MOSAIK: An Agent-Based Decentralized Control System with Stigmergy for a Transportation Scenario

Sebastian Schmid[1(✉)], Daniel Schraudner[1], and Andreas Harth[1,2]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Chair of Technical Information Systems, Nuremberg, Germany
{sebastian.schmid,daniel.schraudner,andreas.harth}@fau.de
[2] Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits IIS, Division Data Spaces and IoT Solutions, Nuremberg, Germany
andreas.harth@iis.fraunhofer.de

Abstract. We investigate possibilities for implementing the decentralized control of transporters with Semantic Web agents to fulfill a given transportation task. We present the MOSAIK framework as a system to build and simulate agents to control transporters using stigmergy for communication, and self-organize based on local decisions. Our framework uses Semantic Web technologies because the communication paradigm of stigmergy directly maps to the REST constraints of the application architecture of the web. The system achieves self-organization by implementing a combination of simple reflex web agents that coordinate using web resources as environment for stigmergy. Finally, we evaluate our system compared to an agent-based simulation and discuss requirements of decentralized systems on the Semantic Web using stigmergy.

Keywords: Self-organization · Transportation · Multi-agent system

1 Introduction

Research for industrial transportation and intralogistics focuses on decentralized control in transportation systems, promising scalability, adaptivity, and flexibility [6,27], in contrast to centralized control that is limited in data processing and scalability, and imposes the threat of a single point of failure [7].

The most promising approaches for decentralized control in transportation are multi-agent systems (MAS), where decision making to solve transportation orders happens in agents locally [27]. MAS are considered appropriate to solve transportation problems [35], are robust, scalable, and can cope with changing environments efficiently [6]. The question remains, how control of these transporter agents can be assured without losing the advantages of decentralized systems by introducing centralized control? Existing MAS approaches already

This work was funded by the German Federal Ministry of Education and Research through the MOSAIK project (grant no. 01IS18070A).

C. Pesquita et al. (Eds.): ESWC 2023, LNCS 13870, pp. 697–714, 2023.
https://doi.org/10.1007/978-3-031-33455-9_41

try to implement decentralized control, e.g. based on auctions [8], forecasting [22], negotiation [10], or hybrid approaches like local coordination and plan merging [1], but often centralized components remain as part of control.

We believe that an agent-based decentralized control system that fulfills the advantages of scalability, adaptivity, and flexibility needs to possess at least the following properties:

1. **Stateless, reactive agents**: Agents shall possess no internal states and only react to their perceived environment, as so-called simple reflex agents (SRA) [25]. SRAs scale well as SRAs do not get more computationally expensive with an increased problem size and still give a coherent outcome [34].
2. **Indirect communication**: The results of SRA's actions are not saved as agent state, but applied to the environment. Agents use their common environment as medium to communicate indirectly with each other, which is called stigmergy [20]. Indirect communication reduces coupling between agents, keeps the population flexible, and leads to resilience against breakdowns of individual agents by distribution of knowledge [3].
3. **Loose coupling between agents**: Agents shall not make assumptions about the agent population and only focus on their own task. For a single agent there shall be no difference if only itself or multiple agents are in the system. The system's adaptive behavior emerges from indirect interactions between agents [18].
4. **Use of local information**: Agents shall use only and thus adapt to locally available information to decide on their actions [17]. Together with stigmergy, the use of local information agents may achieve an overall goal by self-organization via explicit, directive stigmergy [33] that communicates opportunities and goal accomplishments between agents.

We put an emphasis on statelessness, indirect communication and loose coupling between agents, as we believe these form the major requirements for a decentralized control system.

To motivate the problem, we introduce the example of a driverless transportation systems (DTS) whose transporters shall be controlled by agents we realize with the MOSAIK framework [5]. We discern active components, agents, and reactive components, called artifacts that respond to agents' actions, and form together the agents' environment. The agents shall self-coordinate to explore the shop floor in a decentralized manner and use exclusively indirect communication. We assume that all data for agents to decide on is available in RDF (Resource Description Framework) in their environment. The environment shall be accessible for agents via a RESTful Read-Write Linked Data interface as well. We introduce our running example of a transportation scenario in Sect. 2.

We propose to use the advantages of stigmergy, that uses only indirect, decentralized communication for decentralized self-organization and control to keep the advantages of the overall decentralized system. Here, the Semantic Web offers a uniform interface for interaction between agents and artifacts as environment to share data with a common understanding by using established web technologies, as also the rising recent interest in the combination of autonomous

agents and the Semantic Web shows [4]. Our presented combination of SRAs as stateless and independent agents, stigmergy for indirect communication, and the exclusive use of local information leads to a decentralized control system that consists of flexible software agents that can run in the cloud as well as on the edge, use local information without the need to crawl RDF graphs extensively, scale easily with problem sizes, and are resilient with graceful degradation, as agents' data survives in the environment.

We present our system, based on the MOSAIK model [5], consisting of transporters and workstations, which are reactive machines (called artifacts), and transporter agents, which control transporter artifacts via Linked Data technologies that is RDF as data model and HTTP for communication via RESTful interfaces. We built a decentralized transportation system according to the paradigms of stigmergy and Read-Write Linked Data, fulfilling the demands of Industry 4.0 for an agent-based system on state-of-the-art web technology level. Our contributions are:

- We implement the MOSAIK data model and interface for decentralized agents to control decentralized transporter artifacts via Linked Data.
- We present a formalization of the system's demanded behavior and implement the behavior in form of condition-action rules and derivation rules for agents, and state change rules and request processing rules for artifacts.
- We evaluate the performance of our system for a transportation task and compare the outcome to an agent-based simulation.

