# A Procedure for Conceptualizing and Implementing Spade Agents

Henning Gösling[1], Dennis Maecker[1], Tom Pieper[1], Timon Sachweh[2], and Christoph Heinbach[1]

[1] German Research Center for Artificial Intelligence, Osnabrück, Germany
{henning.goesling,dennis.maecker,tom.pieper, christoph.heinbach}@dfki.de
[2] TU Dortmund University, Germany
timon.sachweh@tu-dortmund.de

**Abstract.** In this paper we present our approach for conceptualizing and implementing software agents as part of a multi-agent system (MAS). The procedure consists of four steps: (1.) defining the relevant types of software agents, (2.) specifying the components of the software agents, (3.) conceptualizing each component of the software agents, and (4.) implementing the different components of the software agents. Our approach is derived from the experience in the ongoing research project Gaia-X 4 ROMS in which we build a MAS for the real-time control of various actors involved in parcel transports. After following step 1-3, we are currently implementing the MAS using the development environment Spade.

**Keywords:** Agent-Oriented Analysis and Design, Intelligent Agents, Spade Development Environment.

## 1 Introduction

In our ongoing research project, we are currently developing a multi-agent system (MAS) consisting of six types of software agents that represent essential actors in parcel transportation: booking agents, freight agents, parcel delivery robot agents, trailer agents, depot agents, and workshop agents. The MAS we are currently developing will be used for the real-time scheduling and control of robots, trailers, depots, and workshops. For this purpose, the software agents need to be connected with operators, vehicles, booking platforms, other software, and other software agents. Several methodologies exist, that can be followed for the conceptualization of such a MAS: AAII, Gaia, Prometheus, etc. (Wooldridge 2009). Moreover, there are models that describe the basic components of software agents such as the BDI Agent, Practical Reasoning Agent, Situated Automata, InteRRaP, Stanley, etc. (Wooldridge 2009). Besides these agent-oriented models, there are those that describe the basic components of autonomous systems which can be applied for the conceptualization of software agents, too (Wahlster 2017). Finally, there are development environments for the actual implementation of a MAS, such as Jade, Spade, or ROS2.

As a model for our software agents' internal structures, we decided to use the reference architecture of Wahlster (2017) since it explicitly covers all the components we needed for our software agents. Besides, the Python-based Spade development environment (Palanca 2024) was chosen early on in the project as we wanted to be able to integrate state-of-the-art Python libraries (e.g., for reinforcement learning and optimization) into our project. Hence, we developed our own procedure for conceptualizing software agents using Wahlster (2017) and for implementing these software agents using Spade. It consists of four steps: (1.) defining the types of software agents, (2.) specifying the components of each software agent using Wahlster (2017), (3.) conceptualizing the components of the software agents, and (4.) implementing the components of the software agents in Spade. Currently, we are in the fourth step of implementing the MAS. In this paper, we want to elaborate on the procedure in detail.

## 2 Background

In the research project Gaia-X 4 ROMS, we automate various processes in parcel transports. More precisely, we automate the booking process of transport resources involved in the pick-up of the parcel, its main-haul transport between depots, and its delivery. The involved actors are physically distributed, embedded in their environment, owned by different companies, and work asynchronously. Thus, software agents were selected as a technology that naturally fits to the use case (Heinbach et al. 2022, Maecker et al. 2023). Software agents are considered a valuable technology for managing logistics resources (Gath 2016).

Research in software agents has produced different methodologies that provide step-by-step guides for the design of software agents and MAS: AAII, Prometheus, Gaia, Tropos, Agent UML, and Agents in Z (Wooldridge 2009). For example, following the AAII methodology, firstly the roles and agents are defined and captured in a so-called external model, secondly the software agents' internal models are specified according to their beliefs, desires, and intentions. The Prometheus methodology consists of the system specification step, the architectural design step, and the detailed design step. In the system specification step, the system's goal, interfaces, and functionalities are defined that are necessary for the use case. In the architectural design step, the software agent types are formed by grouping functionalities. In the detailed design step, each software agent is broken down into several components, and each component is modeled separately (Wooldridge 2009).

