An HTTP/RDF-based Agent Infrastructure for Manufacturing using Stigmergy*

Daniel Schraudner[0000−0002−2660−676X] and Victor Charpenay[0000−0002−9210−1583]

Chair of Technical Information Systems, Friedrich-Alexander-University Erlangen-Nürnberg, Germany
{daniel.schraudner,victor.charpenay}@fau.de

Abstract. We have built a demonstrator for the communication infrastructure of a multi-agent system controlling a simplified shop floor. We use the concept of stigmergy which does not allow inter-agent communication but only between agents and the environment. Furthermore we keep the agents as simple as possible by using only simple-reflex agents. The environment is modelled as a RDF Knowledge Graph and communication takes place using HTTP request. We carried out experiments regarding the agent scalability and identified open questions.

Keywords: Multi-Agent System · Stigmergy · Knowledge Graph.

1 Introduction

Information systems in many businesses still are following a quite centralized approach – as opposed to the Web which has an inherently decentralized architecture. Many of those systems could profit from the properties and effects that come along with a decentralized system architecture such as more reliability and less complexity.

A common approach for tackling this problem are multi-agent systems which have been around for a long time in the research field of decentralized manufacturing [4]. Not that well established, however, is the concept of stigmergy which is capable of greatly reducing the complexity of a multi-agent system. It is a communication paradigm which does not allow agents to communicate directly with each other but only indirectly by using the environment. Using stigmergy has several advantages including no need for explicit synchronization between the agents and a clear separation of concerns [2].

In our approach we want to take this idea even further and try to shift as much complexity out of the agents into the environment. This idea is (like stigmergy) taken from biological systems in nature where it works quite well. An ant colony, e. g. comprises a very large number of quite simple agents (the ants) which do not communicate directly but only by using their environment

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

(pheromones, etc.) but it is able to accomplish a lot of tasks (like food foraging) very efficiently.

This specific type of agent we want to use for our system is called simple-reflex agent [5] and its functionality can be seen in Figure 1. It is the simplest form of an agent where the current state of the environment is read through sensors. The agent decides based on a set of simple condition-action rules, which action to take next and executes it using its actuators. No internal state of the agent or further reasoning are involved.

To allow agents to actually do things, one needs a robust communication infrastructure which lets the agents read their environment and change its state concurrently and asynchronously. We propose that the environment can be represented using a RDF-based Knowledge graph and that communication between agents and the environment should be carried out using HTTP requests.

For this demonstration we modelled a the scenario of a simple shop floor producing IoT boards which is described in Section 2. We built a working communication infrastructure for the agents. The architecture of this system is explained in Section 3. Furthermore we carried out some experiments regarding the scalability of agents in the system (Section 4) and identified some open questions that we want to address in the future (Section 5).

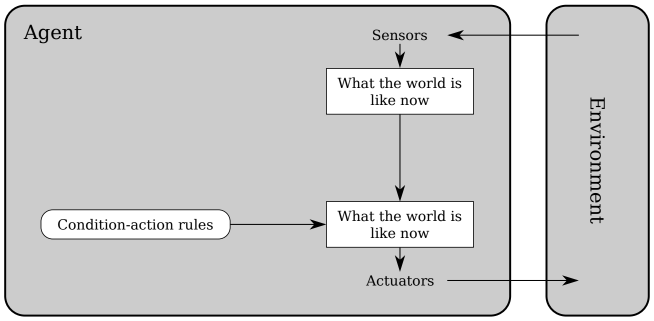

Fig. 1. Schema of a simple-reflex agent based on [5]

2 Scenario

The RDF model of our simplified shop floor scenario is available online[1]. It comprises five different kinds of products in an assembly tree which can be seen in Figure 2. The manufacturing of an IoT board requires specific actions with specific products at different workstations.

The shop floor has five different types of stations. At *Delivery* the base products (*Temperature Sensor*, *Processing Unit* and *Case*) enter the factory. At *Shipping* assembled *IoT Boards* leave the factory. *Soldering Stations* and *Screwing*

[1] https://purl.org/mosaik/demo/arena

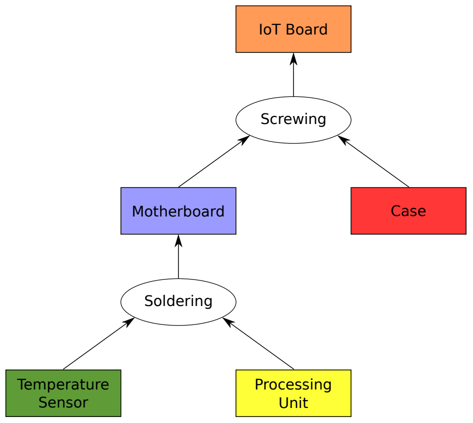

Fig. 2. Assembly tree of an IoT Board

Stations can carry out their respective assemble actions if the needed products are available at the station. Between the stations products are stored in the *Rack* which acts as a buffer. Products do not move by themselves but transport actions have to be carried out by *Forklifts* which can move between the different stations. The general structure of the shop floor can be seen in Figure 3.