2 Running Example and Transportation Scenario

We introduce our example used throughout the paper of a transportation scenario, also depicted in Fig. 1.

Example 1. A shop floor, represented by a 3×6 grid, contains `transporter1` at (2, 2), `transporter5` at (3, 1), and two colored stations `station1` at (0, 2) and `station2` at (5, 2), all as artifacts. `Station1` is a blue station and accepts blue products, `station2` is a green station and accepts green products. Agents control the transporters, here `ldfu5` controls `transporter5`.

`Transporter5` holds a blue product, so `ldfu5`'s reflexes try to find suitable adjacent floor tiles for `transporter5` to move closer to `station1` to deliver the product. Agents may only perceive information that is locally available to their controlled transporter, e.g. for `ldfu5` `transporter5`'s floor tile (3, 1), and the eight adjacent fields. Meanwhile, `ldfu1`, controlling transporter1, evaluated the surrounding floor tiles of `transporter1` and perceived a so-called stigmergy mark (blue arrow) at (1, 2), which helps agents to find the nearest station by following the gradient in the direction of the respective station (see Sect. 3). `ldfu1` creates a stigmergy mark at (2, 2) to extend the stigmergy gradient. `ldfu5` orders `transporter5` to follow these stigmergy marks successively to eventually arrive at `station1`. Therefore, `ldfu5` perceives the now created blue stigmergy mark at (2, 2), concludes that `transporter5` will get closer to `station1` by moving to (2, 2), and thus orders `transporter5` to move to (2, 2).

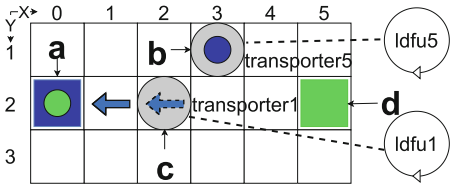


Fig. 1. Detail view of the used shop floor scenario: a) Blue `station1` with green product, b) `transporter5` with blue product, c) empty `transporter1`, d) empty green `station2`. All components are located on the shop floor's tiles (black and white grid). The blue arrows symbolize a gradient of blue stigmergy markers that point the way to the blue `station1` (the dashed stigmergy mark was just created). The respective transporters are controlled by agents (Color figure online).

Example 1 emphasizes our discussed properties from Sect. 1:

1. Stateless, reactive agent: `ldfu5` does not save information in internal states and only reacts to outside stimuli like the sigmergy mark at (2, 2)
2. Indirect communication: `ldfu1` saves stigmergy markers in the environment at (2, 2) to point the way to stations
3. Loose coupling between agents: `ldfu5` requires no knowledge about `ldfu1` and only perceives `ldfu1`'s action of putting a blue stigmergy mark to (2, 2)
4. Local information: `ldfu5` can only perceive the floor tiles that are next to `transporter5`'s location at (3, 1)

Example 1 and Fig. 1 are only a part of the bigger overall scenario, where transporters, as artifacts, have to fulfill transportation tasks [26]: A square shop floor with 7×7 floor tiles has four stations, each with one of four different colors green, red, blue and yellow. A station accepts only products of its own color. Stations randomly produce a product of a different color when their output port is empty. Three transporters have to bring colored products to the correct station of the same color. The agent controlling each transporter evaluates the transporter's own field and the eight adjacent fields.

Agents leave colored marks with values on floor tiles that gets higher the further away the tile is from a perceived station. An agent that follows the descending values will end up at a station [17]. Agents have no model of the shop floor, and do not communicate directly with other agents. The presented, simple scenario can be easily extended to a manufacturing shop floor that produce variants of a product along workstation, as for small batches of customer-specified products are manufactured, e.g. modular smartphones with different processor, memory size, and display as in [28].

3 Theoretical and Technical Background

3.1 Agent-Based Systems and Stigmergy

Russell and Norvig [25] describe the basic agent program for simple reflex agents (SRA), where an SRA is described as simplest possible agent form. SRAs base

their actions only on their current perception during a so called perception-thought-action cycle, but have no memory. A set of condition-action rules gives the behavior, so called reflexes. A reflex triggers an action whenever the respective condition is fulfilled.

Stigmergy is the use of asynchronous interaction and information exchange between agents exclusively through changes in their environment, but not directly with each other [20]. Stigmergy is inspired by the indirect communication of insects like termites [11]. Stigmergy is the base for algorithms coming from ants [9] and has widespread usage [33,34]. The self-organization and coordination of agents is discussed in e.g. [12]. By changing and evaluating their environment, agents influence each others behavior indirectly [17,32]. Our SRAs manipulate their local environment based on their given rules [25]. Agents evaluate their surroundings for manipulations by other agents which then influences their own respective rules, which leads to indirect communication and achieving an overall goal by self-organization via explicit, directive stigmergy [33].

3.2 MOSAIK

We give the definitions of the MOSAIK model [5] for agents and artifacts, with focus on decentralization and stigmergy. Furthermore, we present the resources that were created and used during our research project to realize the model's vision for self-organized transportation in combination with the Semantic Web. We derive our formal definitions from Charpenay et al. [4].

