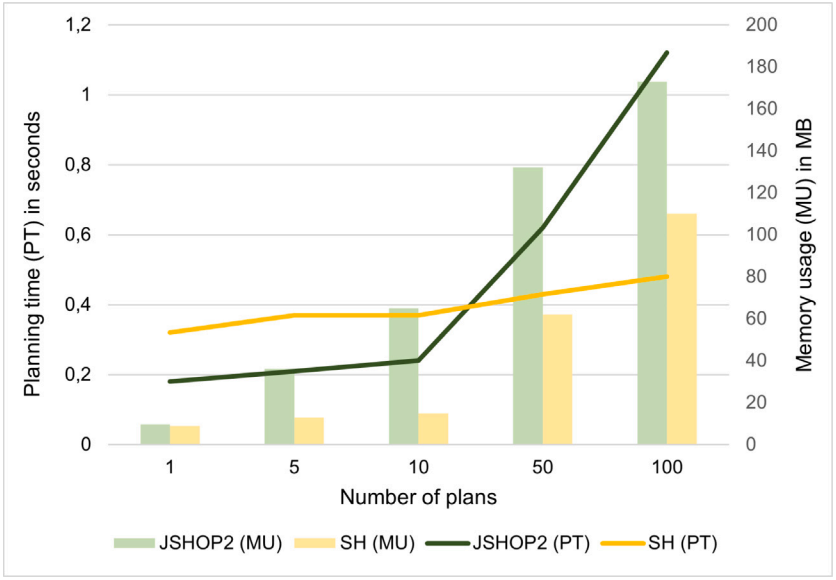


Fig. 3. SH's and JSHOP2's performance when solving a *rover*'s planning problem.

4. Impact

SH stands out as an HTN planning system equipped with a distinctive set of capabilities. These include the ability to represent and reason in real-world domains, modular parsing and seamless object conversion, quick grounding during planning, resource-efficient plan generation, and various mechanisms for use and integration. The combination of these capabilities may not only facilitate the research of underexplored directions, such as AI planning provenance, but also open up new research horizons, such as the consideration of risk when making planning decisions [30]. These attributes position SH as a candidate for addressing multifaceted planning challenges.

SH has played a crucial role in driving scientific research and innovation across several real-world domains. Notably, SH has been instrumental in the realisation of smart offices [31], smart open spaces within buildings [3], and cloud application deployment [5]. Additionally, SH's design philosophy has contributed to advancing methodological aspects of AI planning, namely the use of a standard-based syntax for specifying HTN planning problems [20], incorporation of numeric HTN planning [23], pioneering the idea of AI planning as a service [23,32], and laying the groundwork for planning system's engineering principles now inline with the emerging AI engineering field.

A major part of the development of SH has been at the core of the Energy Smart Offices project, funded by the Dutch Research Council under the Smart Energy Systems program,[4], the BIGS project, funded by the Dutch Research Council under the Smart Energy in Smart Cities

[4] https://www.nwo.nl/en/projects/647000004

program,[5] and the GreenerBuildings project, funded by the European Union under the European Seventh Framework Program.[6] Furthermore, SH has been one of the crucial components in a project funded by the University of Groningen under the Green Mind Award program.[7] Within this initiative, SH played a leading role in investigating the automated control of various aspects of the Bernoulliborg building at the University of Groningen [33]. Over a span of two years, SH was employed in regulating the lighting in the Bernoulliborg's restaurant as part of services offered by Sustainable Buildings, a spin-off company from the University of Groningen. This deployment of SH made a tangible impact by reducing energy consumption, cutting operational costs, and improving user satisfaction with the building automation. For specific metrics, refer to [3].

In the preceding section, we further demonstrated the use and performance of SH in both a real-world domain and a benchmark domain. With the domain of cloud application deployment, we show that SH can tackle real-world domains. To understand the impact of this, consider that while this domain was submitted for the 2020 International Planning Competition,[8] it was not included in the testing benchmarks because its expressiveness surpassed the capabilities of the participating state-of-the-art planners. Furthermore, through the benchmark *rover* domain and the comparative analysis with JSHOP2, we illustrated the competitive edge of SH's functionalities, particularly in terms of scalability and efficient memory usage. In this context, SH comes with a repository of domains, allowing researchers and practitioners to test and evaluate the effectiveness of SH and other HTN planners. Additionally, SH comes with a suite of unit tests, allowing for validation of the integrity and accuracy of its internal mechanisms.

