Does Agentic AI merely incarnate Semantic Web Services?

Anonymous ACL submission

Abstract

The recent publication of agentic AI protocols and systems (such as MCP, A2A, etc.) has sparked tremendous excitement in the AI community, as it enables agents to use and combine tools and services in order to achieve complex goals. This very promise is not new though, and has been made over 20 years ago by an initiative called "Semantic Web Services". In this paper, we point out that (1) the underlying ideas of agentic AI can already be found in the Semantic Web Services vision, and (2) that abstractions, protocols, data models, and descriptions developed in the Web and Semantic Web community within and outside the (Semantic) Web Services strand of research can be beneficially applied in today's endeavour to realise agentic AI. At the same time, we argue that looking back to history reveals certain still unresolved challenges.

1 Introduction

Soon after the invention of the Web by Tim Berners-Lee in the early '90s of the last century, the emergence of tools and services available online have sparked the idea of *automated agents* running and orchestrating such services on our behalf, seamlessly achieving users' goals and supporting our daily lives. This idea is manifested in the renowned "Semantic Web" paper originally published in 2001 in the Scientific American and recently reprinted (Berners-Lee et al., 2023). The very idea behind this original Semantic Web vision was based on well-defined protocols and metadata standards for describing Web content and services in a declarative, interoperable fashion, enabling agents to align and compose services on the fly. The first proposals for such protocols were based on then state-of-the-art Web services technologies (Alonso et al., 2004), suggesting to combine those with the respective Semantic Web metadata standards (Martin et al., 2005; Roman et al., 2005). Yet, despite helping to conceptualise the building blocks of interoperable, machine-processable service ecosystems, these initiatives, dubbed "Semantic Web services", have hardly been adopted in practice.

With the recent advent of generative AI models, and their underlying transformer models (Vaswani et al., 2017), that do not only allow to generate language, but also code, solve mediation and integration problems, even can learn how to use external tools (Schick et al., 2023), and plan courses of action to solve complex goals (Yao et al., 2023), we see the ideas of agentic AI refuelled. Likewise, frameworks and protocols to make generative AI models interact with such tools and services have been put in operation and are gaining traction rapidly, such as Anthropic's Model Context Protool (MCP) (Anthropic, 2025) and Google's Agent2Agent protocol (A2A) (Google and A2A Project Contributors, 2025). As such, this may be a good time to ask ourselves whether indeed all the roadblocks that have prevented earlier successes in agentic AI have been successfully overcome with these recent technological advances?

To this end, the present paper aims at putting the historic efforts towards Semantic Web Services and recent trends and advances in agentic AI side by side. Our goal is to highlight useful "historical" conceptualisations, discuss in how far these are paralleled or incarnated by current agentic AI frameworks, and finally spotlight still unresolved challenges, with potential routes ahead.

In the remainder of this paper we first re-

cap base terminology and concepts from the (Semantic) Web Services Literature, in order to scope what we mean by "service" and other components needed in a truly agentic AI and service ecosystem. Next, we discuss in more detail in how far these concepts are implemented by recent agentic AI protocols. We conclude with challenges in terms of what we consider (still) unresolved challenges.

2 Web Services Terminology

The Web Services (Alonso et al., 2004) and Semantic Web Services (Fensel et al., 2006; Martin et al., 2005; Kopecký et al., 2007) literature has introduced several core concepts for building agentic ecosystems. We briefly recall these concepts and attempt to scope them for our purposes, in order to reflect which of them have been implemented by both preceeeding technologies and current "agentic AI" protocols.

Services, Operations and Functions We herein assume a *service* as a collection of *service operations* accessible via a common access point, or *endpoint*, operated by a remote *agent*, which provides access to the service operations via an *interface*, that expose the services's *capabilities* to the outside. A simple common abstraction of service operations is a *remote procedure call*.

Assuming such a service representing some agent, when taking this responsible agent's capabilities into account, one could also distinguish between fully observable (locally) executable operations (using the agent's own computing power and operating on data from the local state only) as *functions*, as opposed to *remote operations*, that require external services. Petrie called such functions "subroutines"(Petrie, 2016, Section 1.1.5); where we note here that also functions/subroutines could be discoverable and advertised by a service: a recent public effort to provide such a library service for discoverable functions is Wikifunctions.[1]. As opposed to functions, true remote (service) operations, require capabilities *beyond the agent's local control or observability*, according to Preist's definition (Preist, 2004) of a service as the provision of an (abstract) value or *product* in the context of the domain of application through interactions, by one party to another. This lack of observability also typically necessitates some form of agreement or *contract*, before actual delegation to remote service operations, potentially achieved through *negotiation*, i. e., interactions preceding the value delivery, related to non-functional aspects (see below), such as the alignment of the part-taking agents' policies (Kampik et al., 2022).