As each assemble station and forklift can only execute one action at a time, we scaled up the numbers a bit for our experiments (3 *Screwing Stations*, 3 *Soldering Stations*, 10 *Forklifts*).

3 Architecture

The architecture of our demonstrator can be seen in Figure 4.

The client is able to run an arbitrary number of simple-reflex agents. We use Linked-Data-Fu [3] for carrying out the HTTP requests and applying the condition-action rules. The agents decide autonomously whether to carry out a specific action or not, using the rules which are stated in Notation 3 [1] (an extension of RDF with logical operations).

Agents are constantly and asynchronously querying the state of the Knowledge Graph and trying to match their rules. In order for the agents to make sensible decisions, they need a semantically rich representation of the current state of the shop floor which they get in response to their HTTP requests. If they can apply a rule, they send a HTTP POST request to change the Knowledge Graph. More specifically they are proposing an action which should be carried out (in this case either an assemble action for a station or a transfer action for a forklift).

On the server side the state of the environment is maintained as a RDF Knowledge Graph. It comprises the current location of all the forklifts and products as well as the state of the workstations. Furthermore all possible, active and

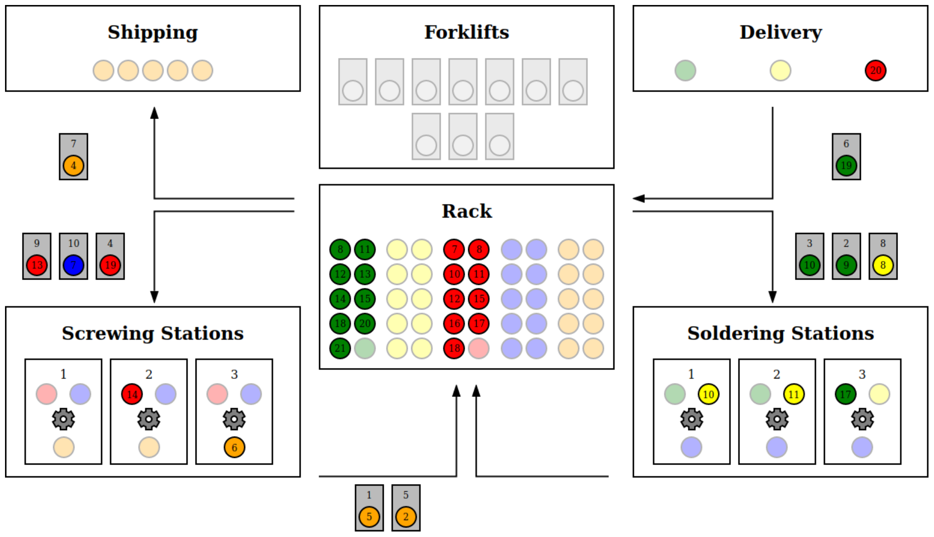

Fig. 3. Structure of the shop floor (screenshot of the GUI)

completed actions are contained. These entities and the relationships between them are represented using Schema.org[2] and, where this was not sufficient, our own vocabulary which is also available online[3].

Also on the server there runs a simulation part which mimics the behavior of the stations and forklifts on the shop floor (i. e. it executes the actions). This is done by constantly observing the Knowledge Graph for new proposed actions (*Action Observer)* and spawning a new *Action Runner* which – after a exponentially distributed time – changes the Knowledge Graph according to the action (e. g. moving the product).

However, as the agents are sending their request asynchronously, some kind of synchronization mechanism has to be implemented such that no conflicting actions are (trying to be) executed (e. g. one product being transported to two different stations at the same time). In our implementation every *Action Runner* checks before the start of the execution of its action, whether there exist conflicting action and, if this is the case, discards its action.

A graphical user interface is also available at the server-side to watch the current state of the shop floor. A screenshot of the GUI can be seen in Figure 3. All elements that are numbered have an URI and are clickable links to their RDF representation in the Knowledge Graph.

