UNIVERSITÄT DES SAARLANDES

Linked Data as Medium for distributed Multi-Agent Systems

by

Torsten SPIELDENNER

A dissertation submitted towards the degree of

Doctor of Engineering (Dr.-Ing.)

of the Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken, 2023

Dekan der Fakultät Mathematik und Informatik: Univ.-Prof. Dr. Jürgen Steimle

Vorsitzende des Prüfungsausschusses: Prof. Dr. Verena Wolf

Erstgutachter: Prof. Dr. Philipp Slusallek

Zweitgutachter: Prof. Dr. Antonio Krüger

Promovierter akademischer Mitarbeiter der Fakultät: Dr. Hilko Hoffmann

Tag des Kolloquiums: 13.12.2023

"It Gets Easier. Every Day, It Gets A Little Easier. But You Gotta Do It Every Day – That's The Hard Part. But It Does Get Easier."

Jogging Baboon, in *Bojack Horseman*

Abstract

The conceptual design and discussion of multi-agents systems (MAS) typically focuses on agents and their models, and the elements and effects in the environment which they perceive. This view, however, leaves out potential pitfalls in the later implementation of the system that may stem from limitations in data models, interfaces, or protocols by which agents and environments exchange information. By today, the research community agrees that for this, that the environment should be understood as well as abstraction layer by which agents access, interpret, and modify elements within the environment. This, however, blurs the the line of the environment being the sum of interactive elements and phenomena perceivable by agents, and the underlying technology by which this information and interactions are offered to agents.

This thesis proposes as remedy to consider as third component of multi agent systems, besides agents and environments, the digital *medium* by which the environment is provided to agents. "Medium" then refers to exactly this technological component via which environment data is published interactively towards the agents, and via which agents perceive, interpret, and finally, modify the underlying environment data. Furthermore, this thesis will detail how MAS may use capabilities of a properly chosen medium to achieve coordinating system behaviors.

A suitable candidate technology for digital agent media comes from the Semantic Web in form of *Linked Data*. In addition to conceptual discussions about the notions of digital agent media, this thesis will provide in detail a specification of a Linked Data agent medium, and detail on means to implement MAS around Linked Data media technologies.

Zusammenfassung

Sowohl der konzeptuelle Entwurf von, als auch die wissenschaftliche Diskussion über Multi-Agenten-Systeme (MAS) konzentrieren sich für gewöhnlich auf die Agenten selbst, die Agentenmodelle, sowie die Elemente und Effekte, die sie in ihrer Umgebung wahrnehmen. Diese Betrachtung lässt jedoch mögliche Probleme in einer späteren Implementierung aus, die von Einschränkungen in Datenmodellen, Schnittstellen, oder Protokollen herrühren können, über die Agenten und ihre Umgebung Informationen miteinander austauschen. Heutzutage ist sich die Forschungsgemeinschaft einig, dass die Umgebung als solche als Abstraktionsschicht verstanden werden sollte, über die Agenten Umgebungseffekte und -elemente wahrnehmen, interpretieren, und mit ihnen interagieren. Diese Betrachtungsweise verschleiert jedoch die Trennung zwischen der Umgebung als die Sammlung interaktiver Elemente und wahrnehmbarer Phänomene auf der einen Seite, und der zugrundeliegenden Technologie, über die diese Information den Agenten bereitgestellt wird, auf der anderen.

Diese Dissertation schlägt als Lösung vor, zusätzlich zu Agenten und Umgebung ein digitales *Medium*, über das Agenten die Umgebung bereitgestellt wird, als drittes Element von Multi-Agenten-Systemen zu betrachten. Der Begriff "Medium" bezieht sich dann genau auf diese technologische Komponente, über die Umgebungsinformationen Agenten interaktiv bereitgestellt werden, und über die Agenten die zugrundeliegenden Daten wahrnehmen, interpretieren, und letztendlich modifizieren. Desweiteren wird diese Dissertation aufzeigen, wie die Eigenschaften eines sorgfältig gewählten Mediums ausgenutzt werden können, um ein koordiniertes Systemverhalten zu erreichen.

Ein geeigneter Kandidat für ein digitales Agentenmedium findet sich im Ökosystem des „Semantic Web", in Form von *„Linked Data", wörtlich („verknüpfte Daten")*. Zusätzlich zu einer konzeptionellen Diskussion über die Natur digitaler Agenten-Media, spezifiziert diese Dissertation „Linked Data" als Agentenmedium detailliert aus, und beschreibt im Detail die Mittel, wie sich MAS um Linked Data Technologien herum implementieren lassen.

Acknowledgements

The journey to finish this document was a long one, and it would not have been possible to complete without the kind help and support of all the people along the way.

First, I would like to express my thankfulness towards my supervisor, Prof. Dr. Philipp Slusallek, for the opportunity and constant support in my endeavour to pursue a PhD in his group at DFKI.

I moreover owe due gratitude towards René Schubotz for lots of very valuable critical and constructive input that definitely helped me sharpen my scientific skills over the years, and whose passion in Semantic Web and Linked Data turned out to be both contagious and truly inspiring in the long run, which undoubtedly truly paved the way for most of the scientific contributions discussed in this document.

I would moreover like to warmly thank Hilko Hoffmann, Ingo Zinnikus, and all other project leads, who through the years happily provided me with the opportunity to both promote and evaluate my research in the scope of the projects under their supervision.

These mentions also definitely extend to all my colleagues and fellow PhD students with whom I had the pleasure of not only many fruitful, constructive and inspirational discussions and collaborations, but also great times outside of the office (special thanks at this place also to Ralph of the Nautilus Bar, who provided the necessary fuel for not few of said discussions).

Finally, I would love to say a huge "Thank You!!" to my family, who never stopped believing in me, and my wife, who never got tired of helping me find again my courage when I was on the brink of losing it.

This journey was an adventure of a lifetime. Thank you all for having been a part of it!

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

AI	**A**rtificial **I**ntelligence
AIS	**A**rtificial **I**mmune **S**ystem
AJAN	**A**ccessible **J**ava **A**gent **N**ucleus
CPS	**C**yber **P**hysical **P**roduction
CRUD	**C**reate **R**ead **U**pdate **D**elete
DAG	**D**irected **A**cyclic **G**raph
ECA	**E**ntity **C**omponent **A**ttribute model
FOAF	**F**riend **O**f **A** **F**riend (RDF ontology)
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
HPC	**H**igh **P**erformance **C**omputing
IoT	**I**nternet **o**f **T**hings
IRI	**I**nternationalized **R**esource **I**dentifier
JEA	**J**ob **E**xecution **A**gent
JSA	**J**ob **S**cheduling **A**gent
JSON	**J**ava **S**cript **O**bject **N**otation
JSON-LD	**JSON** for **L**inked **D**ata
LD	**L**inked **D**ata
LDP	**L**inked **D**ata **P**latform
LDP-RS	**L**inked **D**ata **P**latform **R**DF Source
LDP-NS	**L**inked **D**ata **P**latform **N**on-RDF Source
LuCe	**L**ogic T**u**ple **C**enters
MAS	**M**ulti **A**gent **S**ystem
MOSP	**M**inimize number of **O**pen **S**tacks **P**roblem
MTO	**M**ake **T**o **O**rder
MV*	**M**odel **V**iew * (Design Pattern)
MVVM	**M**odel **V**iew **V**iew**M**odel (Design Pattern)
MVPVM	**M**odel **V**iew **P**resenter **V**iew**M**odel (Design Pattern)
OAA	**O**rder **A**ssignment **A**gent
OHA	**O**rder **H**andling **A**gent
RDF	**R**esource **D**escription **F**ramework
x-RMM	**R**ichardson **M**aturity **M**odel (level-**x** maturity)
ReSpecT	**Re**action **Spec**ification **T**uples
REST	**RE**presentational **S**tate **T**ransfer
TA	**T**ransporter **A**gent
TS	**T**uple **S**pace
TSP	**T**raveling **S**alesman **P**roblem
stigLD	**Stig**mergic **L**inked **D**ata (framework)
stigFN	**Stig**mergy **F**u**N**ction library
WoT	**W**eb **o**f **T**hings

Chapter 1

Introduction

The problem of coordination of agents, tasks, resources, and more, is an ever popular and challenging topic in research. The need for coordination by this arises in most various domains and for as many reasons. The desired effect that is sought by coordinating a system is thereby most often optimization, under the assumption that a perfectly coordinated system behaves optimal with respect to utilization of resources, or time it takes for processes implemented in the system to finish.

Coordination in public transport and traffic may take the shape of properly steering the red and green light phases of traffic lights according to demands: During rush hours in morning and evening, traffic may need to be kept flowing from residential areas to offices and factories, whereas in the evening hours, the most dense traffic flow may be expected when workers returning home merge with citizens who move from their homes to leisure areas in the city center. Another dimension can be added to this by accordingly coordinating public transport routes to support movement of citizens on this routes at best avail.

In industrial manufacturing, coordination algorithms are used to find an optimal load distribution over producing entities, with the goal to avoid bottlenecks on one part of the production while other machines remain unused and idle. Similarly, transport of materials and supplies needs to be coordinated among mobile units. Robots with particular skill sets need to be assigned the proper position within production to provide their required skills to the production cycle at the right time and the right place on the shop floor. Finally, all these entities - production machines, transport, workers - need to work seamlessly hand in hand to ensure a stable and efficient production process.

The approaches to implement coordination and optimization algorithms are manifold. A popular method to implement such coordination scenarios are *Multi agent systems* (MAS) [269, 77]. Multi agent systems, as the name suggests, typically employ a crowd of agents with varying skillsets to jointly observe, interpret, and act on the to-be-coordinated system. The expected benefit is that by distributing the work between a number of agents, the resulting system becomes more flexible, scalable, robust, and efficient.

Conceptually, *agents* are considered to be equipped with sensory capabilities and actuators by which they perceive phenomena in their surroundings, or may inflict changes respectively [186]. This surrounding that is sensed by agents, and upon which the agents act and with which they interact, is commonly referred to as the agents' *environment* [269, 186, 262, 267].

Even though a core feature of agents is autonomy [269], it is commonly agreed upon that an agent's behaviour is not independent of its environment, but that an agent strongly depends on the perceptions and possible interactions offered by it [186]. Despite the crucial significance within multi agent systems, this pivotal role of a properly defined environment was for a long time marginalized, or even overlooked [262]. This leads to an often critical underspecification of how respective perceptions and interactions within the agents' environment are finally being provided to the agents on a technical level. In this respect, Weyns at el. observed in 2015 as result of several surveys that were conducted in the scope of a series of workshops [266] on multi agent systems:

"There is a general agreement in the research community that agent environments are essential for multi-agent systems, yet researchers neglect to integrate the agent environment as a primary abstraction in their models and tools for multi-agent systems." [265]

They suggest to consider the environment not only as a description and state space of the world surrounding the agent, but also as a technical abstraction layer that finally helps design architecture and implementation of a resulting multi-agent system.

While it is true that including the provisioning of environmental data to agents is crucial from an engineering point of view, this consideration, however, blurs the line between two disjunct roles now attached to environment: First the environment being all data, and by this, perceptions, conditions, or processes that are generally observable and accessible to the agent, and by this ultimately define an agents' behavior based on its observations and chosen reactions; and second, the environment being the technological components that provide this information in a way that is accessible for the agents and via which agents may interact with the environment. This observation hints at the necessity to include a third component besides agents and their environment for a proper understanding of well-defined multi-agent systems.

From the field of cyber-physical systems comes another way to conceptually partition multi-agent systems by distinguishing between *agent space* and *artifact space* [43] (see also Fig. 1.1): In the agent space resides all information accessible to agents, such as representational descriptions of processes, plans, or actions that can be taken by the agent. The artifact space then contains all entities residing in the real world,

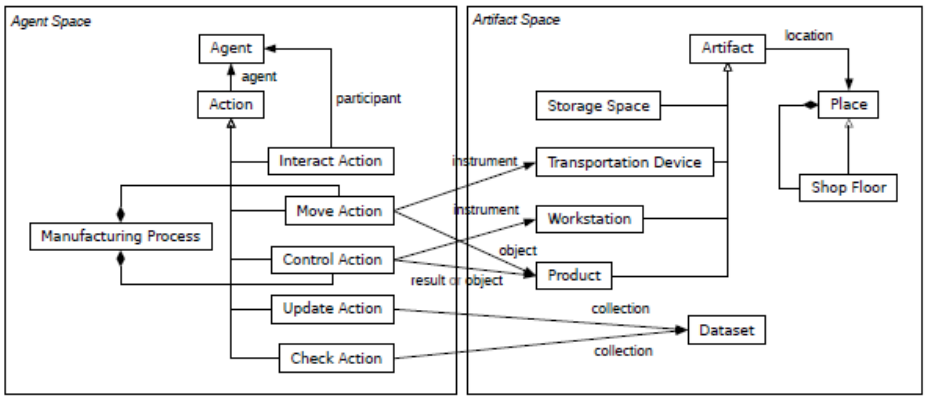

FIGURE 1.1: Separation between virtual agent space and physical artifact space for scenarios from the domain of cyber-physical production [43]

such as for example actual roads in a cities, cars on these, factory machines, or mobile robots. Some of the real world artifacts may provide interfaces to agents to be read and modified, such as machine states, locations of public transport vehicles, factory storage inventory lists, worker shift schedules, and similar.

We have established before that agents gain their autonomy from their capability to sense and interact with their environment. Now considering this model of MAS, it becomes apparent that it is in no way accurate to consider agents to actually sense and directly interact with their *environment*, as "the environment" in terms of relevant conditions, status, or effects is existing in a space that is strictly separated from the virtual space in which agents exist, namely the real world. Agents, in fact, rather sense a *representation of the environment* that is provided to the agents in a for the agents accessible and interpretable way, and interact with the environment *indirectly* via likewise for the agents accessible and invocable handles.

This observation now gives a clearer picture of the third pillar of MAS that was already hinted before: What needs to be taken in to consideration during the design of MAS is a *technology component*, in terms of data models, serialization, and protocols, that is capable of properly representing the environment to agents, from which agents actually get the sensory input about the environment that they need to successfully complete their tasks, and by which agents gain the capabilities to inflict changes to the (real-world) environment. This observation thus decouples the environment as state space of the real world, from the technological ***Medium*** via which agents interact with their environment. This thesis will therefore motivate the notion of a digital ***Medium*** as exactly this technological component that provides interactive environment access to agents.

The expectation is the following: While defining agent models in terms of agents' reactions to perceived phenomena is generally independent of the technical implementation of the final agent program, vice versa, implementing agent-environment

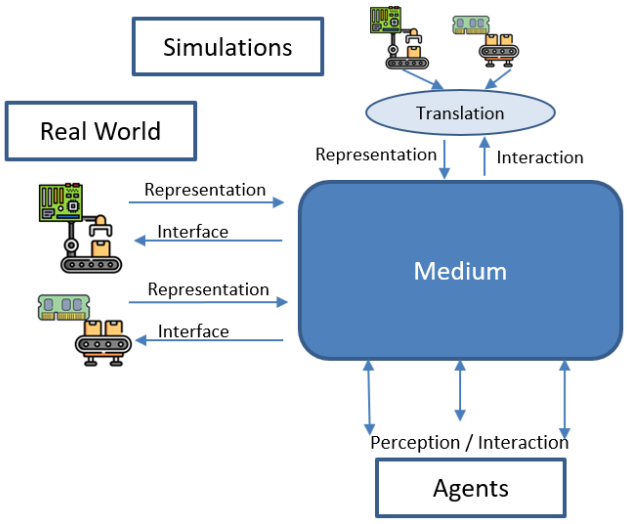

FIGURE 1.2: Medium as technological component that provides a well-defined read/write access to elements in the environment.

interaction is absolutely constraint by technological particularities of the environment implementation. Implementation of MAS can therefore be separated into two distinct tasks: modelling agent *interaction* (the channels by which agents may access environmental data), and modelling agent *behavior* in terms of *reaction* to perceptions. The interface between both steps are agents' *perception* and *actions*, by which agents read data from the environment, or attempt to inflict changes to the environment (see also Fig. 1.2). Providing this interface layer in terms of a well-defined *Medium* component then allows to take into account perception and action access to the environment during design of the agent models. It should be stressed that in this architecture, the medium component only *represents* the environment, and does not implement it. By defining how environmental data may be published to it, the medium component, and by this, agent-environment interaction via the medium, is independent of the represented environment. The architectural choice of introducing a Medium component thus is expected to reduce complexity of implementation of MAS by reducing agent-environment-interaction to the interface provided by the medium. Agent behavior can then be modeled entirely based on perceptions and actions via the medium, allowing for more generic, re-usable, and robustly designed solutions.

When it comes to the application of MAS, a certain class of algorithms have taken into account already a *medium* as tangible, interactive part of the environment: The nature-inspired principle of *stigmergy* [245, 114] models agents entirely around how agents react to influences left in the environment by other agents. This indirect communication mechanism between agents based on traces left in the environment ultimately leads to a self-coordinating and self-organizing system behavior. The resulting systems are by this considered very resilient towards outer influences. The

"medium", the part of the environment that is perceived by agents, and which undergoes changes as result of agents' actions, by this takes such a central role as core of a stigmergic system, that it is also considered the *"true power of stigmergy"* [113]. It can therefore be assumed that a properly defined medium does not only benefit the design of agent-environment interaction, as established before, but that by imposing onto the medium the role of being the center of (indirect) agent interaction, the emergence of optimization and coordination effects in the system can be taken into account early in the design phase of the MAS.

A suitable candidate for a digital agent medium comes from the world of the Semantic Web [12] in the shape of *Linked Data (LD)* [110, 15]. Linked Data refers to a set of best practices to publish structured data on the web, and is strongly revolving around the notions of Web resources ([87, 154]), which are semantically described in terms of RDF (*Resource Description Framework*[1] [147]) data graphs. With HTTP and RESTful operations [86] as underlying protocols and communication mechanisms, Linked Data architectures moreover suggest widely used standards to interact with the provided data. Given that these design choices contributed to that the Semantic Web is widely promoted as a generic integration layer [145, 97, 138, 120], and in particular, Multi Agent Systems [225, 36, 50], this thesis will further discuss Linked Data as suitable base technology for MAS. Based on standardized protocols between Linked Data servers and Linked Data consuming user-agents, the thesis will define a generic interaction pattern between agents and Linked-Data-based media.

1.1 Hypotheses and Research Questions

In the context of above considerations, this thesis derives the following hypotheses and research questions.

1.1.1 Hypotheses

HYPOTHESIS H1. (Main Hypothesis)

Complexity of the implementation of multi-agent systems is reduced tremendously by a properly defined medium, and respective agent-medium interaction. This is due to the possibility to reduce the complexity the agents by exploiting the emergence of optimization and coordination effects in the medium, and reduce the implementation of agents to a most simple agent model.

HYPOTHESIS H2.

Linked Data constitutes a suitable digital agent medium.

[1] RDF 1.1 Primer document (May 2023): `https://www.w3.org/TR/rdf11-primer/`

HYPOTHESIS H3.

A well defined *interactive digital agent medium*, with *medium* referring to the technology, data models, and data access protocols by which an environment is presented to agents, and properly defined interactions between agents and the medium, allows for the emergence of coordination and optimization effects within the medium.

HYPOTHESIS H4.

The quality of medium-centric optimization and coordination is competitive to classic linear and graph-based planning.

1.1.2 Research Questions

The Thesis will verify the Hypotheses **H1. – H4** by answering the following research questions:

R1. What constitutes a suitable digital agent medium?

Judging the quality of a medium w.r.t suitability in multi-agent systems requires an understanding against which features a medium would need to be evaluated. The first research question to be answered is therefore which features a digital medium needs to provide to be considered suitable for MAS.

R2. Is Linked Data a suitable choice for a digital agent medium?

With a set of qualities established by having answered **R1.**, it is now to be shown that Linked Data media actually constitute a suitable agent medium.

R3. How would Linked Data consuming agents interact with Linked Data Media?

Once it is established that Linked Data constitutes a suitable medium for MAS, agent-environment interaction via the medium needs to be defined.

R4. What is a suitable and sufficiently simple agent model to interact with Linked Data Media?

A declared goal of the Thesis is to reduce complexity of agent design by transferring the complexity of the MAS as such to the medium. With the basic interactions of agents and Linked Data media defined after having answered **R3.**, the next question is thus how to model agents in a Linked Data MAS to make best use of the capabilities offered by the medium, while keeping the complexity of the agent itself low.

R5. How can dynamic multi-agent environments be efficiently published in a Linked Data medium?

To this point, the thesis has considered Linked Data media and agents only on a conceptual level. The question remains how to realise the established concepts in actual implementations of interactive agent environments in Linked Data media. A particular challenge lies in representing highly dynamic environments via Linked Data media.

R6. How to implement formally defined agent models in a semantic-preserving fashion?

After having presented means to implement agent environments in Linked Data media, it further needs to be shown how a respective implementation of Linked Data media consuming agents can be realised.

R7. How can continuity of the medium be ensured during link traversal?

Linked Data architectures are decentralized, and make no assumption about where referred resources are hosted, and in which format the respective server returns the content of the resource. It may by this happen, that when agents follow links to discover new resources, the server of a hosting resource ultimately does not provide enough capabilities for the agent to interact with the resource. An example for this is when agents that employ SPARQL [232] queries to interact with resources, see themselves suddenly confronted with a server that does not offer a SPARQL query interface, or does not emit RDF data at all. This case is particularly likely in scenarios where access to IoT devices is embedded into an RDF data graph that describes the overall environment. In these cases, it must be ensured that the agent can still interpret the discovered data in the context of the medium, and ideally, employ Linked Data technologies to retrieve and process the data.

R8. Which means are suitable to incite the emergence of coordination and coordination effects in the medium?

Having defined medium behavior, a proper agent model to interact with the medium, and continuity of the medium during agent interaction, the question remains how to exploit the capabilities of the medium precisely to achieve goal-directed coordination and optimization.

R9. Does off-loading agent complexity into the medium come with a trade-off with respect to solution efficiency?

Just because, by answering **R8.**, it is *possible* to achieve coordination and optimization by properly designing interaction between a rather simple agent model, and an interactive medium, it does not necessarily mean that it is possible to do it *well*.

Consequently, finally, the quality of the found results needs to be evaluated and compared to state of the art results of established coordination and optimization approaches.

1.2 Individual Contributions

With respect to the hypotheses and research questions stated above, this thesis makes the following contributions:

- The thesis strengthens the notion of a *Digital Medium* as interactive component in a multi agent system.
- The thesis suggests *Read-Write Linked Data* as suitable interactive medium for multi agent systems.
- The thesis formally defines *interaction between agents and Linked Data Media*
- For the implementation of both passive (reactive) and dynamic Linked Data Media, the thesis will defines:
  - A suitable data meta model from which dynamic environments can publish their run-time data in real-time in a Linked Data representation
  - A (semi-)automatic mapping from the chosen meta model to a Linked Data representation, along with domain specific semantic annotations
  - Semantic description of Web APIs towards the agents as part of the produced Linked Data representation
  - A micro-service based method to access non-Linked Data data sources via semantic queries
  - A medium component server that actively drives the evolution of dynamic environments
- For the implementation of Linked Data Media consuming agents, the thesis will provide:
  - A mapping from formal agent program descriptions given in the Milner Calculus for Communicating Systems [176], to behavior trees as executable agent programs.
- The thesis will evaluate the aforementioned contributions by application examples, and demonstrate how to achieve coordination and optimization by employing the established methods.

1.2.1 Thesis Overview and Individual Contributions (per Chapter)

This thesis will be structured as follows, and make the following contributions in the respective chapters:

Chapter 2 provides an overview over the theoretical and practical background that provides the foundation of this thesis. First, the concept of Linked Data, underlying design principles, the RDF data model, and the SPARQL query language will be discussed in Section 2.1. Section 2.2 will revisit Milner's Calculus of Communicating Systems (CCS) [176] as formal notation for agent-medium interaction. Finally, Section 2.3 will describe the nature-inspired coordination principles of *stigmergy* ([116, 117]), which will be employed in this thesis to achieve coordination effects in the medium.

Chapter 3 provides a thorough literature review to support the background as presented in Chapter 2, and discusses approaches and results related to the results presented in this thesis.

Chapter 4 establishes Linked Data as a suitable medium for agent-to-agent and agent-to-environment interaction, and will establish Linked Data as a medium of choice for agent-based coordination and optimization. The chapter will moreover formally specify Linked Data Media servers.

Original Contributions:

- A set of requirements towards digital media to provide a suitable medium for agent-based self-organization, focused on princples of *stigmergy* as established in Section 2.3
- Discussion that Linked Data provides a suitable medium w.r.t previously defined requirements
- A formal specification of interactive Linked Data Media servers to host static and dynamic environments

Contents of this chapter were previously published in:

- Spieldenner, Torsten and Melvin Chelli (2018). Linked Data as Stigmergic Medium for Decentralized Coordination. In: *Proceedings of the 16th International Conference on Software Technologies. International Conference on Software Technologies (ICSOFT-2021), July 6-8, Virtual, Pages 347-357, ISBN 978-989-758-523-4*, SCITEPRESS.
- Schubotz, René, Torsten Spieldenner, and Melvin Chelli (2021). "stigLD: Stigmergic Coordination of Linked Data Agents". In: *The 6th International Conference on Bio-inspired Computing: Theories and Applications (BICTA 2021). Taiyuan, China*
- Schubotz, René, Torsten Spieldenner, and Melvin Chelli (2022). "stigLD: Stigmergic Coordination in Linked Systems". In: *Mathematics 10.7 (2022)*, p. 1041

Chapter 5 details on the implementation of dynamic Linked Data Media to describe environments with immanent dynamics, i.e. dynamic behavior that is driven by the environment itself, not by agent-environment interaction. For this, Chapter 5 will present two different solutions: First, a reactive server component that drives environment evolution by Linked Data principles upon client requests. Second, real-time publishing of environment data via a transparent Linked Data lifting approach.

Original Contributions:

- *W.r.t transparent real-time lifting of run-time data*:
  - A formal definition of the established Entity-Component-Attribute data meta model for run-time data from which Linked Data representations of dynamic environments can be generated in real-time.
  - Detailed architecture and implementation of an event-based ECA run-time
  - An automatic structural mapping from ECA run-time data to a Linked Data Platform representation.
  - Semi-automatic augmentation of the structural mapping with domain specific semantics
- *W.r.t reactive environment evolution upon client requests*:
  - Architecture and implementation of a reactive Linked Data media framework
- *W.r.t continuity of the medium during link traversal*:
  - A SPARQL 1.1 Protocol compliant service that allows to transparently query non-RDF endpoints using SPARQL

Contents of this chapter were previously published in:

- Spieldenner, Torsten et al. (2017). "FiVES: An Aspect-Oriented Virtual Environment Server". In: *Proceedings of the 2017 International Conference on Cyberworlds. International Conference on Cyberworlds (CyberWorlds-2017), September 20-22, Chester, United Kingdom*. IEEE Xplore.
- Spieldenner, Torsten et al. (2018). "FiVES: an aspect-oriented Approach for shared Virtual Environments in the Web". In: *The Visual Computer* 34.9, pp. 1269–1282
- Spieldenner, Torsten, René Schubotz, and Michael Guldner (2018). "ECA2LD: FromEntity-Component-Attribute runtimes to Linked Data applications". In: *Proceedings of the International Workshop on Semantic Web of Things for Industry 4.0. Extended Semantic Web Conference (ESWC-2018), International*

Workshop on Semantic Web of Things for Industry 4.0, located at 15th ESWC Conference 2018, June 3-7, Heraklion Crete, Greece. Springer

- Spieldenner, Torsten, René Schubotz, and Michael Guldner (2018). "ECA2LD: Generating Linked Data from Entity-Component-Attribute runtimes". In: *2018 Global Internet of Things Summit (GIoTS)*. IEEE, pp. 1–4
- Schubotz, René , Torsten Spieldenner, and Melvin Chelli (2021). "stigLD: Stigmergic Coordination of Linked Data Agents". In: *The 6th International Conference on Bio-inspired Computing: Theories and Applications (BICTA 2021)*, December 17-19, Taiyuan, China
- Schubotz, René, Torsten Spieldenner, and Melvin Chelli (2022). "stigLD: Stigmergic Coordination in Linked Systems". In: *Mathematics 10.7 (2022)*, p. 1041
- Spieldenner, Torsten (2020). "On the Fly SPARQL Execution for Structured Non-RDF Web APIs". In: *Proceedings of the 16th International Conference on Web InformationSystems and Technologies. International Conference on Web Information Systems andTechnologies (WEBIST-2020), November 3-5*. SCITEPRESS.ISBN: 978-989-758-478-7

Chapter 6 specifies the interaction between Linked Data Media consuming agents, and respective Linked Data Media servers, as presented in Chapters 4 and 5. The chapter then describes the implementation of the formally defined agent models via a semantic preserving mapping from the formal agent model to an executable representation in SPARQL behavior trees [6].

Original Contributions:

- A formal specification Linked Data Media consuming agents and agent swarms
- A semantic preserving translation from formal agent models to behavior trees as visual programming framework for executable agent programs

Contents of this chapter were previously published in:

- Schubotz, René, Torsten Spieldenner, and Melvin Chelli (2021). "stigLD: Stigmergic Coordination of Linked Data Agents". In: *The 6th International Conference on Bio-inspired Computing: Theories and Applications (BICTA 2021). Taiyuan, China*
- Schubotz, René, Torsten Spieldenner, and Melvin Chelli (2022). "stigLD: Stigmergic Coordination in Linked Systems". In: *Mathematics 10.7 (2022)*, p. 1041
- Spieldenner, Torsten, and André Antakli. "Behavior Trees as executable representation of Milner Calculus notations". In: *The 21st IEEE/WIC/ACM*

International Conference on Web Intelligence and Intelligent Agent Technology (WI-AIT), Niagara Falls, Canada. November 2022

Chapter 7 will evaluate the previously defined concepts by application in various scenarios. The versatility, adaptability, flexibility, and efficiency of the developed Prosumer concept as agent implementation for Linked Data media will be demonstrated by use-cases from the fields of classical planning, cyber-physical production, and human-robot collaboration.

Contents of this chapter were previously published in:

- André Antakli; Torsten Spieldenner; Marcel Köster; Julian Groß; Erik Herrmann; Dmitri Rubinstein; Daniel Spieldenner; Ingo Zinnikus "Optimized Coordination and Simulation for Industrial Human Robot Collaborations". In: *Alessandro Bozzon; Jose Francisco and Dominguez Mayo; Joaquim and Filipe. Web Information Systems and Technologies*. Pages 44-68, ISBN 978-3-030-61750-9, Springer International Publishing, 2020.
- Andreas Luxenburger; Jonas Mohr; Torsten Spieldenner; Dieter Merkel; Fabio Espinosa; Florian Reinicke; Julian Ahlers; Markus Stoyke; Tim Schwartz Augmented Reality for Human-Robot Cooperation in Aircraft Assembly. In: *2019 EEE International Conference on Artificial Intelligence and Virtual Reality (AIVR-2019)*, December 9-11, San Diego, CA, USA, Pages 263-266, ISBN 978-1-7281-5604-0, DOI 10.1109/AIVR.2019.00061, IEEE, New Jersey, 12/2019.
- Spieldenner, Torsten and Melvin Chelli (2018). Linked Data as Stigmergic Medium for Decentralized Coordination. In: *Proceedings of the 16th International Conference on Software Technologies. International Conference on Software Technologies (ICSOFT-2021), July 6-8, Virtual, Pages 347-357, ISBN 978-989-758-523-4*, SCITEPRESS.
- Spieldenner, Torsten and Melvin Chelli. "Linked Data as Medium for Stigmergy-based Optimization and Coordination". *Software Technologies: 16th International Conference, ICSOFT 2021*, Virtual Event, July 6–8, 2021, Revised Selected Papers. Springer. 2022, pp. 1–23
- Schubotz, René , Torsten Spieldenner, and Melvin Chelli (2021). "stigLD: Stigmergic Coordination of Linked Data Agents". In: *The 6th International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2021)*, December 17-19, Taiyuan, China
- Schubotz, René, Torsten Spieldenner, and Melvin Chelli (2022). "stigLD: Stigmergic Coordination in Linked Systems". In: *Mathematics 10.7 (2022)*, p. 1041

– Spieldenner, Torsten, and André Antakli. "Behavior Trees as executable representation of Milner Calculus notations". In: *Web Intelligence and Intelligent Agent Technology (WI-IAT-2022)*, November 17-20, Ontario, Canada. Nov. 2022

Chapter 8 will summarize the findings of the thesis, provide a conclusion derived from the individual findings, and close with an outlook over future work.

Chapter 2

Fundamentals

This chapter will revisit core concepts that are elementary for the contributions of this thesis. Section 2.1 will summarize the design principles of Linked Data architectures, as Linked Data is the digital medium chosen in this thesis to represent virtual agent environments. The section will moreover give an overview over the RDF (Resource Description Framework) data model as established resource representation format in Linked Data architectures, and finally, summarize core concepts of the SPARQL query language as one of the main interaction methods between Linked Data clients (such as agents) and Linked Data media. Section 2.2 will revisit Milner's Calculus of Communicating Systems (CCS) which will be used in the remainder of the thesis to formally define the communication between agents and environment via the medium. Section 2.3 will briefly introduce the coordination principle of *stigmergy*, on which medium-based coordination and optimization will be realized in this thesis.

2.1 Linked Data

Linked Data [14, 15] was coined as a term by Tim Berners Lee, and refers to a set of design principles that lay basis for the Semantic Web [12]. In its core, Linked Data can be considered as a publication method for structured data, in which related pieces of information are connected by explicitly modelled links. User agents that consume Linked Data are capable to follow these links, and by this, to autonomously explore Linked Data sets. Apart from links between related data sets, data itself is published in machine readable representations, typically semantically annotated to make data moreover *machine understandable*. The standard format to publish Linked Data and its semantics is the *Resource Description Framework (RDF)*[2] [35], with SPARQL [194] being the standard query language to interact with RDF data.

Loosely based on Berners Lee's design notes[3], implementations of Linked Data services should follow the following core principles:

[2] `https://www.w3.org/TR/rdf11-primer/` *(Visited May 2023)*
[3] `https://www.w3.org/DesignIssues/LinkedData.html` *(Visited May 2023)*

1. Linked Data services should be considered components of a *Resource Oriented Architecture (ROA)*, in which "things" (sic) described by the Linked Data service should be referenced by a unique identifier, specifically, a *unique resource identifier (URI)*.
2. Identifiers should be *resolvable*, in the meaning that users and user-agents may follow an identifier as link to find more information about the identified resource.
3. Resources should provide their content to users and user-agents in a "meaningful" way, following standards like RDF or SPARQL.
4. Finally, resources should provide links to resources with additional information that may be helpful to interpret the current resource's content.

Linked Data by this builds a network of *hypermedia* [27], i.e., a network of pieces of media (data), which are interconnected with related media contents using *hyperlinks*.

2.1.1 Resource Oriented Architectures

In [237], we wrote:

A Resource Oriented Architecture (ROA) is built around the notion of a *Resource* as common representation for and kind of virtual or real-world entities. [87, 154]. A resource is typically characterized by a name (identifier), its representation and links between resource representations [154]. As defined by Fielding, a representation is a sequence of bytes and metadata to describe those bytes. A resource may be described by more than one representation at any given time i.e., provide the same content for example in different serializations formats. More detailed considerations of this architecture can be found in [87].

Resource oriented architectures are typically implemented around RESTful [209] interfaces for CRUD (*Create, Read, Update, Delete*) operations to read and modify the contents of resources, or publish new resources to an existing ROA. Richardson [89] suggested a model to evaluate the maturity of a RESTful architectures with respect to their suitability as hypermedia environment. The following sections will detail more on the respective design principles.

REST

REST (Representational State Transfer) refers to an architecture principle for distributed Web applications [86]. In its core, REST describes principles of how servers manage data and client access to the data in resource oriented architectures. Following REST principles aims at guaranteeing scalability on Internet-scale while providing a unified, well-defined interface across the components.

In „RESTful“ Web applications (Web applications built around REST principles), resources are entirely governed by the server, and in particular, the state of the resources as communicated by the server towards any client is independent of the clients‘ state. The communication between server and clients in RESTful applications is therefore moreover considered *stateless*.

The resource-oriented nature of RESTful architectures allows for decoupled application components that can be developed and evolve independently of each other. Access to the single components is defined by a unified interface as follows:

- The underlying architectural concept is that of a *resource oriented architecture*, i.e., any component, object, or entity is represented by an individual resource.
- The respective resources are unambiguously identfiable by URIs (universal resource identifiers).
- Clients may retrieve a *representation* of the resource, e.g., a JSON or XML serialization of the maintained data. The representation delivered to a client is generally not considered the same as the representation in which the server maintains the resource internally. However, the representations communicated to clients are semantically equivalent to the representation maintained by the server.

CRUD (create, read, update, delete) access on resources is typically implemented via HTTP, with the client specifying the method with which it intends to interact with the resource:

GET : Returns a representation of the requested resource's state, as maintained by the server.

PUT : Create a new resource, or replace a resource state according to the state submitted by the client.

POST : Let the resource handle the submitted data to update the state of the resource according to the logic implemented by the server.

DELETE : Delete the respective resource.

HTTP defines moreover as methods to interact with resources beyond CRUD operations:

HEAD : Similar to `GET`, but returns only the header of the response object, e.g. for clients to check the expected size of the body object before downloading potentially large objects.

OPTIONS : Provides the client with a list of permitted communication objects.

PATCH : Updates the state of a resource partially with the data submitted by the client.

CONNECT : Requests to open a two-way communication tunnel between client and resource, e.g. a TCP or WebSocket connection.

HATEOAS

HATEOAS [193], short for Hyper Media as the Engine of Application State, is a design principle and constraint for RESTful architectures and Web applications. The declared goal of HATEOAS constrained Web applications is that clients understand and are able to interact with the API without any prior knowledge. All information required for clients to interact with the API are provided by the API itself.

This is typically achieved by providing clients with information about available state transitions on a resource, when requesting the current state of the resource. These state change options usually provide callable endpoints that trigger the respective state changes, and a machine-interpretable description of which resource is affected in what way by the state change.

Possible implementations of HATEOAS hypermedia have, for example, been defined in the RFC 5988 Web Linking specification [184], the JSON Hypermedia API Language (HAL) [135], and others [165].

Richardson Maturity Model

The Richardson Maturity Model [89] (RMM) measures the maturity of a Web API with respect to its suitability as hypermedia endpoints. The RMM defines four levels:

Level 0 : The application does not organize its data in resources. Web access mostly reflects remote procedure calls on methods provided by the API. By this, data access is completely proprietary, requiring clients to implement the specific interaction methods as provided by the server. Level 0 is also refered to as the "POX" (Plain Old XML) level, as communication is often implemented using XML encoded messages.

Level 1 : Data is organized in resources, but the operations on the individual resources are encoded within the resource URL via method names and parameters. While Level 1 already achieves a certain level of decoupling application components, access is still defined via proprietary interactions which clients have to implement specifically.

Level 2 : Employs a RESTful architecture, with interaction with resources implemented via HTTP verbs.

Level 3 : Like level 2, but uses hypermedia to inform the client about possible state transitions by providing (semantically annotated) links, and a description of the expected state change, if the client decides to follow the link.

For applications to fulfill the principle of "Hypermedia As The Engine Of Application State" (HATEOAS, [193]), they need to reach in particular Level 3 of Richardson Maturity Model (3-RMM).

In short, a 3-RMM compliant application provides its data as Web Resources, unambiguously identified by resolvable URIs, over which the application state is read and changed using HTTP operations `GET`, `PUT`, `POST`, `DELETE`. Clients can follow links from resources to others to explore data self-driven. Using HTTP operations `HEAD` or `OPTIONS`, clients can learn all necessary information about the state of the resource, and the provided methods to interact with it.

3-RMM compliance is considered a crucial requirement for the design of decoupled Linked Data applications [254, 191].

2.1.2 The RDF Data Model

The Resource Description Framework (RDF) is a W3C Standard to describe resources, and the relation between them, in the Semantic Web [35, 206]. The description format is built around the notion of *triples* of resources which together span *RDF Graphs*. An RDF triple is typically described of a form *(s, p, o)*, with the individual elements being referred to as *subject*, *predicate*, and *object* position respectively. An element within a triple is also commonly referred to as *node*.

An RDF triple encodes a single statement about a resource *s* in subject position by putting it into relation of a resource *o* in object position, with the relation further specified by the linking predicate *p*.

RDF defines three node types: IRIs (*Internationalized Resource Identifiers*) [80] unambiguously identify specific resources by the respective IRI. As known from Web site addresses in the World Wide Web, IRIs may be resolvable by Web clients, such that Web clients may visit the resource and retrieve additional data from the resource identified by the respective IRI. However, IRIs in an RDF Graph do not necessarily need to be resolvable.

Literal nodes directly encode a value within the RDF graph. Literal nodes typically specify the value itself, and may further specify a datatype as which the value is to be interpreted.

And last, *Blank Nodes* indicate the existence of a resource, the content of which is directly explicitly specified within the same graph, but without assigning an identifier to the resource.

Listing 2.1 shows a simple example of an RDF Graph, written in Turtle syntax[4] [55]:

[4] `https://www.w3.org/TR/turtle/` *(Visited May 2023)*

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix rdf:
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

<>  rdf:type  foaf:Person ;
    foaf:knows [
      rdf:type foaf:Person ;
      foaf:name "Alice"^^xsd:string .
    ] .
```

LISTING 2.1: Example of statements about a resource encoded in an RDF graph, using the *foaf* vocabulary, and written in Turtle Syntax.

The example describes the resource that emits the presented RDF graph (denoted as "<>") as information about a *Person*, in the semantic meaning as defined by the FOAF (*Friend of a Friend*) ontology[5]. `rdf:type` and `foaf:Person` are both IRIs, abbreviated using a respective *prefix* as defined in the beginning of the example. The example continues to express that the person described by the resource knows another person, using as predicate `foaf:knows` to define this relationship. The resource describing the other person is not referenced by name, but directly described in terms of a blank node (encapsulated in "`[]`" in Turtle syntax). The blank node specifies the other person's name, "Alice", in terms of a Literal node. The type annotation "`xsd:string`" specifies the value "Alice" to be interpreted as string.

Formal Notation of the RDF Data Model

The remainder of the thesis will employ the following formal notation to refer to data described in terms of RDF Graphs as also described in [234]:

I, **L** and **B** define pairwise disjoint infinite sets of IRIs (Internationalized Resource Identifiers), literals and blank nodes, respectively. We will refer to elements in the union set in $\mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$ as *RDF terms*. The subset $\mathbf{T} = \mathbf{I} \cup \mathbf{L}$ of RDF terms denotes resources in some universe of discourse.

The infinite set of all RDF triples is $\mathcal{T} = (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{T} \cup \mathbf{B})$. Asserting an RDF triple (s, p, o) expresses that some resource, denoted by p, establishes a binary relationship between the resources denoted by s and o.

An *RDF graph* $G \subset \mathcal{T}$ is then a finite set of RDF triples of the form (s, p, o).

[5] `http://xmlns.com/foaf/spec/` *(Visited May 2023)*

2.1.3 SPARQL

The SPARQL Query Language (SPARQL) W3C recommendation is the quasi standard to read and modify RDF datasets[6] [194]. SPARQL queries are built around a graph pattern matching facility, i.e. triple patterns, which form the core of the language. Triples in triple patterns may consist of IRIs, blank nodes, and literal values, as for RDF graphs, and additional *variables* that during query evaluation are bound to actual values from the RDF graph against which the triple pattern is matched.

SPARQL defines several types of queries for different purposes: `SELECT` queries match supplied triple patterns against an RDF graph and return the set of all valid variable bindings. `CONSTRUCT` queries are additionally supplied with a *graph template* into which variable bindings as result of the pattern matching operation are inserted to return a new RDF graph. `ASK` queries return as results a boolean value `true`, if any valid variable mapping was matched against a selected graph, and `false` otherwise. All three types of queries leave the graph against which they were executed untouched.

SPARQL UPDATE[7] moreover defines operations to directly modify the graph against which the query is evaluated: `INSERT` queries, like `CONSTRUCT`, are supplied with a graph template from which a new RDF graph is formed using the resulting variable mappings, but merge the resulting graph with the graph against which the query was evaluated. Conversely, SPARQL `DELETE` operations remove all triples that result from inserting mapped variable bindings into a supplied RDF Graph triple. Listing 2.2 shows the example of a SPARQL UPDATE query, that updates some local agent knowledge for the agent to select some resource to annotate with a marker, out of a selection of previously collected candidate resources.