Definition 1 (SRA). *We define simple reflex agents as proactive components (as opposed to artifacts) that make decisions on their own. Agents implement a perception-thought-action cycle to influence their environment, based on defined condition-action rules. We define an agent A as the tuple $A = \langle G, perc, appl, act \rangle$ where*

$$G, \textit{ set of all RDF graphs, representing the agent's possible knowledge}$$
$$perc : D' \times G \to G, \textit{ perception function, based on perceived environment states,}$$
$$appl : G \to G, \textit{ function for derivation rules,}$$
$$act : G \to O, \textit{ function for condition-action rules,}$$
$$\textit{with } O, \textit{ set of operations, as } \{GET, PUT, POST, DELETE\} \times U \times G$$
$$\textit{and } O_{GET} \textit{ as } \langle GET, u, \emptyset \rangle$$

where D' is the set of all finite datasets (the percepts) and D is the set of all RDF datasets (the environmental states) with $D' \subset D$ (cf. Definition 2).

Agent perception and actions are implemented via HTTP requests. Note that *act* realizes actions to influence the environment via unsafe requests (PUT, POST, DELETE) as well the agent's perception (GET), cf. Charpenay et al. [4].

Definition 2 (Artifact). *We define artifacts as reactive components (as opposed to agents) that form the agents' environment following [4]. Artifacts provide different ways for interaction via HTTP request, e.g. reading the artifact's*

state or manipulating the artifact's state via tasks. An internal logic defines the deterministic reaction to interaction, but artifacts do not act autonomously. We define the set of all artifacts as environment $E = \langle D, d_0, O, transfer, update, evolve \rangle$ *where*

$$D, \textit{set of all possible environment states as set of all RDF datasets}$$
$$d_0, \textit{the environment's initial state}$$
$$transfer : O_{GET} \times D \rightarrow D', \textit{function for agents to retrieve a finite percept } D' \subset D$$
$$update : (O \backslash O_{GET}) \times D \rightarrow D, \textit{effectory function based on operations}$$
$$evolve : D \rightarrow D, \textit{function for the environment's own developing state change.}$$

Artifact behavior is a black box for agents, but the behavior's result is observable via state changes and the *transfer* function. State changes can be triggered not only by agents, but also by the physical environment (e.g. machine error or mechanical overload), represented by *evolve*. In our setting, artifacts are transporter devices, workstations, and manipulable floor tiles.

Table 1 shows as a concluding comparison of agent and artifact rules, how the presented formal definitions of Sect. 1 and 2 map to each other, and how we implemented the rules (cf. Sect. 3.3 and 3.4).

Table 1. Comparison of defined rules for agents and artifacts

	Agents	Artifacts
Rule Definition Implementation	Derivation rules *appl* N3 Rules without HTTP Requests in Header	State change rules *evolve* SPARQL-Insert-Delete
Rule Definition Implementation	Condition-action rules *act* N3 Rules with HTTP Requests in Header	Request processing rule *update* Built-in BOLD rule

Interface. In order for the agents to be able to communicate with the artifacts via HTTP, we defined a common interface in MOSAIK which is based on Linked Data Platform [30] for Read-Write Linked Data. In line with the Interaction Affordances from Web of Things Thing Descriptions [21], agents interact with artifacts by either reading or writing their properties, by submitting tasks[1] to them, or by watching events:

– Agents read properties by sending a GET request to the URI of the artifact, write properties by sending a PUT request to the URI of the artifact.

[1] The difference between writing a property and submitting a task is that writing has to happen instantaneously while the task triggers an action that can take more time.

– A new task can be submitted to an artifact by sending a POST request to the task queue.
– Events can be watched by polling the event container via GET requests.

An extended description of our interface can be found in [29]. The rules to implement this interface on the artifact side can be found in Sect. 4.2. For our scenario we use tasks for transporter movement and properties for setting the stigmergy markers on the floor tiles; events are not used.

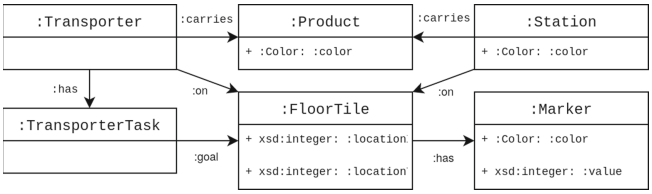


Fig. 2. Class diagram of the data model

Data Model. In our data model (Fig. 2) we have three types of artifacts that can be manipulated by agents: `:Transporter`, `:Station`, and `:FloorTile`, where each `:FloorTile` has a 2D location and each `:Transporter` and `:Station` has a reference to the floor tile they are standing on. Each `:FloorTile` furthermore has stigmergy `:Marker`s for each color that save the stigmergy value for the color. `:Transporters` are moved by agents by submitting `:TransporterTask`s specifying the target `:FloorTile` to the task container of the `:Transporter`. `:Transporter`s and `:Station`s may hold a `:Product`[2]. `:Product`s and `:Station`s have `:Color`s (which must match for a `:Station` to accept a `:Product`). We created the ARENA vocabulary[3] to be able to use the data model from above in RDF. We use as its prefix `arena`.