[2] https://schema.org/
[3] https://purl.org/mosaik/demo/vocab

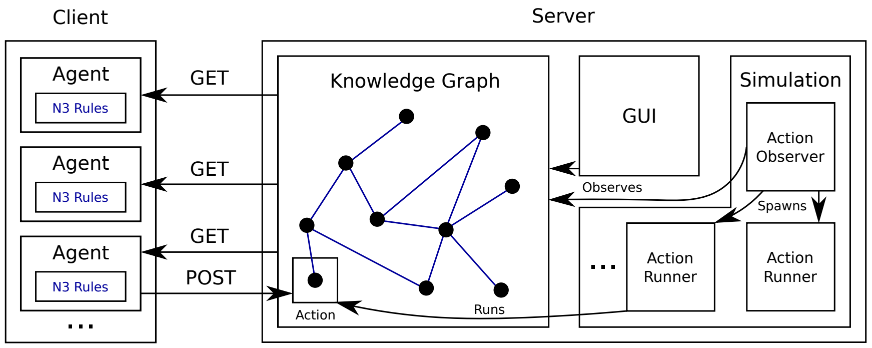

Fig. 4. Architecture of our demonstrator

4 Demo & Experiments

A running instance of our demonstrator is publicly available[4].

We also ran some experiments with a different number of agents and measured those actions that were successful (i. e. they were executed by an *Action Runner*) and those that were conflicting (i. e. they were discarded by an *Action Runner*).

The agents we used can be divided in two classes depending on whether they had a rule for proposing an assemble action or a transport action. The number of assemble and transfer agents always was equal (i. e. *4 Agents* means two assemble agents and two transfer agents.

For each number of agents we ran the experiment ten times and took the averages over those runs to reduce the effect of the random action duration. Figure 5 shows the number of successes over time for each experiment (which directly correlates to the number of assembled IoT boards). There seems to be a optimal number of agents around 24 - 28 (as to few agents are not producing enough action proposals and to much are producing a lot of conflicts which hinder the successful actions).

5 Conclusion & Future Work

From our experiment we can see that the communication architecture we proposed generally seems suited to tackle a problem like in our exemplary scenario. However, further investigation on the optimal number of agents to deploy needs further investigation.

Another interesting question is, whether the performance of the system could be improved if the assignment of agents to action is changed (one or multiple agents have rules for one type of stations or one kind of product instead of all assemble actions, etc.).

[4] https://purl.org/mosaik/demo

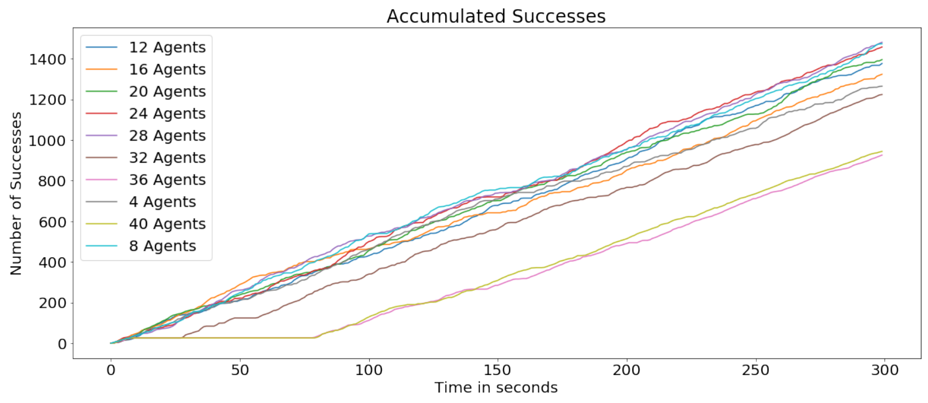

Fig. 5. Successfull actions over time for different numbers of agents

Our synchronization technique (conflict detection) could be further improved or other methods could be applied (e. synchronizing the agents using tokens).

Furthermore it might be interesting to investigate how rules for agents can be written such that the number of occurring conflicts is reduced. It seems likely that a reduced number of conflicts will allow more successful actions and thus lead to a better overall performance.

References

1. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax (2011)
2. Hadeli, Valckenaers, P., Kollingbaum, M., Van Brussel, H.: Multi-agent coordination and control using stigmergy. Computers in Industry **53**(1), 75–96 (jan 2004). https://doi.org/10.1016/S0166-3615(03)00123-4
3. 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 '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3277593.3277641
4. Leitão, P.: Agent-based distributed manufacturing control: A state-of-the-art survey. Engineering Applications of Artificial Intelligence **22**(7), 979–991 (oct 2009). https://doi.org/10.1016/j.engappai.2008.09.005
5. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach Third Edition. Prentice Hall (2010). https://doi.org/10.1017/S0269888900007724