```
DELETE { <http://me> :next ?last . }
INSERT { <http://me> :next ?candidate .}
WHERE {
  <http://me> :next ?last.
   <http://me> :candidates ?candidate .
   FILTER NOT EXISTS {
     <http://me> :marked ?candidate .
    }
}
```

LISTING 2.2: Example of a SPARQL UPDATE query that selects the next candidate for a marker annotation.

[6] W3C SPARQL 1.1 Query Language Recommendation: `https://www.w3.org/TR/sparql11-query/` *(Visited May 2023)*

[7] W3C SPARQL 1.1 Update Language Recommendation: `https://www.w3.org/TR/sparql11-update/` *(Visited May 2023)*

Interaction between Linked Data clients and Linked Data sets using SPARQL via HTTP are defined by in the SPARQL Protocol recommendation.[8] In the following, we briefly re-visit the SPARQL syntax and evaluation semantics, and the SPARQL 1.1 HTTP protocol.

SPARQL Syntax and Evaluation Semantics

Let $\mathbf{V}$ denote an inifinte set of *variable nodes* that is disjoint from $\mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$, with $\mathbf{I}$, $\mathbf{L}$, $\mathbf{B}$ identifier, literal, and blank nodes, as defined in Section 2.1.2. Let E define a SPARQL expression, then we denote the set of variables in E as $var(E)$.

A *triple pattern* is then a tuple of the form $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$[9]. Triple pattern components may be bound, i.e. from set $\mathbf{T}$, or unbound, i.e. from set of variable nodes $\mathbf{V}$.

From triple patterns, SPARQL *graph patterns* are defined recursively using binary operators `AND`, `UNION`, `OPT`, `FILTER`, `SERVICE` and `GRAPH` as follows:

1. A triple pattern $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$ is a *graph pattern*.
2. If P_1 and P_2 are graph patterns, then the expressions $(P_1 \circ P_2)$ with $\circ \in \{\texttt{UNION}, \texttt{AND}, \texttt{OPT}\}$ are graph patterns.
3. If P is a graph pattern and $a \in (I \cup V)$, then the expressions $(a \texttt{ GRAPH } P)$ and $(a \texttt{ SERVICE } P)$ are graph patterns.

The semantics of SPARQL graph patterns is defined in terms of an evaluation function $[\![\cdot]\!]_G^{\mathcal{D}}$ that returns a set of mappings for a given SPARQL graph pattern expression, a fixed and *active dataset* $\mathcal{D} = \{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$ and an *active graph* G within $\mathcal{D}$. A *mapping* from $\mathbf{V}$ to $(\mathbf{T} \cup \mathbf{B})$ is a partial function $\mu : \mathbf{V} \to (\mathbf{T} \cup \mathbf{B})$. The domain $dom(\mu)$ of a mapping is the subset of $\mathbf{V}$ on which μ is defined.

The notion of the SPARQL evaluation function $[\![\cdot]\!]_G^{\mathcal{D}}$ will be used in Section 5.3 to define the evaluation semantics of the service presented in the respective section.

For a detailed formal definition of the evaluation semantics of the different types of SPARQL queries, as well as the keywords `UNION`, `AND`, `OPT`, `GRAPH`, and `SERVICE`, we would like to refer to the original specification, and literature [28].

[8] W3C SPARQL 1.1 Protocol Recommendation: `https://www.w3.org/TR/sparql11-protocol/` *(Visited May 2023)*

[9] This Triple Pattern recommendation allows literal nodes as subjects. Even though literal nodes are not allowed as subject in RDF graphs, the option to support them in triple patterns was discussed by the RDF-core working group, and therefore included in the SPARQL recommendation: `https://www.w3.org/2000/03/rdf-tracking/#rdfms-literalsubjects` *(Visited May 2023)*

	Query via `GET`	**Query via** `POST` (URL enc.)	**Query via** `POST` **(direct)**
Method	`GET`	`POST`	`POST`
Query Param.	`query` (exactly 1) `default-graph-uri` (≥ 0) `named-graph-uri` (≥ 0)	None	`default-graph-uri` (≥ 0) `named-graph-uri` (≥ 0)
Content Type	None	`application/` `x-www-form-urlencoded`	`application/sparql-query`
Body	None	URL-enc, &-sep.: `query` (exactly 1) `default-graph-uri` (≥ 0) `named-graph-uri` (≥ 0)	Unenc. SPARQL query string

TABLE 2.1: HTTP operations as permitted by the SPARQL protocol: Clients may execute SPARQL queries against remote SPARQL processors via HTTP `GET` or `POST` operations[11]

SPARQL Protocol

The SPARQL Protocol W3C recommendation specifies how Linked Data clients may access and modify datasets by supplying SPARQL and SPARQL UPDATE queries in HTTP requests.[10]

For this, clients may send requests via HTTP `GET` or HTTP `POST` against the endpoint of the respective SPARQL processor, which is usually using the URI of the dataset against which the query is to be executed.

Depending on the type of request the clients send, they specify the query which is to be executed, and, optionally, URIs of a specific graph within the data set, as query parameters. The set of allowed operations, respective HTTP headers, and content as expected by the SPARQL processor according to the SPARQL 1.1. protocol specification, is shown in Table 2.1.

The SPARQL 1.1 protocol specification will be referred to in Section 5.3 to define a SPARQL 1.1 compliant HTTP interface for the service presented in the respective Chapter.

2.2 Milner's Calculus of Communicating Systems

The Milner Calculus of Communicating Systems (CCS) is one of many formal calculi to define and describe the behavior of concurrent processes [177]. It describes process in terms of (named) actions and process identifiers, as well as connecting operators. Expressing system behavior in calculus notations like CCS is shown to be beneficial as it allows to evaluate the qualitative correctness of system properties, such as deadlock or live-lock.[78, 26].

[10] W3C SPARQL 1.1 Protocol Recommendation: `https://www.w3.org/TR/sparql11-protocol/` *(Visited May 2023)*.

[11] Source: `https://www.w3.org/TR/sparql11-protocol/`, *(Visited May 2023)*

In the scope of this thesis, CCS will be used to formally define the interactions offered by the Linked Data medium (see Sections 4.3.2 and 4.3.2), as well as the interaction between Linked Data consuming agents and the medium (see Chapter 6).

CCS operators used to describe processes in the thesis are defined as follows:

Action:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P' \quad (K = P)}{K \xrightarrow{\alpha} P'} \tag{2.1}$$

Let α denote a single *action*, P a process, and K a process *label*. Then a Process $\alpha.P$ denotes a process that first executes action α, and then continues as process P. If a process P can continue as a process P' by executing an action α, and P is assigned the label K, then the process K may continue in the same way as process P.

Choice:

$$\frac{P \xrightarrow{\alpha} P'}{(P + Q) \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{(P + Q) \xrightarrow{\alpha} Q'} \tag{2.2}$$

Let P, Q be processes that by executing an action α continue as processes P', Q', respectively, then $P + Q$ denotes a process that may, by executing an action α, continue either as process P' or process Q'. The choice as which process to continue is usually assumed to be non-deterministic.

Parallel Composition:

$$\frac{P \xrightarrow{\alpha} P'}{(P \mid Q) \xrightarrow{\alpha} (P' \mid Q)} \qquad \frac{Q \xrightarrow{\alpha} Q'}{(P \mid Q) \xrightarrow{\alpha} (P \mid Q')} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{(P \mid Q) \xrightarrow{\tau} (P' \mid Q')} \tag{2.3}$$

Let P, Q be processes which may continue as P', Q' after executing an action α respectively. Then $P \mid Q$ denotes a process that executes P and Q simultaneously, or subsequently. However, contrary to $P + Q$, executing either P or Q does not terminate the execution of the second process.

For an exhaustive description of the calculus operators and their semantics, we refer the reader to [177].

We moreover use as action names in this paper: $\overline{req_i}$ for requests sent by an agent via some channel i, res_i for request responses received by an agent via channel i, $\eta(i)$ as function that resolves a resource identifier i to its callable endpoint, $\overline{ans}$ for query answering initiated by the agent, and *qres* for query results received by the agent.

2.3 Stigmergy

Stigmergy originally refers to a coordination principle in nature, mainly found as underlying coordination principle in insect swarms [245], which has also been thoroughly researched for application in AI multi agent systems [250, 251, 105, 59].

In collective stigmergic systems, groups of *agents* perform work by executing *actions* within their environment [114, 116, 117]. An action is considered a causal process that produces a change in the environment. Agents choose actions based on condition-action rules, and perform an action as soon as its condition is found to be met. Conditions are typically based on environmental states as *perceived* by the agent. Examples from nature are the presence of specific (food) resources, semiochemical traces, progress in building nest structures, etc. Which actions an agent can perform, how the agent will perform them, and which condition-action rules an agent will follow, is considered the agent's *competence* [118]. The part of the environment that undergoes changes as a result of executing an action, and the state of which is perceived to incite further actions, is called the *medium*.

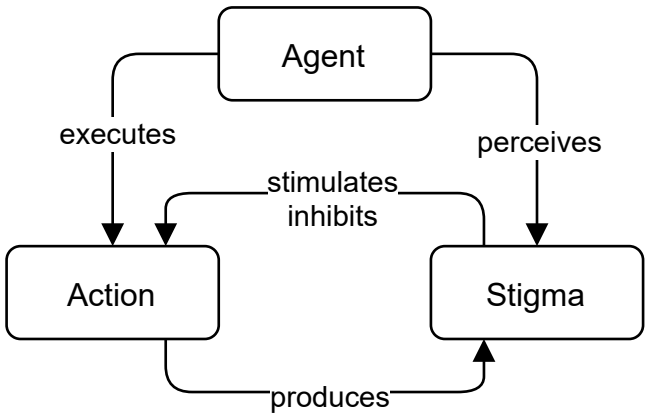

FIGURE 2.1: Action-Mark-Cycle of stigmergic systems, as originally published in [220, 219].

Each action produces, either as byproduct of an action, or the deliberate goal of the action itself, a *stigma* in the medium. Consequently, the behaviour of agents in a collective stigmergic system can be understood as a cycle of executing actions based on existing stigmata, and as result, leaving stigmata that stimulate or inhibit future actions (see Fig. 2.1). In essence, stigmata work as indirect communication mechanism between agents [247], potentially leading to coordination between agents, and, ideally, a self-organising behaviour of the entire system [114, 116, 117]. Based on these core concepts, i.e. *action*, *medium* and *stigma*, stigmergic systems can be further classified [116]. In *sematectonic* stigmergy, a stigma is a perceivable modification of the environment as result of work that was carried out by the agent, e.g. giving some new shape to a working material, or re-arranging order of objects in the world. In *marker-based* stigmergy, stigmata are markers, e.g. semiochemicals, that

are specifically added to the environment as means for indirect communication between agents. When perceiving stigmata, agents may choose their actions based on the mere existence of a stigma in the medium (*qualitative stigmergy*), or also take into account quantities, like semiochemcial concentration levels, number of stigmata left, etc (*quantitative stigmergy*). Moreover, stigmata present in the medium may stay until actively being removed by an agent (*persistent stigmata*) or until dissipated over time due to agent-less processes (*transient stigmata*).

Since the concept of stigmergy was coined as inherent underlying principle of coordination found in nature, it has faced a history of thorough research [245]. There is a profound understanding of the many variations of stigmergic systems, and how these are suited to model and implement efficient, flexible, and scalable algorithms for AI-based coordination and optimization [68, 116, 117]. Stigmergy is recognized as suitable underlying principle for multi-agent systems [106, 105, 250, 247] and is applied in a variety of practical domains, e.g. digital manufacturing [252], robotics [142, 126], or public transport [129, 4].

Stigmergic systems can be considered a variation of *situated agent systems*, in which the interaction of agents with their environment is reduced to direct reaction based on perception, rather than complex knowledge processing and inference [263, 262, 267]. Principles in these systems were also developed around an indirect, influence-based interaction mechanism between agents and their environment as chosen for our proposed stigmergic system [82].

Web technologies have been found a suitable basis for implementation of multi agent systems [52, 51, 121, 127]. Meanwhile, it came to attention that stigmergic principles are the underlying concept of many applications in the World Wide Web [70] including coordination in Web-based IoT systems [201].

Chapter 3

Related Work

This chapter will review literature and existing systems and solutions relevant to the contents of this thesis. Section 3.1 will thoroughly review existing approaches and frameworks to provide data in terms of Linked Data media. Section 3.3 will provide an overview about existing works about hyper-media based multi agent systems. Section 3.2 will present technologies to lift existing data to Linked Data, and by this, allow to provision environment data via Linked Data media. Finally, Section 3.4 will highlight on the concept of medium-based coordination and optimization, putting the focus on the nature-inspired coordination principle of *stigmergy*.

3.1 Linked Data Media

This section reviews existing solutions and technologies to publish contents to a read-write Linked Data medium.

3.1.1 Triple Stores / Graph Store Protocol

Triple store or graph store databases are the default technology to publish Linked Data to the Web. They allow to store potentially large data sets in triple representation, which can be modified by clients either by HTTP operations, or using SPARQL queries to explore and modify the data.

Popular triple store solutions come either as standalone solutions, such as Apache Fuseki[12], Eclipse RDF4j[13], Open Link Virtuoso[14], or as non-persistent in-memory graph store as part of established RDF programming frameworks, as for example Python's RDF Lib[15], dotNetRDF[16], or as in-memory solution of the previously mentioned RDF4j for Java.

[12] `https://jena.apache.org/documentation/fuseki2/` *(Visited May 2023)*
[13] `https://rdf4j.org/` *(Visited May 2023)*
[14] `https://virtuoso.openlinksw.com/` *(Visited May 2023)*
[15] `https://github.com/RDFLib` *(Visited May 2023)*
[16] `https://dotnetrdf.org/` *(Visited May 2023)*

Access to data in triple stores is defined by both the quasi standard of the SPARQL Protocol W3C Recommendation[17], which allows to modify data by submitting respective queries (see also Section 2.1.3), or using the HTTP Graph Store Protocol[18], another W3C Recommendation that allows to access and modify contents in the triple store using the following HTTP Operations:

GET : Returns an RDF serializiation of an entire RDF graph within the requested RDF dataset. The URI of the graph that is to be returned is specified as query parameter.

PUT : Used to send an RDF payload to the server. The server creates a graph within the dataset according to a `graph-uri` parameter submitted as parameter of the query. If a graph with the same URI already exists in the data set, it is replaced by the payload of the latest request.

POST : Merges the payload of the query with the content of the graph which is specified via a `graph-uri` parameter. If no graph exists under the requested URI, the graph store returns a 404 error code (*not found*).

DELETE Deletes the graph which is specified via a `graph-uri` parameter. If no graph exists under the requested URI, the graph store returns a 404 error code (*not found*).

Graph stores provide a simple, but versatile tool to publish Linked Data, and by this, may be considered an important building block for distributed Linked Data media. However, limitations in the scope of multi agent systems come from that graph stores are a completely passive medium, meaning that data is entered into the stores, and modified, exclusively from external services, such as agents.

3.1.2 Linked Data Platform

The W3C Linked Data Platform (LDP) recommendation[19] defines a model to publish distributed data on the Web. This model is based on a data meta model, developed around a minimal set of RDF resources, a set of rules how to access those, and the format, in which the content is to be delivered.

LDP applications follow a resource oriented architecture (see also Section 2.1.1): LDP applications are organized in terms of *containers* that contain *resources*, with the distinction between *LDP RDF Resources (LDP-RS)* and *LDP Non-RDF Resources (LDP-NS)* (see also Fig. 3.1). While the first, LDP-RS, are required to provide their contents in terms of an RDF graph, LDP-NS may provide any type of content, including binary data, such as images or videos.

[17] `https://www.w3.org/TR/sparql11-protocol/` *(Visited May 2023)*
[18] `https://www.w3.org/TR/sparql11-http-rdf-update/` *(Visited May 2023)*
[19] `https://www.w3.org/TR/ldp/` *(Visited May 2023)*

A LDP server hosting LDP resources needs to provide at least HTTP/1.1 access to its hosted resources. For LDP-NS, the server is required to at least allow HTTP `GET` requests, and return the content of the resource, whereas for LDP-RS, the server is required to also support data modification using `PUT`, `POST`, `DELETE` request, the semantics of which are very much aligned with the graph store protocol as presented above.

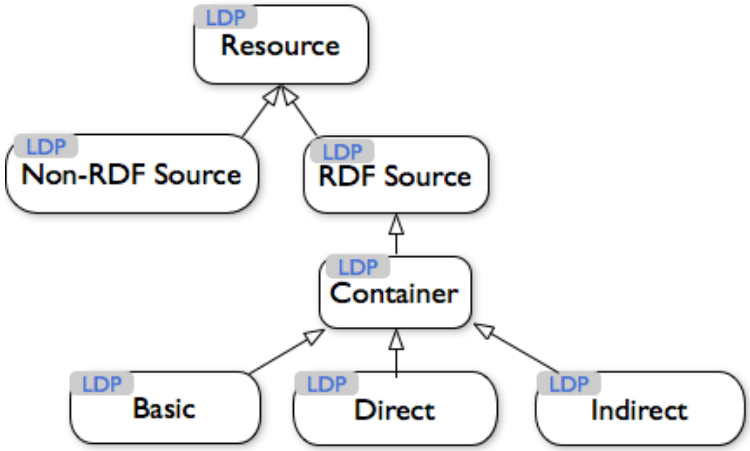

FIGURE 3.1: High level structure of the Linked Data Platform.[20]

LDP *Containers* are a specific type of LDP-RS that specifiy their content in terms of an RDF graph, which in turn contains links to the managed resources, and, depending on the container type, specifies the containment relation between container and resources.

By the defined concepts and interaction patterns, data structured in terms of Linked Data Platform containers allows clients to autonomously explore data, based on the containment relations of the resulting LDP graph. Containers may moreover be used to group subsets of data from larger data sets, and give them additional semantic meaning (see also Fig. 3.2): The LDP container architecture serves as overlay over existing (potentially distrubuted) RDF graphs to group and encapsulate resources with similar roles and semantics. The well-defined relations between containers, their contents, and as result, container hierarchies, enables clients to explore and LDP architectures autonomously to gather the information they are looking for.

A variety of implementations of both LDP servers and clients exist[21], among them the LDP server *Solid*, which has become popular as approach to "decentralize the social Web" [214, 161].

Altogether, the Linked Data Platform is a prime example of how to organize data in a Linked Data medium while fulfilling HATEOAS principles. For this reason, in the scope of this thesis, LDP will be used as underlying technology in some of the

[20] Image source: `https://www.w3.org/TR/ldp/#fig-ldpc-types` *(Visted May 2023)*
[21] `https://www.w3.org/wiki/LDP_Implementations` *(Visited May 2023)*

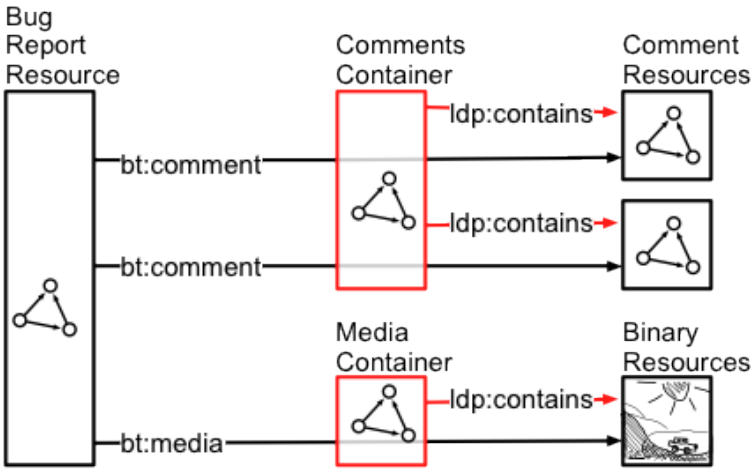

FIGURE 3.2: Example of how to re-organize and enrich existing data with additional semantic meaning using LDP containers.[22]

use cases presented in Chapter 7, and moreover is chosen as target format to publish real-time simulations as Linked Data media in Section 5.2.

3.1.3 Linked Data Notifications

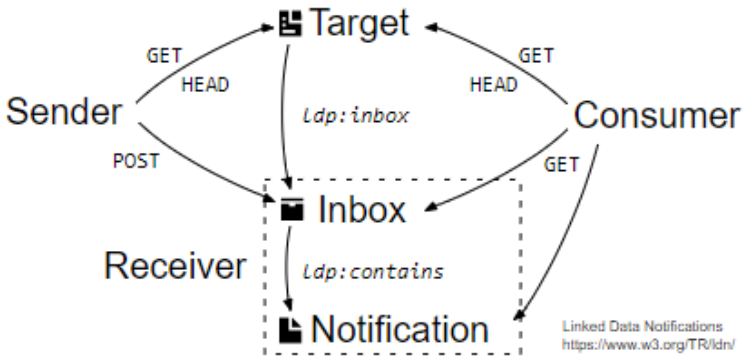

FIGURE 3.3: Linked Data Notifications Architecture.[23]

Linked Data Notifications (LDN) is a W3C recommendation for a protocol for message-based communication between Linked Data applications[24] [38, 29]. The design goal of LDN is to provide messaging capabilities between applications without the communication being tied to specific application technology. LDN are designed around Linked Data Platform principles: Resources in a LDN network provide at least HTTP `GET` endpoints (with *Inbox* resources supporting moreover HTTP `POST` to receive notifications), and provide their contents, which include a self-description of the resource, in JSON-LD [25] format.

[22] Image source: `https://www.w3.org/TR/ldp-primer/` *(Visited May 2023)*
[23] Image Source: `https://www.w3.org/TR/ldn/` *(Visited May 2023)*
[24] Linked Data Notification recommendation: `https://www.w3.org/TR/ldn/` *(Visited May 2023)*
[25] JSON-LD project website: `https://json-ld.org/` *(Visited May 2023)*

The LDN protocol defines a push-pull mechanism for notifications, based on the roles of a notification *sender*, a *target* that provides links to message *receivers*, and a *consumer* (see also Fig. 3.3). *Target* resources advertise message *receivers*, i.e., *inboxes* with a managed set of messages, within the HTTP `Link` header in the response to any HTTP `GET` request. Senders may use the advertised endpoint to send a LDN notification object to the respective inbox via HTTP `POST`, whereas consumers retrieve the collection of previously posted notifications via HTTP `GET`. Inboxes do not provide capabilities to sort or filter messages. Filtering and sorting is left to the consumer clients, allowing them to implement custom filter and sorting algorithms, based on the particular clients' purposes.

Linked Data Notifications have moreover been extended with streaming capabilities [29], and to support different Linked Data serialization formats, such as NGSI-LD [212]. LDN have moreover been studied as a protocol for agent-to-agent communication in multi-agent systems [30].

A number of LDN implementations exists, such as *dokieli* [37], and the *Solid*-based *In-box* [132].

Linked Data Notifications demonstrate how indirect communication in Linked Data media can be implemented.

3.1.4 The Web of Things

The *"Web of Things"* (WoT) is a W3C[26][101], originally recommended by the Web of Things working group to employ Semantic Web technologies as a remedy for the fragmentation of the IoT (Internet of Things) [260, 151] market. The observed fragmentation stemmed from the large number of different protocols, APIs, and data formats that are nowadays present in the zoo of IoT devices [5], a situation that some see to change only slowly, if at all, until 2023 [124]. For this, the WoT working group suggests to use the Semantic Web as integration layer for semantically enriched, interconnected WoT devices [17, 228, 139]. For this, numerous architectures frameworks have been proposed to simplify the publishing of WoT data to the Web of things [190, 160, 11, 13, 139], or for searching for WoT devices in the Web of Things [203, 222].

A standardized data and interaction model for WoT devices ist the Web of Things *Servient*[27] [123] (see also Fig. 3.4).

[26] W3C Web of Things Architecture Specification: `https://www.w3.org/TR/wot-architecture/` *(Visited May 2023)*

[27] `https://www.w3.org/TR/wot-architecture/#sec-servient-implementation` *(Visited May 2023)*

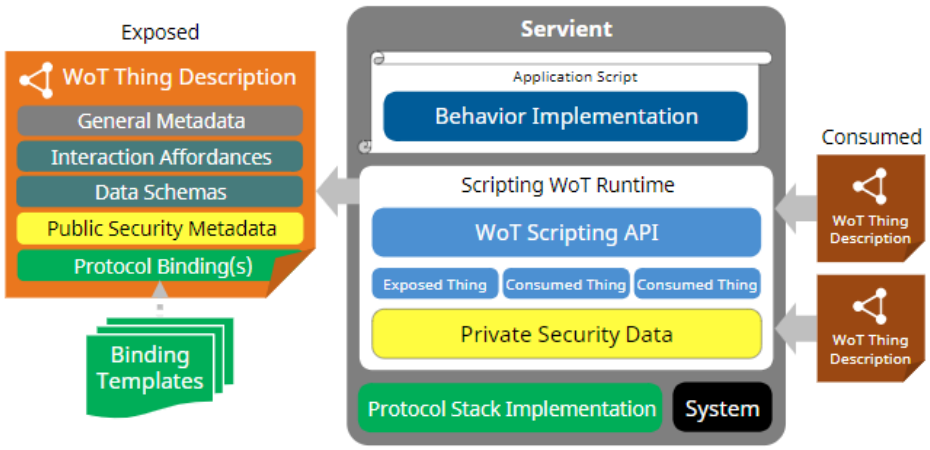

FIGURE 3.4: Architecture of a Web of Things servient[28]:

The word „Servient" is portmanteau and stems from the words „server" and „client", indicating that the respective software element takes the role of both a server, providing WoT thing descriptions to other Servients, and a client, reading data as client from WoT Devices, and server, providing WoT thing descriptions to other Servients.

Servients consume as input *WoT Thing descriptions* that are emitted by other Servients. Based on the received input, the WoT Servient runs its internal behavior, implemented on top of the WoT protocol stack and scripting API, and as a result, emits a WoT Things Description of its own state, capabilities, and intentions.

The Web of Things demonstrates how to implement device interaction via a shared Linked Data medium, following HATEOAS principles: The object self description of WoT objects helps user agents to understand purpose and usage of the respective thing. However, the respective approach is limited to resources describing actual Web Things, whereas this thesis searches a more general approach to employ Linked Data for an agent-understandable representation of MAS environments.

3.2 Linked Data Lifting and Processing

From the beginning of the Semantic Web, the mapping of existing structural data to RDF is an active research area. The so called *lifting and lowering* of data is required to access and modify any non RDF datasource from Linked Data applications. In addition to lifting non-RDF data to an RDF representation, research also investigated means to translate semantic queries into non-semantic ones to allow Linked Data-like interaction with traditional query APIs, such as SQL databases. In the scope of this thesis, related technologies are surveyed as underlying principles to publish environment data to Linked Data media, and to define interaction with Linked Data

[28]Image source: `https://www.w3.org/TR/wot-architecture/#sec-servient-implementation` *(Visited May 2023)*

media. The following literature review has previously been published in similar fashion in [234].

RDF-Lifting approaches

A typical approach to publish static data via Linked Data media is the lifting from relational data base contents to a Linked Data representation.

A prominent example for lifting relational databases to RDF is *R2RML* [58]. R2RML specifies mappings from database schemas to RDF graphs in RDF turtle syntax. A respective mapping file describes table structure and content, and where to inject the data from these tables into a target RDF graph.

Several extensions to R2RML have been presented: The tool *Karma* by Gupta et al. [103] is a visual tool to define R2RML mappings by annotating relational data. Karma supports the user by inference of suitable target mappings from previously annotated data. The resulting mappings can be published as service with REST API for online data conversion. Dimou et al. extended R2RML to *RML* [65, 67, 66], a superset of the R2RML mapping language, that also allows for mapping of structural datasources that are not stored in relational databases. Source formats for RML include CSV, TSV, XML, and JSON.

Slepicka et al. present *KR2RML* [229], a different approach to extending R2RML to support heterogeneous sources, that keeps in mind extendability and scalability with respect to changes in the source data; aspects in which Slepicka et al. see shortcomings in RML. Finally, the implementation *CARML*[29], an extension to RML, allows dynamic input streams as input to an RML mapping, instead of specifying the to be mapped source directly in the mapping file. This is a crucial feature for re-using mappings for a variety of different structurally equivalent source files.

The presented approaches cover the translation of legacy data to Linked Data data dumps. Most of them take offline approaches to lift data to RDF, a way we found impractical for live data. The presented approaches fail in particular to publish dynamic environments to a Linked Data medium. The thesis presents as remedy an online lifting approach in Section 5.2.

SPARQL query interfaces to non-RDF datasources

Instead of creating Linked Data copies from non-Linked Data sources, resarch also investigated technologies to evaluate semantic queries directly on legacy data sources. Respective considerations are relevant in the scope of this thesis to ensure continuity of the medium during link traversal.

In 2004, Bizer et al. presented *D2RQ* [16]. D2RQ provides a mapping language from relational database schemata to RDF, similar to R2RML. Clients may send requests

[29]CARML GitHub Repository: `https://github.com/carml/carml` *(Visited May 2023)*

to the platform to perform SPARQL queries against a database, or explore a database as Linked Data, while the D2RQ platform performs the lifting of the database to RDF transparently, based on a previously defined mapping.

Similar approaches realize SPARQL-to-SQL mappings by employing R2RML based liftings [211, 202, 32]. For non-relational datasources, Michel et al. present an approach to query the document-based MongoDB by employing *xR2RML* [175, 170], an extension for R2RML for non-relational sources.

SPARQL-Generate [148] integrates RDF generation from non-RDF datasources directly into the SPARQL-query itself. This removes the need of separately provided mapping files, however, requires that the target SPARQL processor implemenents SPARQL-generate on top of SPARQL 1.1 .

Finally, *SPARQL-Microservice* [172, 173] provides a SPARQL query interface to wrap existing, JSON-based Web APIs.

Of the reviewed technologies, many target relational databases rather than Web APIs, with D2RQ as one chosen example. However, SPARQL Generate requires an extended SPARQL implementation beyond the generally used specification. SPARQL Microservice requires cumbersome configuration during deploy time. It is moreover limited to JSON-LD as lifting result, which makes it difficult to impossible to use it in scenarios where the source data are not JSON, or when another result representation is needed.

To ensure continuity of the medium during link traversal as sought by Research Question **R7.**, a more generic and flexible approach is required. A respective solution is presented in Section 5.3.

3.3 Hyper Media environments for Multi Agent Systems

The importance of a properly defined medium and its technical implementation has been thoroughly described by Weyns et al. as result of the E4MAS Workshop Series [262, 266, 133].

Soon after the Semantic Web had been envisioned, it has sparked the idea of the Semantic Web being the underlying infrastructure for multi agent systems [112]. However, the adaption of Semantic Web technologies took its time, with actual concepts for Linked Data based agents being published more than 10 years later [64], including the vision of the "Internet of Agents" [272, 198, 197]. Still to this day, despite its promising features, the Web and hypermedia enabled Linked Data agents have yet not been fully embraced to their potential [51, 50].

However, a number of approaches describe Linked Data based agents and multi agent systems: We presented a Linked Data model for robotic components in MAS in [221]. Boztepe et al. presented a framework to implement Linked Data MAS

for mobile devices [22]. Noura et al. describe the implementation of hypermedia driven autonomous agents that interact with Web of Things devices [185]. Hypermedia has been employed to define MAS environments in a way that allows agents to learn new behavior from the provided information [249]. Schraudner et al. presented an HTTP and RDF based infrastructure for agents in manufacturing environments [218], along with an application of the infrastructure for stigmergic coordination [217]. Modelling agent behavior based on Linked Data environment representations as executable Finite State Machines in the shape of behavior trees has been thoroughly described by Antakli et. al [6, 7].

Based on these previous results, it is therefore safe to assume that the Web is suitable to carry information about MAS environments. However, a common understanding of the Semantic Web and Linked Data as a basis for MAS implementations is still lacking. This thesis tackles this shortcoming by strengthening the notion of a *medium* as carrier of environment information, and as an interactive interface to environment elements, and subsequently, describing Linked Data as a suitable digital agent medium.

Research has moreover been conducted to model and formally describe the interaction between agents and a hypermedia environment. In the domain of robotics, formal descriptions of autonomous systems have been thoroughly surveyed in [155]. With *ActivForms* [122, 264] exists a formal model to describe and verify self-adaptive systems, which very much relates to the description of self-coordinating stigmergic system behaviors in this thesis. Käfer and Harth proposed a formal model, based on Abstract State Machines, to define the interaction between Linked Data agents and Linked Data Platform environments [128]. Similarly, this thesis will provide a formal model of agent-environment interaction via Linked Data media. A generic, well-defined interaction model may help to improve modelling MAS behavior, taking the interactive medium component that provides the environment into account. Carefully defining the interaction between agents and Linked Data media in particular may help to further drive research and development of hyper media MAS in the Semantic Web.

3.4 Medium-based and decentralized coordination

3.4.1 Generative Communication and Tuple Spaces

Generative Communication [56] refers to a paradigm that was first introduced by Gelernter and his colleagues in the scope of a distributed programming language for the system "LINDA". [91]

The paradigm is born out of the observation that, to this time, distributed programming languages mostly relied on three main mechanisms: shared variables between processes, direct message passing, and remote procedure calls. Gelernter et al. stated

that while these mechanisms allowed to decouple processes in space (i.e., distribute them over physically detached machines), they were still coupled in time, meaning that remote process invocation happened instantly.

To allow decoupling of processes also in time, they introduced a shared data-space called "Tuple Space" (TS) for inter-process communication. As the name suggests, the TS manages a set of (ordered) tuples which are written to or read from the TS by communicating processes. This concept has later been extended to the use of unordered multi-sets [24], or logic terms [39].

The inter process communication via TS works as follows: Some process A enters a Tuple with j typed parameters of the form $(N, p_2, ..., p_j)$ into the TS via an operation `out(N,`p_2`, ...,`p_j`)`. The first parameter is always assumed to be of string type and defines the name of the tuple. Once entered, the tuple is accessible and can be read by any other process B. Reading from the TS is provided in two ways: processes may either `read()` a Tuple with chosen name in the TS and leave the tuple as such untouched; or processes may withdraw the tuple form the TS and store it internally by calling `in()`.

A formal description of process interactions via Tuple Spaces, and the respective operations have been formalized in a CCS like manner by Ciancarini et al [48].

Tuple spaces had soon been adapted also in other systems but LINDA, in the scope of which TS were originally introduced, and specifically been adapted as indirect communication mechanism for distributed MAS. For this, Tuple Spaces have been extended to *Tuple Centers* which, in addition to just provisioning data, adds logic to specify behavior based on the logic [187]. Examples are the ReSpecT (*Reaction Specification Tuples*) tuple centers [60] that defines a first order logic to describe reactions on tuples within the tuple space, or the system system LuCe [188] (*"Logic Tuple Centers"*), that employs a tuple space for what they call a "coordination medium" (and in this role, relates closely to how the term "medium" is further coined in the scope of this thesis). Finally, Tuple Spaces have lately been extended to a *Spatiotemporal Tuple Model* [41] to express temporal constraints on validity and context, in which tuples are to be interpreted, a model that in its core relates to considerations of time-dependent traces and dynamic environments in Section 4.3.3 of this thesis.

Generative communication, in combination with *aspect oriented software design (AOSD)* [137], has moreover generally been found to be a suitable coordination mechanism for scalable, maintainable, re-usable distributed MAS [143, 23]. Methods of generative communication around the notion of Tuple Spaces and Tuple Centers are to this day used to implement reactive and event-driven distributed Multi Agent Systems [40, 163, 162, 31].

By its originally defined role, Tuple Spaces and Tuple Centers very much take the role of a generic medium for indirect multi agent communication. Being built around the notion of a tuple as information element, it is in a way comparable to Linked

Data provided in terms of RDF graphs and triples. However, TS are not intended to represent interactive, explorable multi agent environments, a gap this thesis intends to close by employing hypermedia Linked Data as medium.

This thesis goes beyond the presented approach by limiting the role of the medium not only to indirect communication, but also as interactive representation for multi-agent environments.

3.4.2 Self-Coordinating and stigmergy-based MAS

In [237], [220, 219], and [43], we reviewed literature about self-coordinating and stigmergy-based MAS as follows.

The concept of stigmergy (cf. Section 4.2.2) has been thoroughly analyzed as underlying coordination priniciple in Multi Agent Sytems [273, 208, 59, 49]. Applications of stigmergic and self-coordinating algorithms include robotics [167, 142, 126] and cyber-physical manufacturing [49, 218]. Inspiration from the concepts of evaporation and replenishment of pheromones in the ant world have moreover led researchers to explore ways of managing the dynamic requirements of telecommunication networks [19]. It has also been shown that aggregated motion data of vehicles in a city can be used to predict traffic hot-spots and thus plan optimal routes for navigation [129, 4].

Ant inspired optimization techniques have been a focus of research since the early 1990s, and various variations have been developed ever since [71, 115]. Bio-inspired algorithms like stigmergy have shown to achieve good performance in decentralized decision making tasks [126]. They also perform well in situations where agents need to recruit peers and coordinate with them for task accomplishment [142]. High dimensional numerical optimization has also been approached using stigmergy (using the Differential Ant-stigmergy Algorithm, DASA) [140]. A distributed variant of the *Hungarian Method* for solving the Linear Sum Assignment Problem (LSAP) was shown by [47], where multiple agents cooperate to find an optimal solution to LSAP without sharing memory, or a central command structure.

Self-organizing multi agent systems and agent systems that rely on stigmergy as coordination mechanism have been exhaustively reviewed in [43]. This review concludes that a common understanding of such systems is widely lacking, and suggests a generic domain model to describe self-organizing system. From the review, we conclude additionally that the interaction between agents and environment is often described only vaguely, and is generally underspecified. As a solution, this thesis provides a formal and generic specification of hypermedia driven agents and the respective agent-medium interaction for stigmergic systems.

3.5 Related Work: Conclusion

The literature review allows the conclusion that interaction of agents with their environment, and between agents, have been to this day an ever present topic of research to improve the quality, efficiency, and scalability of multi-agent systems. It has by this come to attention early that a proper description of how the agent programs utilize the technology that provides information about the environment, and other agents, takes a central role in the overall design of a multi-agent system (Section 3.3).

Research on agent-to-agent communication shows that considerations similar to those done for agent-environment interaction, need to be taken account for communication and interaction between agents as well. *Generative communication* seems to be a promising candidate to establish indirect agent-to-agent communication via a shared medium. It has moreover been shown how, taking inspiration from processes in nature, an effective agent-environment interaction, in combination with an indirect agent-to-agent communication via the environment, may achieve scalable self-coordination and -optimization effects (Section 3.4).

However, for a concise understanding of how the interaction and communication in multi-agent systems can be used as leverage to achieve efficient high quality coordination and optimization effects - after all, the purpose of every multi-agent sytems -, the considerations of the underlying technology need to be extended from mere agent-environment interaction to the other aspects of the MAS as well, namely communication between agents, and the opportunity for coordination effects to arise. These different aspects - environment provisioning, interaction, and communication - have so far been considered mostly separate from each other, whereas systems that synthesize these aspects to a working system, leave out the consideration of the underlying interactive technology that ultimately enables this synthesis in the first place.

This thesis for this endeavours to take into account this technological component that in the end needs to be capable of handling environment provisioning, interaction, and agent-to-agent communication, as third core building block of MAS besides agents and the environment. This technological component will be referred to as the *medium* of the MAS.

As reviewed in Section 3.1, *Linked Data* provides a number of promising technologies that support publication of heterogeneous environments to a Linked Data layer with well defined interaction patterns, be it in Linked Data Platform container architecture, by implementing indirect communication via Linked Data Notifications, or by lifting legacy data to a Linked Data representation in triple stores with graph store protocol interaction mechanisms (Section 3.2).

Consequently, the thesis will define the role of a *medium* for multi-agent systems based on formal definitions of interaction patterns between agents and the medium.

Agent-environment as well as agent-agent interaction are then defined on top of the agent-medium interaction. The thesis moreover discusses Linked Data as suitable candidate technology for MAS media, and demonstrates how to use Linked Data technologies to finally implement a working MAS medium component.

Chapter 4

MAS Environments in Linked Data Media

Despite the thorough analysis of multi agent systems in literature, and sophisticated agent models that were derived from the resulting findings, the pivotal role of a properly implemented, robust, interactive environment is often overlooked.

This chapter will first elaborate on the notion and role of a digital agent medium in Section 4.1, and then derive a set of requirements that digital media need to fulfill to provide a solid shared interactive representation of multi agent environments in Section 4.2. Following, inspired by the concept of *stigmergy* [245, 115] this thesis will discuss Linked Data as suitable medium for multi agent systems. Finally, Chapter 4.3 will provide a formal definition of static and dynamic (stigmergic) Linked Data medium server, as originally published in [220].

4.1 The notion of a digital Agent Medium

The partitioning of a MAS as shown in Figure 1.1 imposes on the medium the role to represent elements in the environment from the Artifact Space in a virtual form to agents, such that agents are able to sense, interpret, and act upon artifacts and their respective individual status.

The medium by this serves as connection point between the purely virtual Agent Space on the one side, and the physical, and potentially completely non-virtual, Artifact Space. It builds the bridge via which virtual agents finally get access to the environment.

The term *medium* may by this very well be understood the way as it used to refer to media by which we as humans broadcast and exchange information: We learn from happenings around us in processed and condensed from from news papers. We follow current events that are outside our own perception via TV broadcasts, or hear about them on radio. And we interact with friends or colleagues indirectly by leaving messages and notes in social networks.

Transferring these concepts to the world of virtual agents, practical examples for a digital agent medium could be the variable space of an application run-time, or the sets of resources in a resource oriented architecture, basically everything from which virtual agents can read, and to which virtual agents can write information back, or generally, anything a virtual agent can directly interact with.

For a medium to fulfill such a role, it has to match a number of requirements: it must be able to properly represent (physical) entities in the environment along with their features, while providing this information in a time critical fashion, such that agents gain the ability to sense the situation of the real-world scenario. In order to influence elements in the Artifact Space by correcting measure during optimization, the medium must also allow agents to interact with the represented entities. Last, the medium should allow multiple agents in the same environment to exchange information and draw agents' attention to relevant pieces of information within the medium.

Summarizing the above, the *medium* is the intersection between the Agent Space and Artifact Space: Agents read from the Medium and interact with it directly, while the Medium provides to agents in an interpretable and interactive fashion all information about, and access to entities in the Artifact Space. As we will see in the following sections, the Medium representing the coordination domain further needs to allow *interaction* between agents and coordinated entities, allow *communication* between agents, and finally, allow *coordination* effects to emerge, as will be elaborated on in the remainder of this section.

4.1.1 Medium as indirect interaction Space

The most naive way of interaction between agents and their digital environment is granting the agents the power to inflict direct changes according to their program: Agents observe a specific feature in the medium, choose a proper reaction according to their program, and write their changes back to the medium. On a very technical level, the inflicted change will most often be implemented by changing respective variables, objects, or resources from which the medium is built. For simple setups with only very limited interaction, this approach is totally valid. The well known simple planner problem of an ape climbing a box to grab a banana, which is typically used as introduction to STRIPS, is such a simple scenario: when solving the problem using AI agents, an agent would move the (simulated) ape's position directly by setting the respective property to "ape on box". The agent thus has the total power over what is happening in the scene

However, it has come to attention early, that such a direct influence of agents causes problems already for only slightly more complex problems that are solved by more than one agent at once. Ferber et al. [82] have illustrated the problem at agents

moving a virtual box. Agents have the power to directly apply changes to the environment according to the desired outcome. Suppose two agents would like to move the box in different directions at the same time. Both agents determine a direction in which the box should move, calculate the final position of box, and set the variables encoding the position of the box accordingly. Now by doing so, the action of one agent would override the action of the other. As a remedy to this problem, and more complex alike ones, Ferber et al suggest an indirect, influence-based interaction mechanism between agents and their environments. Instead of writing the desired effect of their action directly back into the environment, agents rather leave *influences*, and the environment takes care of applying the result of the influence to the entities within it. Applied to the example of moving a box, agents would then, instead of moving the box by directly setting the position, the agents would describe their desired *influence* onto the box, for example by describing a direction and intensity of a force the they would like to apply. The environment, at a given, controlled time would pick up the influence, and evolve the environment and its entities accordingly, e.g. by applying a simulation of force on rigid objects in the given example.

While in the original paper [82], this approach was suggested to avoid erroneous simulation states from conflicting agent messages, we can directly transfer the idea of indirect interaction to the notion of an *Indirect Interaction Space* that is constituted by a proper agent space medium: Obviously, virtual agents can not directly inflict changes from their virtual agent space onto artifacts in the physical artifact space. This is even true for artifacts that provide access to sensors or actuators via programmable interfaces, which is also the core principle for devices to span the *Internet of Things*: It is not the agents that ultimately perform routines on said devices, but the embedded programs in the devices themselves. The agents only give an influence to the device by calling a respective interface function. Deciding whether to accept the call in the first place, and then how to execute it, is entirely depending on how the device is programmed. Interaction with devices via API calls may for example be rejected due to lacking credentials, an improper choice of communication protocol by the agent, insufficient resources on the device itself, or many more. The execution, finally, may be part of a bigger program that also adds side effects that the agent was not aware of, and would not have taken in account, if given the power to directly change the device's parameters.

Based on this observation, the understanding of the Medium being not only a *Representation Space* that describes the Artifact Space to agents, but actually an *Interaction Space*, from which agents can take influence on the world in the Artifact Space, becomes very valid. As Interaction Space, the Medium actually takes two roles:

First, the medium can serve as a layer for indirect interaction between agents in Agent space and artifacts in Artifact Space. This may be achieved by embedding artifact's APIs into the medium, which may take the form of storing callable API

endpoints as Remote Procedure Call (RPC) stubs if the medium is an environment of runtime variables, or by embedding callable HTTP endpoints into a collection of resources in a Web-based architecture.

Second, when agents desire to apply changes to the medium itself - attempting to write variable, add resources, etc. - the medium should be capable of taking the role of ultimately deciding whether or not to accept change requests to allow for an influence-reaction based interaction pattern, similar to the model suggested by Ferber et al. This supports collision resistant collaboration of many agents within the same medium, a crucial requirement for large scale multi agent systems.

4.1.2 Medium as Communication Space

When considering the medium as a space not only for agent-environment interaction, but also for agent-agent interaction, then it easy to attest the medium not only the role of representing the environment, and allowing (stable) interaction with it, but it moreover can take the role of an agent-to-agent Communication Space.

For this, instead of agents directly communicating to each other, they put pieces of information into the medium, which may be picked up, interpreted, and acted upon by other agents. This kind of communication is often refered to as *generative communication* [56], and the method has probably received most attention as underlying communication principle in the agent system LINDA [91] (see also Section 3.4.1).

The advantage of such an approach lies in its robustness, its flexibility, and its scalability. In direct agent-to-agent communication, agents would typically open a communication channel to other agents in the system. Depending on the system design, this communication may be implemented by mere procedure calls, if a group of agents is hosted within the same application or process, it may be handled by some form of inter-process communication, or, for distributed systems that are run on several instances of separate hardware, by network communication channels.

Introducing a new class of agents to such an existing system during design time is considerably difficult. Not only does this new class need to implement protocols and API calls to a number of existing agents, it moreover increases the number of potential communication protocols and APIs that future agents in the system need to speak and understand for the system to operate most effectively. When the system is scaled up and the number of agents is increased during run time, every single agent will have to establish connections to a potentially larger number of other agents, which leads to an exponential growth of total communication channels within the system. If during runtime one of the agent program terminates or crashes for unexpected reasons, the data it received until then is usually lost, which in worst case may lead to a failure of the system as such, if the information communicated to the crashed agent was crucial for reaching the goal.

Employing a shared medium as Communication Space between agents can provide a solution to all three of these issues: First, communication with respect to protocols and APIs is fixed to the set of APIs and protocols as offered by the medium. If these are well defined, then introducing a new agent class reduces to merely an effort of properly modeling a new class of communication piece within the medium. Scaling the system up will put no additional workloads on agents whatsoever, as their endpoint for interaction remains the same, no matter how many agents exist besides them, namely the medium itself. This clearly does require the medium to scale with number of agents that access the medium simultaneously. However, the increase of complexity is only linear per agent.

Depending on the technology on which the communicating MAS is built, an indirect communication system as described above could be achieved in many ways: Information could be stored in shared variables and accessed from various agent processes. Event and messages brokers such as MQTT[30] may offer communication channels to which agents write their information for it to be transmitted to subscribed agents. While originally designed to transmit messages to subscribed entities in real-time, some of those, MQTT actually being one of them, also offer to retain the last message sent via a channel and deliver it to newly connecting clients. This way, agents that join the system later during run-time and sign up for the respective channel would still be provided by the information left by another agent. If the chosen medium is a database, agents may write their messages to the database, where they will be persisted until other agents conveniently retrieve the information. If the database is schema-less, as for example MongoDB or Apache CouchDB, agents are even completely free in the design and content of the message objects they are writing to the storage. Finally, if the medium is designed around a resource-oriented architecture, messages may be written to a dedicated resource container (for example a Notification in a dedicated Linked Data Notifications notification inbox), or modelled as individual resources, with the conveyed information being the content.

While stigmergic systems heavily rely on indirect communication, it should be stressed that in a system that employs medium-based communication, agents are not limited to indirect messaging. Take as example a resource-based medium, built around web technologies: of the communication channels discussed above, message queues, for example, allow to publish their routes in terms of callable web endpoints. Web resources, generally, may serve as HTTP endpoints for request-response communication. By this, agents may as well offer routes for direct communication, and advertise those to other agents using respective artifacts within the medium.

Independent of the chosen implementation, it can be concluded that a properly chosen medium can not only realize the previously discussed agent-to-environment interaction, but it can moreover provide all means for agent-to-agent interaction as

[30]MQTT Website: `https://mqtt.org/` *(Visited May 2023)*

well. It is moreover safe to assume that having agent-environment-interaction encoded in the same medium as agent-to-agent interaction simplifies the implementation of agents communicating about relevant entities or interactions: With messages residing in the same medium as the topics of interest, agents can refer to relevant entities or interaction endpoints by merely referencing to the respective representation within the medium: the corresponding variables, database entries, resources, or any other kind of representation that the medium is using.

4.1.3 Medium as Coordination Space

To this point, above Sections coined the medium as a component of a cyberphysical system in which the virtual elements of the Agent Space manifest, and by this as the component that realizes representation of, interaction with, and communication about the Artifact Space for agents in said cyberphysical system. It now depends on how the agents interact with the system for coordination to arise.

A prime example of agent-environment interaction that follows exactly this motivation, *Stigmergy* [115], comes from the field of nature inspired algorithms. Nature inspired algorithms, as the term would imply, adapt behavior as observed in nature, typically from insect swarms. Stigmergy was coined as a term by Pierre-Paul Grassé [96], and refers to communication and coordination concepts such swarms. The core principle can be summarized as that when carrying out work, members ("agents") in a swarm react to traces left by previous work by other members of the swarm. The work then performed by this particular agent again leaves a trace that subsequently will function as incentive for particular actions of other agents. It can be observed that from this influence-reaction cycle, a goal directed swarm behavior emerges: Termites, for example, are inclined to build upon structures that were constructed by their fellows before, which ultimately results in the typical majestic clay buildings. Ants, when foraging for food, leave traces of pheromones that attract other ants of the swarm. If by chance one ant finds a food source, more and more ants of the swarm will follow the same trace to carry food from the source back to the hive, with each ant increasing the pheromone concentration, and thus attracting more ants. In the end, the hive will have established roads that lead directly to the most lucrative food sources. Stigmergic behaviour is by this never goal directed, or planned. Agents react only to their direct environment, without the capabilities of keeping memory, or planning, and in particular unaware of any concept of *goal* [116, 117].

Such behavior has been thoroughly analyzed in computer science, and been transferred to multi agent systems as a means to achieve coordination in many fields, be it coordination of robot swarms [167, 142], coordination of cyber-physical production, storage and supply optimization [43, 218, 237], public transport [129, 4], and many more. An interesting observation that can be made is that in stigmergic systems – be

it in nature or in computer science – the actual coordination emerges from the interaction of the *with the medium* [113]: It is the medium where agents leave their trace, and it is the medium that agents work on. Be it the clay from which the termite hive is formed, the ground that ants leave their traces and walk on, or the set of variables, objects, or resources that virtual agents write and modify as "trace" for other agents to follow. Coordination of the underlying system is then rather a side-effect of the agents' mutual interaction, and the agent-medium interaction. This point of view on stigmergic system may become even clearer when taking into account the aforementioned concept of indirect, influence based agent-environment interaction: In a stigmergic system, it is not that agents deliberately execute routines on an IoT circuit with the intention to switch a traffic light from red to green, or move a conveyor belt of a production line forward. In a system coordinated by stigmergy, agents will only be led to the interaction endpoint offered by the device in the medium, and, if incited to interact with it, leave an influence on this endpoint that is finally executed by the device itself. The result of the interaction can in the following be observed by other agents that may – intuitively – react to it in a way of which they are totally unaware of the result, but that may have a correcting effect on the overall system. For obvious reasons, systems coordinated by stigmergic principles, are often considered *self-organizing*, and have been found to be particularly scalable, adative, and robust towards changes within the optimization scenario [99, 20, 159, 181, 158].

Obviously, a medium that fulfills the roles discussed before – space for representation of and interaction with entities of the artifact space, and communication space for indirect agent-to-agent communication – also fulfills all requirements for a working self-organizing system based on stigmergic coordination: It allows agents to observe and interpret their environment, which serves as incentive for actions carried out by the agents. It provides a space for agents to leave pieces of information as trace for other agents to interpret and follow. And it provides means for agents to perform influence on elements in the artifact space via indirect interaction, which allows coordination effects to transfer from the medium to the actually to-be-coordinated scenario. By this, given that a medium provides a proper representation of an artifact space to the agent space, allows (indirect) interaction of the agents with entities in the artifact space, and allows agents to use the medium for (indirect) agent-to-agent communication, the originally sought goal, namely coordination, can be achieved directly in the medium, and therefore the medium can, besides a representation, interaction, and communication space, be considered a *coordination space* to achieve self-organized coordination.

4.1.4 Medium: Summary

To summarize, research about agent systems almost exclusively revolves around the concept of agents, underlying agent models, or representation and model of environments in which agents dwell and with which them interact. However, from above

discussions, it is valid to assume that the medium that provides data to agents, and to which agents write data back, deserves at least as much attention as said agents and environments: While MAS strongly depend on how agent-to-agent and agent-to-environment interaction can be provided in a fault tolerant, efficient, and scalable fashion, it is the exactly the medium that enables those features: The environment is represented to agents by representing the environment within the medium; ideally, the medium allows interaction with the environment via elements that themselves are part of the medium; and embedding agent-to-agent interaction into the medium finally enables coordination effects to emerge.

4.2 Linked Data as Medium for (Stigmergic) Multi Agent Systems

In the previous sections, it has been established that a properly chosen medium would be taking the role of an (indirect) *interaction and communication space* for agents, and would thus allow for coordination effects to emerge within the medium. This use of a medium is reflected in systems that employ the nature-inspired coordination principle of stigmergy, in which an emerging goal-directed swarm behavior of (living) agents is observed from the agents' interaction with their environment, further driven by indirect communication via traces left in said environment [245, 115].

With stigmergic systems relying on all three desired features for digital agent media – (indirect) interaction, communication, and by this, lastly, coordination – this section discusses Linked Data as suitable *digital stigmergic medium*, and by this, generally as a suitable medium in multi agent systems.

This section is a revised version of the original publication, *"Linked Data as Stigmergic Medium for Decentralized Coordination"*. [237]

4.2.1 Motivation

Among research that focuses on improving application of classic AI agent-based planning and optimization techniques, there is a specific trend that employs nature inspired algorithms to solve coordination and optimization problems [46, 248]. A re-occurring element from this class of algorithms is the use of concepts of *stigmergy* [115] (cf. Section 4.1.3). Such nature inspired algorithms have been found a promising approach for more flexible, fault-tolerant, and scalable coordination of complex systems in above mentioned domains [142, 208, 126, 59].

However, the majority of stigmergic systems in literature are highly use-case specific, or even implementation specific. It is in particular noticeable, that while focusing on specific algorithms and their implementations, existing work marginalizes or ignores the importance of a proper *medium*. The medium is the environment in which agents leave traces by performing actions, i.e., where stigmergic effects

emerge, and it is considered the *"the mediating function that underlies the true power of stigmergy"* [113]. The question of what a proper medium constitutes is particularly interesting in the world of AI optimization and coordination, where the medium has no tangible physical manifestation, but where agents operate entirely in an environment that is completely digital, while having only abstract correspondences to real world entities.

We believe that both research and application of stigmergic coordination algorithms can benefit greatly from a concise common understanding of a digital stigmergic medium. This medium should be provided by employing widely accepted open standards to be independent of specific use-cases, domains, or technologies that implement the algorithm. It should be based on a well-defined, thoroughly formalized and established foundation to allow for soundly defined, general, transferable solutions.

Such a set of standardized and well-defined tools comes from the world of the *Semantic Web* [12]. Based on the notion of *Linked Data* [14] and typically modeled in terms of the *Resource Description Framework* (*RDF*)[31][147] , the Semantic Web is commonly promoted as a generic integration layer for applications from various domains.

With the Internet, being the maybe largest digital medium that relies on a wide variation of stigmergic effects [70], and being built around technologies and principles closely related to those of the Semantic Web, one may expect that a machine readable and writable Linked Data layer (*read-write Linked Data)* in the Semantic Web may as well provide a widely standardized, established, domain-independent interactive medium for stigmergy-based coordination [69, 201].

To this end, this chapter makes the following contributions: We derive from relevant literature a *set of requirements towards digital media to serve as proper stigmergic medium*; we *establish read-write Linked Data as suitable medium* for stigmergic systems; and we *demonstrate the application* of read-write Linked data as stigmergic medium *with an example from the domain of virtual manufacturing*.

4.2.2 Role of media in Stigmergic Systems

Relevant results of research in the field of stigmergy-based self-organization have been very concisely summarized by Heylighen [115]. We discuss stigmergy based on the findings from Heylighen's article. However, we would like to emphasize that by this work being a very thorough survey over the topic that covers research from several decades, we well include findings from many different researchers with different view on the general topic.

Heylighen derives from his findings his own definition of stigmergy as an:

[31] RDF 1.1 Primer document: https://www.w3.org/TR/rdf11-primer/ *(Visited May 2023)*

*"indirect, mediated mechanism of **coordination** between **actions**, in which the **trace** of an action left on a **medium** stimulates the performance of a subsequent action" [115, p. 5].*

The definitions of the core components of stigmergy as described in [115] are as follows. An *action* is considered as a causal process that produces a change in the world. The *medium* is the part of the world that undergoes changes because of the action, and also whose state is sensed to incite further actions. A *trace* is the perceivable change made in the medium by an action that can trigger subsequent actions. A trace that stimulates agents to perform a specific action, i.e. *affords* the action, is called *Affordance*. Affordances typically encode *condition-action* rules that trigger an agent to perform an action once a certain condition is met. A trace that keeps agents from performing a particular action, is called *Disturbance*.

Heylighen further identifies different variations of stigmergy, depending how the agents interact with the medium (pp. 19 – 27).

This includes the observation that a medium may be worked on by either a single agent, or crowds (*individual vs collective stigmergy*), that agents may take into account either mere existence of certain features in the medium, or quantities of those (*qualitative vs quantitative stigmergy*), that agents may react to direct results of work in their general environment (*sematectonic stigmergy*), or to markers that were deliberately left by other agents (*marker-based stigmergy*), that traces left by agents in the medium can stay until actively being removed by other agents (*persistent traces*) or over time dissipate and vanish (*transient traces*), and finally, that traces in the medium may be observable by every agent working on the medium (*Broadcast*), or only to a limited number of specific agents (*Narrowcast*).

For a very thorough elaboration on the various aspects that we covered here in a very shortened manner, we refer the reader to the original paper [115].

4.2.3 Linked Data as digital stigmergic Medium

In nature, the notions of *agent* and *medium* are determined by nature itself: Ants, for example, follow traces of pheromones left by other ants towards lucrative sources of food. Here, ants are the agents, and the ground and food are the medium. Termites use clay as medium, steered by how the shape of their nest (made of this very clay) has already formed.

When implementing stigmergy-based algorithms for coordination, *"agents"* are usually considered to be software AI user-agents. These agents operate on a *digital representation* of the to-be-coordinated concepts, which may correspond to real-world physical artifacts (e.g. physical production machines or robots in manufacturing scenarios, cars and traffic lights in traffic). This partition between digital and real world is common in agent-based coordination algorithms [68], with the partitions lately labeled as *Agent Space* for the digital, and *Artifact Space* for the physical representation space [43].

Requirements for (digital) Stigmergic Media

From the notions and variations of stigmergic media in the Section 4.2.2, we derive the following requirements that a digital medium should fulfill to be suited for use in stigmergy-based systems:

R1 ***(Representation)***: The medium must be able to *represent* entities of the domain in which the coordination algorithm performs, and relations between them. The medium thus serves as *Agent Space*. If artifacts in Artifact Space are target of coordination, the medium must be able to provide a *representation* of the physical entity in the Agent Space, and allow *access* to the physical entity via the mediums, e.g. to switch a real-world traffic light, or start a production process on a production machine.

R2 ***(Accessibility)***: The medium must be *accessible* to the agent in the meaning that an agent must be able to enter the medium to perform actions on it. Furthermore, the agent must be able to navigate through the medium to the point where an action is to be performed.

R3 ***(Observability)***: The medium must be *observable* (readable) for the agent to recognize conditions of condition-action rules to be fulfilled in the medium. For this, the agent needs to be able to at least observe the *existence of effects* (for qualitative stigmergy). The medium should further be suitable to provide:

R3.1 *(Interpretability)* of observed effects in the medium w.r.t the domain for the agent to correctly set the observed effects into relation with each other.

R3.2 *(Quantities)*: The medium must be able to express *quantities* for coordination by *quantitative* stigmergic effects.

R4 ***(Consistency)***: For *collective stigmergy*, the medium must consistently deliver the same information to different agents at the same point of time. In particular, if an agent induces a change in the medium, following this change, all other agents must observe the changed state as actual state of the medium.

R5 ***(Malleability)***: Agents must be able to *form* and *change* elements in the medium as result of their action. This includes both changing the state of existing entities within the medium (comparable to continuing the construction of a begun nest by a swarm of termites), and add or remove entities from the medium (comparable to leaving pheromone markers, or dissipating markers over time). However, changing the medium should happen in a controlled manner, leading us to the requirement of *Stability*:

R5.1 *Stability*: Any change to the medium must be possible *without inflicting unwanted effects* to resources outside the scope of a performed action. "Unwanted" is in this case not to be confused with changes that an agent "unintentionally" left as a trace, but to be understood as an effect that changes

the state of an entity beyond what was intended by the algorithm.

R6 ***(Scopes)***: The medium must be able to limit visibility of entities and effects in terms of scopes to allow *Narrowcast* of stigmergic effects.

Linked Data as Stigmergic Medium

We in the following show that read-write Linked Data fulfills all requirements towards a digital stigmergic medium.

Representation is achieved by the notion of representation space of resources (see also Section 2.1): Read-write Linked Data being built around resource oriented architectures provides both the tools and best practices of how to represent both real-world and virtual entities in terms of addressable resources. Connections to physical artifacts is established by having callable HTTP endpoints represented as resources within the medium.

Accessibility is achieved by building Linked Data around HATEOAS principles. Agents can access resources and read information from them by HTTP GET requests. All information needed to interact with a resource is provided by the server that manages the resource. Furthermore, Linked Data defines query interfaces as a common interaction method with linked data graphs. By employing graph query engines like SPARQL[32], agents can identify relevant resources as a result of the queries. Moreover, by following links between related resources, agents may explore Linked Data graphs autonomously. It is not required to host the medium being hosted on a single physical server instance to ensure accessibility. In fact, Linked Data principles state that resolution of URIs should happen transparent, and agents do not need to make assumptions where the actual data is hosted. This allows the medium to be hosted on distributed servers. Agents then only need to be pointed to an entry point (e.g. managing server), and may discover new information autonomously by following links provided in the Linked Data medium. For queries, SPARQL supports the integration of data from different distributed endpoints via federated queries using the `SERVICE` keyword.[33] Consequently, despite the distributed nature of the data, agents do not require previous knowledge about where relevant data may be hosted.

Observability: "Existence" of an effect is manifested in Linked Data by existence of a respective triple pattern in the Linked Data graph. By this, the existence of an effect as precondition for an action can be verified by matching expected triple patterns against the Linked Data graph via SPARQL queries. The statements encoded

[32]W3C SPARQL 1.1 Query Language Recommendation *(Visited May 2023)*: `https://www.w3.org/TR/sparql11-query/`

[33]SPARQL Federated Queries: *(Visited May 2023)*: `https://www.w3.org/TR/sparql11-federated-query/`

by triples are moreover *semantically interpretable* by software agents, as commonly established for Linked Data graphs.

Quantities can be modelled in Linked Data graphs either by using respective literal nodes that express a quantity in terms of a fitting datatype directly, or by the number of triples rendered to a respective resource.

Consistency is achieved by the notion of state and representation of resources in a Linked Data system as outlined in Sections 2.1 and 2.1.1. Access to resources, the represented concept, and modes of interaction are managed by the Linked Data server and provided to clients following HATEOAS principles. By the communication between clients and server being stateless (by following REST and HATEOAS principles), the state of the resource as communicated by the server towards clients is independent of particular clients, and by this, consistent among all clients.

Malleability is achieved by write-capabilites of read-write Linked Data. On resource-level, agents may change the state of a resource by PUT / POST / DELETE requests. On graph level, linked data provides possibilities to change elements in the medium using SPARQL UPDATE requests with INSERT and DELETE statements. The WHERE body of these states moreover allows to take into account preconditions that need to be fulfilled when performing the update.

Stability during updates is achieved by unambiguous identification of relevant resources via IRIs. By default, operations on resources do not have side-effects on other resources, and by this, will not inflict undesired changes to resources other than those that the action was performed on. On RDF graph level, stability during write operations is ensured by that adding triple statements to a resource does not interfere with triples already present: by adding triples, statements about a resource may only become more specific, but never eliminate statements that were present before new triples were added.

Scope can be expressed in read-write Linked Data either by specific triple statements on resources by which agents can filter for specific resources, or by using mechanics of Linked Data datasets and named graphs.[34] Different scopes, i.e. named graphs, are then accessed by agents for example by using `FROM` and `FROM NAMED` clauses in the respective SPARQL queries.

By finding all requirements **R1 – R6** fulfilled by and materialized via concepts of read-write Linked Data systems, we derive that read-write Linked Data is without limitations a suitable generic digital medium for stigmergy-based coordination.

[34] `https://www.w3.org/TR/rdf-sparql-query/#rdfDataset` *(Visited May 2023)*

4.3 Stigmergic Linked Systems: Definition

After having established Linked Data as suitable medium for MAS, this section will formally define the means by which agents may interact with MAS environments via LD media.

This chapter recaptures contributions from the article *stigLD: Stigmergic Coordination of Linked Data Agents* and *stigLD: Stigmergic Coordination in Linked Systems* [220, 219]

4.3.1 Motivation

As thoroughly discussed in Sections 4.2.2 and 4.2.2, the concept of stigmergy provides an indirect and mediated feedback mechanism between agents. By modelling agents in terms of condition-action rules based solely on perception of the agents' environment, it enables complex, coordinated activity without any need for planning and control, direct communication, simultaneous presence or mutual awareness. In this context, the *medium* has been established as the crucial part of a stigmergic system, given that *"it is its mediating function that underlies the power of stigmergy"* [117].

Accounting for the importance of distributed hypermedia environments as first-class abstractions in hypermedia MASs [267] and the environment's pivotal role in stigmergic systems, we examine the use of hypermedia-enabled Linked Data as a general stigmergic environment. For this, we propose in this Chapter to use a value-passing fragment of Milner's Calculus (CCS, see also Section 2.2) to formally specify generic, hypermedia-driven Linked Data agents and the Web as their embedding environment. Linked Data agents and their environment will be composed into a *Linked System* [108] (or equivalently a hypermedia MAS).

This chapter focuses on the formal definition of Linked Data media servers for static and dynamic MAS environments. The presented formal medium model will then serve as basis to define respective Linked Data agent models in Chapter 6.

4.3.2 Static Linked Systems

We first define a *Static Linked System* using the notation of Milner's Calculus for Communicating Systems (CCS). Please refer to Section 2.2 for a description of the employed notation.

We conceive a Linked Data server $\texttt{SERVER}_k$ as a reactive component that maintains an RDF graph G. It receives requests to perform a CRUD operation $op \in \texttt{OPS}$ on a resource i via channel req_k

$$\texttt{SERVER}_k(G) \quad = \quad req_k(op, i, G').\texttt{PROC}_k(op, i, G', G)$$

where $G' \subset \mathcal{T}$ is a (potentially empty) request body. The server employs a constrained set of operations to process client-initiated requests for access and manipulation of the server-maintained RDF graph G

$$\begin{aligned}
\texttt{PROC}_k(\texttt{GET}, i, G', G) &= \texttt{RESP}_k(\texttt{OK}, (\emptyset, \texttt{descr}(i, G)), G) + \texttt{RESP}_k(\texttt{ERR}, (\emptyset, \emptyset), G) \\
\texttt{PROC}_k(\texttt{PUT}, i, G', G) &= \texttt{RESP}_k(\texttt{OK}, (\emptyset, \emptyset), (G \setminus \texttt{descr}(i, G)) \cup G') + \texttt{RESP}_k(\texttt{ERR}, (\emptyset, \emptyset), G) \\
\texttt{PROC}_k(\texttt{POST}, i, G', G) &= \texttt{RESP}_k(\texttt{OK}, (\{i'\}, \emptyset), G \cup G') + \texttt{RESP}_k(\texttt{ERR}, (\emptyset, \emptyset), G) \\
\texttt{PROC}_k(\texttt{DEL}, i, G', G) &= \texttt{RESP}_k(\texttt{OK}, (\{i\}, \emptyset), G \setminus \texttt{descr}(i, G)) + \texttt{RESP}_k(\texttt{ERR}, (\emptyset, \emptyset), G)
\end{aligned}$$

where $i' \in \mathbf{I}$ is a "fresh" IRI with $\eta(i') = k$. The server responds to requests via channel $\overline{res}_k$

$$\texttt{RESP}_k(rc, rval, G) = \overline{res}_k(rc, rval).\texttt{SERVER}_k(G)$$

with return code $rc \in \texttt{RET}$ and with a linkset and response graph in $rval \in (2^{\mathbf{I}} \times 2^{\mathcal{T}})$.

4.3.3 Dynamic and Stigmergic Linked Systems

Dynamic system behavior can be achieved by including the dynamics into a simulation and publish the respective simulation directly as Linked Data. In this case, the environment evolution is driven by the underlying environment simulation, rather than the medium. A respective approach is implemented in detail in Section 5.2.

If the evolution of the environment, however, is limited to elements that are created by agents in the medium, e.g. marker traces in stigmergic systems, the environment dynamics can be modelled as part of the behavior of Linked Systems. In this case, environment evolution is governed by the medium and may be driven by queries that encode the evolution of respective medium elements. This section extends the notion of Static Linked Systems from the previous section to include such a reactive medium component. We described the extension of static Linked Systems to dynamic Linked Media in [220, 219] as follows:

A static linked system SERVER as specified previously provides an indirect, mediated mechanism of coordination between agents. Its static and passive nature enables the realisation of sematectonic and *persistent* marker-based stigmergy, as in these cases, changes in the environment are exclusively inflicted by agents. However, when considering some of the prime examples of stigmergy, e.g. ant colony optimization [75, 73, 72, 76] and termite colony optimisation methods [111], it becomes apparent that a purely reactive environment is insufficient for the implementation of *transient marker-based stigmergic* mechanisms.

In fact, a stigmergic environment typically demonstrates some immanent dynamics that may modify the environment's state independent of any agent's actions [116, p. 24]. These endogenous dynamics, e.g. diffusion, evaporation, dissipation, atrophy or erosion of stigmata, constitute a crucial component of transient marker-based

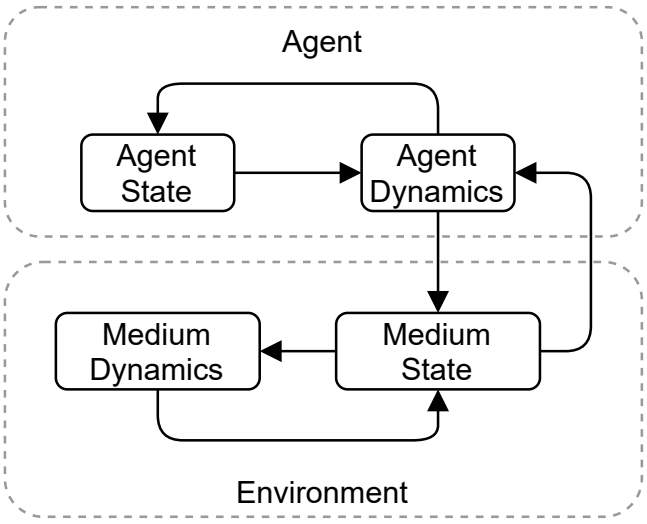

FIGURE 4.1: Stigmergic system components

stigmergic systems ([253], cf. Fig. 4.1), and more importantly, they are *not* subjected to agent-driven processes.

We call the part of a stigmergic environment that, in addition to being malleable and perceivable by all agents under coordination, *actively* drives the evolution of such agent-less dynamic processes a *stigmergic medium*.

Taking into account the notion of a stigmergic medium, we define a *stigmergic Linked System* as the parallel composition of the MEDIUM itself, the ENVIRONMENT with its dynamics and laws that drives the medium evolution, and the AGENTS that observe the medium state and influence its evolution:

$$\texttt{STIGMERGIC-LINKED-SYSTEM} \quad = \quad (\texttt{AGENTS} \parallel (\texttt{MEDIUM} \parallel \texttt{ENVIRONMENT}))$$

where the stigmergic $\texttt{MEDIUM} = \texttt{MEDIUM}_1 \parallel \cdots \parallel \texttt{MEDIUM}_l$ relates to the parallel composition of a collection of *extended* LD server components.

A $\texttt{MEDIUM}_k$ component is a Linked Data server that offers a constrained set of operations to access and manipulate server-provided resource states, but *in addition*, generates server-side side-effects [35]

$$\begin{aligned} \texttt{MEDIUM}_k(G) &= req(op, i, G').\texttt{PROC}_k(op, i, G', G)) \\ \texttt{RESP}_k(rc, rval, G) &= \overline{res}(rc, rval).\texttt{MEDIUM}_k(G) \\ \texttt{PROC}_k(\texttt{GET}, i, G', G) &= \texttt{EVOLVE}_k(i, G) \end{aligned}$$

as evolution $\texttt{EVOLVE}_k(i, G)$ of the environment during the handling of safe and idempotent agent-initiated resource request. The generation of such side-effects is subjected to an *internal* process

$$\texttt{EVOLVE}_k(i, G) = \texttt{RESP}(\texttt{OK}, (\emptyset, \texttt{descr}(i, G')), G'') + \texttt{RESP}_k(\texttt{ERR}, (\emptyset, \emptyset), G) \tag{4.1}$$

[35] We emphasise that this conception is not in violation with HTTP semantics [84, sections 4.2.1, 4.2.2][85].

where the result of executing an *evolutional query* $q_{\texttt{EVO}_k}$ over a given RDF graph G is given by $G' = \texttt{ans}(q_{\texttt{EVO}_k}, G)$ and the server state after an evolutional state update is $G'' = G \setminus \texttt{descr}(i, G) \cup \texttt{descr}(i, G')$. Executing an evolutional query drives the endogenous dynamics of $\texttt{MEDIUM}_k$ over time, e.g. diffusion and evaporation of semiochemicals, irrespectively of *agent-initiated requests for* resource state *change*.

Chapter 5

Implementation of Dynamic Environments in Linked Data Media

Providing static, merely passive agent environments via Linked Data media is rather trivial. These can be hosted by established triple store solutions (cf. Chapter 3.1). Changes within the environment are driven by requests from client user-agents against triple store APIs, be it RESTful APIs offered by the server (e.g. Linked Data Platform), or by clients using SPARQL interfaces to explore and manipulate the datasets.

Such solutions, however, fail to encode inherent dynamics within the environment, and thus fail to capture active components, or dynamics of an evolving environment as defined in Chapter 4.3.3. Following the specification of dynamic and stigmergic Linked Systems from the previous chapter, this chapter will provide two approaches of actual implementations of dynamic environments in Linked Data media:

- Section 5.1 presents a server component that employs a reactive evolution upon receiving, and before handling, client requests as formally defined in Section 4.3.3.
- Section 5.2 presents an approach for simulation-driven evolution of the environment with real-time provisioning of the respective Linked Data representation.
- Section 5.3 presents an approach that allows to employ semantic SPARQL queries over legacy endpoints that do not emit RDF data. This approach allows to include data that was originally not published with Linked Data support in mind, into the Linked Data medium.

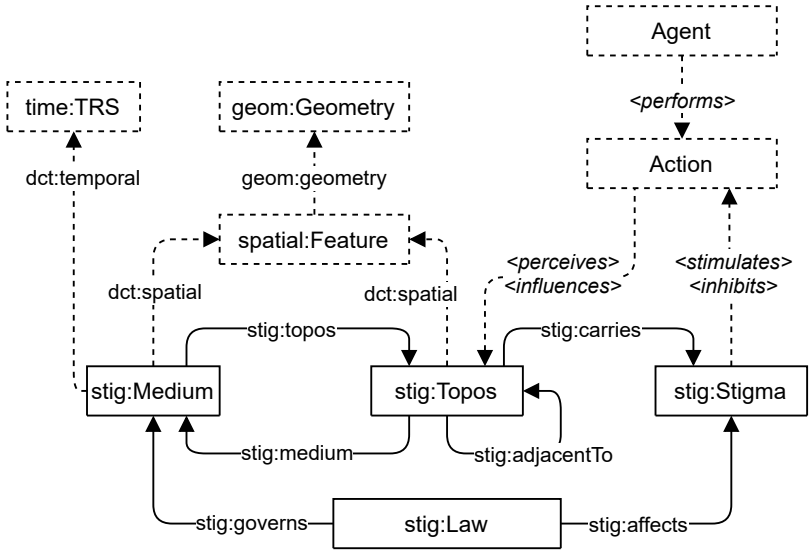

FIGURE 5.1: The StigLD domain model for stigmergic Linked Data Applications [220, 219]

5.1 `stigLD`: A dynamic stigmergic Linked Data Environment Server

In [220] and [219], we presented an implementation of a reactive Linked Data medium server to drive medium evolution according to the formal `MEDIUM` model as defined in Section 4.2.3.

The design and implementation of the resulting system, *`stigLD`*, consists of an RDF domain model to effectively model dynamic environments in terms of Linked Data Resources (Section 5.1.1), a set of SPARQL functions implemented as ARQ filter functions [36] to support evaluation of time-varying decay and diffusion functions (Section 5.1.2, and finally, a medium server framework that, based on these concepts, transparently drives medium-based evolution before handling client requests, following the concept of a reactive `MEDIUM` as defined in Section 4.2.3.

The results in this Chapter were originally published in the scope of [220] and [219].

5.1.1 Domain Model for Linked Systems

The domain model for stigmergic systems is designed around four core concepts (see also Fig. 5.1): `stig:Medium`, `stig:Topos`, `stig:Stigma` and `stig:Law`, with the following semantic meaning:

A resource that classisified as `stig:Medium` provides an interactive, explorable, and malleable representation of an environment, or a subset of this environment, to user-agents. Elements that are maintained by the respective `stig:Medium` (i.e., Topoi and Stigma, as will be detailed out in an instance) undergo changes either via direct input

[36] ARQ Filter Functions Documentation: `https://jena.apache.org/documentation/query/writing_functions.html`, accessed May 18th 2022

from agents, or by the medium's endogeneous dynamics, for instance, diffusion and evaporation effects.