SH has found its place in the PlanX Universe, a collection of resources for engineering AI planning systems.[9] It also plays an active role in teaching and research activities at the University of Stuttgart, with multiple students utilising SH in their projects. Our ongoing efforts are directed towards further enriching SH's capabilities and integrating SH into our living lab.

5. Conclusions

We have introduced SH, an open-source software system for HTN planning. SH distinguishes itself as a domain-independent planning system that solves state-based HTN planning problems expressed in first-order logic and functions. With its various capabilities, SH offers means for parsing planning problems described in HPDL, expressing and reasoning for real-world domains, quick grounding during planning, resource-efficient plan generation, and use and integration of its functionalities in other systems. A hallmark of SH is its modular software architecture that allows the replacement of individual components and extensions with alternative capabilities and new functionality. These features empower both academic and industrial users to tailor the planning system to their specific requirements. SH has already made its mark in both research and commercial settings, and it continues to be a valuable resource for ongoing research and educational activities.

[5] https://www.cs.rug.nl/ds/websites/bigs/

[6] https://greenerbuildings.eu/

[7] https://www.rug.nl/news/2012/04/object974926682?lang=en

[8] https://www.icaps-conference.org/competitions/

[9] PlanX Universe currently consists of the PlanX Toolbox, which is a set of services for composing classical planning systems [34], the PlanXFlow, which is a software development life cycle for engineering AI planning systems [35], and the SH planning system. See https://github.com/PlanX-Universe.

CRediT authorship contribution statement

Ilche Georgievski: Writing – review & editing, Writing – original draft, Visualization, Supervision, Software, Project administration, Methodology, Investigation, Data curation, Conceptualization. **Akash V. Palghadmal:** Writing – review & editing, Software. **Ebaa Alnazer:** Writing – review & editing, Software, Project administration. **Marco Aiello:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Akash V. Palghadmal reports financial support was provided by German Federal Ministry for Economic Affairs and Climate Action. Ilche Georgievski reports financial support was provided by Dutch Research Council. Ilche Georgievski reports financial support was provided by European Commission. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is included in the code repository.

Acknowledgements

We thank Pieter Noordhuis for implementing the initial program that served as a basis for developing the SH planning system and Alexander Lazovik for allowing us to work on that program. This work was partially supported by the German Federal Ministry for Economic Affairs and Climate Action via the project SofDCar (grant number 19S21002).

References

[1] Georgievski I, Aiello M. HTN planning: Overview, comparison, and beyond. Artificial Intelligence 2015;222:124–56.
[2] Humphreys T. Game AI pro 360: Guide to architecture. CRC Press; 2019 [chapter Exploring HTN planners through example].
[3] Georgievski I, Nguyen TA, Nizamic F, Setz B, Lazovik A, Aiello M. Planning meets activity recognition: Service coordination for intelligent buildings. Pervasive Mob Comput 2017;38(1):110–39.
[4] Alnazer E, Georgievski I, Prakash N, Aiello M. A role for HTN planning in increasing trust in autonomous driving. In: IEEE international smart cities conference. 2022, p. 1–7.
[5] Georgievski I, Nizamic F, Lazovik A, Aiello M. Cloud ready applications composed via HTN planning. In: IEEE international conference on service oriented computing and applications. 2017, p. 23–33.
[6] Sacerdoti ED. A structure for plans and behavior (Ph.D. thesis), Standford University, AI Center; 1975.
[7] Tate A. Project planning using a hierarchic non-linear planner. Tech. rep. 25, Department of Artificial Intelligence, University of Edinburgh; 1976.
[8] Wilkins DE. Can AI planners solve practical problems? Comput Intell 1991;6:232–46.
[9] Tate A, Drabble B, Kirby R. O-Plan2: An open architecture for command, planning and control. In: Intelligent scheduling. 1994, p. 213–39.
[10] Erol K, Hendler J, Nau DS. UMCP: A sound and complete procedure for hierarchical task network planning. In: International conference on AI planning systems. 1994, p. 249–54.
[11] Nau DS, Ilghami O, Kuter U, Murdock JW, Wu D, Yaman F. SHOP2: An HTN planning system. J Artif Int Res 2003;20(1):379–404.
[12] Asunción Mdl, Castillo L, Fernádez-Olivares J, García-Pérez O, González A, Palao F. SIADEX: An interactive artificial intelligence planner for decision support and training in forest fire fighting. Artif Intell Commun 2005;18(4):257–68.
[13] Magnaguagno MC, Meneguzzi FR, de Silva L. HyperTensioN: A three-stage compiler for planning. In: International planning competition: Planner and domain abstracts. 2021, p. 5–9.
[14] Höller D, Behnke G, Bercher P, Biundo S. The PANDA framework for hierarchical planning. KI - Künstliche Intell 2021;35:391–6.
[15] Pellier D, Fiorino H. PDDL4J: A planning domain description library for java. J Exp Theor AI 2018;30(1):143–76.