3.3 Implementation of Agents

The data processing system Linked-Data Fu (ldfu)[4] [16] can retrieve, process and modify Linked Data based on logical rules and production rules in Notation3 (N3) [2]. We use N3 to implement the condition-action rules describing the behavior of the agent which controls the transporters, and interact with the artifacts via RESTful interfaces. ldfu has rules of the form $\{b_1 ... b_n\}$`log:implies`$\{h\}$., where we use `log:implies` (from the namespace `log`[5]) to express implication [15]. These rules realize the ldfu's *internalize* and

[2] `:Product`s are not manipulated by the agents directly, but merely indirectly as consequence of manipulations of `:Transporter`s or `:Station`s.
[3] https://solid.ti.rw.fau.de/public/ns/arena#.
[4] https://linked-data-fu.github.io/.
[5] http://www.w3.org/2000/10/swap/log#.

act functions to influence the agent's internal state or send HTTP requests as explained below. The antecedent of the rule, $b_1 ... b_n$, and the conclusion, h, are RDF triple patterns (s, p, o) as

$$(s, p, o) \in (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$$

where a given U is the set of URIs, B a set of blank nodes, and L a set of literals. V is the set of variables that may replace constants to give patterns. The agent's rule set R is a finite set of rules. A conclusion h follows from a graph $g \in G$, if $h \in g$ or $\{b_1 ... b_n\} \subseteq g$.

ldfu has an internal graph (cf. $g \in G$ in Definition 1) that holds the agent's knowledge in volatile storage. During each cycle, ldfu starts by trying to instantiate its given N3 rules. ldfu may send an HTTP request to a URI given in h to interact with its environment, either to perceive (GET) or manipulate artifact states via unsafe requests (PUT, POST, DELETE). We use the namespaces `http`[6] and `httpm`[7] to express these requests. ldfu sends safe requests (cf. *act* O_{GET} in Definition 1) to all defined URIs in valid conclusions and merges the result to its knowledge (cf. *perc* in Definition 1) to apply the N3 rules again (cf. *appl* in Definition 1) - this is done as long as no fixpoint is reached, that is as long as there are still GET requests left to send and new triples are inserted in ldfu's graph. Only then, ldfu instantiates any unsafe requests e.g. to change its environment (cf. *act* with $O \backslash O_{GET}$ in Definition 1). Finally, the internal graph of ldfu is deleted and ldfu will start all over in the next cycle. Note that ldfu does not save triples or graphs between cycles internally (that is, ldfu has no persistent state), but has to apply results to the environment such that they are not lost.

Example 2. `ldfu5` wants to steer `transporter5` to deliver the green product, so `ldfu5` needs information about the current position. To get the information about the transporter's tile, `ldfu5`'s N3 program (see below) has a rule that states whenever `ldfu5` notices that a resource ?a has a `arena:tile` relation to another ressource ?b, `ldfu5` will send an HTTP GET request to the URI of ressource ?b. As `ldfu5` retrieves a triple `</transporters/5> arena:tile <shopfloor/3/1>` for transporter5, `ldfu5` sends HTTP GET to `<shopfloor/3/1>`.

```
# Follow all foor:tile properties
{
  ?a arena:tile ?b .
} => {
  [] http:mthd httpm:GET ;
     http:requestURI ?b .    } .
```

[6] http://www.w3.org/2011/http#.
[7] http://www.w3.org/2011/http-methods#.

3.4 Implementation of Artifacts

BOLD[8] is a simulation environment that implements artifacts according to Definition 2. The artifacts' states are multiple RDF graphs, collected by BOLD in a single named graph representing the environment state D. Agents can read and manipulate artifacts via HTTP requests to HTTP resources under the URI of the named graph, as defined with *transfer* and *update*, cf. Definition 2. Each time step of the simulation, BOLD updates its graph by defined SPARQL INSERT / DELETE queries that take transferred states of agents into account (e.g. moving a transporter via a task), or by the environment's own development as defined in *evolve* (e.g. creating products). We give an example of a SPARQL query that BOLD applies for updates:

Example 3. `Transporter5` was busy as `ldfu5` has sent a task before. Thus, BOLD's graph contains that `transporter5` has still status busy via `</transporters/5> arena:status arena:busy`. But meanwhile `transporter5` executed the movement, so no tasks are left in `transporter5`'s task container. Hence, BOLD sets `transporter5`'s status to `arena:idle`.

```
# Set transporter to idle when no task available and busy
DELETE {  # delete triple with arena:busy as object
    GRAPH ?transporter {
        ?transporter arena:status arena:busy . }
} INSERT { # insert triple with arena:idle as object
    GRAPH ?transporter {
        ?transporter arena:status arena:idle . }
} WHERE {
    ?transporter a arena:Transporter ; #resource is a transporter
        arena:status arena:busy ; #resource has status busy
        arena:tasks ?taskContainer
    FILTER NOT EXISTS { #has no tasks in container
        ?taskContainer ldp:contains ?task .
    } };
```

We use BOLD as a centralized back end for our artifacts as we simulate the behavior and interactions among different artifacts. The (simulated) artifacts can influence each other (e.g. transporters blocking each other) and thus a decentralized simulation would continuously have to synchronize all artifact states. The interfaces offered by the artifacts, however, are independent of each other and thus can be easily distributed.

4 System Behavior

Below, we present an overview of the agent and artifact rules from Example 1, to show how `ldfu5` perceives stigmergy marks and gives a task for movement

[8] https://github.com/bold-benchmark/bold-server.

to `transporter5`, and how `transporter5` reacts to the task. All rules including code examples can be found in our online repository[9].