```
<> a st:Law , a st:LinearDecay ;
   st:decay "0.5"^^xsd:double ;
   st:query </decay.sparql> .
```

LISTING 5.1: Simple example of a stig:Law that describes a linear decay

A medium may provide a client-readable and -interpretable description of its endogeneous dynamics by respective instances of stig:Law resources. Such a stig:Law description may, for example, include parameters for diffusion or evaporation equations as employed by the medium (see also example in Lst. 5.1). A stig:Law may link to an *evolutional query* which may be used to calculate the evolution of the medium's endogenous dynamics.

A medium may, but does not necessarily have to, reflect potential spatio-temporal characteristics of the represented environment, and declare those in the context of its self-description.

However, the medium must at least declare topology by interconnected stig:Topos instances. A stig:Topos is a resource to which a (situated) agent (see also Chapter 6) may navigate, perceive the state, and with which the agent finally interacts as result of its (situated) perception. Each stig:Topos instance provides a (potentially empty) set of outward edges that connect to other stig:Tops instances via the RDF predicate stig:adjacentTo. Agents may follow these edges to navigate to other Topoi in the current Topos' neighbourhood, as result of the agent's situated perception. Furthermore, a stig:Topos may be identified with any domain- or application-specifc resource using an owl:sameAs link and optionally detail on its spatial characteristics.

Each Topos maintains a (potentially empty) set of stig:Stigma instances. A stig:Stigma is a perceivable change that results from an agent's action in the stig:Medium. The perception of a stig:Stigma may stimulate (or inhibit) the performance of a subsequent action, i.e. the presence of a stig:Stigma makes the performance of this action more (or less) likely. Hence, actions stimulate (or inhibit) their own continued execution via the intermediary of stig:Stigma instances.

Thus, the presented domain model introduces all concepts required to implement a *feedback loop* of stigmergic systems (cf. Fig. 2.1).

5.1.2 stigFN Function Library

In order to facilitate the implementation of endogeneous dynamics based on physical diffusion or evaporation process to, for example, support transient marker-based

stigmergy in the dynamic environment, we supplement our domain model with the `stigFN` SPARQL function library. The `stigFN` SPARQL function library provides fundamental operations required to support the computation of spatio-temporal evolution effects within a SPARQL query, and by this finally enables to drive the medium state evolution in terms of *evolutional queries* as motivated in Section 4.2.3. The function library for this implements:

Handling spatial and temporal values:

Decay and diffusion functions require arithmetic operations on temporal data, e.g. `xsd:duration`, `xsd:dateTime` or `xsd:time`. Due to lack of built-in support in SPARQL and XPATH, we provide `stigFN:duration_secs` and `stigFN:duration_msecs` for conversions from a `xsd:duration` value to (milli)seconds. Additionally, `stigFN:dist_manhattan` is provided as a means to find the Manhattan distance between topoi when the medium is discretised into grids.

Decay functions:

Elements in the environment may be subject to dissipation processes. An example from stigmergic systems are transient markers, that, once left by an agent in the medium, reduce in concetration over time until they ultimately vanish. `stigFN` supports linear and exponential decay as two standard decay models, implemented as functions `stigFN:linear_decay` and `stigFN:exponential_decay`, respectively (see also Lst. 5.2) .

```
{SELECT (SUM(?c_i) as ?c) ?topos WHERE {
  ?topos st:carries [ a ex:NegFeedback;
        stig:level ?lvl;
        stig:created ?then;
        stig:decayRate ?d ].
        BIND(stigFN:linear_decay(?then, now(), ?d, ?lvl) as
   ?c_i)
} GROUP BY ?topos}

```

LISTING 5.2: Use of the `stigFN:linear_decay` SPARQL function to evolve and aggregate concentrations of linearly decaying markers on a topos.

Diffusion functions:

In diffusion processes, the intensity of elements in the medium, e.g. stigmata, does not decay over time but rather spreads over a spatial dimension from the point of its deposition. `stigFN` implements the 1D diffusion equation as SPARQL function `stigFN:diffuse_1D` (see also Lst. 5.3).

```
?source  a stig:Topos;
  pos:xPos ?source_x; pos:yPos ?source_y;
  stig:carries ?stigma .

?stigma a stig:Stigma;
  stig:created ?then;
  stig:decayRate ?decay ; stig:level ?srcLevel .

?aoe  a stig:Topos;
  pos:xPos ?effect_x; pos:yPos ?effect_y

BIND(NOW() AS ?now
BIND(stigFN:duration_secs(?then, ?now) AS ?duration
BIND(abs(?effect_x-?source_x) + abs(?effect_y-?source_y) AS
   ?dist)
BIND(stigFN:diffusion_1D(?stigma, ?dist, ?duration,
   ?srcLevel, ?decay) AS ?diffusion)

```

LISTING 5.3: Use of the `stigFN:diffuse_1D` SPARQL function to calculate the concentration of a diffusing stigma in an affected area..

`stigFN` was implemented using SPARQL user-defined functions in Apache Jena. Documentation and source code is publicly available[37]:

`https://github.com/dfki-asr/StigLD-Demo`.

5.1.3 Server Framework

The stigLD server framework consists of a `stigFN` server component as back end that provides an Apache Jena Fuseki triple store[38] with `stigFN` function support to support the RDF representation of the dynamic medium, and a *medium server* component that serves as communication endpoint to clients (see Fig. 5.2).

The medium server component populates the `stigFN` server with the initial state of the environment in terms of `stig:Topos` resources. Furthermore, the medium server defines evolution law parameters, and maintains respective SPARQL queries to drive medium evolution according to the defined parameters. The medium server populates information about these laws on in terms of `stig:Law` resources in the stigFN Server Fuseki triple store.

The medium server provides an HTTP endpoint via which clients may request to read the environment state via an HTTP `GET` request, or request the evaluation of a

[37] Author of stigFN source code: Melvin Chelli

[38] Apache Jena Fuseki home page: `https://jena.apache.org/documentation/fuseki2/` *(Visited May 2023)*

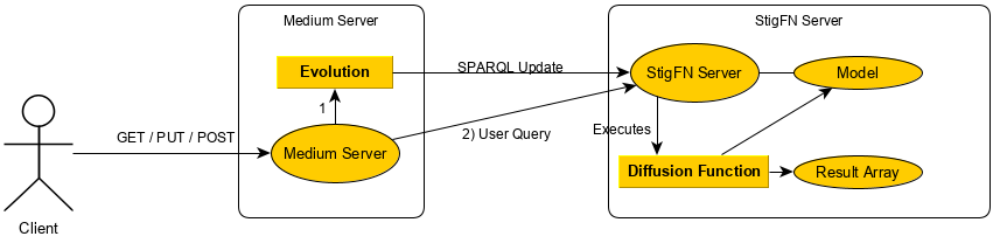

FIGURE 5.2: High-level architecture of the StigLD server framework

SPARQL query via HTTP `GET` or `POST`, following a subset the SPARQL 1.1 Protocol specification (see Section 2.1.3).

Upon receiving a client request, the Medium server first sends each of its managed evolutional queries to the stigFN server to drive the evolution of the environment model as maintained by the Fuseki triple store. These queries may employ decay and diffusion functions as defined in the previous section, which are implemented as SPARQL user-defined function in the stigFN server's Fuseki triple store. As soon as the stigFN server has finished evaluation of the evolution query, the medium server proceeds to finish the original client request by either returning the current medium state to the client (in case the client requested it via HTTP `GET`), or by evaluating the query as supplied by the client on the environment state after evolution.

This protocol implements directly the interaction model as defined for `MEDIUM` components, as defined in Section 4.2.3.

Application and evaluation of systems that employ the concepts and frameworks as presented above are presented in detail in Sections 7.4 and 7.6.

Documentation and source code of the server framework is publicly available[39]: `https://github.com/dfki-asr/StigLD-Demo`.

5.2 ECA2LD: Real-Time RDF lifting of large-scale simulation environments

The previous approach supports mainly the temporal evolution of elements in the medium according to specifically provided evolution rules. This approach, however, does not take into account dynamics within the environment that are not driven by environmental laws, but by underlying simulations of a complex environment.

While there are numerous works investigating (semi-) automated mappings from heterogeneous data structures and serializations to the RDF data model [16, 8, 171, 168, 174] for static data, little research has been conducted on the dynamic mapping of simulation runtime environments to RDF [189, 119]. This chapter presents, as solution, an algorithm that, based on an *Entity-Component-Attribute* data meta model,

[39] Authors of stigLD server framework: Melvin Chelli and Torsten Spieldenner

provides a real-time translation of simulation data to an interactive Linked Data layer.

The contents in this chapter were mainly published at the International Workshop on Semantic Web of Things for Industry 4.0., co-located with the Extended Semantic Web Conference 2018 [239], and presented at the Global IoT Summit [240], as well as the Joint International Semantic Technology Conference [238], in particular contents of Sections 5.2.2 and 5.2.4.

The event model is formally defined originally in the scope of this thesis. The formal definition is based on the event model as defined by the FiVES Virtual Environment Server [242, 241].

5.2.1 General Architecture: Model-View-Presenter-ViewModel Pattern

This section will motivate a design pattern to implement the publication of dynamic environment data via Linked Data media. The chosen *Model-View-Presenter-ViewModel* pattern achieves to decouple agent logic from the business logic of environment simulations by providing access to the simulation via a well defined *View* on the simulation data. The View is moreover compiled from a generic *ViewModel*, which, again, is defined independently of the data model of the underlying environment simulation. The connection between simulation business logic, ViewModel, and Model, is finally established using an independent *Presenter* component.

The chosen architecture achieves the following: Agent interaction is defined entirely against the *View*, making the View the medium via which agents interact with the environment simulation. With View and ViewModel being defined independently of a specific business logic data model, the so defined agent models remain operational even if the underlying data model changes, allowing for versatile, transferable agent programs.

This section will further detail on the chosen design pattern, and from it, develop a respective architecture to represent real-time environment simulation in (Linked Data) media.

Model-View*-Patterns

Model-View-* patterns, originally presented in the 1980's with the *Model-View-Controller* pattern ([141]), describe a class of software design patterns that originally aim at improving the development of User Interfaces (UI) on application data.

MV*-patterns decouple a suitable representation of application data (*Model*) from the respective User Interface (*View*), usually introducing another type of component, generally denoted as '*'. The exact nature of component '*' varies depending on the version of MV* pattern that is implemented. In any of these versions, '*' is ultimately

responsible for building the View, and keeping the View consistent with the Model if the Model changes, or transferring user interaction with the View back into the Model.

Since the introduction of the Model-View-Controller pattern, MV* patterns were extended continuously to achieve a more flexible decoupling of application modules, resulting in patterns of *Model-View-Presenter* ([21]), *Model-View-ViewModel* ([231]), and finally, the *Model-View-Presenter-ViewModel* pattern[8] (see also Figure 5.4).

While originally designed to facilitate the connection and communication between backend, business logic, and graphic user-interface (GUI) for human end users, the gained benefits also transfer to non-human user-agents interacting with an improved data representation of the original Model:

Instead of an interactive graphic user interface for human-application interaction, we consider the *Semantic Linked Data Integration* Layer as View representation of the environment data towards user-agents. The analogy of it being "like a GUI" towards user-agents is strikingly fitting: Like comprehensive icons, tooltips, labels help human users understand the modes of interaction as provided by the respective GUI, the Semantic Linked Data layer, provided in a machine-readable and -interpretable representation, helps machine user-agents identify relevant points of interaction, guides them through the process of using the API most effectively, and tells them what data the application that offers the API expects, both in terms of structure, and semantic meaning.

Model-View-*-ViewModel Patterns

Translating the original data model directly to the view is not always trivial, and moreover couples the view component tightly to the Model.

A couple of variations have therefore been proposed over the time. One of them, the *Model-View-ViewModel Pattern*[40] [95, 231], introduces as additional component a *ViewModel* (see Fig. 5.3).

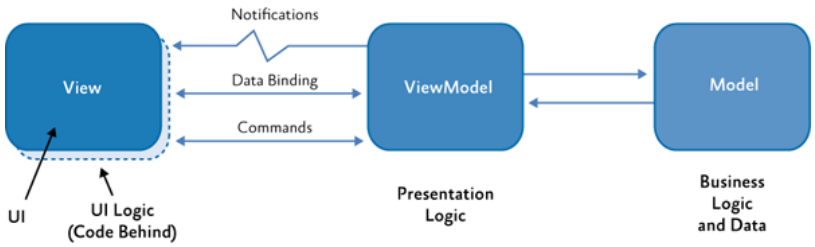

FIGURE 5.3: The Model-View-ViewModel pattern. (Source: [5])

[40]MVVM Pattern *(Visited May 2023)*: `https://docs.microsoft.com/en-us/previous-versions/msp-n-p/gg405484(v=pandp.40)`

The ViewModel is a specific, optimized representation of the original data Model. The ViewModel moreover is the logic component that builds and manages the View, allowing the View to define data and command bindings against the ViewModel, rather than directly to the Model. When users interact with the View, the View forwards the input to the ViewModel, which in turn calls respective methods in the Model. The indirect interaction between View and Model allows to change parts of the model without directly affecting Logic of the View.

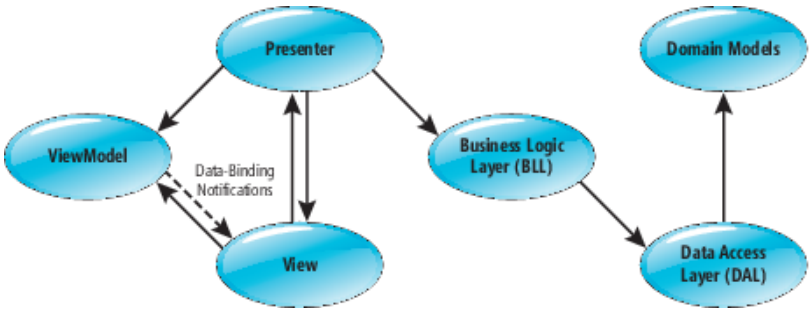

FIGURE 5.4: The Model-View-Presenter-ViewModel pattern as by Bill Kratochvil[6]; diagram source: [6].

As for the MVVM pattern, the *Model-View-Presenter-ViewModel* (MVPVM)[41] employs a *ViewModel* as optimized representation of the application's underlying domain objects for the presenter to render the *View*. However, compared to the MVVM pattern, the MVPVM pattern achieves an even higher level of independence between modules by moving the translation logic from the ViewModel into a separate *Presenter* component, and by this, decoupling the ViewModel from the Logic.

In this pattern, simulation *Business Logic* interacts with data within the *ViewModel* via the *Presenter*, using the *Data Access Layer* to transparently retrieve data from persistence spaces. *View* and *ViewModel* are kept consistent using suitable data bindings between both representations.

Digital Multi Agent Media as MV* Applications

In the context of publishing real-time environment data, the various components of the MVPVM pattern can be identified as follows:

Business Logic: The *Business Logic* implements all logic specific to the simulation domain and services as offered by the simulation. The business logic moreover drives the state of the simulation, and updates the state of entities maintained by the simulation accordingly. As a result of its routine, the Business Logic regularly generates new data during its lifetime.

Model: The *Model* is the internal data representation of the environment simulation on which the Business Logic operates directly. As common for MV* Patterns, the

[41] MVPVM Pattern *(Visited May 2023)*: `https://msdn.microsoft.com/en-us/magazine/hh580734.aspx`

Model is to be considered different from View or View Model. The choice of the Model is rather influenced by constraints set by the Business Logic, starting with the choice of Programming Language, over in-memory backends, employed third party libraries and frameworks, and many more.

In the context of environments in Linked Data media, we make no assumptions towards what Model a respective application uses. The goal is rather to abstract from the Model used in the Business Logic, and provide a shared Linked Data-based abstraction Layer for read-write interaction that is decoupled from Business Logic and its Model. We do, in particular, not require the Model to implement concepts of the RDF graph model of any kind.

View Model: Rendering the View directly from the Model is not necessarily an easy task, as both are heavily constrained by their individual purpose. And as established before, this task of translation is instead to be carried out by a *ViewModel*, which can be considered an Interlingua used by both Model and View, to easier translate between their individual translations.

The ViewModel thus is to be chosen that on one hand, it is easy to transfer data from the *Model*, i.e., modify the View Model from the Business Logic, and on the other, transparently render the interactive View from the ViewModel.

In the context of the proposed Linked Data medium architecture, that means that the ViewModel needs to be based on an underlying (meta-)datamodel that allows for a transparent lifting to a Linked Data representation, i.e., to a suitable representation as resource graph in RDF. Moreover, the model should either support to be easily lowered to from the chosen RDF representation, or be chosen in such a way that both data structure and contents can be directly derived and indexed from the equivalent RDF graph.

This thesis will define, establish, and implement a respective ViewModel component based on an Entity-Component-Attribute (ECA) data meta model that specifically allows to publish the application Model of the simulation as Linked Data.

View: As outlined before, traditionally, the *View* component in a MV* Pattern refers to a human-readable, interactive layer that is rendered transparently from the application *Model*. It serves as interaction layer between user and application, also guiding the user through possible interactions by means like visual elements, tooltips, and more, to help the user understand the possibility of the actions they can take, and the expected result which a specific interaction should cause. Similarly, a digital agent medium does not provide an interaction layer to human users, but to software user-agents.

This thesis will define as View for interactive Linked Data media a Linked Data Platform representation (cf. Section 3.1.2) that is Thus, the interaction layer needs to

be enriched by an equivalent to all the visual aids and tooltips that are provided to human users in traditional MV* Patterns.

Presenter: The *Presenter* component is finally the component that connects View and ViewModel to the Business logic, and takes care of keeping both consistent in state with the business logic, and with each other.

5.2.2 ViewModel: Entity-Component-Attribute Runtime Data Model

The endeavored decoupled nature of components in the MVPVM pattern reflects requirements as seen for changeable software architectures [134].

We further derive requirements towards changeable software from the notion of *aspect-oriented software design* as presented by Kiczales et. al [137]. According to this, it is necessary to avoid *cross-cutting concerns* in the code. As main pitfalls that break changeability of software, aspect-oriented design distinguishes:

Code scattering: Code that implements a concept or logic is distributed over several modules or classes. As a result, adding, changing, or removing logic from an application requires to change several modules at once. Code scattering is avoided by modeling software and its data in distinct modules for every task.

Code tangling: While code that implements a certain feature may be entirely contained in its own module, dependencies between modules can still break changeability of software. This happens for example if code of one module refers to, or makes calls to, code in another module. If one module changes its interface to which other modules make calls, all other modules need to be changed as well. A way to avoid code tangling is a data-centralistic approach by which separate software modules operate on a shared data layer, without direct calls between the modules.

Code tangling is then achieved by modules not being required to include code and dependencies of other modules directly, as it would be necessary if inter-module communication is, for example, directly implemented in Java Code. Changes in code and APIs of one module would then require changes of calling code in any other module, whereas, when communicating via a shared data layer, modules only depend on the definition of that layer's API and data objects. Specific code of other modules is no longer directly referenced, or included.

Entity-Attribute Models

Above requirements are met by Entity-Attribute based software design. The understanding of Entity-Attribute models varies in literature. In the following, we summarise available variants and formalize the notation of Entity-Attribute models.

Common for all variations is the notion of an *entity* as an empty data container which is closer specified by a set of typed *attributes* that carry the actual values.

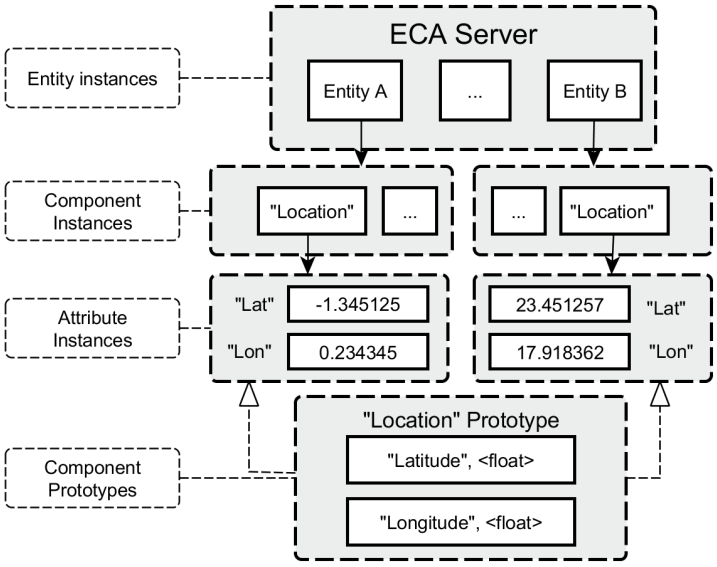

FIGURE 5.5: Entity-Component-Attribute model based on the example of a component that defines a "Location".

The IoT Context Broker by Moltchanov et. al [178] keeps to the levels of entities and attributes.

The systems RealXtend [3, 57] and FiVES [242, 241], as well as popular game engines, such as Unreal Engine[42] and Unity3D[43], include the notion of *components* (*Entity-Component-Attribute pattern, ECA*; see also Fig. 5.5). Components can be considered as prototypes of attribute sets that belong to the same concept. When a component is attached to an entity instance, a new instance of the component and the respective attribute set is created from this prototype.

Entity-Component-Attribute Model: Definition

In the following, we will give an outline of requirements we see for the design of changeable large-scale software projects. We give a formal definition of the established Entity-Component-Attribute model and discuss how it is suitable to fulfill the requirements towards changeable software.

RealXtend equips components with application logic that is directly contained as code in the component implementation. Similarly, components in game engines like Unreal Engine and Unity3D allow to attach executable scripts as part of the component[44,45]. This allows to equip game objects with scriptable behavior that directly depends on the attributes of a component. Examples for such behavior may be collider scripts that define a collider volume in the components, a velocity component

[42] `http://www.unrealengine.com` *(Visited May 2023)*
[43] `http://www.unity3d.com` *(Visited May 2023)*
[44] `https://docs.unity3d.com/ScriptReference/Component.html` *(Visited May 2023)*
[45] `https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Components/` *(Visited May 2023)*

that, once attached, automatically moves an object based on the attached attribute values, or a camera component that defines rendering parameters via the given attributes.

The work by Wiebusch et al. [268] as well as the FiVES server system consider the Entity-Component-Attribute model as data model only. Logic is implemented in independent *systems* (referred to as *plugins* in FiVES), with a careful design of how external logic accesses the data.

We adapt the understanding of components as by Wiebusch, and FiVES, with logic implemented separately to avoid the tight coupling between components and their specific implementation as in RealXtend or game engines. We derive from this the following formal definition of the ECA architectural pattern.

Let $\mathbf{e}$ denote an entity instance, and $\mathcal{P}_\mathbf{C}$ denote the set of all component prototypes. Then we define the following sets:

- $\mathbf{E}$ is the set of all entity instances. An entity instance is defined as $\mathbf{e} = (n_\mathbf{e}, \mathbf{C_e})$, with $n_\mathbf{e} \in \Sigma^+$ being the unique identifier for $\mathbf{e}$ over alphabet Σ, and $\mathbf{C_e}$ being the set of component instances attached to $\mathbf{e}$.
- $\mathbf{C_e}$ is the set of all component instances attached to an entity instance $\mathbf{e}$. A component instance is defined as $\mathbf{c} = (n_\mathbf{c}, p_\mathbf{c}, \mathbf{A_{c,e}})$, with $n_\mathbf{c} \in \Sigma^+$ being the unique identifier for $\mathbf{c}$, $p_\mathbf{c} \in \mathcal{P}_\mathbf{C}$ being the prototype that $\mathbf{c}$ is an instance of, and $\mathbf{A_{c,e}}$ being the set of all attribute instances attached to $\mathbf{c}$.
- $\mathbf{A_{c,e}}$ the set of all attribute instances attached to a component instance $\mathbf{c}$. An attribute instance is defined as $\mathbf{a} = (n_\mathbf{a}, v, t)$, with $n_\mathbf{a} \in \Sigma^+$ being the unique identifier for $\mathbf{a}$, v denoting the attribute instance's current *value*, and t denoting the ECA runtime type of $\mathbf{a}$.

The role of an entity within the application is by this entirely determined by the set of attached components. Components are implemented and operate independent of each other. This avoids the issue of code scattering. The *composition-over-inheritance* principle of the model also avoids code-tangling, as it eliminates class inheritance and attribution hierarchies.

5.2.3 ECA-Model: Implementation

The fist version of the implementation of the ECA model was originally published in the scope of the Virtual Environment server *FiVES* (***F**lexibel **Vi**rtual **E**nvironment **S**erver*) ([242, 241]). However, both the concepts of the model as employed in that implementation, and the way they were finally implmeneted, had over time been deemed so useful that they were extracted from the FiVES server framework as

standalone solution in the form we see it here today [46]. The ECA model library is implemented in C# as programming language due to .Net's powerful concepts of Reflection and Delegates, which are very suitable for the dynamic typing of component blueprints.

The following section has originally appeared in the publications stated above. The terminology has slightly changed over time. Respective corrections have been applied to the following contents, as for general adaptions to the text to fit it into the form and style of this Thesis.

Data Model

Components are registered as *Prototypes* in a *PrototypeRegistry* data structure. A Component Prototype consists of its name and a list of Prototypes of its Attributes, again identified by name. These Attribute Prototypes describe the type of the Attribute, currently in terms of a C# interface that is passed when defining an Attribute, and a default value that is returned when the Attribute is read before any value was written to it. The library tries to map to an Attribute type when trying to assign an attribute value from or to another type implementation. At the current state, this mapping is done by naively mapping field names between types, which allows for example to use different implementations of a Vector class in two different plugins, and still operate on the same Attribute. More sophisticated mapping strategies, based for instance on semantic descriptions of Attribute types, are under investigation.

Applications may register new Component Prototypes during run-time. As soon as a Prototype is registered, its Attributes can directly be accessed on an Entity using the names of Component and Attribute. The process of evaluating a component access, and type check of the attribute, is shown in Figure 5.6.

A Component is not instantiated on an Entity until it is requested for the first time to keep memory foot print in server run-time and during communication low.