[16] Schreiber D. Lilotane: A lifted SAT-based approach to hierarchical planning. J Artif Int Res 2021;70:1117–81.
[17] Lesire C, Albore A. pyHiPOP – Hierarchical partial-order planner. In: International planning competition: Planner and domain abstracts. 2021, p. 13–6.
[18] Papazoglou MP, Georgakopoulos D. Introduction: Service-oriented computing. Commun ACM 2003;46(10):24–8.
[19] Odersky M, Spoon L, Venners B, Sommers F. Programming in Scala: A comprehensive step-by-step guide. 5th ed. Artima Incorporation; 2021.
[20] Georgievski I. HPDL: Hierarchical planning definition language. JBI preprint 2013-12-3, Uni. of Groningen; 2013.
[21] McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, Weld D, Wilkins D. PDDL - The planning domain definition language. Tech. rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control; 1998.
[22] McCluskey TL, Vaquero TS, Vallati M. Engineering knowledge for automated planning: Towards a notion of quality. In: Knowledge capture conference. 2017, p. 1–8.
[23] Georgievski I. Coordinating services embedded everywhere via hierarchical planning (Ph.D. thesis), University of Groningen; 2015.
[24] Russell SJ, Norvig P. Artificial intelligence: A modern approach. Pearson Education; 2003.
[25] Richardson L, Ruby S. Restful web services, 1st ed. O'Reilly; 2007.
[26] Georgievski I. HTN planning domain for deployment of cloud applications. In: International planning competition: Planner and domain abstracts. 2021, p. 34–46.
[27] Alnazer E, Georgievski I, Aiello M. On bringing HTN domains closer to reality - The case of satellite and rover domains. In: International conference on automated planning systems (ICAPS) workshop on scheduling and planning applications. 2022.
[28] Ilghami O. Documentation for JSHOP2. Tech. rep., Department of Computer Science, University of Maryland; 2006.
[29] Vallati M, Chrpa L, McCluskey TL, Hutter F. On the importance of domain model configuration for automated planning engines. J Automat Reason 2021;65(6):727–73.
[30] Alnazer E, Georgievski I, Aiello M. Risk Awareness in HTN Planning. Tech. rep. 2204.10669, arXiv; 2022.
[31] Georgievski I, Nguyen TA, Aiello M. Combining activity recognition and AI planning for energy-saving offices. In: IEEE international conference on ubiquitous intelligence and computing. 2013, p. 238–45.
[32] Georgievski I. Towards engineering AI planning functionalities as services. In: service-oriented computing – ICSOC 2022 workshops. Lecture notes in computer science, Springer International Publishing; 2022, p. 225–36.
[33] Nizamic F, Nguyen TA, Lazovik A, Aiello M. GreenMind – an architecture and realization for energy smart buildings. In: International conference on ICT for sustainability. 2014, p. 20–9.
[34] Georgievski I. PlanX: A toolbox for building and integrating AI planning systems. In: IEEE international conference on service-oriented system engineering. 2023, p. 130–4.
[35] Georgievski I. Software development life cycle for engineering AI planning systems. In: International conference on software technologies. 2023, p. 751–60.