Research in software agents has also produced a lot of models structuring the components of software agents: BDI Agent, Practical Reasoning Agent, Situated Automata, InteRRaP, or Stanley. For instance, the rather sophisticated Stanely architecture (that was used to build a software agent embodied in a car) consists of the sensor interface layer, the perception layer, the planning and control layer, the vehicle interface layer, the user interface layer, and the global service layer (Wooldridge 2009). Beyond, there is the reference architecture for autonomous systems by Wahlster (2017), which can be applied to software agents and robots. Wahlster (2017) differs between components for (1.) self-regulation, (2.) perception, (3.) learning, (4.) planning and plan recognition, (5.) collaboration, (6.) saving knowledge, (7.) communicating with the environment, (8.) communicating with humans, (9.) sensing, (10.) acting, (11.) operators to stop the operation at any time, (12.) operators to take over the operation, and (13.) operators to influence the operation. The reference architecture of Wahlster (2017) is shown in Figure 1.

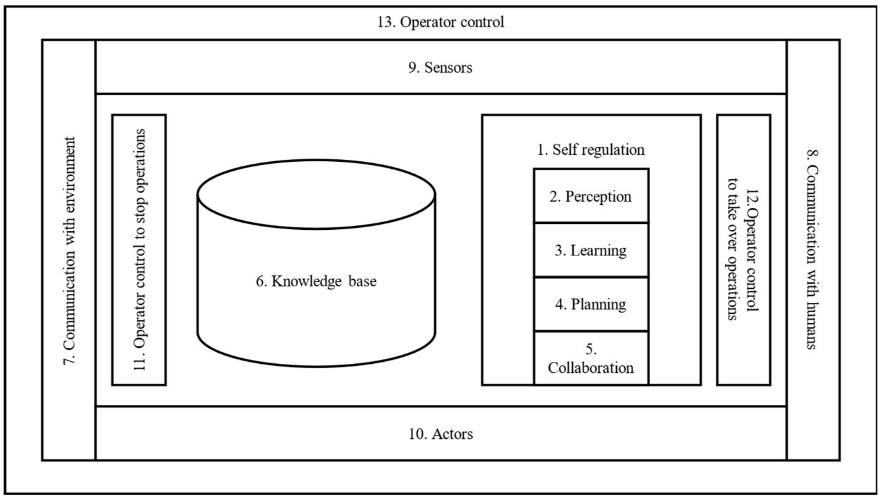

13. Operator control
9. Sensors
7. Communication with environment
11. Operator control to stop operations
6. Knowledge base
1. Self regulation
2. Perception
3. Learning
4. Planning
5. Collaboration
12.Operator control to take over operations
8. Communication with humans
10. Actors

**Figure 1.** Reference architecture for autonomous systems adapted from Wahlster (2017). (Adaption by numbering the components and by translating from German.)

## 3 The Procedure Applied in the Gaia-X 4 ROMS Project

### 3.1 Defining the Types of Software Agents

In the first step, several interviews were conducted between an expert in the field of MAS and an expert in the field of transport logistics. From the insights of these interviews, the main roles in our parcel transport use case were defined: a booking management for the interaction with customers, parcel delivery robots for the first-mile transport, so-called bordero management for the creation of parcel lists for scheduled long-haul transports between depots, trailers for the long-haul transport, parcel delivery robots for the last-mile transport, depot management for the assignment of ramps to robots and trailers, depots for the cross-docking between first-mile, long-haul, and last-mile transports, and workshops for the maintenance of the transport resources. Each role in the transport system got an own software agent type: (i) a booking agent for booking management, (ii) a parcel delivery robot (PDR) agent for the management of first-mile transports and last-mile transports, (iii) a bordero agent for the long-haul transport management, (iv) a trailer agent for the trailer management, (v) a depot agent for the ramp assignment and cross-docking management, (vi) and a workshop agent for the workshop management. In a word document, a table was set up for each type of software agent. In each table, the two experts defined the software agents' main responsibilities, added a detailed description of each responsibility, marked it as a basic function or as an additional function, and linked it with other software agents, different operator roles, and remote software systems. In the following Table 1, an excerpt of the booking agent table is shown that was defined as part of the procedure's first step. The result of this first step can be seen as the so-called external model to be created when applying the AAII methodology. The external model is a system-level view that describes the agents, their responsibilities, and their interactions (Wooldridge 2009).