```
1 ComponentPrototype spatial = new ComponentPrototype("spatial");
2 spatial.AddAttribute<Vector>("position");
3 spatial.AddAttribute<Quat>("orientation");
4 ComponentRegistry.Instance.Register(spatial);
```

LISTING 5.4: Example of registering a Component Prototype for spatial data.

[46] The standalone library has been published to GitHub: https://github.com/tospie/eca-base-model

Component and Attribute Access

Access to components and attributes is provided via their registered names on instatiated entities and components respectively (see also Lst. 5.2.3). Attempted access to attributes and components is then evaluated in a multi-staged process (cf.Fig 5.6):

First, the `PrototypeRegistry` data structure is checked for any registered component blueprint under the requested name. If no blueprint with the requested name is present, an exception is thrown. Otherwise, the registered Component Prototype is checked for an attribute of the requested name, throwing an exception if no according attribute was registered.

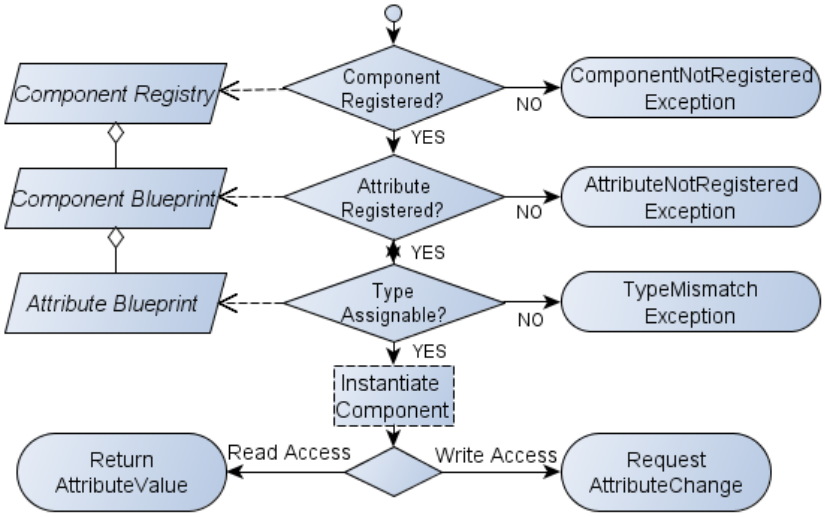

FIGURE 5.6: Multi-staged access to Attributes via Components on Entities.

It is then checked if, for read-access, if the type of the attribute value is assignable to the variable to which it is supposed to be assigned. For write-access, it is checked if the supplied value is assignable to the type of the attribute. The implementation attempts to cast values from variable to attribute type, and vice versa, where possible. If the value is un-assignable, a respective exception is thrown. If the component has to this point not yet been instantiated on the entity, it will be instantiated now. In case an attribute is requested for the time during read-access, and thus, freshly instatiated upon the first read operation, it is initialised with a *default value* that may be declared when defining the Attribute.

Read-Write-Access to attributes is provided by properties `Value` for read-access, and `Set` for write access [47] (see also Lst. 5.2.3). While `Value` returns the current attribute value directly, `Set` will invoke an indirect, event-based writing process, as further detailed in the next section.

[47] It is apparent that choosing `Value` for read-access instead of `Get` is somewhat inconsistent. However, the decision for this design choice was made a long time ago, and for reasons which are, by today, unfortunately forgotten.

```
var e = new Entity();
e["spatial"]["position"].Set(new Vector(1.0, 2.0, 3.0));
e["spatial"]["orientation"].Set(new Quat(0.0, 0.0, 1.0, 0.0));

Vector currentPosition = e["spatial"]["position"].Value;
```

LISTING 5.5: Example of accessing Spatial Components as defined in Lst. 5.2.3 to write position and orientation Attributes.

Event System

Application logic and data managed by the ECA meta data model are loosely coupled by *a)* an indirect write access to attributes, and *b)* and event model that informs application logic over changes in the attributes.

Let $a = (n_a, v, t)$ be an attribute with name n_a and value v as defined above, c a component with name n_c that contains attribute a, and e an entity on which component c and attribute a are instantiated respectively.

Then operations on attribute a can be formally defined as a CCS process `ATTRIBUTE` (cf. Sec. 2.2) as follows:

$$\texttt{ATTRIBUTE}_a = \texttt{READ}_a + \texttt{WRITE}_a(in) \tag{5.1}$$

$$\texttt{READ}_a = get_a.\overline{ret}_a(v).\texttt{ATTRIBUTE}_a \tag{5.2}$$

$$\begin{aligned}\texttt{WRITE}_a(in) = {} & set_a(in).\overline{before}_a(in) \\ & .(\texttt{HANDLE}_a(in) + mod(in_m).\texttt{HANDLE}_a(in_m))\end{aligned} \tag{5.3}$$

$$\texttt{HANDLE}_a(in_h) = [v = in_h].\overline{changed}_e(n_c, n_a, v) \| \overline{changed}_a(v).\texttt{ATTRIBUTE}_a \tag{5.4}$$

Let subscript a denote a channel via which `READ` and `WRITE` access to attribute a are provided. When application logic requests to *get* the current attribute value via `READ`, the process simple returns the current value v of a.

Requests to *set* the attribute's value to a provided input *in* via `WRITE` is handled in a two-stage process: As soon as an attribute receives a *request to set* the attribute value, it emits an event $\overline{before}$ via its channel to inform application logic about the requested attribute change, with the requested *in*put as parameter. Application logic may inflict modifications on the supplied input, and return a modified input value in_m[48]. `ATTRIBUTE` then proceeds to `HANDLE` the modified input in_m, or, in case no

[48]The design may lead to multiple modules of the application logic trying to modify a requested attribute change at the same time, by reacting to the same $\overline{before}$ event. This conflict was resolved in the implementation FiVES in a module that allowed to define the order in which application logic modules apply changes, with subsequent changes being cumulative. This approach, however, exceeds the scope of this Thesis and will therefore not be handled in detail. Interested readers are invited to read the documentation of the respective implementation in Google Docs: `https://rb.gy/3pwif4` *(Visited May 2023)*

application logic reacted to the $\overline{before}$ event, the original input in. `HANDLE` finally assigns the supplied value to handle in_h (in, or in_m, depending on whether application logic reacted to the $\overline{before}$ event) to the attribute value v of a, and subsequently emits two events $\overline{changed}$, with names n_c and n_a of component and attribute respectively, as well as the updated attribute value v via a channel provided by the entity managing the attribute, and with the updated value v only on the attribute channel itself.

This two-staged interaction process models the process of *indirect environment interaction* as described in Section 4.1.1, and by this implements the model of environment evolution based on agent influences as defined in Section 4.3.3: Application logic in general, and specifically, user-agents, request updates of an attribute as *influence* to the environment. This influence, however, does not change the environment directly, but the actual effect is decided by the application itself. $\overline{changed}$ events provide observability of the environment, and achieve a loose coupling between application logic and data model. For this, the automatic type mapping simplifies the use of the attribute value in native code of application logic.

5.2.4 Presenter: Linked Data Lifting Algorithm

In the following, we detail on the automated structural mapping between ECA runtime environments and W3C LDP compliant Linked Data servers. By repeated application of a set of mapping rules (cf. ① to ④), structural interoperability [226] between ECA runtimes and LDP servers is established.

We assume the existence of functions $\nu : \Sigma^+ \to \texttt{IRI}$ and $\rho : \mathcal{P}_\mathbf{C} \to \texttt{IRI}$ for minting fresh `IRI`s from identifiers and component prototypes. Although several guidelines exist for minting `IRI`s[49], we do not make assumptions on ν or ρ.

$$\text{①}\quad \frac{(n_\mathbf{e}, \mathbf{C_e}) \in \mathbf{E} \quad \forall (n_\mathbf{c}, p_\mathbf{c}, \mathbf{A_{c,e}}) \in \mathbf{C_e}}{\begin{array}{l}\nu(n_\mathbf{e})\ \texttt{rdf:type}\ \texttt{ldp:BasicContainer}\ .\\ \nu(n_\mathbf{e})\ \texttt{dct:identifier}\ \text{“}n_\mathbf{e}\text{”}\texttt{\^\^xsd:String}\ .\\ \nu(n_\mathbf{e})\ \texttt{ldp:hasMemberRelation}\ \texttt{dct:hasPart}\ .\\ \nu(n_\mathbf{e})\ \texttt{dct:hasPart}\ \nu(n_\mathbf{c})\ .\end{array}}$$

① Each entity instance $\mathbf{e} = (n_\mathbf{e}, \mathbf{C_e})$ is mapped to a `ldp:BasicContainer` with `IRI` $\nu(n_\mathbf{e})$. This entity container maintains a membership triple $(\nu(n_\mathbf{e}), \texttt{dct:hasPart}, \nu(n_\mathbf{c}))$

[49] `https://www.w3.org/TR/cooluris/` *(Visited May 2023)*

for each component instance $(n_{\mathbf{c}}, p_{\mathbf{c}}, \mathbf{A}_{\mathbf{c},\mathbf{e}})$ attached to $\mathbf{e}$.

② $$\frac{(n_{\mathbf{c}}, p_{\mathbf{c}}, \mathbf{A}_{\mathbf{c},\mathbf{e}}) \in \mathbf{C}_{\mathbf{e}} \quad \forall (n_{\mathbf{a}}, v, t) \in \mathbf{A}_{\mathbf{c},\mathbf{e}}}{\begin{array}{l} \nu(n_{\mathbf{c}})\ \texttt{rdf:type}\ \texttt{ldp:BasicContainer}\ . \\ \nu(n_{\mathbf{c}})\ \texttt{dct:identifier}\ \text{“}n_{\mathbf{c}}\text{”}\texttt{\^\^xsd:String}\ . \\ \nu(n_{\mathbf{c}})\ \texttt{dct:isPartOf}\ \nu(n_{\mathbf{e}})\ . \\ \nu(n_{\mathbf{c}})\ \texttt{ldp:hasMemberRelation}\ \texttt{dct:hasPart}\ . \\ \nu(n_{\mathbf{c}})\ \texttt{dct:hasPart}\ \nu(n_{\mathbf{a}})\ . \\ \nu(n_{\mathbf{c}})\ \texttt{rdfs:isDefinedBy}\ \rho(p_{\mathbf{c}})\ . \end{array}}$$

② Each component instance $\mathbf{c} = (n_{\mathbf{c}}, p_{\mathbf{c}}, \mathbf{A}_{\mathbf{c},\mathbf{e}})$ is mapped to a `ldp:BasicContainer` with `IRI` $\nu(n_{\mathbf{c}})$. This component container uses `dct:isPartOf` to indicate its containing entity container $\nu(n_{\mathbf{e}})$ and maintains a membership triple $(\nu(n_{\mathbf{c}}), \texttt{dct:hasPart}, \nu(n_{\mathbf{a}}))$ for each attribute instance $(n_{\mathbf{a}}, v, t)$ attached to $\mathbf{c}$. In addition, we use `rdfs:isDefinedBy` to indicate an authoritative resource $\rho(p_{\mathbf{c}})$ semantically defining the component container $\nu(n_{\mathbf{c}})$. We detail on $\rho(p_{\mathbf{c}})$ in the next section.

③ $$\frac{(n_{\mathbf{a}}, v, t) \in \mathbf{A}_{\mathbf{c},\mathbf{e}}}{\begin{array}{l} \nu(n_{\mathbf{a}})\ \texttt{rdf:type}\ \texttt{ldp:RDFResource}\ . \\ \nu(n_{\mathbf{a}})\ \texttt{dct:identifier}\ \text{“}n_{\mathbf{a}}\text{”}\texttt{\^\^xsd:String}\ . \\ \nu(n_{\mathbf{a}})\ \texttt{dct:isPartOf}\ \nu(n_{\mathbf{c}})\ . \\ \nu(n_{\mathbf{a}})\ \texttt{rdf:value}\ \nu(n_a^v)\ . \end{array}}$$

③ Each attribute instance $(n_{\mathbf{a}}, v, t) \in \mathbf{A}_{\mathbf{c},\mathbf{e}}$ is represented by a `ldp:RDFResource` with `IRI` $\nu(\mathbf{n}_{\mathbf{a}})$. This attribute resource uses `dct:isPartOf` to indicate its containing component container $\nu(n_{\mathbf{c}})$. The triple $(\nu(n_{\mathbf{a}}), \texttt{rdf:value}, \nu(n_a^v)$ encodes the attribute's publishes the attribute's value in an additional resource with URI $\nu(n_a^v)$. This allows the server to specify further interaction methods on Attribute values, as described in Sections 5.2.5 and 5.2.5.

④ Since the RDF datatype abstraction is compatible with XML Schema, we rely on the data type support between an ECA runtime enviroment and XML Schema Types for datatype conversion. Given an attribute $(n_{\mathbf{a}}, v, t) \in \mathbf{A}_{\mathbf{c},\mathbf{e}}$, we denote by $\nu(t)$ the datatype `IRI` of the RDF-compatible XSD type corresponding to t. The lexical form $\mu(v)$ may be any lexical form, ie. a Unicode string in Normal Form C, from $\nu(t)$'s lexical space that represents the same value as v. Extensions that handle domain-specific or user-defined datatypes beyond the RDF-compatible XSD types are expected to behave as outlined here.

Finally, a *collection of entities* is published in terms of a resource with URI $\nu(n_{\mathbf{n}_{\mathbf{e}}})$. We for this assume an entity collection $\mathbf{E}$ to be assigned a unique name $n_{\mathbf{E}}$.

$$\text{⑤}\quad \frac{\forall(n_{\mathbf{e}}, \mathbf{C_e}) \in \mathbf{E}}{\begin{array}{l} \nu(n_{\mathbf{E}})\ \texttt{rdf:type}\ \texttt{ldp:BasicContainer}\ . \\ \nu(n_{\mathbf{E}})\ \texttt{ldp:contains}\ \nu(n_{\mathbf{n_e}})\ . \end{array}}$$

The respective Entity collection resource then serves as entry point for client applications to explore the server data.

The resulting RDF modeled Linked Data Platform representation allows to further augment the resources with domain semantic information. For this, the RDF description of the data structure serves as input for RDF mapping vocabularies like SPIN SPARQL[50], RIF in RDF[51], the LDIF framework[52] or the R2R framework[53]. For a detailed explanation of domain semantic augmentation, we refer to the original paper [239]. In the context of our approach, additional domain semantic information on top of the structural description allows clients to identify relevant resources based on domain semantic information.

Figure 5.7 shows an example of how additional domain semantic information can be added to the automatic structural mapping using an R2R mapping file:

Whereas the original structural mapping identifies created resources only in terms of their respective LDP classes. The domain semantic information that is implicitly included in terms of component and attribute names is at this stage not accessible to machine clients in an interpretable way, but subject to proper explicit interpretation by domain experts.

This explicit domain knowledge is in a second step supplied in terms of an R2R mapping. In the given example, the mapping file describes that any resource that is created for a component the prototype of which defines "latitude" and "longitude" values is to be interpreted as a `geo:SpatialThing` RDF class, with the attributes being correctly classified as latitude and longitude as defined in the Basic Geo vocabulary[54].

By running the mapping file by a R2R compiler engine, the initially pure structural representation of ECA data in terms of LDP resources can be leveraged to an expressive semantic interpretation that is fully interpretable by user agents.

[50] `https://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/` *(Visited May 2023)*
[51] `https://www.w3.org/TR/rif-in-rdf/` *(Visited May 2023)*
[52] `http://ldif.wbsg.de/` *(Visited May 2023) (Visited May 2023)*
[53] `http://wbsg.informatik.uni-mannheim.de/bizer/r2r/spec/` *(Visited May 2023)*
[54] `https://www.w3.org/2003/01/geo/` *(Visited May 2023)*

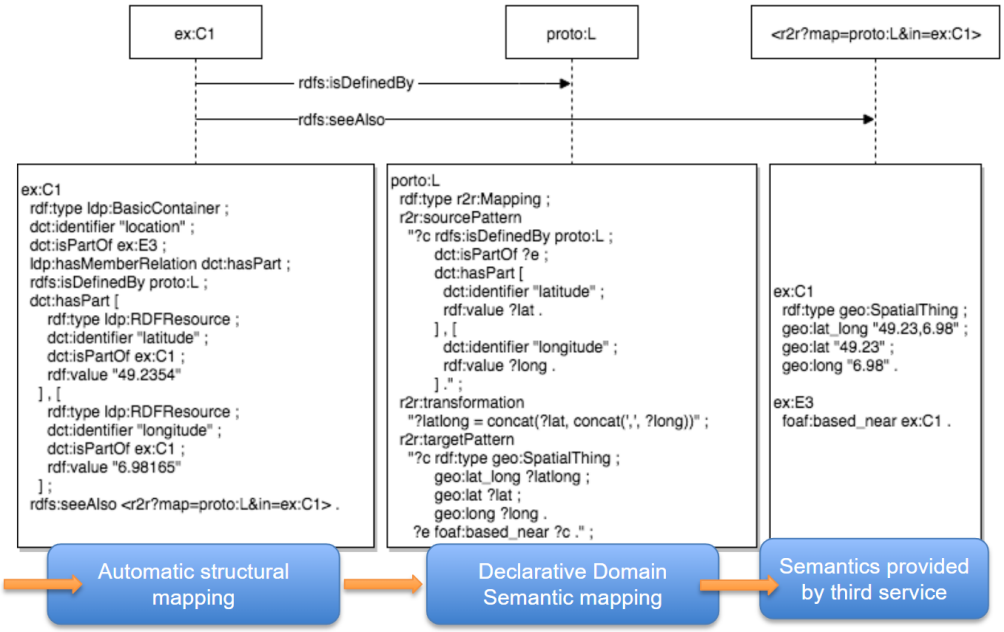

FIGURE 5.7: Example of an augmented domain semantic mapping on top of the automatic structural mapping from ECA to RDF LDP data.

5.2.5 Network API

The Linked Data View that is generated by the Presenter from the ECA ViewModel provides mainly two means of interactions: First, it supports HTTP CRUD operations in a request-response manner. Second, each generated resources provides subscription endpoints via which connected clients are informed about changes in the underlying simulation data in real-time.

Interactive View: RESTful operations on runtime data:

Each of the resources generated according to the rules in the previous section provide an HTTP REST compliant endpoint that is maintained within the ECA2LD library. The route to the resource is determined by the minting functions $\nu : \Sigma^+ \rightarrow \texttt{IRI}$ and $\rho : \mathcal{P}_\mathbf{C} \rightarrow \texttt{IRI}$.

HTTP operations `GET`, `POST`, `PUT`, `DELETE` allow to retrieve, (re)place, amend, or delete the triple sets as produced by ① – ④ respectively, and also to read and update Attribute values on the respective Attribute value endpoints. Information about further interaction possibilities with the resource can be obtained by using the `HTTP OPTIONS`[55] operation.

Respective functionality is already implemented in the ECA2LD library on Entity, Component, and Attribute level. We extended the implementation to support respective operations on Entity Collection and Attribute value level.

[55] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS` *(Visited May 2023)*

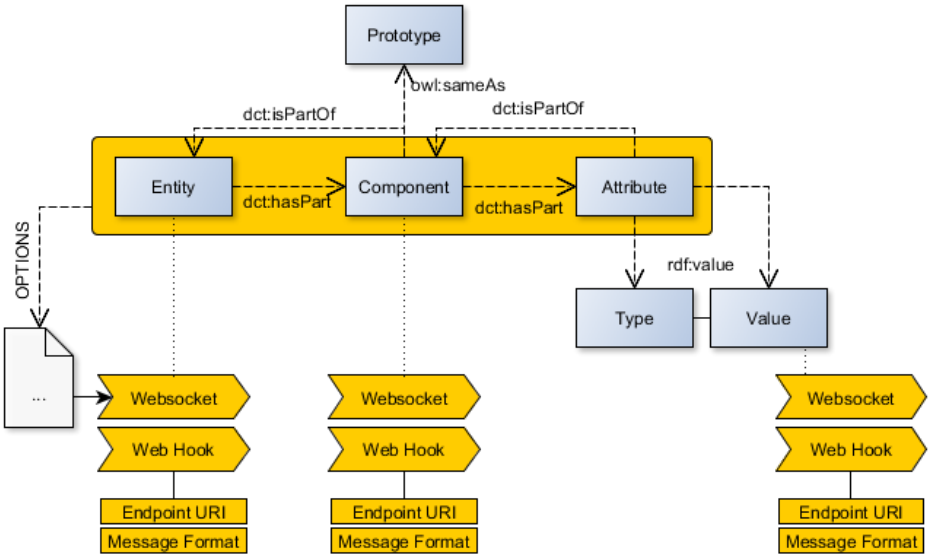

FIGURE 5.8: Web resources and respective endpoints that are generated for each of the elements of the runtime application.

Realtime subscription Pub/Sub:

The mapped structure from steps ① – ④ remains mostly static during execution of an application. Realtime run-time data is aggregated in `Attribute` resources as provided by ③, and can be retrieved using the respective resource indicated by the Attribute's `rdf:value`:

$$\text{⑥}\quad \frac{\forall \texttt{Attribute}\ n_a}{\nu(n_a)\ \texttt{rdf:value}\ \nu(v_a)}$$

We extend the modes of interaction beyond the HTTP operations as described in Section 5.2.5 by providing as part of the returned RDF datagram a description of provided real-time subscription endpoints according to the following rule:

$$\text{⑦}\quad \frac{\forall \text{Attribute Values } v_a, \texttt{Protocol}\ \mathbf{p}}{\begin{array}{l}\nu(n_v^a)\ \texttt{sub:endpoint}\ \sigma_s(v_a)\ .\\ \sigma_s(v_a)\ \texttt{sub:protocol}\ p.\\ \sigma_s(v_a)\ \texttt{rdf:format}\ f.\end{array}}$$

⑦: For each `AttributeValue` v_a of `Attribute` a, a resource with resolvable URI $\nu(n_v^a)$ is created that contains a forward link to a resolvable resource $\sigma_s(v_a)$. The predicate `sub:endpoint` describes the resource with URI $\sigma_s(v_a)$ as endpoint for clients to open a subscription channel with protocol $\mathbf{p}$, p a suitable RDF predicate to refer to $\mathbf{p}$, for

realtime updates of attribute value v_a of attribute a. `rdf:format` points to a description f of the message serialization format emitted by subscription resource $\sigma_s(v_a)$. Defining suitable representations f of the message format beyond xsd compatible atomic types is subject to future work.

We extended the ECA2LD library such that the respective subscription endpoint is wired to the respective Attribute value in the ECA model by an observer pattern. In our specific implementation, we use the `Event` mechanisms in C#[56]. By this, subscribing to a subscription resource $\sigma_s(v_a)$ by its URI as offered by the RDF description in ⑤, clients can keep their local model consistent with the server data.

Figure 5.8 shows the set of created resources, their relation to each other, and examples of further interaction methods that can be retrieved by clients after performing the steps outlined in Section 5.2.2.

5.2.6 The ECA2LD Framework: Implementation

The ECA2LD framework is finally implemented in terms of three modules: The ECA base model implementation that implements Entity Collections, Entities, Components and their Prototypes, as well as Attributes, and the Event model, as detailed in Section 5.2.3 ; *Linked Data Points*, a library that transparently creates the Network API with HTTP and streaming capabilities as defined Section 5.2.5. *Linked Data Points* is extensible in the sense that developers may easily define new request-response or streaming protocols to be supported on the endpoints of Entities, Components, Attributes, or Collections respectively. And finally, the *ECA2LD* lifting library publishes an RDF description of Entitites, Components, and Attributes according to the production rules defined in Section 5.2.4, and publishes the resulting RDF graphs via the endpoints generated by the *Linked Data Points* library.

The different modules are coupled to the underlying ECA model via its event system as presented in Section 5.2.3. Developers who wish to use the ECA2LD lifting library only need to instantiate a Linked Data Point instance on an entity they wish to publish via ECA2LD, or, alternatively, instantiate a Linked Data Point on an Entity Collection. Listing 5.6 shows an example of how to instantiate a Linked Data Point on an Entity, or an Entity Collection, respectively.

The ECA2LD framework is fully published on Github: `https://github.com/dfki-asr/eca2ld` (accessed May 19th, 2022).

[56] `https://docs.microsoft.com/en-us/dotnet/standard/events/` *(Visited May 2023)*

```
// Instantiating a new Linked Data Point on an Entity during runtime
var bp = new Entity();
bp["blueprint"]["about"].Set("cpu:product");
bp["blueprint"]["step"].Set("steps:dispense");
new EntityDatapoint(bp, basePath + "blueprint");

// Instantiating a new Linked Data Point on an Entity Collection during runtime
var worldDatapoint = new EntityCollectionDatapoint(CEC.Instance, basePath);
```

LISTING 5.6: Example of instantiating a Linked Data Point on an Entity (top), or Entity Collection (bottom).

Unity 3D Extension

ECA2LD was moreover implemented as extension to the Unity3D game engine[57]. Unity3D manages contents in a ECA-like fashion, using the term *"Game Object"* to refer to the equivalent of entities. ECA2LD is implemented in a set of C# Unity *Scripts* that reflect the feature set of the Linked Data Points and ECA2LD standalone implementations.

However, as Unity3D couples application logic with Component definitions, a common approach to implement the ECA model, and moreover, Unity3D does not implement attributes in a distinct class, but defines attributes as C# properties of the respective component implementation, publishing the Unity3D ECA model to Linked Data requires a couple of additional steps.

It defines a script class `LDComponent`, which extends `MonoBehavior`, the base class for Unit3D component scripts. It further defines a property annotation `IsLD` to annotate C# properties of `LDComponent` scripts which should be published as Linked Data Platform Resources.

Finally, attaching an `LDEntity` script to a Unity game object publishes the respective game object, its attached `LDComponent` scripts, and the `IsLD` annotated attributes of the component, to a Linked Data Platform representation according to the production rules in Section 5.2.4. Listing 5.7 displays the definition of a `LDComponent` that publishes the game object's pose to the Linked Data layer, further specified by two attributes, *Position* and *Orientation*, which are annotated to be published to the Linked Data representaion with the `IsLD` annotation. The resulting LDP resource RDF graph is shown in Listing 5.8.

[57] Official Unity3D website: `https://unity.com/` *(Visited May 2023)*

```
namespace Assets.Scripts.eca2ld_unity.ld_components
{
    public class LDPose : LDComponent
    {
        [IsLD]
        public LDVector Position = new LDVector();

        [IsLD]
        public LDQuat Orientation = new LDQuat();

        public void Update()
        {
            Position.Set(gameObject.transform.position);
            Orientation.Set(gameObject.transform.rotation);
        }
    }
}
```

LISTING 5.7: Implementation of a ECA2LD Unity3D Component. LD Compontents, along with their annotated attributes, will transparently be published in the ECA2LD Linked Data Platform format.

```
<> dct:hasPart </Orientation/>, </Position/>;
    dct:identifier "LDPose"^^<xsd:string>;
    dct:isPartOf <http://localhost:8080/example/>;
    a ldp:DirectContainer;
    rdfs:isDefinedBy <http://localhost:8080/prototypes/LDPose/>;
    ldp:hasMemberRelation dct:hasPart.
```

LISTING 5.8: LDP representation of an instatiated LDPose component with two Attributes, Orientation and Position, as defined via Lst. 5.7

Figure 5.9 displays ECA2LD scripts attached to a game object in the Unity3D editor: It is sufficient to supply the game object with the `LDEntity` script, and specify endpoint and entity name under which the entity is to be published. Any attached `LDComponent` script, in the example, `LDPose`, publishes its annotated attributes automatically.

The ECA2LD Unity extension is published on Github: `https://github.com/dfki-asr/eca2ld-unity` (Last accessed Jul 2022). An example of the application of the ECA2LD Unity exception is given in Section 7.1.

5.3 SPARQL API Wrapper

The previous Section described how to publish simulation runtimes to an interactive Linked Data representation. However, in real world applications, it is often

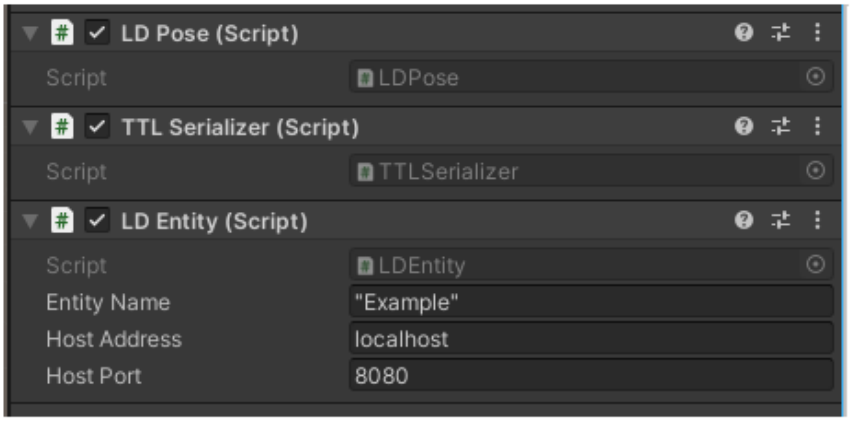

FIGURE 5.9: Publishing an Entity to Linked Data via the ECA2LD Unity3D extension.

required to include components for which the presented approach is not applicable, as the software components are not under governance of the system designer. Linking to such resources as is directly from the medium may lead to non-Linked-Data-conform resources being exposed to Linked Data conforming clients. From this problem arises Research Question R7: *How can continuity of the medium be ensured during link traversal?* (cf. Section 1.1.2)

A remedy is proposed in this Section. By the presented technology, non-RDF emitting resources may be queried by Linked Data clients via SPARQL queries.

5.3.1 Motivation

To this day, there is still a tremendous number of resources being published as structured data that does not yet follow Linked Data principles. For many text-based structured resource representations like XML, JSON or CSV, generic approaches have been developed to map them to RDF ([58, 216, 65, 175]). These approaches, however, are commonly used to create a copy of the original data as Linked Data data dump. This is not practical for data that is changing fast, such as live feeds or streams. For these, data dumps may soon become inconsistent with the original live data when freshness of the data would be crucial.

The problem of integrating live data that is not yet provided in a semantic Linked Data representation into a Linked Data application is mostly untackled.

As a remedy, we present in this paper a system, implemented as microservice, that provides a SPARQL 1.1 Query[58] service behind an API that is fully compliant to the SPARQL 1.1 protocol interface.[59] It allows to specify remote sources, perform a

[58]W3C SPARQL 1.1 Query Language Recommendation *(Visited May 2023)*: `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/`

[59]W3C SPARQL 1.1 Protocol Recommendation *(Visited May 2023)*: `https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/`

TABLE 5.1: Expected parameters for a SPARQL 1.1 Query service call, based on the original SPARQL 1.1 query protocol specification. The structured datagram γ takes the role of both Dataset $\mathcal{D}$ and (default) graph $\mathcal{G}$.

Method	**Query Parameters**	**Content-Type**	**Message Body**
`GET`	`query`=P (exactly 1), `source`=γ (ex. 1)	None	None
`POST` (URL enc. Parameters)	None	`application/ x-www-form-urlencoded`	URL-enc., &-separated: `query`=P (exactly 1), `source`=γ (exactly 1)
`POST` (direct)	`source`=γ (ex. 1)	`application/ sparql-query`	Unencoded SPARQL query string

provided query against them, and return as result a SPARQL query result in RDF representation.

The contents of this section have been originally published in the paper „On the fly SPARQL Execution for Structured non-RDF Web APIs" ([234]). For the contents in this chapter, the original paper has been slightly adapted, where appropriate.

5.3.2 Service Definition

From the notions of the previous section, we will now derive a formal definition of the service in terms of RDF liftings and SPARQL query execution. We first define the query execution as function on a (mapped) data source. Second, we define a SPARQL 1.1 query interface with the notions of the defined formal query approach.

Formal Definition

With the notions from Sect.2.1, we define moreover the following concepts:

We define

$$\mathcal{Q} = (\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V}) \tag{5.5}$$

as a shortcut for the set of all Triple Patterns.

Let Σ denote a *machine readable alphabet*, Σ^* the set of all words over alphabet Σ, and $\Gamma \subset \Sigma^*$ a set of *datagrams* in a given structured data format encoded in alphabet Σ. Such structured data formats could for example be comma separated value (CSV), JSON, or XML documents, as emitted by a Web resource, but also binary streams following a deterministic structure or protocol.

A datagram $\gamma \in \Gamma$ is then a valid structured piece of data that is encoded in a machine readable alphabet Σ.

We then define a ***mapping function***

$$\mathbf{m} : (\Gamma \subset \Sigma^*) \rightarrow (\mathcal{G} \subset \mathcal{T}), \mathcal{T} = (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{T} \cup \mathbf{B}) \tag{5.6}$$

as a function that translates a given structured datagram γ into an RDF Graph $\mathbf{g} \in \mathcal{G}$.

Let $\Omega_{\mathcal{Q}}$ denote the set of result mappings of a triple pattern $P \in \mathcal{Q}$ according to ([28]). A *service call* **s** is then a function **s** with the following properties:

$$\mathbf{s} : (\Gamma \times \mathcal{Q}) \rightarrow \Omega_{\mathcal{Q}} \tag{5.7}$$

$$\mathbf{s}(\gamma, P) = [\![P]\!]^{\mathcal{D}}_{\mathbf{m}(\gamma)},\ \mathbf{m} : (\Gamma \subset \Sigma^*) \rightarrow (\mathcal{G} \subset \mathcal{T}) \tag{5.8}$$

Means, a service accepts as call parameters a tuple that consists of a datagram γ from a datagram syntax Γ, and a triple Pattern $P \in \mathcal{Q}$ in some SPARQL syntax.

The result of the service call is an evaluation of the triple pattern P against the result of a lifting operation **m** on datagram γ.

Service SPARQL Query API

Following, we define the API to the SPARQL Wrapping Service as superset on the W3C SPARQL 1.1 Protocol[60] specification. We define parameters to specify a SPARQL query that is to be evaluated, as well as structured legacy data on which to evaluate the query, or URIs that point to endpoints from where to retrieve the data respectively. The subset of parameters that provides necessary information for the execution of the SPARQL query should completely comply with the SPARQL 1.1 Protocol specification.

Requests:

The SPARQL 1.1 Protocol Recommendation specifies three modes of query requests: Query by `HTTP GET` request with Query String parameters, by query via `HTTP POST` request, either with message body included as URL encoded query parameters, or as direct `POST` operation with all information contained in the message payload. Accordingly, a service call **s** is performed by an HTTP request with the following methods and parameters (see also Table 5.1):

Query via `GET`: The request is sent by the client via `HTTP GET` request to the service endpoint with no Content-Type header set, as request body is empty. The endpoint accepts as parameters *query* and *source*, with *query* being the properly serialized and URL encoded triple pattern P according to SPARQL 1.1. protocol specification, and

[60] https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/ *(Visited May 2023)*

source a reference to a resource from which the datagram γ can be retrieved, or a datagram γ as url-encoded string respectively.

Query via `POST` with URL encoded parameters: The request is sent by clients via `HTTP POST` to the service endpoint with Content-Type header set to `application/x-www-form-urlencoded`. The service accepts as parameters `query` and `source`. Parameters are URL-encoded and ampersand-separated. `query` contains the SPARQL triple pattern P, and `source` the datagram γ (or an URI from which γ can be retrieved).

Query via direct `POST`: Clients send `HTTP POST` requests with Content-Type header set to `application/sparql-query`. The source datagram γ is provided as encoded URL parameter, either inline, or as URI from which γ can be retrieved. The SPARQL query P is provided as unescaped string within the message payload.

Obviously, the above definition satisfies the SPARQL 1.1 protocol specification with respect to necessary parameters. Our API does not yet consider specification of an RDF dataset $\mathcal{D}$ against which the query should be executed in terms of a `default-graph-uri` or `named-graph-uri`. However, according to the SPARQL 1.1 protocol recommendation, these parameters are optional, and the specification states that, *"if an RDF Dataset is not specified in either the protocol request or the SPARQL query string, then implementations may execute the query against an implementation-defined default RDF dataset"*.[61] This default dataset is in our case the result of the mapping operation $\mathbf{m}(\gamma)$.

Responses:

Following the SPARQL 1.1 protocol specification, a query request to a service **s** returns the SPARQL query result with a success status code *(2xx)*.

The service moreover returns codes *400 (Bad Request)* and *500 (Internal Server Error)* in case of a malformed query, or a failure to execute the query respectively, in accordance with the SPARQL 1.1 Protocol specification. The service moreover returns a 400 error code if the supplied datagram γ is syntactically incorrect. The service may moreover return:

422 (Unprocessable Entity), if γ, either provided directly via `HTTP POST`, or as URI reference for download, is syntactically correct, but the mapping **m** returns an error for some reason, or any parameter specifying γ is missing.

502 (Bad Gateway), if γ is provided by URI reference, and the service under `source-uri` returns an error.

[61] `https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/#dataset` *(Visited May 2023)*

5.3.3 Implementation

This section will describe in detail the actual implementation of the previously defined service as a microservice. The overall architecture, and the components it is composed from, is provided in Section 5.3.3. 5.3.3 details out the service API beyond the SPARQL interface. We will give an overview of employed frameworks and libraries in our prototype implementation in Section 5.3.3.

Service Architecture

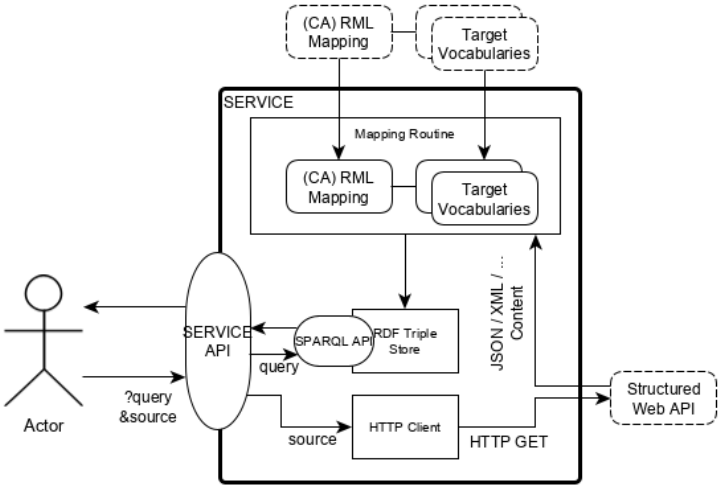

FIGURE 5.10: Architecture of the SPARQL API service

We build the query service around the components as shown in Figure 5.10. Client requests are received by an *HTTP API* endpoint. This API accepts HTTP GET and POST Requests with parameters for `query` and `source` according to the specification in Section 5.3.2. Upon receiving a client request, an *HTTP Client* component sends HTTP GET request to the endpoint as specified by the `source` parameter. If this request returns an error, this error is forwarded to the requesting client according to the error handling routine as described in Section 5.3.2.

In case the remote source returns valid data, it is used as input for an *RML Mapping* component. The respective mapping is provided (for example in terms of an RML mapping file) by the service itself, and can be inspected by clients via an HTTP GET request to the respective resource according to Section 5.3.3.

The result of the mapping is stored in an *in-memory RDF Triple Store* that provides a SPARQL query API to the service application. If the mapping was successful, the query as provided by the client as parameter is executed against the triple store that contains the mapping result. Otherwise, an error according to Sec. 5.3.2 is returned.

Finally, the result of the query is returned to the client as result of its request (or an error, if execution of the query was not successful).

Service self information

```
{ "definitions": {
  "Bike": {
    "type": ["object"],
    "properties": {
      "bike_id": {"type": "string"},
      "lat": {"type": "number"},
      "lon": {"type": "number"},
      "is_reserved": {"type": "integer"},
      "is_disabled": {"type": "integer"}
  }},

  "BikeData": {
    "type": "object",
    "properties": {
      "bikes": {
        "type": "array",
        "items": {"$ref":
          "#/definitions/Bike"}
  }}} # end of BikeData

}, # end of definitions

  "type": "object",
  "properties": {
    "data": {"$ref":
      "#/definitions/BikeData"}
  },
  "required": ["data"]
}
```

LISTING 5.9: Example of a JSON-Schema description of information conveyed by a `free_bike_status.json` datagram according to NABSA/GBFS General Bike Feed Specification.

In the current version, the service provides, listed as result of an HTTP `OPTIONS` request, routes to the following resources:

Under the route `/sourceformat/`, clients may retrieve the expected structure of source data. The source format is specified in JSON- or XML-Schema format, depending on the format that the service maps (see also Listing 5.9). The provided description may be used by clients to validate whether the service is capable of querying the intended legacy API according to JSON-/XML-Schema documentation.[62]

[62] `https://json-schema.org/`, `https://www.w3.org/XML/Schema` *(Visited May 2023)*

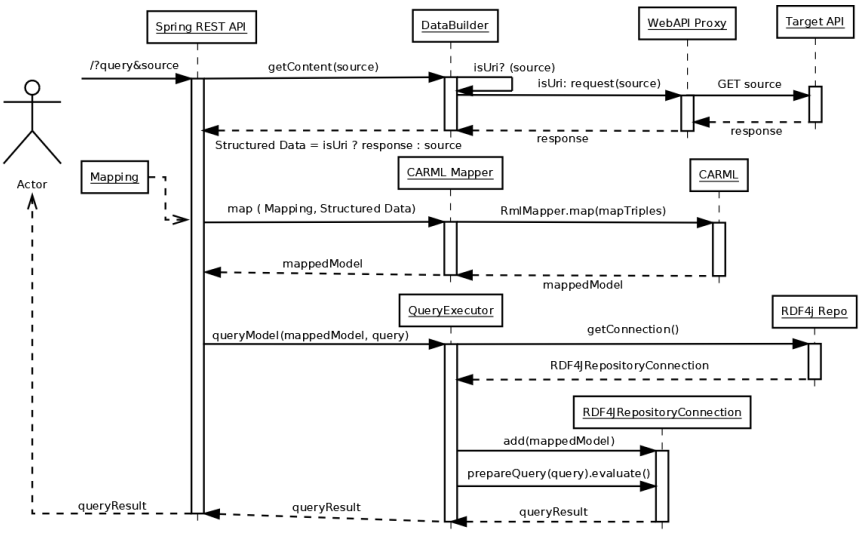

FIGURE 5.11: Call sequence between the different service components upon a client request to the API as defined in Sect. 5.3.2.

Moreover, the employed RML mapping file is provided under the route `/mapping/` for reasons of documentation. From the mapping file, client developers may learn employed ontologies or vocabularies in the resulting RDF SPARQL response, as returned by the service.

For future versions, we moreover plan to provide a definition of the output RDF format, for example in SHACL[63], to help clients to validate their local RDF representation automatically against the output that is generated by the service.

Prototype Implementation

Our prototype implementation is based on the Java Spring Framework[64] for the Web Service HTTP interface (*"SERVICE API"* in Fig. 5.10). RDF features are provided by the RDF4j[65] RDF library, using the RDF4j Repository API[66] for SPARQL Queries. The RDF4j Sail API[67] serves as temporal in-memory triple store to contain RDF mapping results against which the SPARQL Queries are executed (*"SPARQL API"* and *"RDF Triple Store"* in Fig. 5.10 respectively). The mappings are performed by the CARML[68] mapping framework. CARML extends RML mapping routines by the capability of defining a dynamic input stream as input the mapping, unlike RML, which expects a route to a fixed source.

Figure 5.11 shows the function call sequence between components of the service as chosen for our implementation: The HTTP interface provided by the Java Spring application server API receives a client request that specifies parameters `source` and

[63] `https://www.w3.org/TR/shacl/` *(Visited May 2023)*
[64] Spring Framework Website *(Visited May 2023)*: `ttps://spring.io/`
[65] RDF4j Website *(Visited May 2023)*: `https://rdf4j.org/`
[66] RDF4j Repo. API *(May 2023)*: `https://rdf4j.org/documentation/programming/repository/`
[67] RDF4j Sail API *(Visited May 2023)*: `https://rdf4j.org/documentation/sail/`
[68] CARML GitHub Repository: `https://github.com/carml/carml` *(Visited May 2023)*

`query` as either URL parameters or payload of a HTTP POST query. A `DataBuilder` class checks whether the supplied `source` argument is an URI, or already the datagram γ itself. In case of a it being an URI, a `WebAPIProxy` is used to retrieve γ from the provided API. γ then serves as `Structured Data` input for the mapping process.

The (CA)RML `Mapping` file is provided by the service itself. It is fed to a `CARML Mapper` class that, after some preprocessing steps, uses the `CARML` mapping library to translate γ to a `mappedModel`, which will be in RDF Graph form.

The `mappedModel`, and the `query` as provided by the client, are used as input for an `QueryExecutor`. The `QueryExecutor` first opens a connection to a temporary `RDF4j Repo`, loads the `mappedModel` into it, and executes the `query` via the RDF4j Sail API.

The `queryResult` of this operation is finally returned to the client as result of the client's initial query request.

5.3.4 In-Use Examples

Following, we demonstrate the usage of the service using examples from public transport and bike rental domain, the main application domain of the funding project SmartMaaS.

Second, we show how the presented service can be used to infer additional information from distributed data sets by employing distributed SPARQL queries over several SPARQL Wrapper services with the `SERVICE` keyword.

SELECT query on JSON data

The following example demonstrates a simple SELECT query against a JSON data endpoint. The query as shown in Listing 5.10 is sent as `query` parameter to the service endpoint, using the JSON data as shown in Listing 5.11 as input. The RDF result is shown in Listing 5.12.

The overall execution time of the query in the example was about 180ms (milliseconds) for a dataset of 63 bike sharing station items. Of these 180ms, 30ms were spent on the CARML lifting process, and 10ms on the SPARQL query execution (measured on a Intel Core i7-4770k, 3.5GHz). The remaining time was spent to retrieve the source data from the provided URI as `source` parameter.