Provider and Requester; Invocation In the context of a service ecosystem, the literature typically distinguishes between *service provider*, as for the agent providing a service and respective operations to other agents, and *service requester*, as for an agent that *invokes* (or: requests) services, beyond its own capabilities or for other reasons preferably delegated. It is important to note that this distinction, while putting the requester's goal in the focus, largely ignores the provider's goal(s).

Service Descriptions Services can be described for different purposes, e. g. to facilitate discovery, composition, binding and execution, as outlined below. In the simplest case, service descriptions contain overall *capabilities* of the service, its exposed operations as well as syntactical *interface* descriptions of *inputs* and *outputs* of specific operations, next to the address of the endpoint, and necessary authentication. These more functional aspects may be augmented by a description of *non-functional aspects*. We may understand those aspects as – abstract or explicit - (not necessarily fully observable) *conditions* and *rules*, over the internal states of different agents relevant to the service ecosystems (requester, provider, legislator, etc.), for instance *preconditions* that need to be fulfilled to successfully invoke a particular service operation, or *postconditions* that are guaranteed upon such successful invocation. Some even argued to include behavioural information in the description, i. e. in which order different operations should be invoked. Non-functional aspects may additionally refer to e. g. policies and regulations to execute an operation (legally) compliant, payment modalities, service level agreements or alike. In sum, non-functional aspects refer to (not fully observable) *conditions* and *rules*, over the internal states of different agents (re-

[1] https://www.wikifunctions.org/

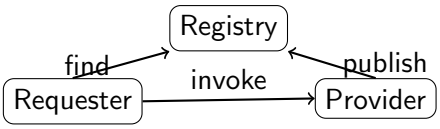

Figure 1: The Publish-Find-Invoke Pattern

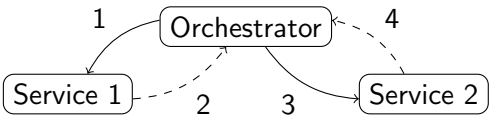

Figure 2: Service Orchestration. An orchestrator makes two sequential calls (solid) to two services, and receives corresponding responses (dashed).

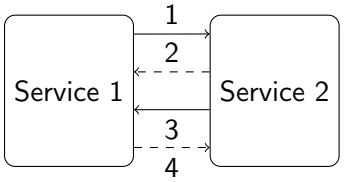

Figure 3: Service Choreography. Two services call (solid) and respond to (dashed) each other sequentially.

quester, provider) or also rules governing the service ecosystems as a whole (legislator, organization etc.).

Discovery For services and their operations to be invocable, their availability must be known to potential requesters. The process of acquiring knowledge about relevant services is called *service discovery*. On the service provider side, the facilitation of discovery may involve advertisement, in the form of either decentralised self-advertisement or *publish*ing service descriptions into dedicated *repositories* or *registries*, which represent separate services. To *find* the right services' descriptions for downstream tasks including composition and invocation, this discovery requires *search* capabilities of *service descriptions*. Related problems include those of service *matchmaking* (to determine which of the discovered services fulfil the requester's requirements) and *selection* (to finally choose from the discovered services). Those steps can be facilitated by methods including string similarity measures or ontological reasoning.

Corresponding registries facilitate the implementation of the publish-find-invoke (or: -bind) pattern for composition (Endrei et al., 2004; Papazoglou and van den Heuvel, 2007), which is commonplace in many service-oriented approaches, including UPnP, Web Services, and CORBA (Aiello, 2018), see Figure 1. In this pattern a Provider publishes descriptions about the services they provide at a Registry, where a Requester can find descriptions, and use the descriptions' information for invocations.

Composition, Choreography, Orchestration In case a single service call does not achieve a given goal, a composition of different available services and service operations may achieve the said goal.