**Table 1.** Excerpt of the booking agent table, specified in several expert interviews.

| **Nr.** | **Responsibility** | **Description** | **Basic function vs. add-on** | **Interaction with operators, robots, or remote software** | **Interaction with other software agents** |
|---|---|---|---|---|---|
| 1 | Collection of shipping orders | The booking agent receives the transport orders from the booking platform, which are divided into three partial orders (first-mile, long-haul and last mile) and contain information on depots, time restrictions and release times. | Basic function | Booking platform | |
| 2 | Validation of shipping orders | The booking agent uses a verification mechanism to check the transport orders for type of goods, destination, weight, plausibility, and speed (same day/hour, overnight, etc.). If necessary, a query is made for dangerous goods, prohibited substances, live animals or organisms. Transports are only approved for the transportation of goods within Germany. | Add-on | | |
| 4 | Determination of release times for partial orders | The booking agent determines the release times for the partial orders. | Add-on | | |
| 5 | Distribution of the individual partial orders | The booking agent distributes the partial orders via a multi-agent-specific coordination mechanism as soon as the release times have been reached. | Basic function | | PDR agents<br><br>Bordero agents<br><br>Trailer agents<br><br>Depot agents |
| … | … | … | … | … | … |

After the tables for all types of software agents were completed, the word document with the tables circulated among several practitioners involved in the research project.

The practitioners made comments in the document. Using the practitioners' feedback, the document was adapted and finalized. Afterwards, the document was used as the starting point for the next step of our procedure.

## 3.2 Specifying the Components of each Software Agent using Wahlster (2017)

In the second step, we defined the architecture for each type of software agent based on Wahlster (2017). When specifying the architecture, we tried to match the standard components of a software agent with the responsibilities of each software agent defined in the first step of the procedure. We also tried to formulate the architectural designs in line with the work of Gregor et al. (2020) who differ between users, aim, context, mechanisms, and rationale of a so-called design principle. As an example, the architectural designs for the booking agent and the PDR agent are (partly) presented in Table 2. The result of the second step can be seen as an extension of the so-called internal model to be created when applying the AAII methodology. The internal model in the AAII methodology is concerned with the agents' beliefs, desires, and intentions (Wooldridge 2009).

**Table 2.** Architectural designs for booking agent and PDR agent (excerpt).

| **Software agent** | **Architectural design** |
|---|---|
| Booking agent | To allow transport order operators (users) to automate the process of placing and modifying bookings for parcel deliveries from/to customers (aim) when parcel delivery robots are used for first- and last-mile transports in urban areas and telematics-enabled trailers are used in main-haul transports between depots (context), a booking agent should be available that consists of the following functions (mechanisms) in line with Wahlster (2017) (rationale):<br>• a self-regulation mechanism (see 1. in Figure 1) to orchestrate the different functions of a booking agent and their interactions with a booking agent's knowledge base (6.)<br>• a collaboration mechanism (5.)<br>  o with PDR agents to assign transport orders to PDR agents<br>  o with bordero agents to assign freight orders to bordero agents<br>• a communication mechanism (7.)<br>  o to publish booking specifications<br>  o to subscribe to messages of bordero agents, trailer agents, PDR agents, and depot agents to collect the current state of the parcel deliveries<br>• a user interface (8./13.)<br>  o to start and stop a booking agent<br>  o to create bookings for parcel deliveries<br>  o to show the current state of parcel deliveries<br>  o to change booking specifications and cancel bookings |
| PDR agent | To allow transport order operators (users) to automate the process of assigning transport orders to parcel delivery robots (aim) when these mobile robots are used for first- and last-mile transports of parcels in |

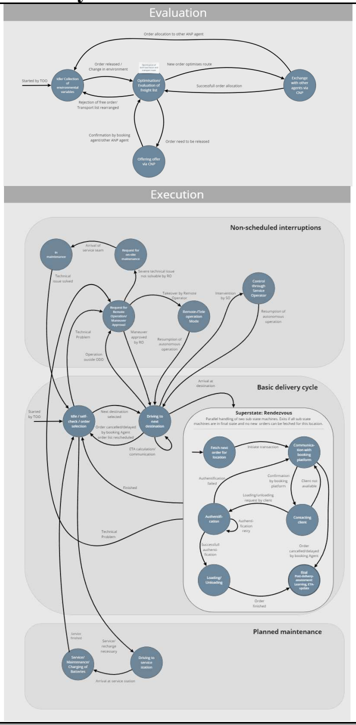
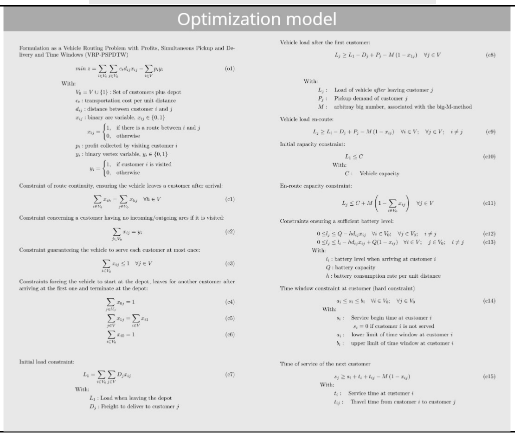
| | |
|---|---|
| | urban areas, each PDR should be represented by a corresponding agent with the following functions (mechanisms) in line with Wahlster (2017) (rationale):<br>• a self-regulation mechanism (1.) to orchestrate the different functions of a PDR agent and their interactions with a PDR agent's knowledge base (6.)<br>• a planning mechanism (4.)<br>  o minimizing the marginal costs when adding a transport order or a battery charging order to the internal tour list<br>  o … |
| … | … |