```
SELECT ?name ?lat ?lon WHERE {
  ?station a gbfs:Station ;
  gbfs:name ?name ;
  wgs84_pos:lat ?lat ;
  wgs84_pos:long ?lon .
}
```

LISTING 5.10: A SPARQL SELECT query that reads location information for bike sharing station from a GBFS ervice endpoint

```
{"last_updated": 1595835393,
 "ttl": 60,
  "data": {
   "stations": [
   {
     "station_id": "10044279",
     "name": "Bahnhof Beuel",
     "short_name": "4741",
     "lat": 50.739211,
     "lon": 7.126598,
     "region_id": "547"
   },
   {
     "station_id": "10044287",
     "name": "Haltepunkt Bonn-West",
     "short_name": "4742",
     "lat": 50.7367675,
     "lon": 7.0809567,
     "region_id": "547"
   }, ... ]
 }}
```

LISTING 5.11: Input provided as example for a simple SELECT query (excerpt; source: `https://gbfs.nextbike.net/maps/gbfs/v1/nextbike_bf/de/station_information.json`)

```
<results>
  <result>
   <binding name='name'>
    <literal>Bahnhof Beuel</literal>
   </binding>
  <binding name='lon'>
   <literal datatype='xsd:double'>
     7.126598
   </literal>
  </binding>
  <binding name='lat'>
   <literal datatype='xsd:double'>
    50.739211
   </literal>
  </binding>
  </result>
 <result>
  ....
</results>
```

LISTING 5.12: Query result of the Query in Listing 5.10 against the data in Listing 5.11

Federated queries

The design of the Service API over parameterized, SPARQL 1.1 compliant request URLs also allows for federated SPARQL queries using the `SERVICE` keyword, as described in the respective W3C recommendation document.[69]

In the formal W3C SPARQL 1.1 Grammar Recommendation,[70] a `ServiceGraphPattern` (entry `59` in the respective grammar document[71]) is defined as

`ServiceGraphPattern := 'SERVICE' 'SILENT'? VarOrIri GroupGraphPattern`

Accordingly, a federated query against a SPARQL Wrapper Service endpoint can be performed by employing as `VarOrIri` a valid URI against the SPARQL Wrapper Service API according to Section 5.3.2, and as *ServiceGraphPattern* the query that is to be executed against the dataset γ that is referred to as `source` (acc. to API definition in Section 5.3.2), employing the mapping $\mathbf{m}(\gamma)$ that is provided by the service under the URI that is provided as `VarOrIri` element in the query. The parameter `query` can be omitted in this case, as the Triple Pattern P that describes the query is already

[69] W3C SPARQL 1.1 Federated Query recommendation: `https://www.w3.org/TR/sparql11-federated-query/`

[70] SPARQL 1.1 Grammar: `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#grammar`

[71] As at current date, June 2020

provided in terms of the `GroupGraphPattern` that follows the `VarOrIri` element of the federated query.

Listing 5.13 shows an example of a federated query. For brevity, the complete route to the service endpoint, as well source datagram γ are omitted, and given as `#srvpath` and `#srcpath` respectively.[72]

```
CONSTRUCT {
  ?station a gbfs:Station ,
           wgs84_pos:SpatialThing;
    gbfs:name ?station_name ;
    wgs84_pos:lat_lon ?lat_lon_pos.

  ?bike_id a gbfs:Bike ;
    wgs84_pos:location ?station .
}
WHERE {
  ?bike wgs84_pos:lat ?lat ;
        wgs84_pos:long ?lon .

  SERVICE <#srvpath/?source=#srcpath> {
    ?station gbfs:name ?name ;
    wgs84_pos:lat ?station_lat;
    wgs84_pos:long ?station_lon .
  }
  FILTER (
    ABS(?lat-?station_lat)<0.001 &&
    ABS(?lon-?station_lon)<0.001)
  BIND
    (CONCAT(str(?lat),",",str(?lon))
      as ?lat_lon_pos)
}
```

LISTING 5.13: Example of a federated SPARQL query

[72]The complete URI used for the given example was `http://sparql-wrapper.service/?source=https://gbfs.nextbike.net/maps/gbfs/v1/nextbike_bf/de/station_information.json`

```
<http://foo.bar/stations/10044540>
  a gbfs:Station,
    wgs84_pos:SpatialThing;
  gbfs:name "Juridicum";
  wgs84_pos:lat_lon "50.73009232899485,
                7.108277678489685" .

<http://foo.bar/bikes/44608> a gbfs:Bike;
  wgs84_pos:location
    <http://foo.bar/stations/10044540> .

<http://foo.bar/bikes/45448> a gbfs:Bike;
  wgs84_pos:location
    <http://foo.bar/stations/10044540> .

<http://foo.bar/bikes/45337> a gbfs:Bike;
  wgs84_pos:location
    <http://foo.bar/stations/10044540> .
```

LISTING 5.14: Query result (excerpt) as returned by the federated SPARQL query example

In our evaluation, the query given in Listing 5.13 was performed against an endpoint that emits information about the location of rentable bikes in GBFS JSON format. The `GraphGroupPattern` in the `SERVICE` clause merges that information with information about the location of rental bike stations of the same provider. The respective query response is shown in Listing 5.14. Note that the displayed information (*Which bike is currently parked at which station?*), is originally not provided by how the GBFS datamodel is defined. Deriving this information via semantic queries over the originally not semantically enriched GBFS data is a direct benefit from lifting queries against the GBFS data to a semantic representation.

The total execution time of the construct query was 560ms for a dataset of 63 bike station entries, and 649 items for free bikes respectively. Of these 560ms, approx. 10ms each were spent for the lifting process of both the datasets, and another 30ms to perform the query against the station information data in the SERVICE clause. The remaining times were spent to retrieve the source data from the provided URIs.

5.3.5 Conclusion and Future Work

In this paper, we have presented a novel service that allows to perform SPARQL queries against non-RDF datasets. Unlike existing solutions, the presented service is not limited to a certain source format, as long as the source to be queried is structured. The service offers a SPARQL 1.1 protocol HTTP query API, that is also suitable to be used as endpoint for federated SPARQL queries.

Based on the original SPARQL 1.1 Protocol, we have derived a formal query API, and provided a formal design of the presented solution. We have moreover presented a proposal for an actual service architecture, based on the CARML non-RDF-to-RDF mapping engine.

Finally, we outlined our protocol implementation, and concluded with an evaluation of the prototype implementation. We moreover demonstrated the applicability of the service in the scope of federated SPARQL queries.

The presented implementation is published on Github under `https://github.com/SmartMaaS/sparql-api-wrapper`.

The current design and implementation so far neglect named graphs. How to include this concept in the presented service design is subject to future work.

So far, the service supports SPARQL SELECT, ASK and CONSTRUCT queries. As future work, we also plan to investigate how SPARQL UPDATE queries against a non-RDF endpoint may be used to modify a remote non-RDF dataset, given that the remote endpoint allows data modification.

We have discussed our solution with respect to use-cases from the domain of traffic and public transport. We see however, and plan to evaluate, a clear applicability in use-cases from industrial domains, as well as Smart City, Smart Grid, and Smart Living scenarios.

In the context of this thesis, the presented technology allows agents to interact with resources as if they were published via Linked Data media, even though the original resource may not emit RDF data.

This allows agents to semantically interpret data in the context of the information given in the Linked Data medium, even though a semantic (RDF) representation of the data was originally not available. This may occur in particular when encountering external (remote) data endpoints which are not under the control of the system designer. By providing the capabilities of service via an interface that follows the SPARQL 1.1 protocol specification, the provisioning of the originally non-semantic structural data is, for the agent, transparent.

The presented service was applied in Use Cases 7.2 and 7.3 with exactly this purpose.

Chapter 6

Linked Data Media Consuming Agents

After the detailed description of possible implementations of dynamic environments in Linked Data media in the previous chapter, this chapter will provide a formal framework to define the behavior Linked Data Media consuming agents. The definition of Linked Data agents will be formulated using the MCC Calculus (cf. Section 2.2). The chapter will then proceed with a detailed translation between MCC behavior notations, and a semantically equivalent notation as (SPARQL) behavior trees [125, 6, 7].

To this end, this chapter makes the following contributions to facilitate the implementation of Linked Data consuming agents:

Section 6.1 builds upon the formal definition of interactive Linked Data Media in Section 4.3 to define the interaction between agents and the medium. This behavior model will be formal and generic, and by this abstract from technical details of the implementation. The models will rely on basic interaction patterns as specified for RESTful resource oriented architectures, and will follow HATEOAS principles to fulfill Richardson maturity as described in Chapter Background. Contents of this chapter are a revised versions of the original publications *stigLD: Stigmergic Coordination of Linked Data Agents* [220].

Section 6.2 provides a detailed formal model of behavior trees, and describes a semantic preserving mapping between formal agent definitions in MCC, and behavior trees representations. With behavior trees being not only a formal model, but actual executable programs with powerful parallel semantics, this Section by that lays foundation to directly transfer formal agent models to provable correct executable agent programs. Contents of this section have been original published in *Behavior Trees as executable representation of Milner Calculus notations* [235].

6.1 Generic Linked Data Agents

6.1.1 Definition of a single tropistic agent

We specify a *tropistic* [92, section 13.1] Linked Data agent $\texttt{AGENT}_k$ as an active component

$$\texttt{AGENT}_k \quad = \quad \texttt{PERC}_k(i \in R, G = \emptyset, L = \{i\})$$

being initially situated at a resource $i \in R$ without a-priori agent knowledge ($G = \emptyset$) and a linkset $L = \{i\}$ restricted to i. Our specification of $\texttt{AGENT}_k$ puts emphasis on a direct response to its perceptions and favours to employ *situated perceptions* [230] of the environment as the basis for deciding which action to perform next. We model situated perception in CCS-style as

$$\begin{aligned}\texttt{PERC}_k(i, G, L) = &\overline{req}_{\eta(j)}(\texttt{GET}, j, \emptyset).res_{\eta(j)}(rc, (L', G')).\\ &\Big(\texttt{PERC}_k(i, G'', L'') + \texttt{REACT}_k(i, G'', L'')\Big)\end{aligned} \tag{6.1}$$

where $\texttt{AGENT}_k$ - while being situated at i - will at first issue a $\texttt{GET}$ request for a resource j in its current linkset L via channel $\overline{req}_{\eta(j)}$ and then awaits the server's response via channel $res_{\eta(j)}$ with return code $rc \in \texttt{RET}$, response linkset $L' \subset \mathbf{I}$ and response graph in $G' \in \mathcal{T}$. Subsequently, the agent executes (i) a *perceptional query* $q_{\texttt{PERC}_k}$ over G' in order to update its situational knowledge to

$$G'' = G \cup \texttt{ans}(q_{\texttt{PERC}_k}, G')$$

as well as (ii) a *navigational query* $q_{\texttt{NAV}_k}$ over its updated knowledge graph in order to update its linkset to

$$L'' = L \cup L' \cup \texttt{sel}(q_{\texttt{NAV}_k}, G''))$$

On the basis of G'' and L'', $\texttt{AGENT}_k$ chooses to either recurse into its situated perception process $\texttt{PERC}_k(i, G'', L'')$ or to enter the process $\texttt{REACT}_k(i, G'', L'')$ in order to select an action on the basis of a local, short-time view of its environment. An action selected only on the basis of a situated perception is called a *reaction*.

We model the process of selecting reactions in the following way

$$\begin{aligned}\texttt{REACT}_k(i, G, L) = &\ \texttt{PERC}_k(j \in L, \emptyset, \{j\}) +\\ &\sum_{m \in \texttt{OPS} \setminus \{\texttt{GET}\}} \Big(\textbf{if } \texttt{ask}(\widehat{q}_{\texttt{m}_k}, G, L) \textbf{ then } \texttt{m}_k(i, G, L) \textbf{ else } \texttt{REACT}_k(i, G, L)\Big)\end{aligned} \tag{6.2}$$

In essence, an agent may choose to either

1. re-situate and perform situated perception of resource $j \in L, j \neq i$ with the implication that its situational knowledge and linkset will be reset; hence it does neither maintain a long-term internal model of its environment nor pursues explicit goals;
2. request the execution of operation $m \in \texttt{OPS} \setminus \{\texttt{GET}\}$ against resource i given

that the *conditional query* $\widehat{q}_{\mathtt{m}_k}$ over its knowledge graph G holds; possible instantiations of $\mathtt{m}_k(i,,L)$ are given by

$$\begin{aligned}
\mathtt{PUT}_k(i,G,L) &= \overline{req}_{\eta(i)}(\mathtt{PUT},i,\mathtt{ans}(q_{\mathtt{PUT}_k},G)).res_{\eta(i)}(rc,(\emptyset,\emptyset)).\mathtt{REACT}_k(i,G,L)\\
\mathtt{POST}_k(i,G,L) &= \overline{req}_{\eta(i)}(\mathtt{POST},i,\mathtt{ans}(q_{\mathtt{POST}_k},G)).res_{\eta(i)}(rc,(L',\emptyset)).\mathtt{REACT}_k(i,G,L\cup L')\\
\mathtt{DEL}_k(i,G,L) &= \overline{req}_{\eta(i)}(\mathtt{DEL},i,\emptyset).res_{\eta(i)}(rc,(L',\emptyset)).\mathtt{REACT}_k(j\in L\setminus L',G,L\setminus L')
\end{aligned}$$

where $\mathtt{ans}(q_{\mathtt{m}_k},G)$ is the result graph of executing an *effectual query* $q_{\mathtt{m}_k}$ over the agent's knowledge graph G with $m\in\{\mathtt{PUT},\mathtt{POST}\}$.

6.1.2 Extension to agent swarms

Following, we model extend the notion of tropistic Linked Data agents from the previous chapter, and in [220] to *hysteretic* [82] *agent swarms*.[73]. Unlike tropistic agents, hysteretic agents possess the capability to maintain memory, which we exploit to supply newly spawned agents with an initial knowledge K (whereas, for tropistic agents, the initial knowledge is assumed to be empty).

Following [220], with R defining the set of all resources, G the knowledge of the agent represented as RDF graph, and L a link set of known resources, a hysteretic agent of a class c can then defined by a process AGENT in CCS as:

$$\mathtt{AGENT}_c(i\in R,K) = \mathtt{SENSE}_c(i,G=K,L=\{i\}).\mathtt{PERC}_c(G',L').\mathtt{REACT}_c(i,G'',L'') \tag{6.3}$$

$$\mathtt{SENSE}_c(i,G,L) = \overline{req}_{\eta(i)}(\mathtt{GET},i).res_{\eta(i)}(L',G') \tag{6.4}$$

$$\mathtt{PERC}_c(G',L') = \overline{ans}(q_{\mathtt{PERC}},G).qres(G'',L'') \tag{6.5}$$

with SENSE and PERC processes by which the agent receives sensory input from resource i, and perceives (i.e. filters) the input respectively. G'' and L'' refer to updated knowledge and link sets as result of executing perception and navigational queries respectively [220, p.5]. The class c of an agent determines the exact set of queries by which the agent will perceive, interact with, and navigate between resources, with agents of the same class always using the exact same queries. As REACTion, the agent either navigates to another resource in its updated link set L'' and continue perception there, or it may ACT on its current resource and SPAWN a number of new agents of a chosen class with some initial knowledge K^i:

$$\begin{aligned}
\mathtt{REACT}_c(i,G'',L'') &= \mathtt{AGENT}_c(j\in L'',K,\{j\})\\
&\quad +(\mathtt{ACT}_c(i,G'',L'')||\mathtt{SPAWN}_c(c,K,L^s)) &(6.6)\\
\mathtt{SPAWN}_c(c,K,L^s) &= \Big|\Big|_{j\in L^s}\mathtt{AGENT}_c(j\in L^s,K^i)\,, &(6.7)
\end{aligned}$$

L^s denotes an updated link set that is determined by evaluating a *spawn query* $q_{\mathtt{SPWN}}$ on the updated Graph G'' after perception: $L^s = ans(q^c_{\mathtt{SPWN}}(G''))$.

[73] The respective extension of the tropistic agent model was originally published in [235]

6.2 Implementation of Linked Data Agents as SPARQL behavior trees

Having formally defined Linked Data agents, this section will now present a possible agent implementation, based on the visual programming paradigm of "Behavior Trees" (BT) [125]. We will see that behavior trees will allow a semantic preserving, one-to-one mapping from agent models in CCS to actually executable agent programs. This mapping in particular preserves the generic nature of the presented agent models. By building the implemention on a BT implementation that is optimized towards Linked Data, SPARQL Behavior Trees [6, 7], the established concepts for Linked Data agents directly transform to concepts inherent to the behavior trees: While (labeled) processes have a one to one correspondence to (named) behavior trees, specialised queries, such as perception or navigational queries, are conveniently supplied as leaf nodes within the behavior tree.

The following contents have originally been published at the "21st IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology" (WI-IAT'22), under the title "Behavior Trees as executable representation of Milner Calculus notations". [235]

6.2.1 Motivation

Behavior Trees (BT) [125] are a popular tool to model agent behaviors. Behavior trees constitute an executable visual programming paradigm that is comparable to Finite State Machines (FSM) or Hierarchical Task Networks (HTN). BTs have recently been extended to specifically implement the behavior of distributed agents in hypermedia MAS [50] by supporting SPARQL queries[74] against RDF[75] knowledge bases [6]. Given that, likewise, the execution semantics of process notations in established calculi can be broken down to a tree structure [215], and both calculus expressions and BTs have representations as FSMs [183, 130, 156], we see strong opportunities in investigating the similarities between formal notations of agent programs in the Milner Calculus, and equivalent executable agent programs, encoded as BTs.

In this context, this Section makes the following contributions: A formal model of BTs for hypermedia driven Multi-Agent Systems; a formal notation of SPARQL Behavior Tree (SPARQL-BT) nodes for hypermedia MAS in Milner Calculus notation; a formal proof for a semantic preserving mapping of agent behavior descriptions from process calculus notation to executable BTs; and an example of how to practically apply the results to model agent swarms in an Industrie 4.0 cyber-physical production scenario.

[74] SPARQL 1.1 Specification: `https://www.w3.org/TR/sparql11-query/` *Visited May 2023*
[75] RDF 1.1 Primer: `https://www.w3.org/TR/rdf11-primer/` *Visited May 2023*

6.2.2 Related Work

Behavior Trees originate from the game industry and were first mentioned in 2005 in [125]. BTs were primarily developed to intuitively model Non-Player Character (NPC) behavior in a modular and reusable way. Nowadays they are used more frequently in industry, e.g. for robot control [166, 182]. This paradigm with loops, sequences, parallels, and an implicit knowledge base is often described as a combination of Decision Trees (DT) with FSMs [164]. Generally there is no clearly recognized definition respectively version of this paradigm [164], but [54] provides a formal description of BTs in order to synthesize them and to guarantee their correct execution. Formal representation of agent behavior for the sake of verifying the correct behavior of MAS is a topic in research to this day [1]. Such representations include generic labeled transition systems [42], or in situation calculus [10], the *pi*-calculus [79, 261, 149, 215], or Milner Calculus [220].

Analogies between Milner Calculus and FSMs are known already for several decades [130]. Luttik et al. have shown that executable behaviors, which can generally be mapped to finite Turing Machines, always have a representation in the π-Calculus (while the converse does not generally hold) [156]. This inspired us to investigate for similar analogies between BTs and calculus expressions. π-calculus expressions have been translated into executable Java code [150], to assembly language [53], and to executable code in specialised frameworks [244, 104]. The programming language *pseuCo*[76], strongly based on CCS, allows to display executable programs in terms of Milner Calculus processes [90].

Our research on the topics concluded that while both formal calculi for concurrent software systems and BTs as powerful tool to efficiently model MAS have strong research communities, both fields have to this point been considered more or less independently. In particular, the transferability of provable and verifiable calculus expressions to directly executable BTs has so far hardly a been topic of research. We thus endeavor with this paper to motivate BTs as executable representation of calculus notations.

6.2.3 Formal Behavior Tree Model

Let $\mathcal{K} \subseteq 2^{\mathcal{T}}$ denote the set of *Knowledge Bases* maintained by an agent, $\mathcal{B}$ denote the set of all BTs, $\mathcal{N} = \mathcal{I} \cup \mathcal{L}$ the set of *Nodes*, itself composed from inner nodes $i \in \mathcal{I}$ and leaf nodes $l \in \mathcal{L}$, and $\mathcal{E}$ a set of (directed) *edges* connecting nodes in a BT respectively.

We can then describe a BT $b \in \mathcal{B}$ as a directed (acyclic) graph via a tuple

$$b = \Big(N \subseteq (\mathcal{N} \cup \varnothing), E \subseteq \mathcal{E}\Big) \tag{6.8}$$

[76] `https://pseuco.com/` *Visited May 2023*

that contains of a set of nodes N which are connected by a set of (directed) edges E. $\varnothing$ denotes the root node of the BT.

An edge $e \in \mathcal{E}$ can be expressed as a tuple linking a single node to another node in the BT:

$$e = (l \in \{\varnothing\} \cup \mathcal{N}, r \in \mathcal{N}) \tag{6.9}$$

In this notation, l refers to the *source* of the directed edge, r to the *sink*. l and r are chosen as names, as in the SPARQL-BT editor of our choice, AJAN[77] [6], sources usually appear at the *left* end of the edge, whereas sinks appear on the *right* end.

If there are two edges $e_i, e_o \in \mathcal{E}$ with $e_i = (n_l, n), e_o = (n, n_r)$ (e_i for *incoming* edge, e_o for *outgoing*), we can moreover express a *node* $n \in \mathcal{N}$ of a BT in terms of the node's *left-hand side* n_l and *right-hand side* n_r:

$$n = \left(n_l \in \mathcal{I} \cup \{\varnothing\}, n_r \in 2^{(\mathbb{N} \times \mathcal{N})} \cup \varnothing\right) \tag{6.10}$$

A node can thus connect any parent with a number of tuples of the form $(i \in \mathbb{N}, r \in \mathcal{N})$. We refer to $n_l \in \mathcal{I}$ as *left-hand side (LHS)* of the node, and to $n_r \in 2^{(\mathbb{N} \times \mathcal{N})}$ as *right-hand side (RHS)* of the node. In case of a leaf node, the RHS is empty as denoted via $\varnothing$. $i \in \mathbb{N}$ determines the *evaluation order* of RHS elements, i.e., for two elements r^i, r^j with orders i and j, it holds that if $i < j$, node r^i is evaluated *before* node r^j.

6.2.4 Behavior Tree Execution Semantics

We denote as $\circ_l(\phi, \kappa) : (2^{\mathcal{P}} \times 2^{\mathcal{K}}) \rightarrow \{\texttt{SUCCESS}, \texttt{FAIL}\}$ the execution of a *leaf node* $l \in \mathcal{L}$ with $\phi \subseteq \mathcal{P}$ a set of node specific parameters, and $\leq \subseteq \mathcal{K}$ a set of knowledge bases as targets of the node execution. We further define for execution of a *composite node* $c \in \mathcal{C}$ a function $\circ_c(R, \phi) : (2^{(\mathbb{N} x \mathcal{N})} \times 2^{\mathcal{P}}) \rightarrow \{\texttt{SUCCESS}, \texttt{FAIL}\}$, with R RHS elements as defined above, and ϕ a set of parameters analogue to the leaf node case. Finally, *decorator nodes* modify the result of a single child node, i.e. $\circ_d(r, \phi) : (\mathcal{N}, 2^{\mathcal{P}}) \rightarrow \{\texttt{SUCCESS}, \texttt{FAIL}\}$. By this, we can describe the execution semantics of particular BT nodes as follows:

Sequence: A *sequence* composite node `seq` executes each of its RHS nodes in execution order until the first node fails. `seq` returns `SUCCESS` only if each of its children return `SUCCESS`, and `FAIL` otherwise:[78]

$$\circ_c = \texttt{seq}(R, \varnothing) \equiv \bigwedge_{k=0}^{min_j[(j,r_j) \in R] | n_j(R_j, \phi_j) = \texttt{FAIL}} n_k(R_k, \phi_k) \tag{6.11}$$

[77] `https://github.com/aantakli/AJAN-service` *Visited May 2023*

[78] For the chosen notation, we interpret `SUCCESS` and `FAIL` as boolean values, i.e $\texttt{SUCCESS} \wedge \texttt{FAIL} \Rightarrow \texttt{FAIL}$, $\texttt{SUCCESS} \vee \texttt{FAIL} \Rightarrow \texttt{SUCCESS}$

Priority: A *priority* composite node `prio` returns `SUCCESS` as soon as the first of its children return `SUCCESS`, and `FAIL`, if none of the children return `SUCCESS`

$$\circ_c = \texttt{prio}(R, \emptyset) \equiv \bigvee_{k=0}^{min_j[(j,r_j)\in R]|n_j(R_j,\phi_j)=\texttt{SUCCESS}} n_k(R_k, \phi_k) \tag{6.12}$$

Parallel: A *parallel* composite node `par` executes all of its RHS children independent of the return status of the individual child node, and returns `SUCCESS` only if all children returned `SUCCESS`:

$$\circ_c = \texttt{par}(R, \emptyset) \equiv \bigwedge_{k=0}^{|R|} n_k(R_k, \phi_k) \tag{6.13}$$

Parallel Priority: A *parallel priority* composite node `parprio` executes all of its RHS children, and returns `SUCCESS` if any of the children returned `SUCCESS`:

$$\circ_c = \texttt{parprio}(R, \emptyset) \equiv \bigvee_{k=0}^{|R|} n_k(R_k, \phi_k) \tag{6.14}$$

Executor: A *executor* composite node `exec` picks a single one out of any RHS child nodes, and execute it, while discarding any other node. In SPARQL-BT, executor nodes execute the n^{th} child node, with parameter n either supplied directly, or by determining the parameter as result of a SPARQL `SELECT` query.

$$\circ_c = \texttt{exec}(R, k \in \mathbb{N}) = n_k(R_k, \phi_k)\ ,\ 0 \leq k < |R| \tag{6.15}$$

Condition: A *condition* leaf node `cond` evaluates a supplied condition (in SPARQL-BT, encoded as SPARQL `ASK` query) against any of the knowledge bases of the agent to a parameter $b \in \mathbb{B}$. It returns `SUCCESS`, if the condition holds, and `FAIL` otherwise:

$$\circ_l = \texttt{cond}(b \in \mathbb{B}, K \in \mathcal{K}) \equiv \textbf{if}\ b = \texttt{true}\ \textbf{then}\ \texttt{SUCCESS}\ \textbf{else}\ \texttt{FAIL} \tag{6.16}$$

SPARQL-BTs and the implementation AJAN[6] moreover define a series of leaf nodes to communicate with hypermedia environments, and update and manipulate the knowledge bases of an agent: `QUERY` nodes to retrieve information from a remote resource, `MESSAGE` nodes to send requests to remote endpoints, `WRITE` nodes to write information from one local knowledge base into another, and `UPDATE` nodes to modify the content of a single knowledge base.

With $\mathcal{Q}$ the set of all queries, a query $q^{\texttt{X}}$ a SPARQL query of type $\texttt{X} \in \{\texttt{SEL}, \texttt{CON}, \texttt{ASK}, \texttt{UPD}\}$, $op \in \{\texttt{GET}, \texttt{PUT}, \texttt{POST}, \texttt{DELETE}\}$ HTTP operators, and *rc* some return code as returned by a server upon request, we can describe these nodes in

CCS as:

$$\begin{aligned}
\texttt{QUERY}(\{q^{\texttt{SEL}}\} &\subset \mathcal{Q}, \{K, L\} \subset \mathcal{K}) && (6.17)\\
&= \overline{req}_{\eta(j)}(\texttt{GET}, j, \emptyset).res(rc, (\cdot, G)).[K = K \cup G]\\
\texttt{MESSAGE}(\{q^{\texttt{SEL}}, p^{\texttt{CON}}\} &\subseteq \mathcal{Q}, \{K, L\} \subset \mathcal{K}) && (6.18)\\
&= \overline{ans}(p^{\texttt{CON}}, K).qres(G).\overline{req}_{\eta(j)}(op, j, G).res(rc, \cdot, \cdot)\\
\texttt{WRITE}(\{q^{\texttt{CON}}\} &\subset \mathcal{Q}, \{K, L\} \subset \mathcal{K}) && (6.19)\\
&= \overline{ans}(q^{\texttt{CON}}, K).qres(G).[L = L \cup G]\\
\texttt{UPDATE}(\{q^{\texttt{UPD}}\} &\subset \mathcal{Q}, K \in \mathcal{K}) = \overline{ans}(q^{\texttt{UPD}}, K).qres(K').[K = K'] && (6.20)
\end{aligned}$$

`QUERY` and `MESSAGE` nodes return `SUCCESS`, if the remote end returns a `OK` return code, and `FAIL` otherwise. `WRITE` and `UPDATE` return `SUCCESS`, if the respective queries were executed against the supplied knowledge base successfully, and `FAIL` otherwise.

6.2.5 Equivalence of CCS and SPARQL-BT

We now show equivalence between CCS operators as summarized in Section 2.2, and BT evaluation semantics:

Action ($\alpha.P \xrightarrow{\alpha} P$): We show that sequential node execution complies with the semantics of action execution in CCS. Consider a `sequence` composite node $\texttt{seq}_\texttt{c}$ with $|R| = m$ executed right hand side elements. We can then chose a $0 \leq l < m$ to decompose Eqn. 6.11 as $\texttt{seq}_\texttt{c}(R, \emptyset) = \bigwedge_{i=0}^{l} n_i(\cdot) \wedge \bigwedge_{j=l}^{m} n_j(\cdot)$. For $l = 0$ we get:

$$\begin{aligned}
\texttt{seq}_\texttt{c}(R, \emptyset) = \bigwedge_{i=0}^{0} n_i(\cdot) \wedge \bigwedge_{j=1}^{m} n_j(\cdot) \quad &= \quad n_0(\cdot) \wedge \bigwedge_{j=1}^{m} n_j(\cdot)\\
&= n_0(\cdot) \wedge \texttt{seq}_c(R \setminus \{0, n_0\})
\end{aligned}$$

With action $\alpha := n_0$ and remaining process $P := \texttt{seq}_c(R \setminus \{0, n_0\})$, it thus holds that by executing node n_0 as action $\alpha : \texttt{seq}_c(R, \emptyset) \equiv \alpha.P \xrightarrow{\alpha} P$. □

The same holds for Priority nodes as can be shown by equivalent proof. Values are passed between actions resp. processes indirectly, by each action reading values from the specified knowledge bases. Process execution scope can be preserved by using the same shared knowledge base for each action in a process.

Parallel Composition in CCS relates to parallel and parallel priority nodes, as can be derived directly from Eqns.6.13 and 6.14, as both execute all children independently from each other, i.e. for a node $\texttt{prio}(\{(0, P), (1, Q)\})$ with P and Q child processes or actions encoded as composite nodes or leaf nodes, it holds that both P and Q are evaluated, and thus $\texttt{par/parprio}(\{(0, P), (1, Q)\}) \equiv P|Q$ □

Choice (by executor): Let B a BT, P, Q be two processes resp. BTs in the right-hand side of an executor node $n_e \in B$, with P being first in execution sequence, and with n_p and n_q root nodes of P and Q respectively, i.e. $n_e = \texttt{exec}\big(R = \{(0, n_p), (1, n_q)\}, k \in \{0, 1\}\big)$.

$$\texttt{exec}\big(R = \{(0, n_p), (1, n_q)\}, k \in \{0, 1\}\big) = \begin{cases} n_p(R_p, \phi_p) & , k = 0 \\ n_q(R_q, \phi_q) & , k = 1 \end{cases} \equiv P + Q$$

i.e. the process B continues as process P or process Q, depending on the choice of parameter k. As shown for Action and Parallel execution, the nodes n_p and n_q may encode single actions in case of them being leaf nodes, or processes, in case of n_p and n_q being composite nodes. □

Choice (by priority and conditioned sequence):

Consider a priority node $\texttt{prio}\big(\{(0, p), (1, q)\}, \emptyset\big)$, with p a sequence node $\texttt{seq}\big(\{(0, n_{\texttt{cond}})\} \cup P \subseteq 2^{(\mathcal{N} x \mathbb{N})}, \emptyset\big)$, i.e., with a condition node preceding a set of right-hand side nodes as first element of the sequence. Let us first assume that $n_{\texttt{cond}}$ evaluates as `false`. It holds by Eqn. 6.11 that $n_{\texttt{seq}}$ returns `FAIL` directly after executing $n_{\texttt{cond}}$, i.e., the remaining process P is discarded.

Moreover, by Eqn. 6.12, it holds that $n_{\texttt{prio}}$ evaluates its children until the first child node returns `SUCCESS`, i.e., $n_{\texttt{prio}}$ will continue evaluating the process encoded by node q. Conversely, if $n_{\texttt{cond}}$ evaluates as `true`, process P of node p will be executed entirely, whereas the process encoded by q will be skipped due to early exit of the priority node. By this it holds that □

$$\texttt{prio}\Big[\big\{\big(0, \texttt{seq}\big[\{(0, \texttt{cond}), (1, P)\}, \emptyset\big]\big), (1, Q)\big)\big\}, \emptyset\Big] \equiv P + Q$$

Process labelling in CCS translates to naming behavior trees, which follows directly from that by all of the above, any Process P in CCS notation can be transferred to an equivalent BT $B \in \mathcal{B}$, and thus any name K assigned to BT B can be considered a process identifier K for the equivalent process P. Referring to P via K then corresponds to calling BT **B** from a parent process using a respective BT node. □

6.2.6 Conclusion

In this Section, we demonstrated how to formally transfer notations of agent programs in Milner Calculus (CCS) [176] into executable Behavior Trees (BT) [125, 6]. For this, we first provided a formal notation for BTs that takes into account the topological relation between BT nodes within the tree. We then continued to formalize BT node execution semantics in Milner Calculus. Consequently, we could prove that there exists a semantic preserving mapping between BTs and CCS process notations. This finding allows to transfer formally specified Multi-Agent Systems into

executable equivalents, while ensuring to maintain execution semantics as notated in the formal calculus. For future work, we plan to investigate means to automatically compile SPARQL-BT from formal CCS notations. Conversely, we are interested to examine possibilities of automated model checking of agent behavior described as BTs, by transferring respective approaches for CCS processes to BTs, while exploiting the semantic preserving mapping as presented in this Section.

The application of the concepts will be demonstrated in Section 7.3 in the scope of a cyber-phyiscal production scenario in an Industry 4.0 setting. The definition and implementation of that use-case was as well published in the scope of "Behavior Trees as executable representation of Milner Calculus notations". [235]

Chapter 7

Application and Evaluation

This chapter will demonstrate the application of the previously presented solutions in actual use case scenarios. The scenarios are chosen to show how a proper agent-environment interaction via Linked Data media achieve both *coordination* of a crowd of agents, as well as *optimization* of agent behavior.

All of the presented scenarios intend to provide examples of how to employ the concepts of reactive Linked Data agents, and (dynamic) Linked Data media, to achieve the desired results. Sections 7.2, 7.3 and 7.4 will for this cover scenarios that require *multi agent coordination* in the domain of cyber physical production.

Among the coordination scenarios, Section 7.2 employs a dynamically lifted simulation using ECA2LD (cf. Section 5.2), as underlying technology to implement the dynamic Linked Data environment, specifically the Unity 3D implementation as described in Section 5.2.6. The use case in Section 7.3 employs a Linked Data Platform server (see also 3.1.2 as medium to provide the environment, and implements hysteretic agent swarms using AJAN behavior trees as presented in Section 6.2. The use case presented in Section 7.4 is implemented against the reactive `stigLD` server component, as presented in Section 5.1.

The optimization scenario in Section 7.5 does not require environment dynamics, and is therefore implemented using Apache Fuseki[79] as Linked Data endpoint. Section 7.6 uses the `stigLD` server component for dynamic environment evolution, as presented in Section 5.1.

Each of the individual Sections will first outline the chosen scenario, and give a detailed description of how the respective scenario is modelled and published in a Linked Data medium. Each scenario will moreover provide a detailed definition of the agent models, and their interaction with the environment. Finally, each Section will define evaluation metrics for the presented scenario, and show efficacy and performance of the demonstrated solutions by evaluating them against the chosen metrics.

[79] `https://jena.apache.org/documentation/fuseki2/` *Visited May 2023*

7.1 USE CASE 1: Cyber-physical airplane assembly

Linked Data as medium, provided from real-time simulations using ECA2LD in both its standalone and Unity3D implementation (cf. Section 5.2), in conjunction with hyper-media agents modelled by AJAN behavior trees (cf. Section 6.2), have been successfully applied to implement a series of applications from the domain of cyber-physical airplane assembly [157, 6, 7].

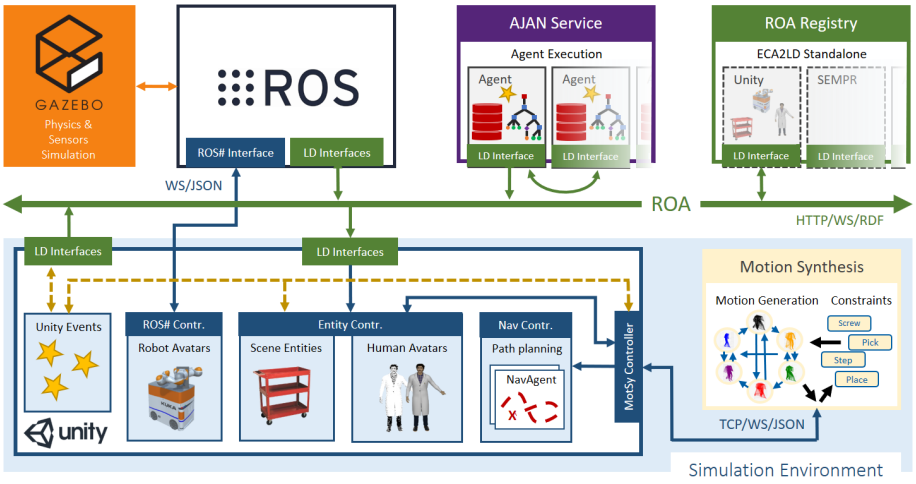

FIGURE 7.1: High level architecture of applications from the domain of cyber-physical airplane assembly: The Linked Data medium ("ROA", "Resource Oriented Architecture") serves as integration layer for connected services. (Image source: [6]

The general architecture of the use-cases is reflected in Figure 7.1 (originally published in [6]). The Linked Data medium (*"ROA"* in the diagram, for "Resource Oriented Architecture") serves as representation and interaction point for a variety of connected applications. The selection of applications varied depending on the implemented scenario, with Fig. 7.1 showing the selection of tools of the wing assembly scenario (see [6], and Section 7.1.1).

Within the domain, use-cases from different scenarios were implemented: human-robot collaboration for wing assembly [6] (see Fig. 7.2), and coordination (Fig. 7.3) [7] and execution [157] (Fig. 7.4) of Cobot [195] supported quality control during the process of hull construction.

7.1.1 Wing Assembly

The first use-case implemented coordination of robots to support human workers in ergonomically challenging over-head tasks during the installation of cable raceways in airplane wings [6] (see Fig. 7.2).

The system employed a number of established robotic programming, simulation, and visualization tools, such as the ROS robotic operation system, and Gazebo robotic simulation, with the goal that designers of robotic systems with knowledge of the

respective systems could use the resulting framework to evaluate worker tasks and possible support by robotic units with respect to ergonomics of the specific tasks.

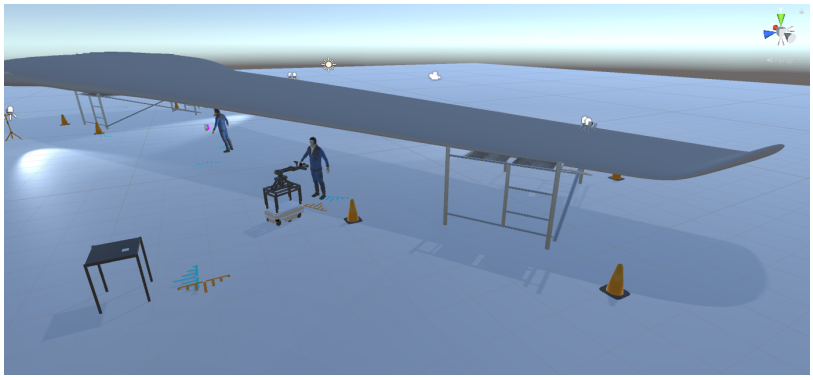

FIGURE 7.2: Simulation of raceway installation task by a team of human and robot workers (originally published in [6])

In the context of the different roles of the Linked Data medium as elaborated on in Section 4.1, the medium served as representation and (indirect) interaction space for the connected software systems, whereas environments artifacts were managed by Unity3D and ROS instances.

The Real-time access to entities in the environment simulations was established via the HTTP and publish-/subscribe protocol as presented in Section 5.2.5.

7.1.2 Coordination of Cobot supported quality control

The second use-case covered installation of fortification stringers in the airplane hull, particularly the installation and quality inspection of fixation rivets before applying sealant, and performing the sealing as final step.

In the implemented scenario in [7], crucial steps are carried out by human worker who are supported by Cobot units, with the Cobot units performing a first quality check of installed rivets after the human worker finished his work (see Figure 7.3). Detected faults are then reported to a second worker for a closer inspection. As soon as closer inspection is finished, the second worker marks the respective rivets as "OK", which serves as signal to a second Cobot to apply the sealant and finish installation of the particular rivet.

The system presented in [7] covered the coordination of the Cobots using a high performance optimizer to align the Cobot programs with the steps being carried out by the worker. Autonomy of the human workers to keep full control of the scenario was a key challenge.

The environment was simulated using Unity3D as game engine. Entities simulated by Unity3D were published to the Linked Data layer using the Unity3D ECA2LD extension (cf. Section 5.2.6). The planner operated on the environment representation

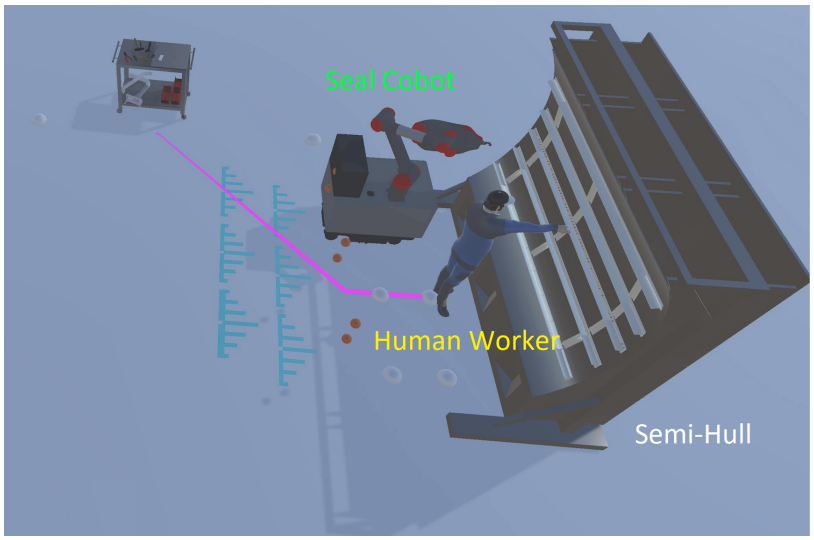

FIGURE 7.3: Simulation of of quality check of rivet sealant application during fortification stringer installation (originally published in [7])

as served by the Linked Data medium, while interaction with simulated workers and Cobots was provided via the medium by the protocols as described in Section 5.2.5.

The resulting scenario simulation allowed to evaluate various criteria, e.g. time to fulfill a specific task sequence, reachability of positions and objects, or ergonomic assessment of poses based on visual appearance with respect to different body sizes.

7.1.3 AR support for Cobot supported quality control

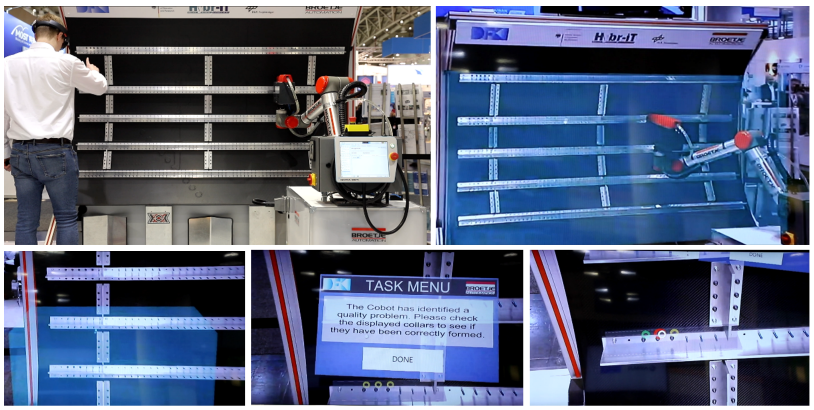

FIGURE 7.4: AR supported quality check with human-robot collaboration. (Originally published in [157])

The third demonstrator focused on the interaction between human workers and Cobots. In the presented scenario, the initial quality check by the Cobot unit was visualized towards the human worker using a Holo Lens AR device (see Fig. 7.4). Once the worker checked the state of highlighted rivet units, they marked them as ready for sealing via the AR via hand gestures [157].

The scenario employed the same Linked Data based architecture as the previous two use-cases. The rivets were organized in several levels of detail, ranging from hull, over hull segment, to fortification stringer, and finally the single rivets. The LoD areas and rivets were in this scenario hosted using an ECA2LD based standalone server (cf. Section 5.2.6).

The Linked Data medium, provided as ECA2LD standalone solution, handled 45 stringer frames in 3 hull regions with 810 rivet instances in total, resulting in a number of 3964 resources, encoded in a total of more than 23.000 RDF triples. SPARQL query response times were in the range of 200ms to up to 800ms.

7.1.4 Summary

The presented use-cases demonstrated the use of a Linked Data medium as interactive representation and interaction space for various connected heterogeneous simulations and AI systems. While the presented scenarios did not yet cover multi-agent based coordination and optimization, the suitability of Linked Data as real-time medium for interactive environment representation could be convincingly shown, showcasing the technologies presented in the scope of this thesis.

7.2 USE CASE 2 (Coordination): Shopfloor scheduling

The following application scenario was originally published in [237]. It demonstrates how to employ a read-write Linked Data layer as stigmergic medium in the agent space by an application example from the domain of digital manufacturing. The example is loosely based on the use case presented in [218]: A (simulated) factory receives orders for simple IoT modules on a "batch size 1" production line as commonly envisioned in Industry 4.0 [146, 180]. Once an order is received, it is carried out by employing machines that provide the capabilities to perform manufacturing steps necessary for particular steps during the production process, e.g. providing plastic casts for casings, soldering electric circuits, or fixing the final model (see Figure 7.5).

Orders are executed by AI agents. The need for coordination arises as machines are shared between simultaneously executed orders.

The purpose of the presented coordination algorithm is to find a suitable distribution of machines between agents working on different orders with the goal to complete each order in the shortest possible time.

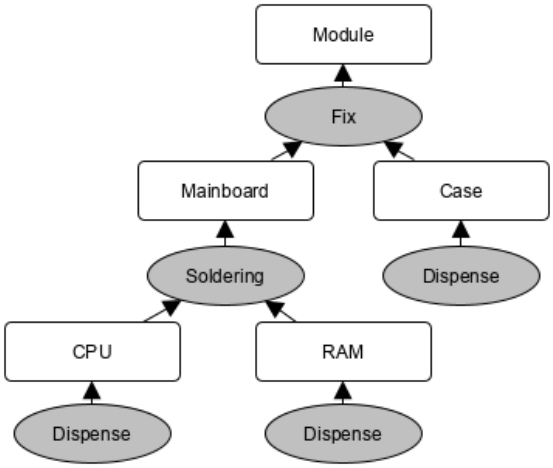

FIGURE 7.5: Process of IoT module production used as example.

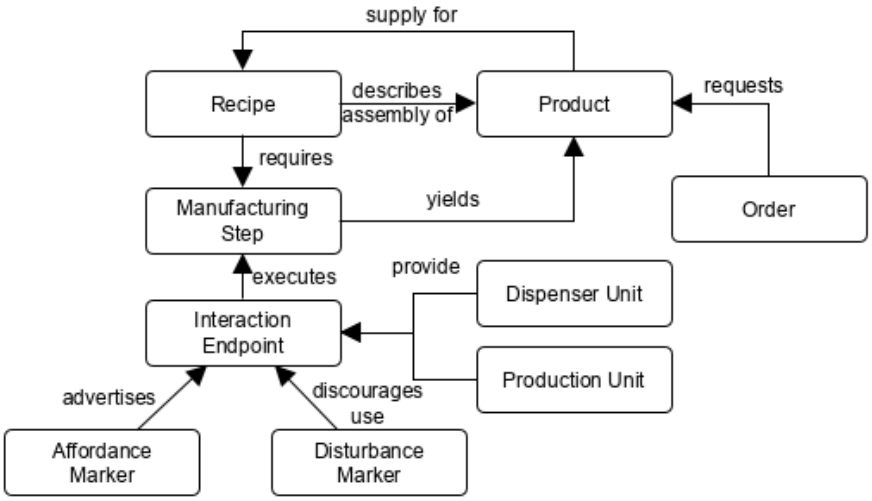

FIGURE 7.6: Domain model of the chosen application example.

7.2.1 Domain Model

The domain model for our application example is shown in Figure 7.6: An *order* requests a certain product to be produced. How a product can be assembled is described in *recipes*. These recipes specify what other products are needed as prerequisite, and which manufacturing step is necessary to assemble supply products to a higher level product. Manufacturing steps are provided by units on the shop floor, and can be executed by network interaction endpoints. As common in models for automated production, and also assumed in ([43]) and ([218]), we assume production units to have callable network endpoints by which their particular production step can be executed. Affordance- and disturbance markers are used to encourage or discourage the use of a certain interaction endpoint.