Establishing a viable composition to goal completion may involve/ be preceded by *negotiations* establishing compliance with the rules and conditions –yet in practice negotiations hardly happen and customers are left with standardised service-level agreements to which they can agree or not–, and *planning* the sequence of invocations in the correct order. This planning may be done by humans who manually craft an orchestration, or by automated planners –yet this automation is not done at scale, probably due to the high requirements of such planners regarding formalisation of corresponding goals and semantic web service descriptions.

A composition may be described in different ways, in terms of an explicit *workflow* or *process*, that decomposes the overall goal into subtasks down to single service or function calls –often called an *orchestration*, see Figure 2–, or more declarative means which rather define a certain order or parallel execution of services, or, more generally, a *protocol* –often called *choreography*, see Figure 3.

Execution, Binding Finally, for practical effect, a composition of services needs get *executed*. To this end, values, services, and communication channels need to be chosen (also called *binding*), and the services invoked, all governed by the composition. Depending on the nature of the composition, orchestration or choreography, there is or is not necessarily a central entity to govern the composition. In case of the orchestration, this entity is the orchestrating service requester, in case of a choreography there may or may not be a ded-

icated middleware in place that can serve as message broker or even check compliance to the protocol.

Mediation is the task of resolving mismatches in data (e. g. terminology or message format mismatches), processes (e. g. behavioural mismatches), or functionality (e. g. capability mismatches) between different services to enable them to work together. Mediation can occur during search, composition, and invocation.

In the following, we will reflect in how far traditional Web service technologies, as well as Semantic Web services have implemented the envisioned automation of the above-mentioned steps, followed by a discussion of agentic protocols.

3 Web Services Technologies

We introduce core technologies from the WS-* stack, and Semantic Web Services, and briefly summarise.

3.1 WS-* Technologies

The Web services stack proposed to enable automation mostly through technologies like SOAP, WSDL, and UDDI, providing standardized protocols, interface descriptions and registries (with limited search functionality), and a method to describe orchestrated execution workflows through BPEL-WS.

Invocation SOAP (Gudgin et al., 2007) is a protocol for remote procedure calls that are tunnelled through other protocols, most prominently HTTP POST requests, with the service endpoint as target URI of the request. The content of such a request is an XML document containing a SOAP envelope element, which is subdivided into header and body elements, to describe the remote procedure call.

Description The WSDL (Web Services Description Language) is an XML standard to describe service operations, including message formats, typically in terms of XML schema format descriptions, and bindings to a specific protocol, essentially wrapping remote procedure calls into XML. As such, the descriptions were, despite human readable comments as part of the description, mostly focused on enlisting API operations, and – borrowing from XSD – descriptions of the service input and output messages' grammar.

Discovery UDDI (Universal Description Discovery and Integration) as an early standard to advertise services and implement a respective *registry*, distinguished between different levels of description relating to general service description, such as contact information (White pages), business description (in terms of e.g. USPSC business/sector categories), and technical descriptions, e.g., linking to a WSDL web service description.Although publicly running UDDI registries were operated (e.g. by IBM, Microsoft and SAP), all of these seem to have stopped operation before 2010.

Orchestration The Web Services Business Process Execution Language (Alves et al., 2007) is an XML-based standard that allows to describe an orchestration of services in a process, i. e. calls to service operations, made by one single service requester, put into order by some control flow (e. g. sequential, parallel ordering). For software that executes service orchestrations there are many commercial offerings.

Mediation In an orchestration, mediation can be done manually, yet facilitated by the inclusion of corresponding XML standards in BPEL, e. g. functionality to transform variables (using XSLT transformations) and value selection (using XPath).

Around the core WS-* standards, a plethora of other standards emerged, such as WS-Security, which introduced security features, WS-BPEL4People, which introduced human tasks into service compositions, and WS-Policy, which allows to describe requirements of a service regarding, e. g. security, quality of service, etc.

Overall, the Web services stack provided very limited support for automation by agents, and was rather focused on enabling human developers to operate, compose, and invoke SOAP/XML-based services, and executing pre-defined orchestrated executions.

3.2 Semantic Web Services Technologies

In an attempt to overcome theses deficiencies, Semantic Web Services attempted to enrich

Service descriptions, filling in the blanks, while still building on the Web Services ideas and base technologies.