4.1 Agent Behavior

An ldfu program defines the agent behavior, whereas transporter, floor tiles, products, and stations are artifacts, simulated by BOLD. The ldfu agent reads the information provided by the artifacts and interacts with the artifacts, which in turn respond to the agent's tasks. Agents get a fixed entry URI of BOLD, the root resource `/`, to start their exploration, and obey their rule set in Notation3 to move the transporter and build a grid of stigmergy marks.

1. GET all links that are accessible as objects in triples at root
2. GET all tiles, their marks, and the stigmergy values
3. GET all links to transporter's neighboring tiles
4. If the transporter carries a product, GET the link to that product
5. If the transporter has a product, POST a task to move to the neighboring tile with the lowest stigmergy value of matching color to the transporter's task container, thus following the descending gradient of stigmergy values

4.2 Artifact Behavior

Transporters have internal reflexes as reaction to interactions from the outside, e.g. by the agent. These reflexes are given in SPARQL, as artifacts have a persistent state over the simulation time that we manipulate in a non-monotonic way (SRAs again have no internal state over the simulation time, so monotonic reasoning with N3 rules is sufficient), and handle move orders (given as `arena:TransporterTask`) and the pickup and delivery of products.

1. INSERT that the transporter is busy and DELETE that the transporter is idle, if the transporter is idle and has tasks in its task container
2. INSERT the goal tile given in a task as the transporter's new position and DELETE the old one, if the tile is free and adjacent
3. INSERT all neighboring tiles according to the transporter's current position
4. INSERT that the transporter is idle and DELETE that the transporter is busy, if the transporter is busy but has no tasks in its task container
5. DELETE all tasks that point to the transporter's current tile (e.g. after the transporter moved successfully to the goal tile)

The stations' reflexes, given in SPARQL, handle the creation and consumption of products:

1. If station's output port is empty, INSERT a new product of other color
2. If station's input port contains a link to a product of the station's color, DELETE the product

[9] https://github.com/wintechis/mosaik-runtime-documentation.

4.3 System Implementation

Our implementation as well as a demo video of our system in action are available in our online repository[10]. The overall system itself works as follows:

- ldfu executes the perception-thought-action cycle:
  - ldfu GETS the given entry point, a resource that links (indirectly) to all artifacts
  - ldfu GETS the current state of its transporter, follows the link to the transporter's current tile, and to all neighboring tiles and the markers.
  - ldfu derives that all perceived shop floor tiles without stigmergy markers get an internal value of 1000. Thus, tiles with a smaller stigmergy value are more attractive for ldfu to go to.
  - ldfu creates new unsafe HTTP requests (i. e. POST, PUT, or DELETE), according to derived statements in its knowledge graph.
- BOLD executes SPARQL queries periodically on its internal RDF graphs and updates the graphs of artifacts according to the stated behavior. BOLD checks also if the transporters have new tasks in their task containers and will initiate to execute the tasks.

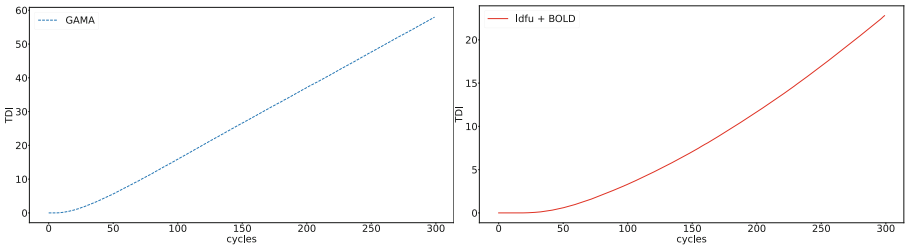
Note that ldfu and BOLD are separate, concurrent systems. BOLD manages the shop floor as a grid with x- and y-coordinates, where all tiles are addressable by URIs. All fields are pre-initialized with value 1000 for color marks. Tiles outside the valid area are set to arena:nil and values to 1001, so ldfu discerns between invalid tiles outside the shop floor, tiles that were never visited and tiles that lead to a station, as these tiles have a smaller stigmergy mark value.

5 Evaluation

We evaluate our system compared to a simulation of agents that represents the expected behavior, implemented in GAMA, a modeling and simulation environment for building explicit agent-based simulations that provides an integrated environment for creating and testing distinct agents [31]. With the given algorithms of our agents and artifacts from ldfu and BOLD, GAMA can simulate multiple repetitions to show the system's correct behavior with statistical significance. As measurement for performance, we measure the cumulative amount of delivered products over time (TDI). The setup is the quadratic shop floor as presented in Sect. 2. Adjacency of transporters and stations is defined by the 3×3 floor tiles around them. We measure time in passed cycles during simulation, where one time step equals one cycle. We chose a cycle time of five seconds for ldfu. All repetitions were run with randomly chosen starting points for transporters on the shop floor. Figure 3 shows the TDI for 300 cycles with 1000 repetitions and the average development of the systems. On average, the GAMA simulation delivered 57.95 products after 300 cycles, where the first successful delivery was achieved after about 23 cycles. In comparison, MOSAIK

[10] https://github.com/wintechis/mosaik-runtime-documentation.

with ldfu and BOLD delivered 22.92 products after 300 cycles, with the first delivery after about 16 cycles. Note that the qualitative development of both systems resembles a "hockey stick", showing the state of the system without and with only few stigmergy marks (low performance), and with more marks to create direct paths (increasing performance). We focus on the implementation of MOSAIK in the application domain and include a general running example without comparison of different agent-based approaches or parameter optimization, as this is out of scope. For a discussion of different agent approaches in the DTS domain with a dynamic environment, see e.g. [26].