```
recipes:main-module rdf:type schema:HowTo ;
  schema:about mainboard:product ;
  schema:step steps:solder ;
  schema:supply [
    rdf:type schema:HowToSupply ;
    schema:item cpu:product ] ,
  [ rdf:type schema:HowToSupply ;
    schema:item ram:product ] .
```

LISTING 7.1: Example of a production recipe using schema and steps vocabularies.

A recipe specifies the produced artifact by the `schema:about` predicate, the required supply from which the product is created (indicated the `schema:supply` predicate), and the production step that needs to be performed to combine the specified supplies to the resulting product via the `schema:step` predicate (see Listing 7.1). `schema:` denotes the namespace of the schema.org ontology [80]. We assume a set of supply materials to be provided to the factory without the need for specific production. These supply materials will be provided by dispenser units, and do not require any additional supplies.

Products are described in terms of an RDF class, e.g. `(<#product>, rdf:type, cpu:product)`.

Dispenser and production units specify their (callable) execution endpoint as `td:InteractionPattern` in a set of triples that is referenced via `td:providesInteractionPattern`. The Interaction Pattern specifies the step carried out by the respective unit (see also Listing 7.2). Dispensers refer to the class of dispensed products via a triple `(<#unit>, schema:yield, <#productClass>)`, with `<#productClass>` referring to the RDF class of the produced product. Dispenser units can dispense more than one class of products. Production units do not specify particular products that are produced at the unit to allow for various products that are produced with the same step to be produced at the same machine. Instead, they provide information about the type of provided production step via a triple `(<#unit>, schema:step, steps:<type>)`. An example of a simple soldering unit is shown in Listing 7.2.

Interaction Patterns on machines describe particular *actions* that agents may perform by to trigger the execution of the respective production step on the physical machine. This is done by resolving the URI that is provided by the respective resource, and that is identified by the property path `td:isAccessibleThrough/td:href`, with `td:` denoting the namespace of the Web Thing Description ontology [81].

[80] `https://schema.org/` *Visited May 2023*
[81] `https://www.w3.org/2019/wot/td` *Visited May 2023*

```
sol:station-1 a td:Thing ;
  td:thingName "Soldering Station 1"^^xsd:string ;
  td:providesInteractionPattern sol:soldering .

sol:soldering a td:InteractionPattern ;
  td:interactionName "solder"^^xsd:string ;
  schema:step steps:solder ;
  td:isAccessibleThrough [
    td:href <http://10.2.100.17/solder/>
  ] .
```

LISTING 7.2: Example of a simple description of a workstation that performs a soldering step. The soldering action is executed by calling the respective referenced URI.

Affordances and Disturbances

```
<urn:uuid:a526>
    a stigmergy:marker ;
    stigmergy:marked  sol:soldering ;
    stigmergy:scope  order:module ;
    schema:supply  cpu:product ,
                   ram:product ;
    schema:yield  mm:product .
```

LISTING 7.3: Example of an affordance marker resource that advertises a `steps:soldering` interaction as relevant for the current order

Affordances will advertise `td:InteractionPattern` resources as callable endpoint to some executing agent. Affordances are *markers* that are left on a `td:InteractionPattern`.

Listing 7.3 shows an example of such a marker: The marker gives information about which Interaction Pattern it marked (via `stigmergy:marked`), for which order the respective pattern needs to be executed (via `stigmergy:scope`), whether or not the respective step needs particular supplies to be present to be executed (`schema:supply`), and finally, which product will be the result of calling the respective InteractionPattern resource (`schema:yield`). A marker can link to one or more interaction patterns. If more than one interaction pattern is marked, it is up to an executing agent to choose which of the endpoints to call.

Disturbance markers will discourage agents from visiting a marked resource. If an affordance marker links to several resource endpoints, an executing agent will decide for an endpoint that is the least influenced by disturbance markers.

7.2.2 Agent models

We now show how to realize a stigmergy-based coordination algorithm based on the definition of the Linked Data layer from the previous section.

Marker agents will employ *affordance markers* to advertise and encourage interaction with an endpoint in the Linked Data layer to agents. When interacting with afforded resources, builder agents will leave *disturbance markers* to discourage agents from further interaction with the marked endpoint to avoid the same resources being called by agents disproportionally often. By the employed stigmergic principles, the approach is intended to find a balanced, correct production in an online fashion, during execution.

The algorithm to execute a coordinated production process for multiple orders is implemented in two steps by two classes of agents. We divide agents into *marker* agents and *builder* agents. Marker agents traverse graphs in the agent space and generate *production markers* as affordances on resources in the Artifact space as shown in Listing 7.3. Builder agents are attracted towards the respective endpoints by the affordances left by the marker agents and execute those production endpoints that were marked in the scope of the current order, given that the production requirements (supplies) are met.

Marker Agents

The goal of a marker agent to identify all *suitable* production units that will be involved in the process of producing a particular order. For this, marker agents will traverse recipe resources and leave affordance markers on resources as follows:

The agent maintains a list `unvisited` of nodes it would like to visit, but has not yet.

1. Check for *order* resources that have not yet been handled, i.e., do not carry a mark. Follow the link via the `schema:orderedItem` property to the resource that represents the class of the ordered product and add it to `unvisited`. Mark the order as *handled*.

2. From a resource *r* in `unvisited`, find a respective recipe blueprint *b* that contains a triple (*b*, `schema:about`, *r*), i.e., the recipe for the respective product.

3. Check for a `schema:step` link, and *visit all interaction patterns i* matching the `schema:step`; if the step is `steps:dispense`, find the respective interaction patterns of dispenser units that `schema:yield` *r* .

4. Leave an *affordance marker* on each visited *i* (for both production and dispenser, cf. Listing 7.3).

5. For each resource *s* in `schema:supplies` of *b*, add *s* to `unvisited`. If no `schema:supply` is specified, or resource points to an empty set (`rdf:nil`), do nothing. Remove the current resource *r* from `unvisited`.

6. If `unvisited` is empty, **terminate**; else, **go to 2**.

The mark which is left by the agent in Step 4. follows the structure of the example shown in Listing 7.3. It includes information about the order in the scope of which it was placed, and will moreover specify the required supplies *s* for this step. Markers may be scoped by order, using a triple (*marker*, `stig:scope`, *order*) on the marker resource (*Narrowcast*), or unscoped and by this visible for every other agent (*Broadcast*). Following the given algorithm, a marker agent is solely driven by the structure of the knowledge graph that is formed between product and blueprint descriptions. Each subsequent step is solely decided by the state of the currently visited resource. Its behavior can by this be classified as a *sematectonic stigmergic* agent (cf. Section 2.3, [117]).

Builder Agents

Builder agents are attracted to markers left by marker agents and call the respective `InteractionPattern` endpoints. A Builder agent for this proceeds in the following manner:

1. Scan for all markers *m* left by marker agents. If the builder agent is bound to a specific scope (i.e. fulfilling a particular order), it will only follow markers in its scope (i.e., with a matching (*m*, `stig:scope`, *order*) triple present.
2. For each *m*, the checks, e.g. via a fitting SPARQL query, if for each *supply s* specified by the marker via (*m*, `schema:supply`, *s*), there exists a product *p* that is a product of class *s*, as encoded by a triple (*p*, `rdf:type`, *s*).
3. For each *m* for which supplies are fulfilled, visit the InteractionPattern resource *i* that is marked via (*m*, `stig:marked`, *i*) and that carries the *least amount of disturbance markers*. Execute the action endpoint that is identified via `td:isAccessibleThrough/td:href`.
4. Leave a disturbance marker on the interaction pattern resource and removes the affordance marker.

7.2.3 Discussion

In the scope of the original publication [237], the use-case was intended to demonstrate the proper materialization of stigmergic self-coordinating principles. The algorithm was therefore against two criteria: first, does it implement stigmergic principles, as defined in the original publication. Second, does the algorithm ultimately lead to a correct result.

Correctness of the Algorithm

An order specificies the expected result of one instance of the algorithm, namely the specific product that is to be produced in the end. By marker agents starting from

the expected goal (the ordered product), and following links backwards through the needed supplies, it is ensured that over the total production process, all needed supplies will be available eventually. Marker agents do not need to keep memory of the goal they are following, but are guided entirely by the structure of the Linked Data medium. It can be easily shown that the marker agent's algorithm will terminate as soon as all dispensable supplies (leaf nodes in the graph in Figure 7.5) are provided with a marker. The builder agent's algorithm will terminate when the last marker is consumed. Builder agents are not restricted to follow only markers of a specific order. Consuming a marker and triggering the respective action will always result in a product that was previously found to be a requirement by some marker agent. Eventually, by builder agents executing endpoints for products with rising complexity as supplies are more and more met, the ordered product will be produced.

If several orders are executed in parallel, production units (and dispensers) will simply receive several independent markers. By having separate markers per order, and having builder agents removing the marker they followed after executing the production step, it is ensured that for every order, every production step is executed exactly once. The concept may be extended for products to require more than one instance of a supply product. In this case, a marker agent would leave a marker per required instance of a supply.

The opportunity for coordination arises in Step 3. of the builder agent algorithm: For every recipe, markers are left on every machine that is capable of carrying out the needed production step. When following the marker trace, builder agents have a choice which of the marked machines they actually execute. The decision for which machine to call for to execute the step is based on the number of disturbance markers left on the resource: The more agents visit, means, the busier the machine already is with executing orders, the more disturbance markers are left on the machine, and agents will be more likely to divert to other machines to complete their order.

The algorithm could further be extended to take into account failures of production units. Builder agents that observe a failure, for example by requests to an executable endpoint timing out, receiving error responses from the endpoint, or by not observing the expected result of their actions for a while, would leave a respective information on the faulty resource which inhibits other agents to consider the endpoint entirely, and highlight the unit represented by the marked resource for troubleshooting to maintenance staff.

The algorithm at this point ignores transport of products on the shop floor. A more sophisticated heuristic may take into account also transport times between machines between the different steps.

Implementation of Stigmergic Principles

In the presented algorithm, several agents with different competences (marker and builder agents) jointly solve the given problem. These agents react both to markers left by other agents, as well as to direct results of their own work: the supply products that become avaialble in the process of production.

The present approach by this implements *collective stigmergy* in a stigmergic system with *marker-based* and *sematectonic* elements.

In the following, we analyse the use of Linked Data as stigmergic medium based on findings from the algorithm w.r.t. benefits of stigmergic systems (see also ([115, pp.13-14])):

In the presented scenario, agents do *not plan or anticipate*, but only follow links between resources in the Linked Data medium. The condition-action-rules that determine which resources to visit are generic. Agents do not need to make per-resource decision whether following the rule is beneficial for the goal, or not. The "goal" (production of a specific order) is, in particular, not known to an agent, and reaching any goal is no condition for termination of the algorithm.

Memory-less agents are sufficient by storing all relevant information that arises during execution in terms of resources in the medium. Same goes for *communication between agents*, which is eliminated by limiting interaction to following markers left by other agents. Agents are moreover not *aware* of each other, as their interaction is limited entirely to the Linked Data medium. This also implies that agents *do not need to be simultaneously present*.

The correct *sequence* of steps arises naturally by including information about required supplies, both when a marker agent decides which recipe resources to visit next, and when a builder agent decides which marked production resource to visit next. There is no requirement to explicitly model (and by this, impose) sequences during execution of a particular production in a form like *"after you have successfully visited this resource, continue with that specific resource over there"*. Instead, sequence arises implicitly from the condition-action rules encoded in the Linked Data medium.

Non-necessity for commitment is achieved by having no explicit assignment of tasks to agents, but have agents decide which resource to visit, and how to interact with it (e.g., perform their competent action), solely on the state of *resources in the medium*. Any agent can pick up any task at any point in time according to the agents' competence.

Finally, as it is obvious from the algorithms of both marker and builder agents, that there is *no centralized coordination or control* authority that agents need to consult, or by which they are controlled. Coordination arises solely from resource states and markers left in the medium, as discussed above.

7.2.4 Implementation

We implemented the example using the Unity 3D game engine to simulate the factory, a Fuseki triple store to host the read-write Linked Data medium, and the AJAN agent platform[82,83] ([6, 7]) to implement the behaviors of both marker and builder agents. All related resources will be published on GitHub:
`https://github.com/dfki-asr/stigmergy-demo`

7.3 USE CASE 3: Shopfloor Scheduling II

In the scope of [235], the previous use-case (Section 7.2) was adapted to demonstrate the formal definition of agent models in terms of Milner CCS expressions and their respective implementation in semantically equivalent behavior trees, as described in Section 6.2. The contents of this section have originally been published as parts of the paper "Behavior Trees as executable representation of Milner Calculus notations" [235].

The use-case defines and implements different agents for a self-coordinating shop floor scenario: A virtual factory shop floor provides production units to produce certain products. Each type of production unit is capable of producing one or more different product kinds. Products assembled from two or more products, also being produced within the same factory, or dispensed by dispenser stations in the factory. The assembly instructions of higher level products are provided in terms of blueprints that specify which products need to be provided as supply to assemble the higher level product.

The shop floor, manufacturing units, product blueprints, and products available on the shop floor are represented as Web resources, following the Linked Data Platform[84] specification (see also Fig. 7.7). Each resource provides an RDF description of its represented entity. For Workstations, this description includes a reference to a callable endpoint by which the workstation can be triggered to produce a product according to its capabilities.

7.3.1 Agent Models

We want to tackle the problem in a self organized approach to find a solution in an online fashion that is adaptable to changes of scenario during execution, as it has been discussed for self organized scenarios.

From the generic description for hysteretic agent swarms in Section 6.1.2, we model three hysteretic agent models, namely *Order Handling Agent*, *Job Scheduling Agent*, and *Job Execution Agent*, by which the described scenario is handled.[85]

[82] `https://github.com/aantakli/AJAN-service` *Visited May 2023*
[83] `https://github.com/aantakli/AJAN-editor` *Visited May 2023*
[84] `https://www.w3.org/TR/ldp/` *Visited May 2023*
[85] Detailed formal models of `JSA` and `JEA` are omitted for sake of brevity.

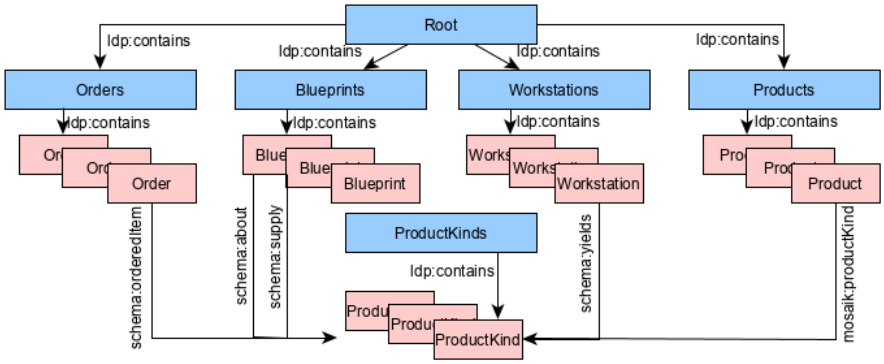

FIGURE 7.7: Hierarchical structure of Linked Data Platform container resources (blue), the contained entity types (red), and their semantic relations in terms of RDF links.

Order Handling Agent:

Order Handling Agents (`OHA`) are initially spawned on the *"Orders"* collection resource. During `PERCeption`, they update their link set to contain references to any `schema:Order` resource in the order collection that is not yet annotated as being processed (namespace "`schema`" abbreviated as `'s'`), and relocates to one of them:

$$L'' = sel(q^{\texttt{OHA}}_{\texttt{NAV}}, G'') \equiv L'' = \{?j | (?\texttt{j} \texttt{ , } \texttt{s} : \texttt{orderStatus} \texttt{ , } \texttt{s} : \texttt{OrderProcessing}) \notin G''\}$$

Upon visiting a `schema:Order` resource j, the agent, as `REACTion`, marks the order as in progress (namespace "`schema`" abbreviated with `'s'`):

$$\texttt{ACT}_{\texttt{OHA}}(j, G'', L'') = req_{\eta(j)}(\texttt{POST}, j, \{(?\texttt{j} \texttt{ , s} : \texttt{orderStatus} \texttt{ , s} : \texttt{OrderProcessing})\}) \text{ ,}$$

and spawns a `JSA` on the blueprint collection resource. The initial knowledge of the `JSA` specifies the `ProductKind` that is assembled by the blueprint j currently visited by the `OHA`:

$$\begin{aligned} \texttt{SPAWN}_{\texttt{OHA}} &= \texttt{SPAWN}(\texttt{JSA}, K^i, L^s), \text{with} \\ K^i &= \{(\_ : \texttt{self}, \texttt{schema} : \texttt{orderedProduct}, ?p) | (?\texttt{i} \texttt{ , schema} : \texttt{about} \texttt{ , } ?p) \in G''\}, \\ L^s &= \{?\texttt{j} | (?\texttt{j} \texttt{ , a , } : \texttt{BlueprintContainer})\} \end{aligned}$$

After acting on an order resource, and spawning the respective `JSA`, the `OHA` relocates back to the order collection resource.

Job Scheduling Agent:

Job Scheduling Agents (`JSA`) are spawned on the collection of all product blueprints, with their initial knowledge containing the ProductKind p for which it is supposed to schedule production, i.e. $K = \{(\_ : \texttt{self}, \texttt{schema} : \texttt{orderedProduct}, p)\}$. Upon `PERCeption` of the blueprint collection, it will add to its link set the resource within the collection that is described

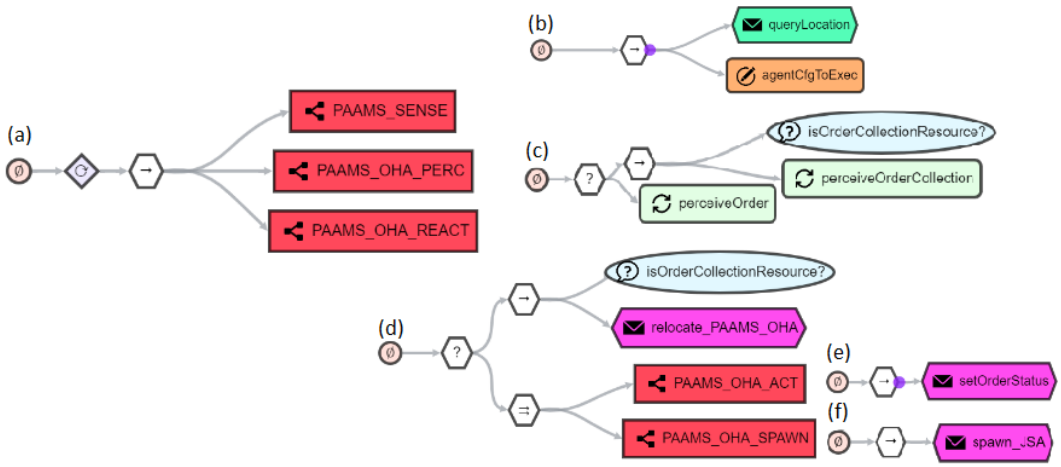

FIGURE 7.8: Behavior trees modelling the processes of the Order Handling agent: Agent as Sense, Perception, Reaction sequence (a), sensing via `QUERY` nodes (b), perception using `UPDATE` (c), and reaction (d) as either relocation (by re-initiating the agent using `MESSAGE`), or parallel resource interaction (e) and spawn action (f).

to `schema:yield` the requested `ProductKind`[86], and relocates to this resource.[87]

During `REACT`ion, it `SPAWN`s for every required `schema:supply` an offspring of Job Scheduling agents with initial knowledge $K^i = \{(\_ : \texttt{self}, \texttt{schema} : \texttt{orderedProduct}, s)\}$, and moreover a `JEA` (see below) with initial resource i the container of all Workstation resources, and initial knowledge $K = \{(\_ : \texttt{self}, \texttt{schema} : \texttt{orderedProduct}, p)\}$, i.e., the ordered `ProductKind` for which the `JSA` was created.

Job Execution Agent:

Job Execution Agents (`JEA`) are spawned on the Workstation container resource, with initial knowledge containing the `ProductKind` they are supposed to produce, and the `schema:supply` that is required to successfully assemble the ordered product. During `PERC`eption, the `JEA` will update its link set to contain all Workstation resources that are described to `schema:yield` the requested product, of which the specified `schema:supply` is fulfilled[88], and relocates to one of them. While located on a Workstation resource, the agent will identify, during `PERC`eption, the callable workstation endpoint, and issue a request to produce the required product during `ACT`.

7.3.2 Implementation

We implemented the above defined agent models as AJAN SPARQL-BTs. Fig. 7.8 shows the set of BTs that we derived from the `OHA` model. By understanding BTs as

[86] We make use of the SPARQL `SERVICE` keyword to include linked resources into the perception of a local resource via federated queries.

[87] We at this point assume that for each product, there is only one blueprint that describes assembly of a product.

[88] At the current state of the algorithm, we abstract from location information, and assume any present product of a fitting kind being ready for input as `schema:supply`

(labeled) processes, mapping operators to composite nodes, as shown in Sec. 6.2.5, and mapping request actions to the respective leaf node types for SPARQL-BT as described in 6.2.3, we could successfully transfer all 3 agent models to semantically equivalent behaviors. The BTs are published on GitHub, along with all necessary resources to run the application example[89].

7.4 USE CASE 4 (Coordination): Make to order pickup and delivery

The second evaluation use-case demonstrates how to apply tropistic agents in Linked Data Media to a Make-to-Order (MTO) fulfilment process from the production domain. This use-case was originally published in [220].

As for the algorithm showcased in Section 7.2, the presented approach is applicable to problems that target a "batch size one" approach [146, 180], this time covering shop floor logistics, rather than job scheduling.

7.4.1 Problem Definition

Let us consider a shop floor area that is represented by a discrete grid; in each grid cell is a shop floor location and can accommodate a single production resource (see also Fig. 7.9. We distinguish between three types of production resources: machines, output slots assigned to individual machines and transporters.

Machines produce a product of not further specified kind in response to a confirmed order received for it from a final customer. Whenever a machine finishes production of a product, the product is placed into an output slot awaiting pickup by a transporter unit. *Output slots* have limited capacity. If any of the output slots are full, the associated machine cannot produce any new products until the output slot is emptied by the transporters. *Transporters* are initially situated in idle locations spread throughout the grid; they can move to any unoccupied location within their respective Manhattan distance neighbourhood. Their task is to pick up finished products from the output slots of machines, so that production can go on without significant interruptions.

A visualization of the shopfloor model is shown in Figure 7.9. The visualization displays the 10 by 10 grid, with cells labeled with their respective identifiers in the range from 1 to 100, machine units labeled as *M*, and transporter units labeled as *T*, respectively. Green colored cells encode the presence of transient and diffusing semio-chemical markers, with color intensity encoding marker concentration on a particular grid cell (cf Section 7.4.3).

[89] `https://github.com/dfki-asr/wiait22-demo` *(Visited May 2023)*

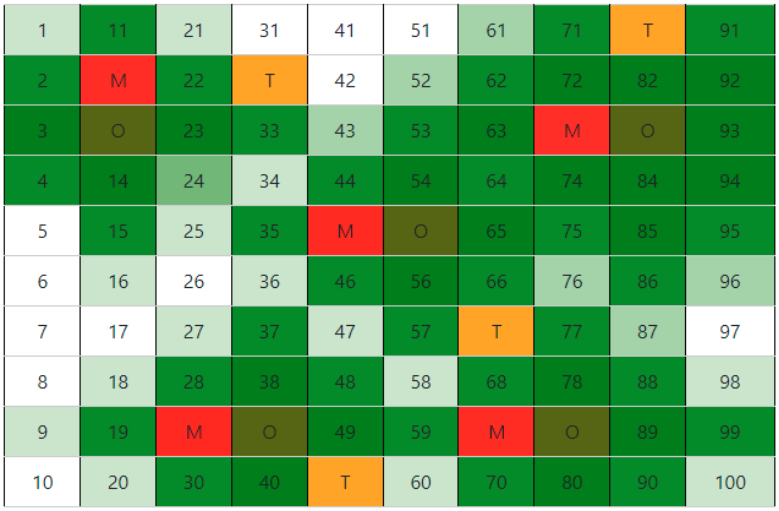

FIGURE 7.9: Visualisation of the shopfloor environment of the make to order usecase with spreading markers to attract transport unit.s

The shop floor will continuously receive new customer orders. Whenever a new order is received by the simulation, the order is entered into a queue of one of the machines of the shop floor. Producing the product in the machine once the order has moved to the head of the queue takes a fixed amount of time. We aim to coordinate the MTO fulfilment process such that customer orders should be assigned to machines in such a way that the overall machine work load is balanced, and *mean times to deliver* – the time from start of production to delivery of the finished product – should be minimized. More specifically, we are interested in improving the following metrics

- average number of steps moved by the transporters
- average maximum and minimum machine loads
- deviation in maximum load experienced by machines
- average time between start of production of a product until pickup by a transport unit

All material needed to set up and run the example were provided online[90] along with an interactive demo instance[91] in the scope of the original publication.

7.4.2 Shop Floor Representation in StigLD

In our example, the `stig:Medium` represents the overall shop floor area as a 10x10 grid of `stig:Topos` instances. Neighborhood relations depend on the type of agent that is exploring the medium (see also Section 7.4.3): For transporter agents that navigate the shopfloor, each `st:Topos` links via `stig:adjacentTo` predicates to the

[90] `https://github.com/BMBF-MOSAIK/StigLD-Demo`
[91] `http://mosaik.dfki.de`

`stig:Topos` instances in its Manhattan distance neighborhood. Order assignment agents ignore spatial information, and consider all topoi that carry a machine unit as mutually connected. Production resources are assigned to their individual `stig:Topos` instances using `stig:locatedAt` link predicates; the Transporters' idle locations – the grid cells to which they return after having finished a pickup – are given by `ex:idlePosition` link predicates.

7.4.3 Agent Models

We employ *marker-based stigmergy* with *transient* semio-chemical marker models to achieve the desired coordination. For this, we employ two types of agents: one type assigns open orders to available machines on the shop floor, the other controls transport units.

Order Assignment Agents: Transient stigmergy based on linear decay

The task of Order Assignment Agents (`OAA`) is to ensure equal distribution of production workloads over machine units on the shopfloor. To this end, order assignment agents will assign newly received orders to machine units, and place a *negative feedback* marker on the topos that accommodates the respective machine. If multiple feedback markers are assigned to a single machine, their concentration will be added. Consequently, the more products are assigned to a single machine, the higher the concentration of negative feedback markers will be. For subsequent orders, Order Assignment Agents will prefer machines with the *least concentration* of negative feedback markers. By negative feedback markers decaying linearly over time, machines that received markers will become more attractive again the more time passes.

Following the original paper [220], using the notion of tropistic stigmergic agents from the same publication, the behavior of an Order Assignment Agent `OAA` can be specified as follows:

For an open order, an order assignment agent $\texttt{OAA} = \texttt{PERC}(i, G = \emptyset, L = \emptyset)$ is placed on a randomly chosen topos i that is accommodating a machine.

During situated perception, the agent will first update its set of perceived topoi by a perceptional query $q_{\texttt{PERC}}$ (cf. Eqn 6.1). It will then proceed to relocate to one the topos that, among the perceived topoi, accommodates a machine unit, and has the lowest concentration of negative feedback markers This situated perception is described in [220] as:

$$(G'' = \texttt{ans}(q_{\texttt{PERC}}, G')) \quad \equiv \quad (\forall t \in G' \Rightarrow t \in G'')$$

$$(L'' = \texttt{sel}(q_{\texttt{NAV}}, G'')) \quad \equiv \quad (L'' = \{j \,|\, \underset{j}{\operatorname{argmin}} \begin{pmatrix} \texttt{<j>} & \texttt{stig:carries [} & \texttt{stig:level ?val;} \\ & & \texttt{a ex:NFMarker];} \\ & \texttt{\^{}(stig:locatedAt)} & \texttt{[a ex:Machine].} \end{pmatrix}\})$$

When selecting its reaction (cf. Eqn. 6.2)

$$\texttt{REACT}(i, G, L) = \textbf{if } i \notin L \textbf{ then } \texttt{PERC}(j \in L, \emptyset, \emptyset) \textbf{ else } \texttt{MARK}(i, G, L)$$

the agent `OAA` will either (i) re-situate to a topos with lower concentration of negative feedback or (ii) leave a *negative feedback marker*[92] on its current topos:

$$\begin{aligned} \texttt{MARK}(i, G, L) &= \overline{req}_{\eta(i)}(\texttt{PUT}, i, \texttt{ans}(q_{\texttt{PUT}}, G)).res_{\eta(i)}(rc, (\emptyset, \emptyset)).\mathbf{0} \\ \texttt{ans}(q_{\texttt{PUT}}, G) &\equiv \texttt{descr}(i, G) \cup \{\texttt{<i> stig:carries [a ex:NFMarker; stig:level 1.0].}\} \end{aligned}$$

Negative feedback markers will decay linearly over time; the system's endogenous dynamics with respect to negative feedback markers is given by Eqn. 4.1 with

$$\texttt{ans}(q_{\texttt{EVO},G}) \equiv \begin{pmatrix} \texttt{?i stig:carries [a ex:NFMarker; stig:level ?c; stig:decayRate ?d].} \\ \Downarrow \\ \texttt{?i stig:carries [stig:level stigFN:linear\_decay}(\Delta\texttt{t, ?d, ?c)].} \end{pmatrix}$$

Leaving a negative feedback marker *inhibits* future selection of a machine, and increases the likelihood of balancing machine workloads during the MTO process.

Transporter Agents: Transient stigmergy based on diffusion

Transporter agents `TA` steer transport units on the shopfloor. Their task is to collect finished products that are waiting for pickup in machine output slots. Transporters have to find a trade-off between minimizing mean times to deliver, and minimizing steps of transporter units on the shopfloor: Picking up products early may result in a short time to deliver for the particular product. However, performing an individual pickup run for every product results in high number of transporter steps, and potential delay when transporters have to start a new pick up run from their idle positions to the machines for later products. Conversely, waiting for the output slot to fill up and to collect a number of products at once reduces total move distance of the transporters, but may result in long delivery times for single products if they have to wait in the output slot for longer time.

To accommodate for both constraints, `TA` follow traces of *Transportation Markers (TMarkers;* represented as `ex:TMarker` in the Linked Data medium), which over time diffuse spatially. TMarkers are generated on machine output slots whenever a finished product wanders into the respective slot. Following immanent medium updates as described in Section 4.3, the temporal updates of spatial diffusion are triggered by client `GET` requests, and are implemented as evaluation of an evolutional query $q_{\texttt{EVO}}$, with $\texttt{ans}(q_{\texttt{ECVO}})$ as:

[92] – as well as a production task into the respective machine's task queue –

$$\texttt{ans}(q_{\texttt{EVO},G}) \equiv \left(\begin{array}{l} \texttt{?i stig:carries [a ex:TMarker; stig:level ?c;].} \\ \qquad\qquad\qquad\qquad \Downarrow \\ \texttt{?j stig:carries [a ex:TMarker;} \\ \qquad\qquad\qquad\quad \texttt{stig:level stigFN:diffuse1D(} \\ \qquad\qquad\qquad\quad \texttt{?i, stigFN:dist\_manhattan(?i, ?j), ?c, } \Delta\texttt{t} \\ \qquad\qquad\qquad\quad \texttt{)].} \end{array} \right)$$

By the chosen marker model, we achieve the aforementioned trade-off between shorter times and waiting times as follows: The longer a product is waiting in an output slot, the farther the respective marker will spread over the shop floor, eventually attracting a TA for pickup. The more products are waiting in an output slot, the higher the concentration of the diffusion source, leading to transporters being attracted to output slots with higher number of products. In combination, TA will be attracted to output slots, in which *higher amounts of products* await pickup, with longer waiting products attracting potentially more transports, increasing the chance for a soon pickup.

In [220], we modelled Transporter Agents TA that follow TMarker concentrations as follows:

Transporter agents are located on grid cells, with a triple (<t> stig:locatedAt <c>), with (<c> a stig:Topos), encoding the locatedness of a transporter <t> on a grid cell Topos <c>. Transporter agents perceive grid cell Topoi mutually adjacent, as additionally expressed by triples (<c_1> stig:adjacentTo <c_2>).

A transporter agent $\texttt{TA} = \texttt{PERC}(s, G = \emptyset, L = \emptyset)$ is initially situated in its idle location s; the agent performs situated perception as specified in Equation 6.1 with

$$\begin{aligned} (G'' = \texttt{ans}(q_{\texttt{PERC}}, G')) &\equiv (\forall t \in G' \Rightarrow t \in G'') \\ (L'' = \texttt{sel}(q_{\texttt{NAV}}, G'')) &\equiv (L'' = \{l \mid \operatorname*{argmax}_l \left(\begin{array}{l} \texttt{<l> stig:carries [stig:level ?val;} \\ \qquad\qquad\qquad\qquad \texttt{a ex:TMarker].} \end{array} \right)\}) \end{aligned}$$

The Reaction of the agent acc. to Eqn. 6.2 can be described as:

$$\texttt{REACT}(i, G, L) = \textbf{if } i \notin L \textbf{ then } \texttt{PERC}(j \in L, \emptyset, \emptyset) \textbf{ else } \texttt{PICKUP}(i, G, L)$$

Transporter Agents thus navigate to the highest concentration of ex:TMarker stigmata in its neighbourhood, and by this, climb the increasing gradient until reaching the source of the diffusion process: the marker on the output slot with the finished products.

When located at a topos i with highest marker concentration in the neighbourhood, i.e., the product output slot, TA will perform a PICKUP operation. During PICKUP, the agent will remove any products from the output slot, and return to its idle position:

	Random walk	Stigmergic coordination
Avg. number of updates	85	58
Avg. transporter steps	262	132
Mean time to deliver	112 seconds	67 seconds
Avg. max machine load	13	12
Avg. min machine load	6	8

TABLE 7.1: Results of simulations

$$\begin{aligned} \texttt{PICKUP}(i, G, L) \quad = \quad & \textbf{if } \exists p : (\texttt{<p> a ex:Product; stig:locatedAt <i>}) \in G \\ & \textbf{then } \texttt{DEL}(p, \emptyset, \emptyset).\texttt{MOVE}(s, p).\texttt{PERC}(s, \emptyset, \emptyset) \\ & \textbf{else } \texttt{PERC}(j \in L, \emptyset, \emptyset) \end{aligned}$$

7.4.4 Evaluation

We evaluated above scenario with fifty orders for products to be produced and picked up by the transporters from output slots. The shop floor contains five production machines and four transporter artifacts. For the sake of uniformity while running these simulations, all machines have output slots with a capacity of holding five finished products.

We employ the agent models as described in the previous section and benchmark against a simplified transporter agent model that only scans for finished products in its surroundings to initiate pick up, but otherwise move around randomly, i.e. not following any marker trace.

We compare the total number of updates required in each instance to complete producing fifty orders, as well as emptying them from the output slots. In addition, we compare the average number of steps moved by the transporters, the deviation in maximum load experienced by machines in each simulation and the average time that a finished product spends in an output slot before being picked up by transporters. These results can be seen in Table 7.1. The stigmergic coordination based shop floor simulation requires around 30% less updates in order to complete the simulation run of producing fifty orders and transporting them away from the output slots of machines. Also, it takes half as many movements by transporters compared to randomly moving transporters. Moreover, the average time it takes from a product from beginning of production to pickup by a transporter (mean time to deliver) is reduced by 40% in the stigmergy based simulation.

Average maximum and minimum machine loads are comparable in both cases, but slightly worse in the random walk simulations. Ideally, given that we have five machines and fifty orders, the average number of orders at each machine should be ten. But, since the randomly moving transporters often take longer to empty some output slots, the corresponding machines are loaded less relative to the other

machines. Each update query (which includes the implicit diffusion and linear decay of stigmergic markers) takes an average of 500 milliseconds to complete.

7.5 USE CASE 5 (Optimization): Minimize Open Stacks

This use case demonstrates how to employ stigmergy in a Linked Data medium to solve an abstract scheduling problem: the planner domain of the *Minimize Number of Open Stacks* problem *(MOSP)*. The following experiments were in its entirety originally published in the article [236]. This chapter models the problem domain in terms of a Linked Data representation, and defines three different agent types with increasing complexity of stigmergic principles applied for evaluation. The approach will finally be evaluated by comparing the performance with respect to the commonly employed metric, number of open product stacks created, against the verified optimal solutions as published in [9].

The experiments in this chapter will show that by proper application of stigmergic principles, agents will find solutions close to the top ranked planners in the original benchmark. However, while the planners in the original competition employed offline approaches, the solution presented in this chapter is an *online approach*, meaning that a close to optimal, or even optimal solution (depending on the chosen benchmark scenario) can be found in just a single run of the agent program, without any preliminary knowledge or assumption about the optimal solution,

7.5.1 The Minimize Number of Open Stacks Problem

The Minimize Number of Open Stacks Problem (*MOSP*) is a common scheduling problem to evaluate planner tools, and was one of the planner domains in the Constraint Modelling Challenge 2005[93] and International Planner Competition 5 [93]. The problem is known to be NP hard [152].

This problem assumes a fictional factory that is capable of producing products of different kinds. For items of same kind, it can produce batches of arbitrary size. However, there can always only be one product kind produced at the same time. Initially, the factory receives a number of orders that demand for one or more kinds of products. Whenever the factory produced a kind of product that was requested by one of the orders, a *stack* is opened for the respective order. Subsequent items for this order are added to the stack until all items requested by the order have been added to the stack, which is when the stack of the order will be removed. The goal is now to find an order in which the factory produces batches of product kinds, such that the number of open stacks is minimal.

[93] `https://ipg.host.cs.st-andrews.ac.uk/challenge/` *(Visited May 2023)*

7.5.2 Domain representation in Linked Data Medium

We model the scenario in the Linked Data medium as follows: We employ the RDF namespaces `stig:` for elements from the domain of stigmergic principles and effects (i.e., markers and traces), classes from the schema.org Ontology[94] with namespace `schema:` to refer to elements specific to order processes (e.g. orders and products), and `mosp:` as namespace to refer to instances within the minimize open stacks problem. A class of orderable products can then be described by triples of the form `( mosp:RedBox a schema:Product ).`

We employ the notion of situated tropistic agents as described in ([220]), cf. also Chapter 6, i.e., agents reside on (virtual) locations (or *"topoi"*), and react based only on perception of their direct surroundings. As discussed in the original publication, this model relates closely to stigmergic principles in nature, and is a very suitable choice to model stigmergic media in Linked Data.

In the presented case, we for this describe orders as instance of `schema:Order`, as well as a stigmergic *topos*, i.e., a resource in the Linked Data medium that can be visited and inspected by a tropistic stigmergic agent. Orders that share at least one type of product are perceived as *adjacent* by the agent (see also Listing 7.4). Adjancency implies both that an agent can *perceive* state of the adjacent resource, and move from a resource to an adjacent one as result of its perception.

```
mosp:order_1  a      schema:Order , stig:Topos ;
    st:adjacentTo    mosp:order_2 ;
    schema:orderedItem  mosp:RedBox , mosp:PurpleBox .

mosp:order_2  a      schema:Order , stig:Topos ;
    st:adjacentTo    mosp:order_1 ;
    schema:orderedItem  mosp:BlueBox , mosp:PurpleBox .
```

LISTING 7.4: Example of two orders in the minimize open stacks domain that share one common product.

Open stacks for orders are encoded as triples (`<urn>` a mosp:Stack), with `<urn>` a unique resource identifier that was randomly created when the stack was opened, the respective order linked to it via a relation `mosp:forOrder`, and contained products linked to it via the `schema:orderedItem` relation (see also Listing 7.5) .