Description Frameworks like WSMO (Roman et al., 2005), and OWL-S (Martin et al., 2005) are mainly semantic extensions of service descpriptions, annotating overall capabilities, as well as **i**nputs, **o**utputs, **p**re-conditions, and **e**ffects (IOPEs), with declaratively described concepts (e.g using OWL or F-Logic), in order to facilitate ontology-mediated composition by leveraging ontological reasoning to match (by subsumption) descriptions of requester goals to service capabilities. On top, the description can include, e. g. information about how a service makes use of other services in a choreography.

Mediation Especially WSMO put mediation into the focus and defined mediation components, which are modular and reusable (OO, GG, WG, and WW mediators) to connect and reconcile differences between **o**ntologies, **g**oals, and **W**eb services. In OWL-S, mediators are just yet another service (Paolucci et al., 2004).

Overall, Semantic Web Services remained largely an academic endeavour, partially because service providers hardly supplied the required formal and unambiguous annotations: this shortage of real-world service descriptions kept repositories and engines–such as WSMO's reference execution architecture (WSMX) (Haller et al., 2005)–at a mostly conceptual or prototypical level.

3.3 Summary

Both traditional Web Services and Semantic Web Services technologies share that they did not go significantly beyond the simple publish-find-invoke pattern. The addition of workflow-based orchestrations here merely added hard-wired processes, in the case of BPEL-WS, as well as prototypically leveraging declarative AI Planning techniques for composition, e.g. (Pistore et al., 2004).

Since then, XML-based messaging has been largely replaced by JSON as the standard data format, and SOAP has been replaced by RESTful interaction via HTTP, where discovery and composition is facilitated using hypermedia, instead of registries.

In summary, (Semantic) Web Services research was arguably focused on the automation of discovery, mediation and composition from a requester-provider point of view.

4 Agentic AI

We argue that the recent advent of Agentic AI protocols (Anthropic's Model Context Protocol (MCP) and Google's Agent2Agent Protocol (A2A) as the main contenders) resembles many of the promises of (Semantic) Web Services. We thus outline that these "agentic" extensions of LLMs indeed solve main challenges, but also show strong resemblance with (Semantic) Web Services.

Model Context Protocol

MCP enables LLMs to call and coordinate tools where an LLM (Host) acts as orchestrator, who knows MCP clients that connect to MCP servers accessible through JSON-RPC.

Descriptions MCP servers are described in natural language (typically explaining the service operation capability in human-readable terms), along with a template RPC call, illustrating the interface.

Discovery In MCP, discovery is facilitated typically by system prompt, where the descriptions are injected. At the same time, we note that dynamic discovery through a registry can be easily added to the MCP architecture. We illustrate this in Figure 4: to this end, we have implemented a prototypical architecture, i.e., an MCP host using a "repository" MCP server that in turn provides access to a repository of other MCP servers dynamically. As a demonstration use case, we provide a travel booking use case and resp. MCP servers through an anonymized GitHub repository[2].

Composition, Execution When the LLM is triggered with a prompt, that can be understood of a natural language described user *goal*. The LLM can choose to either respond directly or to instantiate one of its known MCP clients to solicit an answer for the prompt. When several MCP client calls are required, this is achieved through, typi-

[2] https://anonymous.4open.science/r/agentic-ai-implementations-D0C2/

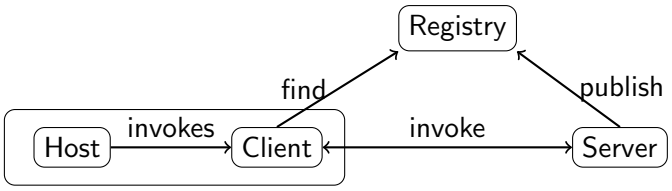

Figure 4: The Publish-Find-Invoke Pattern in MCP.

cally ReACT-style (Yao et al., 2023), repeated decomposition of the goal.

Mediation The MCP host LLM may adapt calls to the MCP servers through LLM-mediated binding, leveraging the description template, and background knowledge.

We thus argue that (i) MCP can be easily extended to implement the above-mentioned traditional WS publish-find-invoke pattern and (ii) solves mediation/binding, by means of relatively lightweight, natural language capability descriptions and the increasing ability of LLMs to interpret those, and (iii) provides dynamic orchestration through ReAct.