(a) GAMA Simulation (b) MOSAIK with ldfu and BOLD

Fig. 3. Average behavior of our DTS compared to a simulation for the cumulative amount of delivered items, measured over 300 cycles with 1000 repetitions.

Example 4 explains the interaction between multiple, distinct agents in MOSAIK:

Example 4. Consider again Fig. 1 with agents `ldfu1` and `ldfu5`, both controlling respectively `transporter1` at (2, 2) and `transporter5` at (3, 1). As `transporter5` is carrying a blue product, `ldfu5` prioritizes movement towards blue stigmergy markers such that the delivery can be fulfilled. At (3, 1), `ldfu5` cannot perceive the blue station at (0, 2) or any marker towards the station, so the next move might be random. However, next to `transporter1`, at (1, 2), is a stigmergy marker pointing towards the blue station. `ldfu1` cannot perceive the station, but perceives the adjacent marker. Agents GET all local information of their artifacts, e.g. `ldfu1` also `transporter1`'s floor tile or state, and meanwhile also adjacent floor tiles and respective stigmergy marks, see no. 1 Fig. 4. `ldfu1`'s rules state to replicate adjacent stigmergy markers with increased value, so it PUTs a new blue marker with value 2 to `transporter1`'s current tile, see no. 2 Fig. 4 (dashed arrow in Fig. 1). Finally, see no. 3 Fig. 4, `ldfu5` GETs `transporter5`'s floor tile (3, 1) and also the surrounding tiles including (2, 2). `ldfu5` GETs (2, 2) and thus the stigmergy mark of blue with value 2, which is smaller than any other blue marker around. Hence, `ldfu5` received the information, without direct communication, that a blue station can be reached, when following the stigmergy mark via the descending gradient and can decide to do so in a subsequent rule.

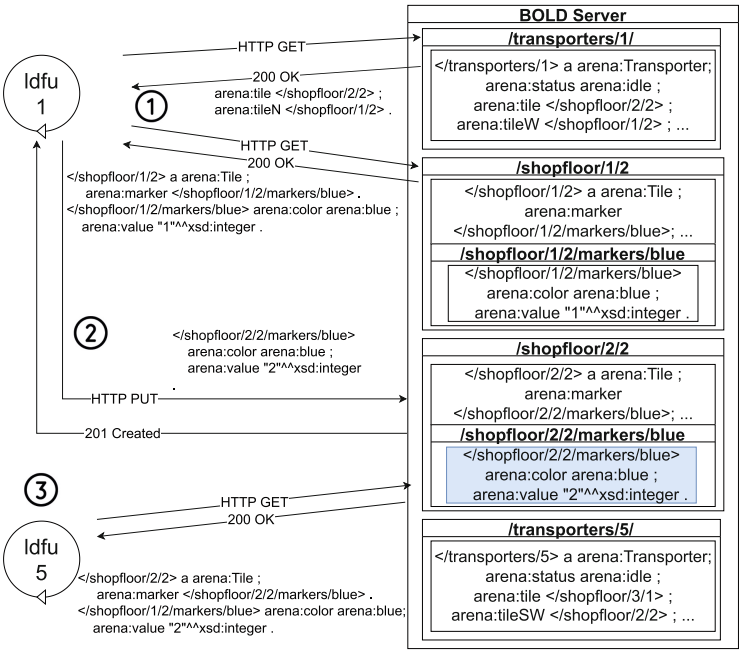


Fig. 4. Interaction between two agents to exchange information via stigmergy

6 Discussion

6.1 Simulation Results and System Performance

Considering the different performances of delivered items, we emphasize how GAMA works compared to BOLD and ldfu: GAMA simulates all agents and artifacts during one simulation cycle, e.g. all marks appear at the same time and can be evaluated in the successive cycle; BOLD+ldfu are unsynchronized, separate programs i.e. the environment's reaction (BOLD) might take longer to appear than the agents' action (ldfu). Thus, agents need additional cycles as they might miss other agents' actions as these are not applied to the environment in time. Still, MOSAIK and GAMA show the characteristic behavior of swarms that use the environment for positive feedback through stigmergy mark placement [14], similar to trails in Ant Colony Optimization [9]. Our agents decide autonomously, according to given rules, when to place marks to reflect the perceived truth. Artifacts act predominantly as producers of data - although their reflexes may also change the environment's state (e.g. picking up products) or its own (e.g. moving to a tile), these actions are only answers to an agent's stimulus.

6.2 Implementation of Distributed Control

MOSAIK discerns active agents (Sect. 4.1) that can perceive and interact with their environment, and reactive artifacts (Sect. 4.2) that represent external, reacting parts that are controlled by agents (the environment). As we focus on a decentralized control system, we demand agents to be stateless, use stigmergy, and have loose coupling (see Sect. 1). Semantic technologies offer sound inference algorithms for our agents, a uniform interface for interactions, and data sharing with a common understanding, implemented by RDF and RESTful interfaces. Our agents use MOSAIK and semantic technologies to self-coordinate and control, which enables other agents to take part pervasively.