```
<urn:>  a      mosp:Stack ;
    mosp:forOrder    mosp:order_1 ;
    schema:orderedItem  mosp:PurpleBox .
```

LISTING 7.5: Example of two orders in the minimize open stacks domain that share one common product.

[94] `https://schema.org`

7.5.3 Agent Models

The presented problem can be solved by a single agent. The general algorithm works as follows: The agent is situated on a `schema:Order` resource *o* as indicated by a triple `mosp:agent stig:locatedAt <Order>`, where `<Order>` is an order as described above. From here, the agent will perform the following steps:

1. Pick from order *o* any *product p* with `(<o> schema:orderedItem <p>); (<p> a schema:Product)` that is not yet part of any `mosp:Stack`, i.e., any type of product that has not yet been produced .
2. For each `schema:Order` $<$ o_i $>$ that requires this product as indicated by a triple `(`$< o_i >$ `schema:orderedItem <p>)`, add *p* to the respective stack s_i: `(`$< s_i >$ `a mosp:Stack ; mosp:forOrder` $< o_i >$ `; schema:orderedItem <p> )` .
3. For every order that is completed, i.e., for *o*, *s* with `(<s> a mosp:Stack ; mosp:forOrder <o>)` , $\forall p_k$: `(<o> schema:orderedProduct <`p_k`>)`$\Rightarrow$`(<s> schema:orderedProduct <`p_k`>)`, *remove* the stack *s*, the order *o*, and their corresponding triples .
4. If there is no order *o* left, **terminate**. Otherwise, move to another order *o* and **restart** from **1**.

The number of open stacks is counted before step 3, i.e., before finished stacks are closed.

We will show how the above behavior can be optimized by influencing the agent in its choice of the subsequent order in step 4. We use stigmergic markers to support the agent in preferring certain orders over others, such that the number of open stacks remains minimal. The approach can thus be classified as a hybrid *marker-based* and *sematectonic* stigmergic system (by the agent reacting both to markers, and results of its own work: the types of products already produced), in an *individual stigmergic system*, as there is only one agent that is steered by the results of its own action. For comparison, we have created three different agent behaviours with increasing selectiveness of subsequent orders as follows:

Random selection: In the simplest case, the agent selects the next order randomly among those that are labeled as `stig:adjacentTo`, i.e., the agent selects any open order that shares at least one kind of product, but neglecting whether the order has already a stack open. The selection of orders in Step 4 of the algorithm works as follows:

4.1 If the order that the agent was situated at was removed in step 3, i.e. there is *no triple* `( mosp:Agent stig:locatedAt <o> )` , choose any order *o* at random .

4.2 Otherwise, select an *adjacent* o_a with `<o> stig:adjacentTo <`o_a`>` .

4.3 **Restart** from **1**.

Favoring orders with stacks: A simple heuristic is to favor orders for which a stack is already open. Producing for an order with an already open stack eliminates the risk that a new stack is created for this specific order. In order to identify orders with open stacks, the agent *marks* in step 2 of the algorithm every order for which it already created a product:

2.1 For each `schema:Order` o_i for which a product was created, create a marker as indicated by the triple `( <`o_i`> stig:carries [ a stig:Marker ] )`.

When moving to another order after step 3, the agent favors orders that carry the *highest amount of markers*, i.e., for which it already produced the most products:

4.1 If the order that the agent was situated at was removed in step 3, i.e. there is *no triple* `( mosp:Agent stig:locatedAt <o> )` , choose any order o as order with the highest amount of markers: $o = \underset{count(\texttt{<m>})}{argmax}(\texttt{o})(\texttt{<o> stig:carries <m>})$

4.2 Otherwise, select an *adjacent* o_a with `<o> stig:adjacentTo <`$\texttt{o}_\texttt{a}$`>` and $o_a = \underset{count(\texttt{<m>})}{argmax}(\texttt{o}_\texttt{a})(\texttt{<o}_\texttt{a}\texttt{> stig:carries <m>})$.

4.3 **Restart** from **1**.

Favoring almost completed orders: The efficiency of the previous heuristic can be further improved if among those orders with already open stacks, the agent prefers those that are close to being finished. Preferring almost closed orders increases the probability that the agent will pick an order of which the stack can be closed in the next step, while reducing the risk of an agent choosing a product kind that opens several new stacks from an order with many open products. To identify respective orders, the agent leaves a marker as follows:

2.1 For each `schema:Order` o_i for which a product was created, create a marker as indicated by the triple `( <`o_i`> stig:carries [ a stig:Marker ; stig:level ?lvl] )`.

`?lvl` refers to the concentration level of the marker, and is equal to the number of remaining products in the stack.

In step 4, when selecting an order to continue with, the agent chooses as follows:

4.1 If the order that the agent was situated at was removed in step 3, i.e. there is *no triple* `( mosp:Agent stig:locatedAt <o> )` , choose any order o as order with the *lowest concentration* of markers:
$o = \underset{\sum(\texttt{?lvl})}{argmin}(\texttt{o})(\texttt{<o> stig:carries [a stig:Marker ; stig:level ?lvl]})$

4.2 Otherwise, select an *adjacent* o_a with `<o> stig:adjacentTo <`$\texttt{o}_\texttt{a}$`>`, and o_a carrying the lowest concentration of markers, as given in 4.1.

	Miller	NWRS1	NWRS2	NWRS3	NWRS4	NWRS5	NWRS6	NWRS7	NWRS8
Ord.	20	10	10	15	15	20	20	25	25
Prod.	40	20	20	25	25	30	30	60	60

TABLE 7.2: Dimensions of orders and products per problem domain

4.3 **Restart** from **1**.

7.5.4 Evaluation

Implementation of stigmergic principles

The presented algorithm implements a single agent. This agent reacts to both markers that were deliberately left on resources, and to results of its work in the environment by checking which kinds of products have already been produced in step 1 of the algorithm. By this, the presented algorithm implements *individual stigmergy* in a *marker-based* and *sematectonic* stigmergic system. The agent does neither maintain memory, nor does it plan or anticipate any future steps, but reacts solely on current observations of its current environment.

Empirical Results

We evaluate above agent models by having them solve 9 instances of MOSP as given in the Constraint Programming Challenge 2005. The problems have increasing complexity. Table 7.2 lists the numbers of orders and different product types per problem.

The sequence that an agent chooses in a single execution of a particular problem is non-deterministic. Typically, in one step, the agent will face several equally attractive resources as next candidates for a visit in step 4, in which case it will choose one at random. As the agent does not plan ahead, while being equally attractive at the instant, the choice of particular paths may have adverse effects later in the execution. We have therefore executed the experiment for each test instance 10 times. Figure 7.10 shows the best solution found by the agent out of 10 runs. Figure 7.11 shows the arithmetic average over the stack sizes as found by a particular agent type over all runs. Both figures also include the proven optimal solution as given in [9].

The experiments show that with increasingly expressive markers, the results of the algorithm improve for all problems, up to very noticable improvements in the more difficult problem instances NWRS 5 to NWRS 8.

In the simpler examples, even the random walk provides results close to the optimal solution. In these examples, orders have only few products in common, and by this share only few connections with other orders. This leaves only few choices to all of the agents which order to visit next, and reliably guides all types of agents over a close to optimal path.

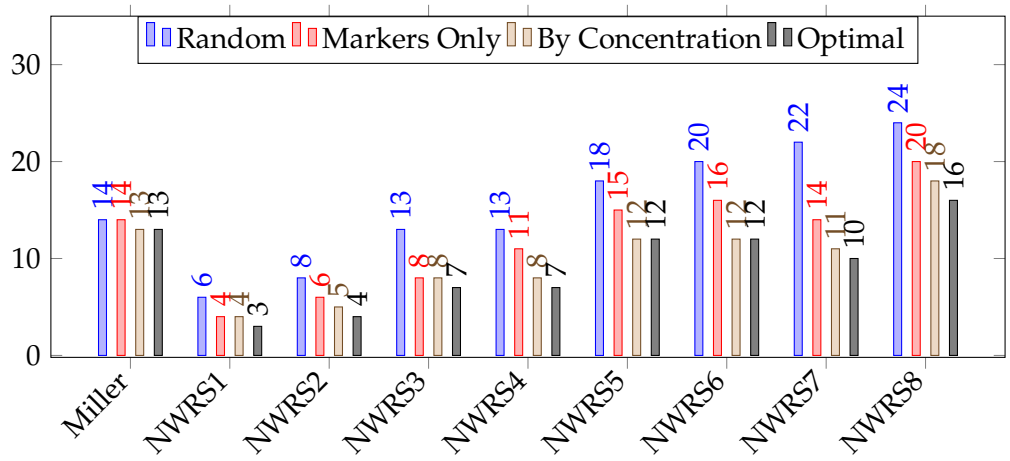

FIGURE 7.10: Minimal stack size found by the different agent models over 10 runs compared to the verified optimal solution.

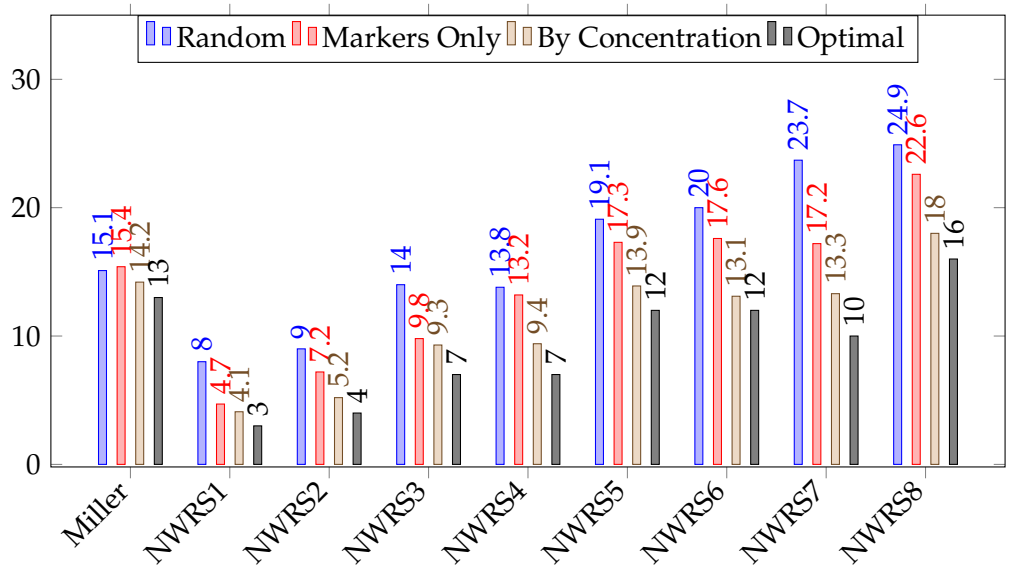

FIGURE 7.11: Arithmetic average of stack sizes out of 10 solutions found by the different agent models compared to the verified optimal solution.

In the more complex examples with larger and denser orders, in which orders share many products with many other orders, the random agent is more likely to pick a suboptimal path. However, with taking more information about its surrounding into account, the quality of the solution improves significantly, up to the most complex agent finding the close to optimal, or even optimal, solution in all cases.

These experiments show that with a sufficiently elaborate interpretation of, and interaction with the environment, stigmergic agents are able to find good quality solutions to complex optimization problems without the need to keep memory, plan ahead, or know the optimization goal at all, as are known benefits of stigmergic systems ([115, pp.13-14]).

7.5.5 Implementation

The above agents were entirely implemented in terms of SPARQL queries that encode the different actions that agents can take. The respective queries are published on GitHub, alongside with the application domain model, and a Postman collection that allows to execute the SPARQL queries against any triple store of choice:
`https://github.com/dfki-asr/stigmergy-mosp`

For evaluation, we used an Apache Fuseki standalone installation.

7.6 USE CASE 6 (Optimization): Trucks World

The following use case demonstrates how to achieve *self-optimizing* behavior by application of dynamic stigmergic markers, and the `stigLD` domain model. The trucks world domain is another problem that was chosen in the Internation Planning Competition [93]. The contents of this chapter have been previously published as part of the publication *"stigLD: Stigmergic Coordination in Linked Systems"* [219].

We demonstrate in this chapter that using stigmergy as coordination mechanism, the problem instances given in the challenge to be solved by linear (offline) planners, can be solved dynamically by an online approach in a single run. Not only will we show that a stigmergic online agent will be able to fulfill every constraint defined for each of the scenarios: moreover, the solution quality will rank among the highest scoring planners in the original competition. In addition, this chapter will demonstrate that our approach will adapt to changes of the problem domain *during runtime*, which is an extension to the original problem that linear offline planners would not be able to solve.

7.6.1 The trucks world domain problem

To demonstrate how to achieve *self-optimizing* behavior by application of dynamic stigmergic markers, and the `stigLD` domain model, we implement a stigmergic

agent system to solve the *(time constrained) trucks problem* as presented in the International Planning Competition 5 (IPC-5) [93].

The International Planning Competition is an annual contest to evaluate various planners on pre-defined sets of problems. Evaluating our stigmergic approach against problems from these competitions thus allows to compare the performance of our system against a set of established and publicly available benchmarks.

The chosen *trucks world* problem consists of a number of mutually connected *locations*. Initially, any of the locations may carry a number of *packages*. Locations moreover may serve as *destination* for packages, as determined by a set of *orders*.

Goal of the problem is for a truck to pick up packages and deliver them to their final location in shortest amount of time. Travelling from one location to another takes the truck a certain amount of time, depending on the distance between two locations, as set by the specific scenario. The space in the truck loading bay is limited, and divided into a set of *areas* that can carry one package each. The loading bay needs to be loaded and unloaded in a last in, first out manner, i.e., packages closer to the end of the truck block unloading of packages farther inside the truck. Loading packages to, unloading from, and finally delivering packages at the destination, each take an action with a fixed time span considerably shorter than the driving times between locations.

Figure 7.12 ([93]) shows a simple instance of the trucks world problem, and respective plans for the displayed problem: An optimal plan (upper right) that prioritizes loading packages over delivering, and a sub-optimal plan that loads only one package before delivery, and thus requires the truck to return to its initial location to pick up the second package.

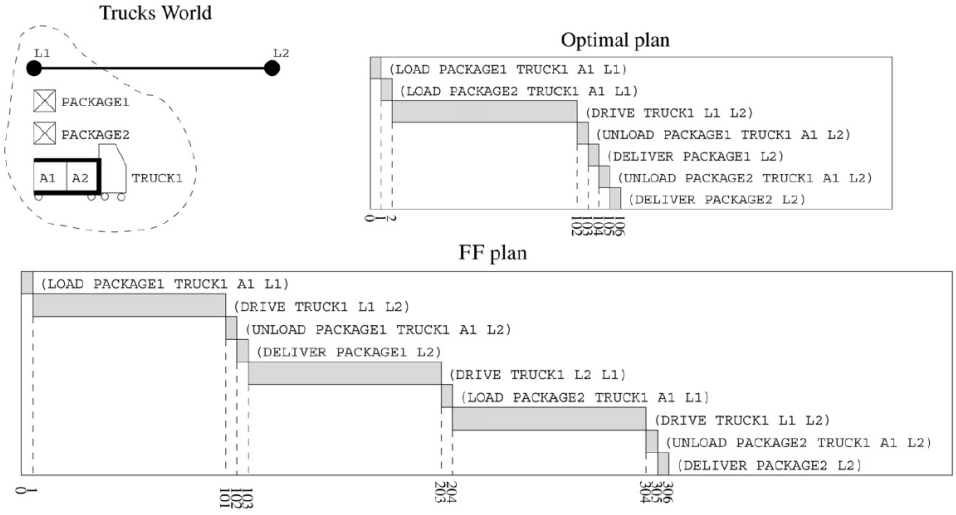

FIGURE 7.12: Schematic picture of the trucks world problem, and Gantt diagrams of respective plans satisfying the problem constraints, as originally presented in ([93])

The above general trucks world problem comes with a series of variants that each introduce additional constrains that need to be considered by planner instances. We evaluated our system against the *time constrained* trucks world problem. In this variant, each package delivery instruction comes with a deadline to which a package has to be delivered latest. If the truck fails to deliver any package within the specified deadline, the problem is considered as not solved. From the deadlines arises an implicit order for pickup and delivery tasks of particular packages.

This variant is particularly challenging to solve for self-coordinating systems, as the presented memory-less tropistic agents have no capabilities to look ahead in time, trace back sequences of actions to find the most optimal out of a number of explored solutions, or explore several potential solutions in parallel. The time constrained trucks world problem moreover allows us to judge the efficacy of our system w.r.t to self-coordination by solely observing if it manages to keep the given deadlines or not, without necessarily aiming for the shortest, (self-)optimized time of completion.

All material needed to run the presented scenario, and reproduce the reported results, is available on Github[95].

7.6.2 Domain representation in Linked Data Medium

Let `stig:` denote the Namespace for classes and predicates from the stigLD domain model, as defined in Section 5.1.1, and `trucks:` the namespace for classes and predicates specific to the trucks world domain.

For a trucks world problem instance, we represent locations that serve as pickup and delivery point for packages as both `stig:Topos` and `trucks:Location` . Locations are mutually linked by the predicate `stig:adjacentTo`, and moreover specify the distance to other locations by triples that are linked via the predicate `trucks:driveTime` (see also Listing 7.6). Trucks are linked to their current location by the `stig:locatedAt` predicate.