Potential and Limitations. Our prototypical tests in using MCP for a typical service orchestration use case, conference travel booking, confirms promising results, but also shows that user control to when and which services are called on behalf of a user goal is still limited, if we leave control to LLMs. Remarkably, we can also show that dynamic enrichment of services published on a dynamic repository is entirely possible, but leaves questions such as reliability of advertisements open: repsective considerations from earlier initiative such as UDDI, or "app stores" to ensure trustworthyness of services on such repositories, should probably be considered.

Also, the delegation of composition/mediation to an LLM may be considered less reliable than the symbolic reasoning and planning techniques envisioned in Semantic Web Services. A final solution to this challenge seems to call for a neurosymbolic approach (Sheth et al., 2023; De Smet and De Raedt, 2025).

We also note that so far, composition is, in most documented MCP use cases, limited orchestration, where a single LLM takes the role of an orchestrator of services that are known at design time.

A2A

A2A by Google was presented as a complementary extension to MCP. A2A does not per se make the same strict client-server distinction as MCP, and also allows for HTTP-based interaction. On top, A2A supports *discovery* by so called AgentCards, which, as opposed to MCP's simple text-based *descriptions*, are more structured (e. g. making capabilities explicit). Also, A2A uses unique IDs for contexts and tasks, which can be shared along with calls, and –similar to session IDs in Web applications– enables (host) applications to incorporate communication state information into compositions, without revealing full internal state details. Additionally, A2A supports some form of authentication and access control. As such, A2A adds "convenience" and conceptual promises on top of MCP that, however, yet have to be proven in practical use cases, for instance in terms of supporting real choreographed, collaborative agent compositions.

However, it seems that –partially due to the novelty of A2A– fully controllable mechanisms or protocols to govern collaborative environments and adding safeguards for secure collaboration of agents, do not seem to be built-in or existing as of yet.

5 Related Work

Together with the recent rise of Agentic AI, papers about Agentic AI are written. Some make economic arguments about the incentives in Agentic AI and the Web (Yang et al., 2025), others write about the notion of agency in Agentic AI from a multi-agent systems perspective (Botti, 2025). Closest to our work, (Petrova et al., 2025) see Agentic AI in the history of FIPA agents, Semantic Web, and multi-agent systems. Yet, for Semantic Web, the paper stays on the vision level, and does not have a conceptual Semantic Web Services

treatment that we provide.

6 Outlook and Challenges

In summary, we believe one can learn important lessons from revisiting traditional Web Services terminology in the light of the current Agentic AI "hype": while the main ideas and principles behind MCP or A2A based agents are not fundamentally new, LLMs and their reasoning capabilities have enabled automating mediation, binding and composition, in a manner and scale that has not been achievable by (Semantic) Web Services. However, some main challenges arise or remain:

- **agency**: i.e., what to delegate to whom? Sensitive aspects to achieve an agent's goal may (not) be delegated to remote agents. In particular, fully-observable *functions* within the agent's own capabilities *could* be executed locally, whereas service operations that require capabilities (i. e. access to the internal state) of another agent *require* remote operations.

- **collaboration**: each agent (and also service), may have own goal(s), i. e., the perspective of a single user agent defining the goal of a composed service execution is over-simplifying. In particular, *orchestration* is only one view of service composition, and which is coordinated by a single agent's goal, where as complex, non-hierarchical interactions of multiple agents need *choreography* by different means of governance.

- **contracting**: non-functional aspects and "service" delivery (in the sense of a business value or product) is not fully observable and may need a-priori negotiations of contracts, as well as decentralized error-handling and dynamic contract renegotiation during execution. Especially, dynamic discovery and execution needs to cater for timeouts or temporal validity of service provision agreements, and may require guaranteed, standardized service-level agreements.

- **mediation**: in current agentic AI systems, mediation between the terminology of the different tools called seems to be implicitly done by the LLM. The inner workings of this mediation, how it works and when it fails, next to the necessary negotiation in case something fails, deserves future study. On top, WSMO had identified other types of mediation, namely between goals and between the different communication paradigms and directions of services. Whether and how those more involved types of mediation can be addressed by LLMs is also an open question.

- **Web**: most digital information and functionality is currently provided via the Web, which is built as a resource-oriented and not a service-oriented architecture, and where hypermedia facilitates discovery and composition, all of which is yet to be leveraged by Agentic AI.

While we hardly see these challenges addressed in the current hype yet, they call for action and revisiting potentially ahead-of-their-time solution proposals from the research community.

Limitations

- The lack of formal treatment in the Agentic AI proposed standards makes it hard to objectively tell what is the technical essence.