Stateless agents amend each others knowledge by creating and changing perceived resources, but do not save any information internally. Thus, the knowledge of the agent population lies exclusively in the environment such that agents decide only on locally available information. RESTful interfaces and Linked Data technologies guarantee a common understanding and exchange of information.

Indirect communication has no dependence on explicit channels and protocols between agents, but agents need a shared medium to pass information, limited by agents' ability to perceive the world and the representation of useful states via stigmergy [20,24]. We use the artifacts to realize the agents' medium. Still, a distortion of the medium can lead to knowledge loss for the population [19].

With loose coupling between agents, the system is flexible and scalable as the population is independent such that single agents can be added to or removed from the population without impacting other agents. Without knowledge about other agents or their architectures, agents do not care if perceived information in the environment come from themselves or any other agent. Thus, new agents can adapt and work right away, as local knowledge is shared and perceivable. Removed agents only lead to graceful degradation of the system [3].

As we use RDF to represent the current state of a floor tile (and all other artifacts), an agent could crawl the environment's entire graph and thus have knowledge about all floor tiles and stations' locations. However, crawling the entire graph becomes infeasible when the environment's graph is very large and agents have a short perception cycle. Instead, we use stigmergy to make global information (possibly anywhere in the RDF graph, many links away) available locally [13]. Thus, agents get up-to-date information about the applied actions of others (or themselves before), and additionally agents can always be sure that the available information is at least a best guess with respect to the last available state of the environment, giving a minimum of resiliency. Unfortunately, the price of possibly suboptimal behavior comes when local information is heavily outdated or wrong, when the environment changed fundamentally, but can be repaired as part of robust self-organization [26].

We conclude that we successfully built a stateless, decentralized control system with MOSAIK building on indirect communication and loose coupling, using state-of-the-art Semantic Web technologies, with RDF as data model and RESTful communication. We designed our system according to the properties stated

in the introduction and emphasize the merits of the Semantic Web for our application domain of decentralized transportation systems:

- Stateless agents can be stopped and started without the risk to loose internal information, as agents write their data to the environment. Agents can be thus be executed anywhere e.g. in an industrial cloud or on mobile transportation devices as edge.
- Loose coupling and indirect communication make it easy to create new agents, so the population can be scaled depending on outside requirements, e.g. if new transportation units have to be available in short time.
- Agents do not need to crawl whole RDF graphs for perception, when all required knowledge for decisions is locally available which reduces the overall network traffic and leads to faster decisions.

We see the following remaining challenges for the DTS domain:

- All artifacts need HTTP interfaces with a stable connection to their agents' network to be perceivable for agents.
- As agents regularly poll artifacts as part of the perception cycle, depending on the duration between polls, agents might not perceive artifact state changes in time. Also, artifacts have to respond to agents which uses electrical power. Thus, mobile transport units with limited battery cells might have an increased power consumption which influences the movement range.
- Precautions have to be taken that agents can find the required local information, otherwise the agent's perception of the local environment degenerates to a randomized perception search [13] of the RDF graph and gets inefficient.

7 Conclusion and Outlook

We realize a self-organizing, decentralized controlled transportation system using MOSAIK. We build on an agent-based system that uses Linked Data and stigmergy as technologies to implement stateless, reactive agents that rely on locally available information, retrieved from distributed artifacts. Agents and artifacts are described via declarative rules and communicate via HTTP. We show our system in the context of a defined transportation scenario that works in an abstract Industry 4.0 shop floor. Further usages include the control of stations, the optimization of path building and assigning transport orders. More research regarding the application of agent-based systems in the Semantic Web is needed, so we identified three problems during our research that will inevitably appear when scaling up our approach and thus must be tackled in the future:

- What is the optimal partitioning of artifacts that shall be controlled among agents? I.e. for how many artifacts should one agent be responsible? Or should agents only be responsible for certain aspects of the control, e.g. one agent for following stigmergy markers and one separate agent for random movement otherwise?

- When multiple agents act on the same artifacts, how to avoid conflicts between the actions agents submit to an artifact (e.g. one agent send the transporter to the left, one sends it to the right as both perceived that the transporter currently has no task)? As solution, HTTP would allow to use optimistic locking via the `If-Unmodified-Since` header, but this method is only efficient if conflicts are rare [23].
- Agent perception is limited to adjacent floor tiles – all other global information in the environment has to be made locally available via stigmergy. For other use cases, a different perception radius might be more efficient, i.e. agents could follow more link hops in the RDF graph to evaluate more distant floor tiles. How to find the optimal radius of the local perception is an open problem.