```
:loc1 a stig:Topos, trucks:Location  ;
  stig:adjacentTo   :loc2, :loc3;
  trucks:driveTime   [ trucks:destination :loc2;
                       rdf:value
   "406.3"^^xsd:double];
  trucks:driveTime   [ trucks:destination ex:l3;
                       rdf:value
   "73.1"^^xsd:double].
```

LISTING 7.6: A location in the trucks world domain, represented in the `stigLD` domain model.

Areas in the truck loading bay are represented as both `trucks:Area` and `stig:Topos`, and denote their position in the truck via the predicate `trucks:position`, linking

[95] `https://github.com/dfki-asr/stigld-trucks-world` *(Visited May 2023)*

to an integer value. Areas are sorted from the front to the back of the truck by increasing `trucks:position` values, as also shown in Figure 7.12.

Packages are represented as both `trucks:Package`, and, as they may carry stigmergic markers as result of the `MEDIUM`'s evolution (cf. Section 7.6.3), also as `stig:Topos`. Packages are assigned to their current location via the predicate `stig:adjacentTo`, i.e., the triple (<p> `stig:adjacentTo` <l>) denotes that a package p is currently located at a location l. Delivery instructions are represented as entities of a class `trucks:Goal`, that further specifies the package that needs to be delivered, the target location, and potential deadlines, by predicates `trucks:payload`, `trucks:destination`, and `trucks:deadline`, respectively.

7.6.3 Marker Model

Markers are generated by `MEDIUM` evolution on Topoi that are moreover classified as `trucks:Pacakage`. Markers denote packages that require urgent pickup or delivery, i.e., for which time runs short to meet the deadline. Packages without deadlines will not incite the creation of markers. The concentration of an individual marker will be calculated from the current time t of the simulation, estimated remaining time to deliver based on distances between pickup and delivery location, and the deadline d at which a package needs to be delivered.

The simulation time is provided by the environment by a triple (`trucks:ClockTime, rdf:value, t`), with t denoting the current time of simulation. The deadline d of a particular package is provided by a respective `trucks:Goal` resource, as described in the previous section.

The RDF graph G encoding of the problem instance is maintained by the stigmergic system's `MEDIUM` component. As described in Section 4.3.3, any `GET` request as part of a `TRUCK` agent's situated perception (cf. Eqn. 6.1) will trigger the following server-side updates

$$q_{\texttt{EVO}} \equiv \begin{pmatrix} \texttt{?i a trucks:Package, stig:Topos .} \\ \Downarrow \\ \texttt{?i stig:carries [a stig:Stigma; stig:level } c_i(t) \texttt{].} \end{pmatrix}$$

and drive the evolution of the system's package markers.

$ans(q_{EVO})$ calculates the concentration c of the marker and updates the concentration of an existing marker, or creates a new marker with concentration c, if the conditions are met, and there is not yet a marker present.

The concentration $c_i(t)$ for a package resource i at time t is calculated as follows: Let δ_p^i denote the time that it will take the truck to arrive from its current location at the *pickup* location of a package i, δ_d^i the drive time from the current location of i to its *destination*, and γ a configurable *confidence parameter*, in our evaluations chosen as $\gamma = 1.5$, and t and d current simulation time and deadline respectively.

The concentration c of a marker for a package i is then calculated as:

$$\begin{aligned} \delta^i &= (\delta^i_p + \delta^i_d), \text{ (minimal total drive time to pickup and deliver)} \\ r &= min(0, d - t - \delta^i), \text{ (remaining time buffer to deliver)} \\ c &= \begin{cases} \frac{1}{1-r}\ ,\ r < \delta^i * \gamma \\ 0,\ \text{else} \end{cases} \text{, (marker concentration)} \end{aligned}$$

A marker is thus assigned a concentration between 0 and 1 as soon as the remaining time left to deliver r falls below the confidently estimated delivery time, with 1 representing most urgent delivery.

7.6.4 Truck Agent Model

A truck agent $\texttt{TRUCK} = \texttt{PERC}(u, G = \emptyset, L = \emptyset)$ is initially situated in a location i, specified by the scenario description. Situatedness of the agent on a resource i is expressed by a triple (`:truck stig:locatedAt <i>`). The agent performs situated perception as specified in Equation 6.1 with

$$(G'' = \texttt{ans}(q_{\texttt{PERC}}, G')) \equiv (\forall t \in G' \Rightarrow t \in G'')$$

Let i be the resource the agent is currently situated on, and `:trucks` denote the resource representing the `TRUCKS` agent. When selecting its reaction (cf. Eqn. 6.2), the agent may perform a `PICKUP`, if situated on a resource representing a package, or `DROP` and `DELIVER` a package from its bay, if situated on a location that receives a package, with:

$$
\begin{aligned}
\texttt{REACT}(i,G,L) \;=\;& \textbf{if } i \notin L \textbf{ then } \texttt{PERC}(j \in L, \emptyset, \emptyset)\\
& \textbf{elseif } (<\texttt{i}> \texttt{ a trucks : Location}) \in G \textbf{ then } \texttt{DROP}(i,G).\texttt{DELIVER}(i,G)\\
& \textbf{elseif } (<\texttt{i}> \texttt{ a trucks : Package}) \in G \textbf{ then } \texttt{PICKUP}(i,G)\\
\texttt{DROP(i,G)} \;=\;& \textbf{if } \begin{pmatrix} \texttt{<p>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:adjacentTo <a> .} \\ \texttt{<a>} & \texttt{a trucks:Area .} \end{pmatrix}, \begin{pmatrix} \texttt{<g>} & \texttt{a trucks:Goal ;} \\ & \texttt{trucks:payload <p>;} \\ & \texttt{trucks:destination <i> .} \end{pmatrix} \in \texttt{G}\\
& \qquad \textbf{or if } \begin{pmatrix} \texttt{<p>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:adjacentTo <i> ;} \\ & \texttt{stig:carries [a stig:Stigma ;} \\ & \qquad \texttt{stig:level ?c}_p \texttt{] .} \end{pmatrix} \in G,\ ?c_p > 0,\\
& \qquad \textbf{and } \begin{pmatrix} \texttt{<q>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:adjacentTo [a trucks:Area] ;} \\ & \texttt{stig:carries [a stig:Stigma ;} \\ & \qquad \texttt{stig:level ?c}_q \texttt{] .} \end{pmatrix} \in G,\ ?c_q = 0,\\
& \qquad \textbf{and } \left| \begin{pmatrix} \texttt{<a>} & \texttt{a trucks:Area ;} \\ & \texttt{trucks:status :empty .} \end{pmatrix} \in G \right| < |\texttt{<p>}|,\\
& \textbf{then } \texttt{MOVE(k,i) ; k} = \underset{?pos}{\text{argmin}}\texttt{<k>} \begin{pmatrix} \texttt{<k>} & \texttt{a truck:Package ;} \\ & \texttt{stig:adjacentTo <a> .} \\ \texttt{<a>} & \texttt{a trucks:Area ;} \\ & \texttt{trucks:position ?pos} \end{pmatrix}\\
\texttt{DELIVER(i,G)} \;=\;& \textbf{if } \begin{pmatrix} \texttt{<p>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:adjacentTo <i> .} \end{pmatrix}, \begin{pmatrix} \texttt{<g>} & \texttt{a trucks:Goal ;} \\ & \texttt{trucks:payload <p>;} \\ & \texttt{trucks:destination <i> .} \end{pmatrix} \in \texttt{G}\\
& \textbf{then } \texttt{DEL(i,}\emptyset,\emptyset\texttt{)}\\
\texttt{PICKUP(i,G)} \;=\;& \texttt{MOVE(:truck, l) ; (<i> stig:adjacentTo <l>)} \in G\\
& \texttt{MOVE(i,a) ; a} = \underset{?p}{\text{argmax}}\texttt{<a>} \begin{pmatrix} \texttt{<a>} & \texttt{a trucks:Area;} \\ & \texttt{trucks:position ?p ;} \\ & \texttt{trucks:status trucks:empty .} \end{pmatrix}
\end{aligned}
$$

During PICKUP, the truck agent will move the package, on which it is currently situated, from its current location to the furthest back free loading area in the bay, i.e., insert a triple (<i> stig:adjacentTo <a>), and subsequently relocate the truck to the location from which it picked up the package.

Conversely, DROP will move the frontmost package from the loading bay to the location on which the truck is currently located. The truck will also DROP the frontmost package as long as there are more urgent packages located on i, than the truck can currently accommodate, but the space in the truck is occupied with non urgent packages. If the location i on which the truck is situated is destination to any of the packages located on i, they are DELIVERed, i.e., removed from the scenario, and the respective delivery goal is marked as completed. Note that whenever the truck reaches a location to which a package needs to be delivered, the truck prioritizes unloading and delivering over any pickup action.

`PICKUP`, `DROP` and `DELIVER` increase the simulation time clock by the duration for these action as defined by the scenario.[96]

If none of the conditions above apply, the truck will update its linkset and relocate to another resource:

Let $(L'' = \texttt{sel}(q_{\texttt{NAV}}, G'')) \equiv L'' = n(\circ)$ denote the evaluation of the navigational query, then as a result of perceiving i, and neighbouring locations j , the agent may further decide to navigate to another location as follows:

1. If `(<p> stig:adjacentTo <i>)` $\in$ `G` , i.e, if there is any package on the trucks current location, then:

$$n(\circ) = \underset{?dist}{\text{argmax}}\texttt{<p>}\{\underset{?c}{\text{argmax}}\texttt{<p>}_{|a|}\{p \mid \begin{pmatrix} \texttt{<i>} & \texttt{a trucks:Location ;} \\ & \texttt{trucks:driveTime [trucks:destination ?q ;} \\ & \texttt{rdf:value ?dist] .} \\ \texttt{<p>} & \texttt{stig:adjacentTo <i> ;} \\ & \texttt{stig:carries [a stig:Stigma ;} \\ & \texttt{stig:level ?c] .} \\ \texttt{<a>} & \texttt{a trucks:Area ;} \\ & \texttt{trucks:status :empty .} \\ \texttt{<g>} & \texttt{a trucks:Goal ;} \\ & \texttt{trucks:destination <d> ;} \\ & \texttt{trucks:payload <p> .} \end{pmatrix} \in G\}\}$$

   Consequently, out of the $|a|$ *most urgent* packages, with $|a|$ being the number of free loading bay areas in the truck, the truck moves to the package with the *farthest distance to its destination*, leading to loading urgent parcels with nearby delivery destinations last.

2. If above condition is not satisfied, i.e. there are no packages on i or the loading bay is full, then if for any package p with `(<p> stig:adjacentTo [a trucks:Area])` $\in$ G there is `( <p> stig:carries a stig:Stigma ; stig:level ?c])` $\in$ `G` with $c > 0$, i.e., any package within the truck's loading bay has exceeded it's delivery confidence value:

$$n(\circ) = \underset{?c}{\text{argmax}}\texttt{1}\{l \mid \begin{pmatrix} \texttt{<p>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:carries [a stig:Stigma ;} \\ & \texttt{stig:level ?c] .} \\ \texttt{<g>} & \texttt{a stig:Goal ;} \\ & \texttt{trucks:destination <l> ;} \\ & \texttt{trucks:payload <p> .} \end{pmatrix} \in G, c > 0\}\}$$

   Consequently, the truck moves to the next location l to which a package needs to be delivered *most urgently*.

[96] In the evaluated use cases, `PICKUP`, `DROP` and `DELIVER` take 1 time step each.

3. If none of the packages needs to be delivered urgently, then:

$$n(\circ) = \underset{?c}{\text{argmax}}\texttt{<l>}\{l \mid \begin{pmatrix} \texttt{<l>} & \texttt{a stig:Location .} \\ \texttt{<p>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:adjacentTo <l> ;} \\ & \texttt{stig:carries [a stig:Stigma ;} \\ & \texttt{stig:level ?c] .} \end{pmatrix} \in G, c > 0\}$$

   Consequently, the truck moves to the location with the package with the highest marker concentration c.

4. If for none of the packages it holds that marker concentration $c > 0$, move to the location where the package in the front of the loading bay, i.e., the package that would be unloaded next, needs to be delivered:

$$n(\circ) = \underset{?pos}{\text{argmin}}\texttt{<l>}\{l \mid \begin{pmatrix} \texttt{<p>} & \texttt{stig:adjacentTo <a> .} \\ \texttt{<a>} & \texttt{a trucks:Area;} \\ & \texttt{trucks:position ?pos .} \\ \texttt{<g>} & \texttt{a trucks:Goal ;} \\ & \texttt{trucks:payload <p> ;} \\ & \texttt{trucks:destination <l> .} \end{pmatrix} \in G\}$$

5. Finally, if the truck is empty and none of the packages in any location carries a marker with marker concentration $c > 0$:

$$n(\circ) = \underset{|p|}{\text{argmax}}\texttt{<l>}\{l \mid \begin{pmatrix} \texttt{<l>} & \texttt{a trucks:Location .} \\ \texttt{<p>} & \texttt{a trucks:Package ;} \\ & \texttt{stig:adjacentTo <l> .} \end{pmatrix}$$

By the behavior as defined above, TRUCK agents react to the stigmata generated and updated during evolution of the environment, as described in Section 7.6.3, or the relation between *package* resources, and other Topoi. With marker concentrations being calculated before queries by the truck agent are evaluated against the environment state, as it is defined for environment evolution in Eqn. 4.1, the truck agent always perceives the accurate marker concentrations, at the relevant topoi, at time of perception.

7.6.5 Evaluation

We evaluate above agents against Problems 1 through 5 of the time constrained trucks world challenge of the IPC-5[97]. These problems each provide scenarios with a single truck agent. An extension of our trucks world model to scenarios with a number of trucks operating simultaneously is ongoing, and publication of the results is subject to future work. To show adaptability of the system and chosen agent model, we use the same agent model and implementation throughout every problem instance, without further adaption towards individual challenges.

[97] The original problem definitions are available for download from `https://lpg.unibs.it/ipc-5/`, under "Resources"

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
# locations	3	3	3	3	3
# Packages	3	4	5	6	7
# Deadlines	3	1	3	3	3

TABLE 7.3: Problem sizes of the Trucks World single agent problem instances

Table 7.3 shows the problem dimensions of the different problem instances: Each of the problems accommodates a set of 3 interconnected locations each, with an increasing number of packages that need to be delivered. In all but the second problem instance, 3 of the delivery instructions are constrained by a deadline, with the second problem specifying only one deadline.

The results of the evaluation are shown in Tables 7.4 and 7.5. Table 7.4 shows the deadlines defined by the individual problem instances, and the time by which the respective packages were delivered to the defined target destinations. Clearly, the agent manages to coordinate the pickup and delivery runs in every problem to meet each given deadline.

Table 7.5 compares the performance of our approach to the two planners that solved the time constraint trucks world problem in the IPC-5[98], MIPS-XXL ([81]) and SGPLAN ([45]). This comparison shows that our solution does not only find valid solutions in every problem instance w.r.t. deadlines, but also performs in the range of efficiency as the reference planners. In all of the problems, we find solutions in at most as many steps as the reference planners, while delivering all packages in the same time ranges as the reference planners. Only in Problem 5, our solution takes one step more than the only other competitor that solved the problem, SGPLAN, whereas MIPS-XXL did not provide a solution at all.

		Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Pack. 1	Deadline	919.7	842.7	616.7	537.3	992.8
	Delivered	432.9	769.1	569.6	491.4	729.2
Pack. 2	Deadline	919.7	N/A	925.1	1026.9	1866.7
	Delivered	842.2	N/A	283.3	942.5	292.6
Pack. 3	Deadline	1813.7	N/A	925.1	2878.2	2878.0
	Delivered	844.2	N/A	285.3	1737.2	2255.8

TABLE 7.4: Deadlines and times of deliveries per package in the different problem instances: the truck agent manages to meet all deadlines in all problems.

[98]Results are available for download from `https://lpg.unibs.it/ipc-5/`, under "Resources"

		MIPS-XXL	SGPLAN	stigLD
Problem 1	Steps	12	14	12
	MakeSpan	845.32	845.23	844.20
Problem 2	Steps	17	17	17
	MakeSpan	1714.57	1711.44	1713.4
Problem 3	Steps	19	19	19
	MakeSpan	1474.29	1470.14	1473.1
Problem 4	Steps	23	23	23
	MakeSpan	2634.63	2629.45	2676.7
Problem 5	Steps	N/A	28	29
	MakeSpan	N/A	1870.06	2255.8

TABLE 7.5: Number of steps and make span taken until the last package was delivered by the compared approaches

	t	Pack 1	Pack 2	Pack 3	Pack 4	Pack 5	
Problem 3	300	925.1	925.1	1792.58	1744.48	1744.48	Deadline
		283.3	285.3	569.6	1136.2	1721.2	Delivered
Problem 4	950	537.3	1026.9	3629.56	4224.76	4224.76	Deadline
		491.4	942.5	1737.2	2979.0	4221.8	Delivered
Problem 5	500	1866.7	1877.32	3762.52	1393.42	1393.42	Deadline
		292.6	1783.8	2692.4	822.4	1349.2	Delivered

TABLE 7.6: Adapted deadlines and time of completion for packages after introduced disturbance at time t. Red: Newly added packages with deadlines, Blue: Deadlines changed from the original problem.

Time constraint trucks world with disturbances

A main feature of stigmergy-based systems is their robustness against disturbances in the environment ([115]). In order to demonstrate that by employing the methods and technologies from this paper, a robust, adaptive agent behavior emerges, we further extended the original IPC trucks challenge. In our extension, after a fixed time, new packages are created at a given locations, and with given deadlines. Deadlines of packages that had not been delivered at the time of disturbance are increased (colored blue in Table 7.6) to keep the problem satisfiable, even if the truck has to pick up newly created packages before those originally in the problem. We omitted problems 1 and 2, as the low number of packages in the original problems did not leave much of a margin to introduce disturbances that would actually disturb the original problem execution.

Time of disturbances and new deadlines are chosen such that the disturbance happens *before* the plan for the original problem has finished, i.e., the truck agent will have to react to the newly created package delivery orders before finishing delivery of the packages that were originally in the problem.

New packages are created on the same location, but are to be delivered to different destinations. We calculated the new deadlines such that while the truck may take one connection between locations twice to deliver each of the packages to a different destination, it is left only one movement to a neighbouring location, leading to that the truck will have to combine deliveries to make the new deadlines. Remaining deadlines after the newly introduced deadlines are shifted accordingly.

The results of our experiments with induced disturbances is shown in Table 7.6. Despite constraints changing while the simulation is already running, the truck manages to meet any given deadline, old ones as well as new ones. In Problem 5, the new deadlines were created after the truck has already delivered the first package, thus deadline and delivery time of that package did not change compared to the original problem. However, as soon as new deadlines are generated, the truck prefers the newly created packages with stricter deadlines over the originally placed ones (see Problem 5). Given that none of the agents maintain memory or plans, we can by this show that the presented system is able to react dynamically to changes in the environment, and without (re-)planning, manages to generate valid solutions in dynamic environments. This is a significant improvement to the existing planners that solved the discussed problem, as those create offline results and by this are not able to alter a plan based on changes in the environment during execution.

Chapter 8

Conclusion and Future Work

This thesis discussed *Linked Data* as suitable interactive digital medium for multi-agent systems. The thesis motivated the presented results by highlighting the importance of taking into account the role of a digital medium as representation, interaction, and communication space, and by this, as third crucial core element in a multi-agent system (Chapter 1). This particular consideration of the medium as third element in a multi agent system is a novelty compared to literature which typically only distinguishes between agents and their environment, whereas the distinction between the environment as the agent surrounding effects, sensations, and actions, and how to technically provide these to the agent, is left open, or marginalized.

Consequently, considerations of what constitutes a proper medium are lacking in literature. With a focus on bio-inspired multi-agent systems that operate by means of *stigmergy*, an indirect coordination principle found in nature, Chapter 4 proposed a set of feature requirements for digital media to be a suitable choice for multi-agent systems. The focus on stigmergic systems may at first seem arbitrary and limiting the final results, but it is justified, as they employ all three indirect communication between agents via the medium, (indirect) interaction of agents with the medium, and by this, finally, coordination and optimization effects emerge within the medium.

The thesis continued to discuss the suitability of Linked Data Architectures to constitute a suitable digital agent medium. The choice for Linked Data as medium to be discussed was after all also made due to the fact that Linked Data is based on a set of well-defined, standardized technologies and interaction patterns, such as HTTP, HATEOAS principles, and last but not least, the RDF data model that allows to represent semantic information about resources to agents in a machine-readable and interpretable way.

Grey is all theory, this is why Chapters 5 and 6 presented in detail possible implementations of both environment representation in Linked Data media, and Linked

Data media consuming agents, respectively. For this, the Thesis first provided formal models of both media servers, and agents interacting with those. For environments in digital media, two approaches were presented that also allow for the representation of environments with inherent dynamics.

Finally, the application of the presented concepts and implementations was demonstrated in the context of practical use-cases and benchmarks from the domains of coordination in cyber-physical production environments, and optimization in abstract planning challenges.

The research questions posed in Section 1.1.2 were answered, and verified the hypotheses stated in Section 1.1 as follows:

- **R1. What constitutes a suitable digital agent medium?**, was answered in Section 4.2.3 by deriving, inspired from stigmergic systems in nature, a set of requirements that a digital medium needs to fulfill to be as suitable medium for stigmergic multi-agent systems specifically, and by this. These findings naturally transfer automatically to any kind of multi-agent systems that use the medium for indirect interaction between agents and their environment, as well as indirect communication between agents.
- **R2. Is Linked Data a suitable choice for a digital agent medium?**, was answered in Section 4.2 by discussing the found requirements with respect to the design principles, features, and interfaces to Linked Data architectures, coming to the conclusion that Linked Data provides a suitable medium by fulfilling all stated requirements.
- **R3. How would Linked Data consuming agents interact with Linked Data Media?** Chapters 4.3.2 and 4.3.3 presented a description of interaction channels of Linked Data media for for both static and dynamic environments, using Milner's Calculus of Communicating Systems, as originally published in [220]. The chosen description exhaustively covers typical operations on RDF data sets using HTTP CRUD operations, and by this provides a formal, well-defined set of interactions as offered by Linked Data media, by thus answers Research question R3.
- **R4. What is a suitable and sufficiently simple agent model to interact with Linked Data Media?** Chapter 6.1 presented agent models for single tropistic agents, and hysteretic agent swarms. Both models were based on the formally defined interaction between agents and Linked Data media. This model keeps the agent behavior simple by its (mostly) memoryless, purely reactive nature, while a the same time, exploiting the expressiveness and interactiveness of the medium to its full extend. The variation of different agent behaviors that can emerge from these simple base models has been demonstrated in Chapter 7.

Research questions 1 – 4 jointly validate **Hypothesis H2.**: By R1. answered, we gained an understanding about requirements towards a digital agent medium, while R2. validated that Linked Data is a suitable choice. Agent-Environment interaction could be well defined in the scope of R3. and R4.

Feasibility of these conceptual considerations on a technical level is furthermore ensured by answering research questions 5 – 7:

- **R5. How can dynamic multi-agent environments be efficiently published in a Linked Data Medium?** Chapter 5 presented two approaches to provide dynamic environments via Linked Data media: First, the stigLD media server as presented in Section 5.1 (see also [220, 219]), which stores environment data in a typical graph store (see also Section 3.1), and drives environment evolution by specific queries. The second approach, as thoroughly described in Section 5.2, employs an intermediate meta data model to transparently lift native application run-time data to a Linked Data Platform representation, thus providing the possibility to publish real-time simulations as Linked Data medium. The respective lifting algorithm has been implemented as standalone server solution, as well as a library for the Unity 3D game engine.
- **R6. How would an implementation of the agent model found in R4 look like?** Chapter 6.2 provided a direct translation from the formal agent model descriptions in Section 6.1 to executable behavior trees as suitable implementation of the defined models. Specific implementations of specialised, use-case oriented agents based on the provided approach have been further detailed out in Chapter 7, specifically in the use-case presented in Section 7.3.
- **R7.How can continuity of the medium be ensured during link traversal?** This specific question was answered by presenting the SPARQL API Service in Section 5.3, which allows agents to evaluate SPARQL queries against non-RDF endpoints, and thus letting agents perceive non-RDF endpoints as if they were part of the Linked Data medium. Usability and efficacy of the approach has moreover been demonstrated in the use-case presented in section 7.3.

Finally, by evaluating the developed approaches in various use-cases in Chapter 7, the thesis answered:

- **R8. Which means are suitable to incite the emergence of coordination and optimization effects in the medium?** Chapter 7 demonstrated how to employ means of *stigmergy* (see also Section 4.2.2), a self-coordination principle inspired by nature, to incite a coordinating and optimizing behaviour of the system, solely by relying on the mediating function of the medium. It was specifically shown how to achieve coordination and optimization effects, using the reactive agent models as defined in Chapter 6, and employing indirect communication mechanisms based on stigmergic markers. Respective agent models have been presented in detail in the specific use-cases in Chapter 7.

– **R9. Does off-loading agent complexity into the medium come with a trade-off with respect to solution quality?** The results in Chapter 7 show that off-loading complexity in the medium, and keeping agent models simple (reactive, memoryless), does not inflict quality of solutions found by the various coordination and optimization algorithms. Not only do the algorithms produce provable correct results (Sections 7.2, 7.3) with clearly visible coordination effects (Section 7.4), but for selected algorithms, the quality of the solution is comparable to equal to that of reference planners (Sections 7.5, 7.6).

By answering research questions 1–8, this thesis validates **Hypothesis H3.**: Based on the definition of both medium and agent-medium interactions (R1. – R3.) , a set of simple agent models (R4., R8.) allowed for the emergence of coordination and optimization effects within the medium.

With R9. answered, the thesis moreover validated By this, the thesis validated **Hypothesis H4.**. **H2.** – **H4.** in conclusion finally validate **Hypothesis H1.**:

„Complexity of the implementation of multi-agent systems is reduced tremendously by a properly defined medium, and respective agent-medium interaction. This is due to the possibility to reduce the complexity the agents by exploiting the emergence of optimization and coordination effects in the medium, and reduce the implementation of agents to a most simple agent model."

8.1 Future Work

The main contribution of this thesis was to strengthen the notion of digital agent media as third component in Multi Agent Systems. Such MAS were mainly designed and evaluated in the domain of cyber-physical manufacturing in Industry 4.0 environments, with the purpose of the systems being coordination or optimization of production or scheduling processes. The contents of this thesis obviously only touch a small fraction of a very complex and promising topic, and they hopefully serve as inspiration for future definitions of MAS and the interaction of agents and environment via a medium technology.

I see several possible directions to which the topic may be further deepened: Digital media may be further analyzed with their capability to serve as shared knowledge base in *distributed machine learning* algorithms [255]. Moreover, digital media may support the interaction between AI planners and human users in AI driven *recommender systems* [207, 153]. The approaches in this thesis were so far applied to use-cases in cyber-physical production in the Industry 4.0. However, given that the general efficacy in abstract coordination and optimization algorithms could as well be shown, it may be worthwhile to evaluate the effectiveness of similarly carefully designed systems and algorithms in other domains. In this Thesis, Linked Data was

taken as one of many possible examples for digital agent media. However, a similar thorough analysis concerning suitability and possible interaction models may strongly support the adaption of new, emerging technologies, such as blockchain technology [63], or directed acyclic graphs (DAG) [200]. Finally, this Thesis focused on stigmergy as indirect coordination mechanism, and provided a well defined agent-environment interaction via Linked Data media to achieve stigmergic self-coordination effects, and moreover a couple of derived agent models for specific use-cases. Similar effort may be taken to investigate other coordination principles.

8.1.1 Media-centered MAS for recommender systems

This thesis primarily described systems in which AI agents brought the underlying system to a coordinated and / or optimized state without human interaction. *Recommender systems* [207, 169, 153, 2], on the other hand, are designed to find, out of a current state of system, the top candidates of state transitions into a next state, with leaving the final decision which state is to be taken to a human.

We experience recommender systems every day when online shopping portals, or video streaming platforms, present us a selection of articles we might be interested in buying, or a selection of movies or shows we might be interested in watching, based on our previous watching habits. Recommender systems are moreover topic of research for application in public and professional domains, such as smart homes [205, 94], smart cities [62, 204], or smart factories in the context of industry 4.0 [44, 109].

With the Internet, built around Web and Semantic Web technology, exists a medium that is equally accessible to both human users, and AI user-agents: While in the Semantic Web, the Web-contents are "rendered" as Linked Data View towards AI agents (as also further detailed in Section 5.2.1), in interactive Web pages, the content is rendered in terms of texts, images, and videos as perceivable, interactive representation for human users of the Internet. The parallels between user-agent and human use of Web content go even that far, that the Internet has in the past been shown to implement stigmergic principles to coordinate and optimize the content provided to individual human users [70].

Consequently, the idea to extend the concepts presented in this thesis as underlying principles to define and implement recommender systems is evident: The agent models in this thesis made extensive use of advertising possible options for interactions to other agents in terms of markers carried by the medium (see for example the application use-cases in Sections 7.2 and 7.3; in Section 7.6, the Trucks optimization domain, the recommendations which packages to pick up or deliver next were even generated by dynamic medium itself). Likewise, instead of marker traces they leave in the environment, agents could render, as results of their observations and perceptions, interactive, descriptive Web elements on a Web site that guide human users

through complex processes by recommending suitable options at every step of the process.

8.1.2 Transfer to other application domains

This Thesis presented a generic model of agent-environment interaction via Linked Data as digital medium. The applicability of this model has been shown by refining the generic agent models to agents that were further modelled to solve specific problems, which were, in the scope of this thesis, majorly chosen from the domain of cyber-physical production.

However, the need for large-scale, distributed coordination and optimization arises in a variety of topics that impose a real challenge to society in the very near future:

Structural changes in energy supply with focus being shifted to potentially unstable renewable energy meets an ever increasing demand. *Smart Grids* [227] have been proposed as an AI supported architecture to respond to fluctuating demands, lately also specifically analysed with their role to cope with the fast increasing number of electric vehicles as consumers [107, 213]. Multi agent systems have moreover been applied to coordinate navigation of vehicle fleets in autonomous driving scenarios [144, 196], with the goal to reduce traffic congestion, and plan vehicle paths according to economical and ecological constraints [224]. *Smart Farming* [100, 259, 179] researches AI approaches to help food production adapt better to a fast growing population, and rapidly changing climate conditions.

Given that the adaptability, scalability, and robustness of self-coordinating and self-organizing systems in these domains have been investigated previously [99, 20, 159, 181, 158], re-visiting these domains with the medium-based MAS model as presented in this thesis in mind may be worthwhile to find more applicable, transferable solutions in the mentioned domains.

8.1.3 Transfer to other media

This thesis analyzed Linked Data as suitable medium for multi-agent systems. In the scope of the thesis, Linked Data was analyzed and discussed with respect to suitability in Section 4.2. Subsequently, a suitable agent-medium interaction model was presented in Sections 4.3.2 and 4.3.3.

A similar analysis for other technologies may be beneficial for the research community to get a deeper understanding about both established and emerging technologies against which MAS are implemented. A comparable discussion of other media as given in this Thesis for Linked Data could moreover ease the transfer of agent models, coordination- , and optimization algorithms to other media. Such a discussion is of particular interest for technologies that put constraints on MAS interaction

by constraints that stem from particularities in design and architecture of the target technologies.

Take as example block chain technology [63], that has peaked interest in society, industry, and research tremendously in the last decade. While having come to the attention of the broad public mainly as underlying technology for the (in)famous Bitcoin crypto-currency [223], and, to this day, hundreds of currencies that build on the same technology[99], it is also vividly discussed for possible application in other fields.

It has been suggested for Industry 4.0 architectures [18, 83] as technology for secure, scalable communication of IoT devices. It is believed that blockchain, together with Web technologies, to "ensure transparency of supply, immutability of records, and it ensures the trust of end to end trading organizations." [136].

Blockchains have moreover drawn noticable attention in the research community around MAS [34]. In this context, blockchains are lately discussed as means to document decision processes of MAS, and by this, make AI-based decision-making more explainable, and by this increase acceptance in public [33]. Moreover, protocols for MAS have been defined for blockchain-based communication within fleets of autonomous vehicles [131], for MAS-coordinated asset transfer [192], and even for blockchain as knowledge representation for MAS in industry 4.0 [199].

A common understanding of blockchain, or other technologies with similar purpose, like IOTA's underlying Directed Acyclic Graph [200], as digital medium, along with a well-defined, generic agent-medium interaction model, may help tremendously to transfer individual results to new approaches.

Blockchain was discussed in detail only as one of many example of other MAS media. Other promising candidates are High Performance Computing (HPC) architectures [256, 210], or interaction with container and cloud environments [243].

Given the attention block chains currently receive as underlying technology for MAS, and the promising results achieved so far, a similar analysis of block chains as agent medium may be worthwile. Not only could an analysis similar to the one done for Linked Data in this thesis strengthen the understanding about block chain technologies as agent medium, but a well-defined interaction pattern between agents and block chain architectures would further facilitate the design of block chain based MAS, allow for transferability of results, and by this benefit the uptake of the topic in research that we can currently observe.

[99] At the time of writing of this Thesis, one of the largest crypto currency platforms, Binance (`https://www.binance.com/en`), listed more than 600 different tradeable tokens. (May 21st, 2022)

8.1.4 Extend to different coordination principles and algorithms

This thesis primarly employed stigmergy as underlying nature-inspired coordination principle. While commonly following this principles by employing situated tropistic agents, the individual algorithms were mostly tailored towards the respective use-case.

Research attempted to solve classic coordination and optimization problems using stigmergy. Examples for this are works that create solutions for the traveling salesman problem (TSP) [88] using ant colony optimization [74, 271], or for general continuous numeric optimization [140]. Analyzing such generic optimization algorithms based on a formal agent-medium interaction model as presented in this thesis would facilitate transferring these generic results to more specific applications.

The efficacy of medium-based coordination and optimization stems from the fact that observations on which agents act are direct symptoms of the state of the underlying environment, and that the agents' endeavor to strengthen positive symptoms while mitigating undesired ones directly affects the underlying environment.

Another nature-inspired algorithm model that is very much designed around the detection and handling of *symptoms* are *artificial immune systems* (AIS) [246, 98]. Artificial immune systems are used for fault prediction and error handling by representing the problem domain into a model that resembles the human immune system. AIS implement virtual "cells" as agents that detect anomalies in the underlying system, and send further cells to "heal" the observed anomalies. AIS have been applied in various domains, such as cyber- security in IoT [270, 25], energy management [274, 181], or traffic [158, 61].

The model of tropistic agent swarms as presented in Section 6.1.2 is a suitable basis from which an AIS-like "cell"-swarm behaviour could be defined, whereas the formal definition of the medium provides the well-defined interface between cell agents and problem domain. Extending this model of medium-based agent swarms to specifically describe a generic AIS model would greatly contribute to an even better understanding of AIS, and support transferability of AIS systems.

Bibliography

[1] Yehia Abd Alrahman and Nir Piterman. "Modelling and verification of reconfigurable multi-agent systems". In: *Autonomous Agents and Multi-Agent Systems* 35.2 (Oct. 2021). ISSN: 15737454. DOI: 10.1007/S10458-021-09521-X.

[2] Charu C Aggarwal et al. *Recommender systems*. Vol. 1. Springer, 2016.

[3] Toni Alatalo. "An entity-component model for extensible virtual worlds". In: *IEEE Internet Computing* 15.5 (2011), pp. 30–37.

[4] Antonio Luca Alfeo et al. "A Stigmergy-Based Analysis of City Hotspots to Discover Trends and Anomalies in Urban Transportation Usage". In: *IEEE Transactions on Intelligent Transportation Systems* 19.7 (2018), pp. 2258–2267. ISSN: 15249050. DOI: 10.1109/TITS.2018.2817558. arXiv: 1804.05697.

[5] Mohab Aly et al. "Is Fragmentation a Threat to the Success of the Internet of Things?" In: *IEEE Internet of Things Journal* 6.1 (2018), pp. 472–487.

[8] Sören Auer et al. "Triplify: light-weight linked data publication from relational databases". In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 621–630.

[10] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. "Abstraction of agents executing online and their abilities in the situation calculus". In: *IJCAI International Joint Conference on Artificial Intelligence*. Vol. 2018-July. 2018, pp. 1699–1706. ISBN: 9780999241127. DOI: 10.24963/ijcai.2018/235.

[11] Vinícius A Barros et al. "An IoT multi-protocol strategy for the interoperability of distinct communication protocols applied to web of things". In: *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. 2019, pp. 81–88.

[12] Tim Berners-Lee, James Hendler, Ora Lassila, et al. "The semantic web". In: *Scientific american* 284.5 (2001), pp. 28–37.

[13] Simon Bienz et al. "Escaping the streetlight effect: Semantic hypermedia search enhances autonomous behavior in the web of things". In: *Proceedings of the 9th International Conference on the Internet of Things*. 2019, pp. 1–8.

[15] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data: The story so far". In: *Semantic services, interoperability and web applications: emerging concepts*. IGI Global, 2011, pp. 205–227.

[16] Christian Bizer and Andy Seaborne. "D2RQ-treating non-RDF databases as virtual RDF graphs". In: *Proceedings of the 3rd international semantic web conference (ISWC2004)*. Vol. 2004. Proceedings of ISWC2004. 2004.

[17] Michael Blackstock and Rodger Lea. "Toward interoperability in a Web of Things". In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. 2013, pp. 1565–1574.

[18] Umesh Bodkhe et al. "Blockchain for industry 4.0: A comprehensive review". In: *IEEE Access* 8 (2020), pp. 79764–79800.

[19] Eric Bonabeau et al. "Routing in telecommunications networks with ant-like agents". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1437.1 (1998), pp. 60–71. ISSN: 16113349. DOI: 10.1007/bfb0053944.

[20] SG Borghini. "Stigmergy in the design of social environments". In: *The European Physical Journal Special Topics* 226.2 (2017), pp. 269–281.

[21] Andy Bower and Blair McGlashan. "Twisting the triad". In: *Tutorial Paper for European Smalltalk User Group (ESUP)* (2000).

[22] İlker Semih Boztepe and Rıza Cenk Erdur. "Linked data aware agent development framework for mobile devices". In: *Applied Sciences* 8.10 (2018), p. 1831.

[23] Zaki Brahmi and Mohamed Mohsen Gammoudi. "Semantic shared space-based complex tasks allocation method for massive MAS". In: *Proceedings - 2009 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009*. 2009, pp. 428–434. ISBN: 9781424445196. DOI: 10.1109/ICCSIT.2009.5234814.

[24] Antonio Brogi and Paolo Ciancarini. "The concurrent language, shared prolog". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 13.1 (1991), pp. 99–123.

[25] James Brown and Mohd Anwar. "Blacksite: human-in-the-loop artificial immune system for intrusion detection in internet of things". In: *Human-Intelligent Systems Integration* 3.1 (2021), pp. 55–67.

[26] Jean-michel Bruel et al. "The role of formalism in system requirements". In: *dl.acm.org* 1.5 (June 2021), p. 35. DOI: 10.1145/3448975.

[27] Peter Brusilovsky. "Adaptive hypermedia". In: *User modeling and user-adapted interaction* 11.1 (2001), pp. 87–110.

[28] Carlos Buil-Aranda et al. "Federating queries in SPARQL 1.1: Syntax, semantics and evaluation". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 18.1 (2013). Special Section on the Semantic and Social Web, pp. 1–17. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2012.10.001.

[29] Jean Paul Calbimonte. "Linked data notifications for RDF streams". In: *CEUR Workshop Proceedings* 1936 (2017), pp. 66–73. ISSN: 16130073.

[30] Jean Paul Calbimonte, Davide Calvaresi, and Michael Schumacher. "Multi-agent interactions on the web through linked data notifications". In: *Lecture*

Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10767 LNAI.January (2018), pp. 44–53. ISSN: 16113349. DOI: 10.1007/978-3-030-01713-2_4.

[32] Diego Calvanese et al. "Ontop: Answering SPARQL queries over relational databases". In: *Semantic Web* 8.3 (2017), pp. 471–487.

[33] Davide Calvaresi et al. "Explainable multi-agent systems through blockchain technology". In: *International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems*. Springer. 2019, pp. 41–58.

[34] Davide Calvaresi et al. "Multi-agent systems and blockchain: Results from a systematic literature review". In: *International conference on practical applications of agents and multi-agent systems*. Springer. 2018, pp. 110–126.

[35] K. Selçuk Candan, Huan Liu, and Reshma Suvarna. "Resource Description Framework: Metadata and Its Applications". In: *SIGKDD Explor. Newsl.* 3.1 (July 2001), pp. 6–19. ISSN: 1931-0145. DOI: 10.1145/507533.507536.

[36] Alda Canito et al. "Semantic web services for multi-agent systems interoperability". In: *EPIA Conference on Artificial Intelligence*. Springer. 2019, pp. 606–616.

[37] Sarven Capadisli et al. "Decentralised authoring, annotations and notifications for a read-write web with Dokieli". In: *International Conference on Web Engineering*. Springer. 2017, pp. 469–481.

[38] Sarven Capadisli et al. "Linked data notifications: A resource-centric communication protocol". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10249 LNCS (2017), pp. 537–553. ISSN: 16113349. DOI: 10.1007/978-3-319-58068-5_33.

[39] Nicholas Carriero, David Gelernter, and Lenore Zuck. "Bauhaus Linda". In: *European Conference on Object-Oriented Programming*. Springer. 1994, pp. 66–76.

[41] Roberto Casadei et al. "Tuple-Based Coordination in Large-Scale Situated Systems". In: *International Conference on Coordination Languages and Models*. Springer. 2021, pp. 149–167.

[42] Walid Chainbi. "Modeling multi-agent systems as labeled transitions systems: A unifying approach". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5559 LNAI. 2009, pp. 131–140. ISBN: 3642016642. DOI: 10.1007/978-3-642-01665-3_14.

[44] Xiaoyu Chen and Ran Jin. "Adapipe: A recommender system for adaptive computation pipelines in cyber-manufacturing computation services". In: *IEEE Transactions on Industrial Informatics* 17.9 (2020), pp. 6221–6229.

[45] Yixin Chen, Chih-Wei Hsu, and Benjamin W Wah. "SGPlan: Subgoal partitioning and resolution in planning". In: *Edelkamp et al.(Edelkamp, Hoffmann, Littman, & Younes, 2004)* (2004).

[46] Raymond Chiong. *Nature-inspired algorithms for optimisation*. Vol. 193. Springer, 2009.

[47] Smriti Chopra et al. "A Distributed Version of the Hungarian Method for Multirobot Assignment". In: *IEEE Transactions on Robotics* 33.4 (2017), pp. 932–947. ISSN: 15523098. DOI: 10.1109/TRO.2017.2693377. arXiv: 1805.08712.

[48] P Ciancarini, R Gorrieri, and G Zavattaro. "Towards a calculus for generative communication". In: *Formal Methods for Open Object-Based Distributed Systems*. Springer, 1997, pp. 283–297.

[49] Vincent A. Cicirello and Stephen F. Smith. "Wasp-like agents for distributed factory coordination". In: *Autonomous Agents and Multi-Agent Systems* 8.3 (2004), pp. 237–266. ISSN: 13872532. DOI: 10.1023/B:AGNT.0000018807.12771.60.

[50] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. "Engineering World-Wide Multi-Agent Systems with Hypermedia". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11375 LNAI. Springer Verlag, 2019, pp. 285–301. ISBN: 9783030256920.

[51] Andrei Ciortea et al. "A decade in hindsight: The missing bridge between multi-agent systems and the World Wide Web". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*. Vol. 3. 2019, pp. 1659–1663. ISBN: 9781510892002.

[52] Andrei Ciortea et al. "Exploiting Interaction Affordances: On Engineering Autonomous Systems for the Web of Things". In: *Second W3C Workshop on the Web of Things: The Open Web to Challenge IoT Fragmentation, Munich, Germany* (2019).

[53] Tiago Cogumbreiro, Francisco Martins, and Vasco T. Vasconcelos. "Compiling the π-calculus into a Multithreaded Typed Assembly Language". In: *Electronic Notes in Theoretical Computer Science* 241.C (2009), pp. 57–84. ISSN: 15710661. DOI: 10.1016/j.entcs.2009.06.004.

[54] Michele Colledanchise, Richard M Murray, and Petter Ögren. "Synthesis of correct-by-construction behavior trees". In: (2017).

[55] World Wide Web Consortium et al. *RDF 1.1 Turtle: terse RDF triple language*. Tech. rep. World Wide Web Consortium, 2014.

[56] Krzysztof Czarnecki and Ulrich W Eisenecker. *Generative programming*. Addison-Wesley, 2000.

[57] Toni Dahl et al. "A Virtual World Web Client Utilizing an Entity-Component Model." In: *NGMAST*. IEEE, 2013, pp. 7–12.

[58] Souripriya Das, Seema Sundara, and Richard Cyganiak. "R2RML: RDB to RDF Mapping Language". In: *W3C Recommendation* September 2012 (2012), pp. 1–34. ISSN: 1098-6596.

[59] Rocco De Nicola, Luca Di Stefano, and Omar Inverso. "Multi-agent systems with virtual stigmergy". In: *Science of Computer Programming* 187 (2020), p. 102345. ISSN: 01676423. DOI: 10.1016/j.scico.2019.102345.

[61] M Vasumathi Devi, E Laxmi Lydia, and Hima Bindu Gogineni. "Artificial Immune Traffic Prediction". In: *Computational Intelligence for Sustainable Transportation and Mobility* 1 (2021), pp. 32–48.

[62] Sergio Di Martino and Silvia Rossi. "An architecture for a mobility recommender system in smart cities". In: *Procedia Computer Science* 98 (2016), pp. 425–430.

[63] Massimo Di Pierro. "What is the blockchain?" In: *Computing in Science & Engineering* 19.5 (2017), pp. 92–95.

[64] Oğuz Dikenelli, Oylum Alatlı, and Rıza Cenk Erdur. "Where are all the semantic web agents: Establishing links between agent and linked data web through environment abstraction". In: *Agent Environments for Multi-Agent Systems IV*. Springer, 2015, pp. 41–51.

[65] Anastasia Dimou et al. "Extending R2RML to a source-independent mapping language for RDF". In: *CEUR Workshop Proceedings*. Vol. 1035. 2013, pp. 237–240.

[66] Anastasia Dimou et al. "Mapping Hierarchical Sources into RDF using the RML Mapping Language". In: (2014).

[67] Anastasia Dimou et al. "RML: A generic language for integrated RDF mappings of heterogeneous data". In: *CEUR Workshop Proceedings*. Vol. 1184. 2014.

[68] Aiden Dipple, Kerry Raymond, and Michael Docherty. "General Theory of Stigmergy: Modelling Stigma Semantics". In: *Elsevier* (2014). DOI: 10.1016/j.cogsys.2014.02.002.

[69] Aiden Dipple, Kerry Raymond, and Michael Docherty. "Stigmergy within Web Modelling Languages : Positive Feedback Mechanisms". In: *eprints.qut.edu.au* (2013).

[70] Aiden Charles Dipple. "Standing on the Shoulders of Ants: Stigmergy in the Web". In: *Proceedings of the 20th international conference companion on World wide web*. 2011, pp. 355–360.

[71] Marco Dorigo and Christian Blum. "Ant colony optimization theory: A survey". In: *Information Sciences* (2005).

[72] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. "Ant algorithms and stigmergy". In: *Future Generation Computer Systems* 16.8 (2000), pp. 851–871. ISSN: 0167739X. DOI: 10.1016/S0167-739X(00)00042-X.

[73] Marco Dorigo and Gianni Di Caro. "Ant colony optimization: A new metaheuristic". In: *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*. Vol. 2. IEEE Computer Society, 1999, pp. 1470–1477. DOI: 10.1109/CEC.1999.782657.

[74] Marco Dorigo and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66.

[75] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. "Ant system: Optimization by a colony of cooperating agents". In: *IEEE Transactions on Systems,*

Man, and Cybernetics, Part B: Cybernetics 26.1 (1996), pp. 29–41. ISSN: 10834419. DOI: 10.1109/3477.484436.

[76] Marco Dorigo and Thomas Stützle. "Ant colony optimization: Overview and recent advances". In: *International Series in Operations Research and Management Science*. Vol. 272. Springer New York LLC, 2019, pp. 311–351. DOI: 10.1007/978-3-319-91086-4_10.

[77] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. "Multi-Agent Systems: A Survey". In: *IEEE Access* 6 (Apr. 2018), pp. 28573–28593. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2831228.

[78] Laurent Doyen et al. "Verification of Hybrid Systems". In: *Handbook of Model Checking* (May 2018), pp. 1047–1110. DOI: 10.1007/978-3-319-10575-8_30.

[79] Y Dumond and C. Roche. "Formal specification of a multi-agent system architecture for manufacture: the contribution of the π-calculus". In: *Journal of Materials Processing Technology* 107.1-3 (2000), pp. 209–215. ISSN: 09240136. DOI: 10.1016/S0924-0136(00)00712-3.

[80] Martin Dürst and Michel Suignard. *Internationalized resource identifiers (IRIs)*. Tech. rep. RFC 3987, January, 2005.

[81] Stefan Edelkamp, Shahid Jabbar, and Mohammed Nazih. "Large-scale optimal PDDL3 planning with MIPS-XXL". In: *5th International Planning Competition Booklet (IPC-2006)* (2006), pp. 28–30.

[82] Jacques Ferber and Jean-Pierre Miiller. *Influences and Reaction : a Model of Situated Multiagent Systems*. Tech. rep. 1996.

[83] Tiago M Fernandez-Carames and Paula Fraga-Lamas. "A review on the application of blockchain to the next generation of cybersecure industry 4.0 smart factories". In: *Ieee Access* 7 (2019), pp. 45201–45218.

[85] Roy Fielding. *Re: draft findings on Unsafe Methods (whenToUseGet-7)*. Online; accessed Apr.2021, 2002. URL: https://lists.w3.org/Archives/Public/www-tag/2002Apr/0207.html (visited on 03/31/2021).

[86] Roy Fielding. "Representational state transfer". In: *Architectural Styles and the Design of Netowork-based Software Architecture* (2000), pp. 76–85.

[87] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. Vol. 7. University of California, Irvine Irvine, 2000.

[88] Merrill M Flood. "The traveling-salesman problem". In: *Operations research* 4.1 (1956), pp. 61–75.

[89] Martin Fowler. "Richardson Maturity Model: steps toward the glory of REST". In: *Online at http://martinfowler. com/articles/richardsonMaturityModel. html* (2010).

[90] Felix Freiberger and Holger Hermanns. "Concurrent Programming from pseuCo to Petri". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11522 LNCS. 2019, pp. 279–297. ISBN: 9783030215705. DOI: 10.1007/978-3-030-21571-2_16.

[92] Michael R Genesereth and Nils J Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 2012.

[93] Alfonso E Gerevini et al. "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners". In: *Artificial Intelligence* 173 (2009), pp. 619–668. DOI: 10.1016/j.artint.2008.10.012.

[94] L Mary Gladence et al. "Recommender system for home automation using IoT and artificial intelligence". In: *Journal of Ambient Intelligence and Humanized Computing* (2020), pp. 1–9.

[96] PP Grassé. "Les phénomènes sociaux chez les animaux". In: *Cahiers de l'Institut de Science économique appliquée* (1963), pp. 7–23.

[97] Markus Graube et al. "Linked Data as integrating technology for industrial data". In: *International Journal of Distributed Systems and Technologies (IJDST)* 3.3 (2012), pp. 40–52.

[98] Julie Greensmith, Amanda Whitbrook, and Uwe Aickelin. "Artificial Immune systems". In: *Natural Computing Series*. Vol. 28. 2015, pp. 301–332. DOI: 10.1007/978-3-662-43631-8_16.

[99] Stefan Grobbelaar and Mihaela Ulieru. "Holonic stigmergy as a mechanism for engineering self-organizing applications". In: *3rd International Conference of Informatics in Control, Automation and Robotics*. 2006, pp. 1–5.

[100] Abi Grogan. "Smart farming". In: *Engineering & Technology* 7.6 (2012), pp. 38–40.

[101] Dominique Guinard and Vlad Trifa. "Towards the web of things: Web mashups for embedded devices". In: *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*. Vol. 15. 2009.

[103] Shubham Gupta et al. "Karma: A system for mapping structured sources into the semantic web". In: *Extended Semantic Web Conference*. Vol. 7540. Springer, Berlin, Heidelberg, 2012, pp. 430–434. ISBN: 9783662466407.

[104] Bastiaan Haaksema. "Executable Specifications of Message-based Concurrency in Maude". PhD thesis. University of Groningen, 2021.

[105] Karuna Hadeli et al. "Multi-agent coordination and control using stigmergy". In: *Computers in Industry* 53.1 (2004), pp. 75–96. ISSN: 01663615. DOI: 10.1016/S0166-3615(03)00123-4.

[106] Karuna Hadeli et al. "Self-organising in multi-agent coordination and control using stigmergy". In: *International Workshop on Engineering Self-Organising Applications*. Springer. 2003, pp. 105–123.

[107] Ali-Mohammad Hariri, Maryam A Hejazi, and Hamed Hashemi-Dezaki. "Investigation of impacts of plug-in hybrid electric vehicles' stochastic characteristics modeling on smart grid reliability under different charging scenarios". In: *Journal of Cleaner Production* 287 (2021), p. 125500.

[108] Andreas Harth and Tobias Käfer. "Towards specification and execution of linked systems". In: *CEUR Workshop Proceedings*. Vol. 1594. 2016, pp. 62–67.

[109] Mark Hawkins et al. "Cyber-physical production networks, internet of things-enabled sustainability, and smart factory performance in industry 4.0-based manufacturing systems". In: *Economics, Management, and Financial Markets* 16.2 (2021), pp. 73–83.

[110] Tom Heath and Christian Bizer. "Linked data: Evolving the web into a global data space". In: *Synthesis lectures on the semantic web: theory and technology* 1.1 (2011), pp. 1–136.

[111] R. Hedayatzadeh et al. "Termite colony optimization: A novel approach for optimizing continuous problems". In: *2010 18th Iranian Conference on Electrical Engineering*. 2010, pp. 553–558. DOI: 10.1109/IRANIANCEE.2010.5507009.

[112] James Hendler. "Agents and the semantic web". In: *IEEE Intelligent systems* 16.2 (2001), pp. 30–37.

[113] Francis Heylighen. "Mediator Evolution: a general scenario for the origin of dynamical hierarchies". In: *Worldviews, Science and Us.(Singapore: World Scientific)* 44 (2006), pp. 45–48.

[114] Francis Heylighen. "Stigmergy as a generic mechanism for coordination : definition , varieties and aspects". In: *Cognition* (2011), pp. 1–23. ISSN: 13890417.

[115] Francis Heylighen. "Stigmergy as a Universal Coordination Mechanism: components, varieties and applications". In: *Human Stigmergy: Theoretical Developments and New Applications; Springer: New York, NY, USA* (2015).

[116] Francis Heylighen. "Stigmergy as a universal coordination mechanism I: Definition and components". In: *Cognitive Systems Research* 38 (2016), pp. 4–13. ISSN: 13890417. DOI: 10.1016/j.cogsys.2015.12.002.

[117] Francis Heylighen. "Stigmergy as a universal coordination mechanism II: Varieties and evolution". In: *Cognitive Systems Research* 38 (2016), pp. 50–59. ISSN: 13890417. DOI: 10.1016/j.cogsys.2015.12.007.

[118] Francis Heylighen and Clément Vidal. "Getting things done: the science behind stress-free productivity". In: *Long Range Planning* 41.6 (2008), pp. 585–605.

[119] Guillaume Hillairet, Frédéric Bertrand, Jean Yves Lafaye, et al. "Bridging EMF applications and RDF data sources". In: *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, SWESE*. 2008.

[120] Pascal Hitzler. "A review of the semantic web field". In: *Communications of the ACM* 64.2 (2021), pp. 76–83.

[121] Edmund R. Hunt, Simon Jones, and Sabine Hauert. "Testing the limits of pheromone stigmergy in high-density robot swarms". In: *Royal Society Open Science* 6.11 (2019). ISSN: 20545703. DOI: 10.1098/rsos.190225.

[122] M Usman Iftikhar and Danny Weyns. "Activforms: Active formal models for self-adaptation". In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2014, pp. 125–134.

[123] Markel Iglesias-Urkia et al. "Automatic generation of web of things servients using thing descriptions". In: *Personal and Ubiquitous Computing* (2020), pp. 1–17.

[124] *Internet of Things (IoT) Markets: A Global Outlook*. 2019. URL: https://www.researchandmarkets.com/reports/4768775/internet-of-things-iot-markets-a-global-outlook.

[125] Damian Isla. *GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI*. Last accessed 12 April 2022. 2005. URL: https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai.

[126] Aleksandar Jevtić et al. "Distributed bees algorithm for task allocation in swarm of robots". In: *IEEE Systems Journal* 6.2 (2012), pp. 296–304. ISSN: 19328184. DOI: 10.1109/JSYST.2011.2167820.

[127] Benjamin Jochum et al. "Data-Driven Workflows for Specifying and Executing Agents in an Environment of Reasoning and RESTful Systems". In: *Lecture Notes in Business Information Processing*. Vol. 362 LNBIP. Springer, 2019, pp. 93–105. ISBN: 9783030374525. DOI: 10.1007/978-3-030-37453-2_9.

[128] Tobias Käfer and Andreas 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 27th Web Conference (WWW)*. 2018.

[129] Ryo Kanamori, Jun Takahashi, and Takayuki Ito. "Evaluation of traffic management strategies with anticipatory stigmergy". In: *Journal of Information Processing* 22.2 (2014), pp. 228–234. ISSN: 18826652. DOI: 10.2197/ipsjjip.22.228.

[130] Paris C. Kanellakis and Scott A Smolka. "CCS expressions, finite state processes, and three problems of equivalence". In: *Information and Computation* 86.1 (1990), pp. 43–68. ISSN: 10902651. DOI: 10.1016/0890-5401(90)90025-D.

[131] Aleksandr Kapitonov et al. "Blockchain-based protocol of autonomous business activity for multi-agent systems consisting of UAVs". In: *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. IEEE. 2017, pp. 84–89.

[132] Antonín Karola and Jakub Klímek. *Inbox-a messaging app based on Linked Data Notifications and Solid*. Tech. rep. 2021.

[133] David Keil and Dina Goldin. "Indirect interaction in environments for multi-agent systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 3830 LNAI. 2006, pp. 68–87. ISBN: 3540326146. DOI: 10.1007/11678809_5.

[134] Rudolf K Keller and Reinhard Schauer. "Design components: Towards software composition at the design level". In: *Proceedings of the 20th International Conference on Software Engineering*. IEEE. 1998, pp. 302–311.

[135] Mike Kelly. *JSON Hypertext Application Lanuage*. 2012. URL: https://datatracker.ietf.org/doc/html/draft-kelly-json-hal-00 (visited on 06/04/2022).

[136] Abdul Ghaffar Khan et al. "A journey of WEB and Blockchain towards the Industry 4.0: An Overview". In: *2019 International Conference on Innovative Computing (ICIC)*. IEEE. 2019, pp. 1–7.

[137] Gregor Kiczales et al. "Aspect-Oriented Programming". In: *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. Finland: Springer-Verlag, Berlin, Germany, 1997.

[138] Matthias Klusch et al. "Semantic web service search: a brief survey". In: *KI-Künstliche Intelligenz* 30.2 (2016), pp. 139–147.

[139] Ege Korkan et al. "Safe interoperability for web of things devices and systems". In: *Languages, Design Methods, and Tools for Electronic System Design*. Springer, 2020, pp. 47–69.

[140] Peter Korošec, Jurij Šilc, and Bogdan Filipič. "The differential ant-stigmergy algorithm". In: *Information Sciences* (2012). ISSN: 00200255. DOI: 10.1016/j.ins.2010.05.002.

[141] Glenn E Krasner, Stephen T Pope, et al. "A description of the model-view-controller user interface paradigm in the smalltalk-80 system". In: *Journal of object oriented programming* 1.3 (1988), pp. 26–49.

[142] Michael J.B. Krieger, Jean Bernard Billeter, and Laurent Keller. "Ant-like task allocation and recruitment in cooperative robots". In: *Nature* 406.6799 (2000), pp. 992–995. ISSN: 00280836. DOI: 10.1038/35023164.

[143] Uirá Kulesza et al. "A generative approach for multi-agent system development". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 3390 LNCS. Springer Verlag, 2005, pp. 52–69. ISBN: 3540248439. DOI: 10.1007/978-3-540-31846-0_4.

[144] Imad Lamouik, Ali Yahyaouy, and My Abdelouahed Sabri. "Smart multi-agent traffic coordinator for autonomous vehicles at intersections". In: *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*. IEEE. 2017, pp. 1–6.

[145] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. "A semantic web middleware for virtual data integration on the web". In: *European Semantic Web Conference*. Springer. 2008, pp. 493–507.

[146] Heiner Lasi et al. "Industry 4.0". In: *Business & information systems engineering* 6.4 (2014), pp. 239–242.

[147] Ora Lassila, Ralph R Swick, et al. "Resource description framework (RDF) model and syntax specification". In: (1998).

[148] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. "A SPARQL extension for generating RDF from heterogeneous formats". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10249 LNCS. 2017, pp. 35–50. ISBN: 9783319580678.

[149] Bairun Li, Hui Kang, and Fang Mei. "Implementation of distributed multi-agent scheduling algorithm based on Pi-calculus". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11344 LNCS. 2018, pp. 182–195. ISBN: 9783030057541. DOI: 10.1007/978-3-030-05755-8_19.

[150] Liwu Li. "Implementing the π-calculus in Java". In: *Journal of Object Technology* 4.2 (2005), pp. 157–177. ISSN: 16601769. DOI: 10.5381/JOT.2005.4.2.A5.

[151] Shancang Li, Li Da Xu, and Shanshan Zhao. "The internet of things: a survey". In: *Information systems frontiers* 17.2 (2015), pp. 243–259.

[152] Alexandre Linhares and Horacio Hideki Yanasse. "Connections between cutting-pattern sequencing, VLSI design, and flexible machines". In: *Computers & Operations Research* 29.12 (2002), pp. 1759–1772.

[153] Linyuan Lü et al. "Recommender systems". In: *Physics reports* 519.1 (2012), pp. 1–49.

[155] Matt Luckcuck et al. "Formal specification and verification of autonomous robotic systems: A survey". In: *ACM Computing Surveys (CSUR)* 52.5 (2019), pp. 1–41.

[156] Bas Luttik and Fei Yang. "The π-calculus is behaviourally complete and orbit-finitely executable". In: *Logical Methods in Computer Science* 17.1 (2021), 14:1–14:26. ISSN: 18605974. DOI: 10.23638/LMCS-17(1:14)2021.

[158] Naercio Magaia and Zhengguo Sheng. "ReFIoV: A novel reputation framework for information-centric vehicular applications". In: *IEEE Transactions on Vehicular Technology* 68.2 (2019), pp. 1810–1823. ISSN: 00189545. DOI: 10.1109/TVT.2018.2886572.

[159] Hemant B Mahajan and Anil Badarla. "Cross-layer protocol for WSN-assisted IoT smart farming applications using nature inspired algorithm". In: *Wireless Personal Communications* 121.4 (2021), pp. 3125–3149.

[160] Andrés García Mangas and Francisco José Suárez Alonso. "WOTPY: A framework for web of things applications". In: *Computer Communications* 147 (2019), pp. 235–251.

[161] Essam Mansour et al. "A Demonstration of the Solid Platform for Social Web Applications". In: *Proceedings of the 25th International Conference Companion on World Wide Web*. Association for Computing Machinery (ACM), 2016, pp. 223–226. ISBN: 9781450341448. DOI: 10.1145/2872518.2890529.

[162] Stefano Mariani and Andrea Omicini. "Event-driven programming for situated MAS with ReSpecT tuple centres". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8076 LNAI. 2013, pp. 306–319. ISBN: 9783642407758. DOI: 10.1007/978-3-642-40776-5_26.

[163] Stefano Mariani and Andrea Omicini. "Semantic shared space-based complex tasks allocation method for massive MAS". In: *Springer* 8076 LNAI (2013), pp. 306–319. DOI: 10.1007/978-3-642-40776-5_26.

[164] Chris Martens, Eric Butler, and Joseph C. Osborn. "A Resourceful Reframing of Behavior Trees". In: (Mar. 2018). URL: http://arxiv.org/abs/1803.09099.

[165] Jaime A Martins, Andriy Mazayev, and Noélia Correia. "Hypermedia APIS for the Web of Things". In: *Ieee Access* 5 (2017), pp. 20058–20067.

[166] Alejandro Marzinotto et al. "Towards a unified behavior trees framework for robot control". en. In: IEEE, May 2014, pp. 5420–5427. ISBN: 978-1-4799-3685-4.

[167] Maja J. Matarić, Gaurav S. Sukhatme, and Esben H. Østergaard. "Multi-robot task allocation in uncertain environments". In: *Autonomous Robots* 14.2-3 (2003), pp. 255–263. ISSN: 09295593.

[168] Luciano Frontino de Medeiros, Freddy Priyatna, and Oscar Corcho. "MIRROR: Automatic R2RML mapping generation from relational databases". In: *International Conference on Web Engineering*. Springer. 2015, pp. 326–343.

[169] Prem Melville and Vikas Sindhwani. "Recommender systems." In: *Encyclopedia of machine learning* 1 (2010), pp. 829–838.

[170] Franck Michel, Catherine Faron-Zucker, and Johan Montagnat. "A mapping-based method to query MongoDB documents with SPARQL". In: *International Conference on Database and Expert Systems Applications*. Springer. 2016, pp. 52–67.

[171] Franck Michel, Johan Montagnat, and Catherine Faron-Zucker. "A survey of RDB to RDF translation approaches and tools". PhD thesis. I3S, 2014.

[172] Franck Michel et al. "Bridging Web APIs and Linked Data with SPARQL Micro-Services". In: (2018), pp. 187–191.

[173] Franck Michel et al. *SPARQL Micro-Services: Lightweight Integration of Web APIs and Linked Data*. Tech. rep. 2018, pp. 1–10.

[174] Franck Michel et al. "Translation of relational and non-relational databases into RDF with xR2RML". In: *11th International Conference on Web Information Systems and Technologies (WEBIST'15)*. 2015, pp. 443–454.

[175] Franck Michel et al. "xR2RML: Relational and non-relational databases to RDF mapping language". PhD thesis. CNRS, 2017.

[176] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: 3-540-10235-3. DOI: 10.1007/3-540-10235-3.

[177] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[178] Boris Moltchanov and Oscar Rodriguez Rocha. "A context broker to enable future IoT applications and services". In: *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2014 6th International Congress on*. IEEE. 2014, pp. 263–268.

[179] Vasileios Moysiadis et al. "Smart farming in Europe". In: *Computer science review* 39 (2021), p. 100345.

[180] Beata Mrugalska and Magdalena K Wyrwicka. "Towards lean production in industry 4.0". In: *Procedia engineering* 182 (2017), pp. 466–473.

[181] María Navarro-Cáceres et al. "Application of artificial immune system to domestic energy management problem". In: *2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband, ICUWB 2017 - Proceedings*. Vol. 2018-Janua. 2018, pp. 1–7. ISBN: 9781509050079. DOI: 10.1109/ICUWB.2017.8251010.

[182] Hai Nguyen et al. "ROS commander (ROSCo): Behavior creation for home robots". In: *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 467–474.

[183] R. de Nicola, P Inverardi, and M Nesi. "Using the axiomatic presentation of behavioural equivalences for manipulating CCS specifications". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 407 LNCS. 1990, pp. 54–67. ISBN: 9783540521488. DOI: 10.1007/3-540-52148-8_5.

[184] Mark Nottingham. *Web linking*. 2010. URL: https://datatracker.ietf.org/doc/html/rfc5988 (visited on 06/04/2022).

[185] Mahda Noura, Valentin Siegert, and Martin Gaedke. "WAT: Autonomous Hypermedia-driven Web Agents for Web of Things Devices". In: *Proceedings of the All the Agents Challenge co-located with the 20th International Conference on Semantic Web (ISWC)*. Mar. 2022, pp. 38–43.

[186] James J Odell et al. "Modeling agents and their environment". In: *International Workshop on Agent-Oriented Software Engineering*. Springer. 2002, pp. 16–31.

[187] Andrea Omicini and Enrico Denti. "From tuple spaces to tuple centres". In: *Science of Computer Programming* 41.3 (2001), pp. 277–294.

[188] Andrea Omicini, Enrico Denti, and Vladimiro Toschi. "The LuCe coordination technology for MAS design and development on the internet". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1906. 2000, pp. 305–310. ISBN: 9783540410201. DOI: 10.1007/3-540-45263-x_20.

[189] Eyal Oren, Benjamin Heitmann, and Stefan Decker. "ActiveRDF: Embedding Semantic Web data into object-oriented languages". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6.3 (2008), pp. 191–202.

[190] Benedikt Ostermaier, Fabian Schlup, and Kay Römer. "Webplug: A framework for the web of things". In: *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE. 2010, pp. 690–695.

[191] Kevin R Page, David C De Roure, and Kirk Martinez. "REST and Linked Data: a match made for domain driven development?" In: *Proceedings of the Second International Workshop on RESTful Design*. 2011, pp. 22–25.

[192] Fernando Gomes Papi, Jomi Fred Hübner, and Maiquel de Brito. "A Blockchain integration to support transactions of assets in multi-agent systems". In: *Engineering Applications of Artificial Intelligence* 107 (2022), p. 104534.

[193] Savas Parastatidis et al. "The role of hypermedia in distributed system development". In: *Proceedings of the First International Workshop on RESTful Design*. ACM, 2010, pp. 16–22.

[194] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and complexity of SPARQL". In: *ACM Transactions on Database Systems (TODS)* 34.3 (2009), pp. 1–45.

[195] Michael A Peshkin et al. "Cobot architecture". In: *IEEE Transactions on Robotics and Automation* 17.4 (2001), pp. 377–390.

[196] Alberto Petrillo et al. "Adaptive multi-agents synchronization for collaborative driving of autonomous vehicles with multiple communication delays". In: *Transportation research part C: emerging technologies* 86 (2018), pp. 372–392.

[197] Pablo Pico-Valencia, Juan A Holgado-Terriza, and José A Senso. "Towards an Internet of Agents model based on Linked Open Data approach". In: *Autonomous Agents and Multi-Agent Systems* 33.1 (2019), pp. 84–131.

[198] Pablo Pico-Valencia et al. "Towards the internet of agents: an analysis of the internet of things from the intelligence and autonomy perspective". In: *Ingeniería e Investigación* 38.1 (2018), pp. 121–129.

[199] Pedro Pinheiro et al. "Multi-agent systems approach to industry 4.0: Enabling collaboration considering a blockchain for knowledge representation". In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2018, pp. 149–160.

[200] Serguei Popov. "The tangle". In: *ABA Journal* FEB. (2018), pp. 1–25. ISSN: 07470088.

[201] Gilles Privat. "Phenotropic and stigmergic webs: The new reach of networks". In: *Universal Access in the Information Society* 11.3 (2012), pp. 323–335. ISSN: 16155289. DOI: 10.1007/s10209-011-0240-1.

[202] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. "Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph". In: *WWW 2014 - Proceedings of the 23rd International Conference on World Wide Web*. 2014, pp. 479–489. ISBN: 9781450327442.

[203] Lianyong Qi et al. "Finding all you need: web APIs recommendation in web of things through keywords search". In: *IEEE Transactions on Computational Social Systems* 6.5 (2019), pp. 1063–1072.

[204] Lara Quijano-Sánchez et al. "Recommender systems for smart cities". In: *Information systems* 92 (2020), p. 101545.

[205] Katharina Rasch. "An unsupervised recommender system for smart homes". In: *Journal of Ambient Intelligence and Smart Environments* 6.1 (2014), pp. 21–37.

[207] Paul Resnick and Hal R Varian. "Recommender systems". In: *Communications of the ACM* 40.3 (1997), pp. 56–58.

[208] Alessandro Ricci et al. "Cognitive stigmergy: Towards a framework based on agents and artifacts". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4389

LNAI (2007), pp. 124–140. ISSN: 16113349. DOI: 10.1007/978-3-540-71103-2_7.

[209] Leonard Richardson and Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008.

[210] Juan A Rico-Gallego et al. "A survey of communication performance models for high-performance computing". In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–36.

[211] Mariano Rodríguez-Muro and Martin Rezk. "Efficient SPARQL-to-SQL with R2RML mappings". In: *Journal of Web Semantics* 33 (2015), pp. 141–169. ISSN: 15708268.

[212] Luca Roffia. "Extensions–Linked Data Notifications and NGSI-LD". In: *24th Conference of the Open Innovations Association FRUCT*. 2019.

[213] Alireza Roudbari et al. "Resilience-oriented operation of smart grids by rescheduling of energy resources and electric vehicles management during extreme weather condition". In: *Sustainable Energy, Grids and Networks* 28 (2021), p. 100547.

[214] Andrei Vlad Sambra et al. *Solid: a platform for decentralized social applications based on linked data*. Tech. rep. 2016.

[215] Davide Sangiorgi and Xian Xu. "Trees from functions as processes". In: *Logical Methods in Computer Science* 14.3 (2018), pp. 1–41. ISSN: 18605974. DOI: 10.23638/LMCS-14(3:11)2018.

[216] François Scharffe et al. "Enabling Linked Data Publication with the Datalift Platform". In: *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence* (July 2012).

[217] Daniel Schraudner. "Stigmergic Multi-Agent Systems in the Semantic Web of Things". In: *European Semantic Web Conference*. Springer. 2021, pp. 218–229.

[218] Daniel Schraudner and Victor Charpenay. "An HTTP/RDF-Based Agent Infrastructure for Manufacturing Using Stigmergy". In: 01 (2020), pp. 197–202. ISSN: 16113349. DOI: 10.1007/978-3-030-62327-2_34.

[222] Luca Sciullo et al. "WoT Store: Managing resources and applications on the web of things". In: *Internet of Things* 9 (2020), p. 100164.

[223] Björn Segendorf. "What is bitcoin". In: *Sveriges Riksbank Economic Review* 2014 (2014), pp. 2–71.

[224] Sergey Shadrin. "Affordable and efficient autonomous driving in allweather conditions". In: *FISITA World Automotive Congress*. Vol. 2018. 2018.

[225] M Omair Shafiq, Ying Ding, and Dieter Fensel. "Bridging multi agent systems and web services: towards interoperability between software agents and semantic web services". In: *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*. IEEE. 2006, pp. 85–96.

[226] Amit P Sheth. "Changing focus on interoperability in information systems: from system, syntax, structure to semantics". In: *Interoperating geographic information systems*. Springer, 1999, pp. 5–29.

[227] Pierluigi Siano. "Demand response and smart grids—A survey". In: *Renewable and sustainable energy reviews* 30 (2014), pp. 461–478.

[228] André Luis Meneses Silva, José de Jesús Pérez-Alcázar, and Sergio Takeo Kofuji. "Interoperability in semantic Web of Things: Design issues and solutions". In: *International Journal of Communication Systems* 32.6 (2019), e3911.

[229] Jason Slepicka et al. "KR2RML: An alternative interpretation of R2RML for heterogeneous sources". In: *CEUR Workshop Proceedings*. Vol. 1426. 2015.

[230] Gregory J. Smith and John S. Gero. "What does an artificial design agent mean by being 'situated'?" In: *Design Studies* 26 (5 Sept. 2005), pp. 535–561. ISSN: 0142694X. DOI: 10.1016/j.destud.2005.01.001.

[231] Erik Sorensen and M Mikailesc. "Model-view-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technology". In: *MegaByte Journal* 9.4 (2010), pp. 1–19.

[243] Priyanshu Srivastava and Rizwan Khan. "A review paper on cloud computing". In: *International Journal of Advanced Research in Computer Science and Software Engineering* 8.6 (2018), pp. 17–20.

[244] Prasanna Thati, Koushik Sen, and Narciso Martí-Oliet. "An executable Specification of asynchronous Pi-Calculus Semantics and May Testing in Maude 2.0". In: *Electronic Notes in Theoretical Computer Science*. Vol. 71. 2004, pp. 261–281. DOI: 10.1016/S1571-0661(05)82539-3.

[245] Guy Theraulaz and Eric Bonabeau. "A brief history of stigmergy". In: *Artificial life* 5.2 (1999), pp. 97–116.

[246] Jon Timmis et al. "An overview of artificial immune systems". In: *Computation in cells and tissues* (2004), pp. 51–91.

[247] Luca Tummolini and Cristiano Castelfranchi. "Trace signals: The meanings of stigmergy". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4389 LNAI (2007), pp. 141–156. ISSN: 16113349. DOI: 10.1007/978-3-540-71103-2_8.

[248] Alexandros Tzanetos, Iztok Fister Jr, and Georgios Dounias. "A comprehensive database of Nature-Inspired Algorithms". In: *Data in Brief* 31 (2020), p. 105792.

[249] Danai Vachtsevanou et al. "Long-lived agents on the web: Continuous acquisition of behaviors in hypermedia environments". In: *Companion Proceedings of the Web Conference 2020*. 2020, pp. 185–189.

[250] Paul Valckenaers, Martin Kollingbaum, Hendrik Van Brussel, et al. "Multi-agent coordination and control using stigmergy". In: *Computers in industry* 53.1 (2004), pp. 75–96.

[251] Paul Valckenaers et al. "MAS coordination and control based on stigmergy". In: *Computers in Industry* 58.7 (2007), pp. 621–629. ISSN: 01663615. DOI: 10.1016/j.compind.2007.05.003.

[252] Paul Valckenaers et al. "Multi-agent Coordination and Control Using Stigmergy Applied to Manufacturing Control". In: *ECCAI Advanced Course on Artificial Intelligence*. 2001, pp. 317–334. DOI: 10.1007/3-540-47745-4_15.

[253] H. Van Dyke Parunak. "A Survey of Environments and Mechanisms for Human-Human Stigmergy". In: *Environments for Multi-Agent Systems II*. Ed. by Danny Weyns, H. Van Dyke Parunak, and Fabien Michel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 163–186. ISBN: 978-3-540-32615-1.

[254] Ruben Verborgh et al. "Efficient runtime service discovery and consumption with hyperlinked RESTdesc". In: *2011 7th International Conference on Next Generation Web Services Practices*. IEEE. 2011, pp. 373–379.

[255] Joost Verbraeken et al. "A survey on distributed machine learning". In: *ACM Computing Surveys (CSUR)* 53.2 (2020), pp. 1–33.

[256] Mario Vestias and Horacio Neto. "Trends of CPU, GPU and FPGA for high-performance computing". In: *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2014, pp. 1–6.

[259] Achim Walter et al. "Opinion: Smart farming is key to developing sustainable agriculture". In: *Proceedings of the National Academy of Sciences* 114.24 (2017), pp. 6148–6150.

[260] Rolf H Weber and Romana Weber. *Internet of things*. Vol. 12. Springer, 2010.

[261] Yinxing Wei, Shensheng Zhang, and Jian Cao. "Coordination among Multi-agents Using Process Calculus and ECA Rule". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2480 (2002), pp. 456–465. ISSN: 16113349. DOI: 10.1007/3-540-45785-2_36.

[262] Danny Weyns and Tom Holvoet Agentwise. *A Formal Model for Situated Multi-Agent Systems*. Tech. rep. 2004, pp. 1–34.

[263] Danny Weyns and Tom Holvoet. "Model for simultaneous actions in situated multi-agent systems". In: *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*. Vol. 2831. 2003, pp. 105–118. DOI: 10.1007/978-3-540-39869-1_10.

[266] Danny Weyns, H Van Dyke Parunak, and Fabien Michel. *Environments for Multi-Agent Systems: First International Workshop, E4MAS, 2004, New York, NY, July 19, 2004, Revised Selected Papers*. Vol. 1. Springer, 2005.

[267] Danny Weyns et al. "Environment as a first class abstraction in multiagent systems model of the environment". In: *Auton Agent Multi-Agent Syst* 14.1 (Feb. 2007), pp. 5–30. DOI: 10.1007/s10458-006-0012-0.

[268] Dennis Wiebusch and Marc Erich Latoschik. "Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems". In: *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2015 IEEE 8th Workshop on*. IEEE. 2015, pp. 25–32.

[269] Michael Wooldridge. "Intelligent agents". In: *Multiagent systems* 6 (1999).

[270] Bo Yang and Meifang Yang. "Data-driven network layer security detection model and simulation for the Internet of Things based on an artificial immune system". In: *Neural Computing and Applications* 33.2 (2021), pp. 655–666.

[271] Jinhui Yang et al. "An ant colony optimization method for generalized TSP problem". In: *Progress in Natural Science* 18.11 (2008), pp. 1417–1422.

[272] Han Yu, Zhiqi Shen, and Cyril Leung. "From internet of things to internet of agents". In: *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*. IEEE. 2013, pp. 1054–1057.

[273] Xinjia Yu and Tao Cheng. "Research on a Stigmergy-driven & MAS-based Method of Modeling Intelligent System". In: (2020), pp. 1042–1047. DOI: 10.1109/cisp-bmei51763.2020.9263567.

[274] Yichi Zhang et al. "Artificial immune system based intrusion detection in a distributed hierarchical network architecture of smart grid". In: *IEEE Power and Energy Society General Meeting* 43606 (2011), pp. 1–8. ISSN: 19449925. DOI: 10.1109/PES.2011.6039697.

Own Publications

[234] Torsten Spieldenner. "On the Fly SPARQL Execution for Structured Non-RDF Web APIs". In: *Proceedings of the 16th International Conference on Web Information Systems and Technologies. International Conference on Web Information Systems and Technologies (WEBIST-2020), November 3-5*. Ed. by Massimo Marchiori, Francisco Domínguez Mayo, and Joaquim Filipe. SCITEPRESS, 2020. ISBN: 978-989-758-478-7.

[235] Torsten Spieldenner. and André Antakli. "Behavior Trees as executable representation of Milner Calculus notations". In: *2022 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). Web Intelligence and Intelligent Agent Technology (WI-IAT-2022), November 17-20, Ontario, Canada*. Nov. 2022.

[236] Torsten Spieldenner and Melvin Chelli. "Linked Data as Medium for Stigmergy-based Optimization and Coordination". In: *Software Technologies: 16th International Conference, ICSOFT 2021, Virtual Event, July 6–8, 2021, Revised Selected Papers*. Springer. 2022, pp. 1–23.

[237] Torsten Spieldenner. and Melvin Chelli. "Linked Data as Stigmergic Medium for Decentralized Coordination". In: *Proceedings of the 16th International Conference on Software Technologies - ICSOFT,* INSTICC. SciTePress, 2021, pp. 347–357. ISBN: 978-989-758-523-4. DOI: 10.5220/0010518003470357.

[238] Torsten Spieldenner and René Schubotz. "A Linked Data Model-View-* Approach for Decoupled Client-Server Applications". In: *Joint International Semantic Technology Conference*. Springer. 2019, pp. 67–81.

[239] Torsten Spieldenner, René Schubotz, and Michael Guldner. "ECA2LD: From Entity-Component-Attribute runtimes to Linked Data applications". In: *Proceedings of the International Workshop on Semantic Web of Things for Industry 4.0. Extended Semantic Web Conference (ESWC-2018), International Workshop on Semantic Web of Things for Industry 4.0, located at 15th ESWC Conference 2018, June 3-7, Heraklion,, Crete, Greece*. Springer, 2018.

[240] Torsten Spieldenner, René Schubotz, and Michael Guldner. "ECA2LD: Generating Linked Data from Entity-Component-Attribute runtimes". In: *2018 Global Internet of Things Summit (GIoTS)*. IEEE. 2018, pp. 1–4.

[241] Torsten Spieldenner et al. "FiVES: an aspect-oriented Approach for shared Virtual Environments in the Web". In: *The Visual Computer* 34.9 (2018), pp. 1269–1282.

[242] Torsten Spieldenner et al. "FiVES: An Aspect-Oriented Virtual Environment Server". In: *Proceedings of the 2017 International Conference on Cyberworlds. International Conference on Cyberworlds (CyberWorlds-2017), September 20-22, Chester, United Kingdom*. IEEE Xplore, 2017.

Publications with Contributions from me

[6] André Antakli et al. "Agent-based Web Supported Simulation of Human-robot Collaboration". In: *Proc. of the 15th Int. Conf. on Web Information Systems and Technologies (WEBIST)* (2019), pp. 88–99.

[7] André Antakli et al. "Optimized Coordination and Simulation for Industrial Human Robot Collaborations". In: *Web Information Systems and Technologies*. Ed. by Alessandro Bozzon, Francisco José Domínguez Mayo, and Joaquim Filipe. Cham: Springer International Publishing, 2020, pp. 44–68. ISBN: 978-3-030-61750-9.

[43] Victor Charpenay et al. "MOSAIK: A Formal Model for Self-Organizing Manufacturing Systems". In: *IEEE Pervasive Computing* 20.1 (2021), pp. 9–18. ISSN: 15582590. DOI: 10.1109/MPRV.2020.3035837.

[102] Michael Guldner, Torsten Spieldenner, and René Schubotz. "NEXUS: Using Geo-fencing Services without revealing your Location". In: *Proceedings of the 2018 Global Internet of Things Summit. Global Internet of Things Summit (GIoTS-2018), June 4-7, Bilbao, Spain*. IEEE Communications Society, 2018.

[157] Andreas Luxenburger et al. "Augmented Reality for Human-Robot Cooperation in Aircraft Assembly". In: *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. IEEE. 2019, pp. 263–2633.

[219] René Schubotz, Torsten Spieldenner, and Melvin Chelli. "stigLD: Stigmergic Coordination in Linked Systems". In: *Mathematics* 10.7 (2022), p. 1041.

[220] René Schubotz, Torsten Spieldenner, and Melvin Chelli. "`stigLD`: Stigmergic Coordination of Linked Data Agents". In: *The 6th International Conference*

on Bio-inspired Computing: Theories and Applications (BIC-TA 2021). Taiyuan, China, 2021.

[221] René Schubotz et al. "Requirements and Specifications for Robots, Linked Data and all the REST". In: *Proceedings of 2nd Workshop on Linked Data in Robotics and Industry 4.0. (LIDARI-2017), located at Semantics 2017, Amsterdam, Netherlands*. CEUR, 2017.

[233] Daniel Spieldenner and Torsten Spieldenner. "A Smart Integration Layer for Smart City Applications". In: *4th International Conference on Smart Grids and Smart Cities (ICSGSC 2020), Osaka, Japan*. 2020.

[257] Christian Vogelgesang, Torsten Spieldenner, and René Schubotz. "SPARQλ: A functional perspective on Linked Data Services". In: *Semantic Technology. Joint International Semantic Technology Conference (JIST-2018), November 26-28, Awaji City, Hyogo, Japan*. Springer Nature, 2018.

[258] Christian Vogelgesang, Torsten Spieldenner, and René Schubotz. "SPARQλ: SPARQL as a Function". In: *Lecture Notes in Networks and Systems. Future of Information and Communication Conference (FICC-2019), March 14-15, San Francisco, California, USA*. Springer, 2019.