- Similarly, the Agentic AI proposed standards rarely state which architectural decisions have led to their development, which makes it hard to discuss the standards. This was also what motivated us to write the paper at hand, to investigate the intellectual roots of Agentic AI.

References

Marco Aiello. 2018. *The Web Was Done by Amateurs - A Reflection on One of the Largest Collective Systems Ever Engineered*. Springer.

Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. 2004. *Web Services: Concepts, Architectures and Applications*. Springer.

Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin

Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. 2007. Web services business process execution language version 2.0. Oasis standard. https://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Anthropic. 2025. Model context protocol specification, version 2025-06-18. MCP Specification. Accessed: 2025-09-14.

Tim Berners-Lee, James Hendler, and Ora Lassila. 2023. *The Semantic Web: A New Form of Web Content that is Meaningful to Computers will Unleash a Revolution of New Possibilities*, 1 edition, page 91–103. Association for Computing Machinery, New York, NY, USA.

V. Botti. 2025. Agentic AI and multiagentic: Are we reinventing the wheel? *CoRR*, abs/2506.01463.

Lennert De Smet and Luc De Raedt. 2025. Defining neurosymbolic ai. *arXiv preprint arXiv:2507.11127*.

Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. 2004. *Patterns: service-oriented architecture and web services*. Redbooks. IBM Corporation.

Dieter Fensel, Holger Lausen, Axel Polleres, Jos De Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. 2006. *Enabling semantic web services*. Springer Science & Business Media.

Google and A2A Project Contributors. 2025. Agent2agent (a2a) protocol specification. https://a2a-protocol.org/v0.3.0/specification/. Version 0.3.0.

Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. 2007. Soap version 1.2 part 1: Messaging framework (second edition). W3c recommendation. https://www.w3.org/TR/soap12/.

A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. 2005. Wsmx - a semantic service-oriented architecture. In *IEEE International Conference on Web Services (ICWS'05)*, pages 321–328 vol.1.

Timotheus Kampik, Adnane Mansour, Olivier Boissier, Sabrina Kirrane, Julian Padget, Terry R. Payne, Munindar P. Singh, Valentina Tamma, and Antoine Zimmermann. 2022. Governance of autonomous agents on the web: Challenges and opportunities. *ACM Trans. Internet Technol.*, 22(4).

Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. 2007. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67.

David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. 2005. Bringing semantics to web services: The owl-s approach. In *Semantic Web Services and Web Process Composition*, pages 26–42, Berlin, Heidelberg. Springer Berlin Heidelberg.

Massimo Paolucci, Naveen Srinivasan, and Katia P. Sycara. 2004. Expressing WSMO mediators in OWL-S. In *Proceedings of the Workshop on Semantic Web Services (SWS) at the 3rd International Semantic Web Conference (ISWC)*, volume 119 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Mike P. Papazoglou and Willem-Jan van den Heuvel. 2007. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, 16(3):389–415.

Charles J. Petrie. 2016. *Web Service Composition*. Springer.

Tatiana Petrova, Boris Bliznioukov, Aleksandr Puzikov, and Radu State. 2025. From semantic web and MAS to agentic AI: A unified narrative of the web of agents. *CoRR*, abs/2507.10644.

Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. 2004. Planning and monitoring web service composition. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 106–115. Springer.

Chris Preist. 2004. A conceptual architecture for semantic web services. In *The Semantic Web – ISWC 2004*, pages 395–409, Berlin, Heidelberg. Springer Berlin Heidelberg.

Dumitru Roman, Uwe Keller, Holger Lausen, Jos De Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. 2005. Web service modeling ontology. *Applied ontology*, 1(1):77–106.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Amit Sheth, Kaushik Roy, and Manas Gaur. 2023. Neurosymbolic artificial intelligence (why, what,

and how). *IEEE Intelligent Systems*, 38(3):56–62.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Yingxuan Yang, Mulei Ma, Yuxuan Huang, Huacan Chai, Chenyu Gong, Haoran Geng, Yuanjian Zhou, Ying Wen, Meng Fang, Muhao Chen, Shangding Gu, Ming Jin, Costas J. Spanos, Yang Yang, Pieter Abbeel, Dawn Song, Weinan Zhang, and Jun Wang. 2025. Agentic web: Weaving the next web with AI agents. *CoRR*, abs/2507.21206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.