References

1. Alami, R., Fleury, S., Herrb, M., Ingrand, F., Robert, F.: Multi-robot cooperation in the MARTHA project. IEEE Robot. Autom. Mag. **5**(1), 36–47 (1998). https://doi.org/10.1109/100.667325
2. Berners-Lee, T.: Notation3 logic - an RDF language for the Semantic Web. https://www.w3.org/DesignIssues/Notation3.html
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, Santa Fe Institute Studies on the Sciences of Complexity (1999)
4. Charpenay, V., Käfer, T., Harth, A.: A unifying framework for agency in hypermedia environments. In: Alechina, N., Baldoni, M., Logan, B. (eds.) EMAS 2021. LNCS, pp. 42–61. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97457-2_3
5. Charpenay, V., et al.: MOSAIK: a formal model for self-organizing manufacturing systems. IEEE Pervasive Comput. **20**(1), 9–18 (2021). https://doi.org/10.1109/MPRV.2020.3035837
6. Chen, B., Cheng, H.: A review of the applications of agent technology in traffic and transportation systems. IEEE Trans. Intell. Transp. Syst. **11**(2), 485–497 (2010)
7. Chen, B., Wan, J., Shu, L., Li, P., Mukherjee, M., Yin, B.: Smart factory of industry 4.0: key technologies, application case, and challenges. IEEE Access **6**, 6505–6519 (2018). https://doi.org/10.1109/ACCESS.2017.2783682
8. Choi, H.L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE Trans. Rob. **25**(4), 912–926 (2009). https://doi.org/10.1109/TRO.2009.2022423
9. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006). https://doi.org/10.1109/MCI.2006.329691
10. Giordani, S., Lujak, M., Martinelli, F.: A distributed multi-agent production planning and scheduling framework for mobile robots. Comput. Ind. Eng. **64**, 19–30 (2013). https://doi.org/10.1016/j.cie.2012.09.004
11. Grassé, P.: La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. Insectes Sociaux **6**(1), 41–80 (1959). https://doi.org/10.1007/BF02223791

12. Hadeli, K., et al.: Self-organising in multi-agent coordination and control using stigmergy. In: Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F. (eds.) ESOA 2003. LNCS (LNAI), vol. 2977, pp. 105–123. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24701-2_8
13. Hadeli, K., Valckenaers, P., Kollingbaum, M.J., Brussel, H.V.: Multi-agent coordination and control using stigmergy. Comput. Ind. **53**(1), 75–96 (2004)
14. Hamann, H.: Swarm Robotics: A Formal Approach, pp. 1–32. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74528-2_1
15. Harth, A.: Interoperation between information spaces on the web. In: Grust, T., et al. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 44–53. Springer, Heidelberg (2006). https://doi.org/10.1007/11896548_5
16. Harth, A., Käfer, T.: Linked data techniques for the web of things: Tutorial. In: Proceedings of the 8th International Conference on the Internet of Things. IOT 2018, Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3277593.3277641
17. Heylighen, F.: Stigmergy as a universal coordination mechanism i: Definition and components. Cogn. Syst. Res. **38**, 4–13 (2016). https://www.sciencedirect.com/science/article/pii/S1389041715000327
18. Heylighen, F.: The science of self-organization and adaptivity. Encycl. Life Support Syst. **5**, 253–280 (1970)
19. Heylighen, F.: Stigmergy as a universal coordination mechanism ii: varieties and evolution. Cogn. Syst. Res. **38**, 50–59 (2016)
20. Holland, O., Melhuish, C.: Stimergy, self-organization, and sorting in collective robotics. Artif. Life **5**, 173–202 (1999)
21. Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., Kovatsch, M.: Web of Things (WoT) Thing Description, W3C Recommendation. Accessed 18 Oct 2021 (2020)
22. Klein, N.: The impact of decentral dispatching strategies on the performance of intralogistics transport systems (2012). https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-147739
23. Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. ACM Trans. Database Syst. **6**(2), 213–226 (1981)
24. Parker, L.E.: Multiple Mobile Robot Systems, pp. 921–941. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-30301-5_41
25. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, USA (2009)
26. Schmid, S., Schraudner, D., Harth, A.: Performance comparison of simple reflex agents using stigmergy with model-based agents in self-organizing transportation. In: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), pp. 93–98 (2021). https://doi.org/10.1109/ACSOS-C52956.2021.00071
27. Schmidt, T., Reith, K.B., Klein, N., Däumler, M.: Research on decentralized control strategies for automated vehicle-based in-house transport systems - a survey. Logistics Res. **13**, 10 (2020). https://doi.org/10.23773/2020_10
28. Schraudner, D.: Stigmergic multi-agent systems in the semantic web of things. In: Verborgh, R., et al. (eds.) ESWC 2021. LNCS, vol. 12739, pp. 218–229. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80418-3_34
29. Schraudner, D., Harth, A.: A RESTful interaction model for semantic digital twins. In: Proceedings of the Third International Workshop on Semantic Digital Twins co-located with the 19th Extended Semantic Web Conference (ESWC 2022) (2022)

30. Speicher, S., Arwe, J., Malhotra, A.: Linked data platform 1.0. W3C Recommendation (2015)
31. Taillandier, P., et al.: Building, composing and experimenting complex spatial models with the GAMA platform. GeoInformatica **23**(2), 299–322 (2019)
32. Theraulaz, G., Bonabeau, E.: A brief history of stigmergy. Artif. Life **5**, 97–116 (1999). https://doi.org/10.1162/106454699568700
33. Tummolini, L., Castelfranchi, C.: Trace signals: the meanings of stigmergy. In: Weyns, D., Parunak, H.V.D., Michel, F. (eds.) E4MAS 2006. LNCS, pp. 141–156. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71103-2_8
34. Dyke Parunak, H.: A survey of environments and mechanisms for human-human stigmergy. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2005. LNCS (LNAI), vol. 3830, pp. 163–186. Springer, Heidelberg (2006). https://doi.org/10.1007/11678809_10
35. Zedadra, O., Jouandeau, N., Seridi, H., Fortino, G.: Multi-agent foraging: state-of-the-art and research challenges. Complex Adapt. Syst. Model. **5**(1), 3 (2017)