These architectural designs were discussed during an expert panel of nine participants (3x software developer for logistics systems, 1x cloud architect for logistics systems, 1x innovation manager, 1x product specialist for trailers, 1x researcher for PDR development, 1x transport logistics expert, 1x senior researcher for MAS) as part of our research project. At the end of the panel, they were evaluated by the experts and the feedback was used for their adaption. The finalized instructions were then used as the starting point for the next step of our procedure.

## 3.3 Conceptualizing each Component of a Software Agent

In the third step of our procedure, we defined a separate concept for each component of our software agents. The self-regulation mechanisms were modeled with finite-state machines, the knowledge base with a UML class diagram, the communication and interaction with other software agents, remote software services, user interfaces of operators, and actors/sensors with a UML component diagram, the planning mechanism for optimizing a software agent's schedule with an optimization model, and the collaboration mechanism for assigning tasks with a sequence diagram. For the self-regulation, at least two routines, an evaluation and an execution routine, were defined per agent in accordance with the MAPE-K architecture (IBM Corporation 2006). For the optimization model, we looked for standard problems in the Operations Research literature that could be used as the starting point for the mathematical formulation: for example, the agents for the PDR and for the trailer have the single-vehicle routing problem with pickups and deliveries and time windows while the depot agent has a cross-docking problem. The learning model of a software agent was described without a separate modeling language. Each model and description of a software agent's component was created by a researcher who later was responsible for its implementation (step 4). The creation was done using an interactive Miro board. In Table 3 below, some of the models for the PDR agent are indicated by screenshots from this Miro board. These models and descriptions were also discussed during several expert panels, one panel for each type of software agent. The experts evaluated the models, and the feedback was used for their adaption. The finalized concepts represented the blueprints for the implementation.

**Table 3.** Models for the components of the PDR agent (excerpt). Models are indicated by screenshots from a Miro board.

| **PDR agent component** | **Concept model type** | **Model (indicated by screenshots from Miro board)** |
|---|---|---|
| Self regulation (see 1. in Figure 1) | State machines for evaluation routine and execution routine |  Evaluation<br>Order allocation to other ANP agent<br>Started by TOO<br>Idle/ Collection of environmental variables<br>Order released / Change in environment<br>Optimisation of both load factor and transport route<br>Optimisation/ Evaluation of freight list<br>New order optimises route<br>Successfull order allocation<br>Exchange with other agents via CNP<br>Rejection of free order/ Transport list rearranged<br>Confirmation by booking agent/other ANP agent<br>Order need to be released<br>Offering offer via CNP<br>Execution<br>Non-scheduled interruptions<br>In maintenance<br>Arrival of service team<br>Request for on-site maintenance<br>Technical issue solved<br>Severe technical issue not solvable by RO<br>Request for Remote Operation/ Manouver Approval<br>Takeover by Remote Operator<br>Remote-/Tele operation Mode<br>Intervention by SO<br>Control through Service Operator<br>Technical Problem<br>Maneuver approved by RO<br>Resumption of autonomous operation<br>Operation ouside ODD<br>Resumption of autonomous operation<br>Basic delivery cycle<br>Arrival at destination<br>Started by TOO<br>Idle / self-check / order selection<br>Next destination selected<br>Driving to next destination<br>Order cancelled/delayed by booking Agent order list rescheduled<br>ETA calculation/ communication<br>Superstate: Rendezvous<br>Parallel handling of two sub state machines. Exits if all sub state machines are in final state and no new orders can be fetched for this location.<br>Fetch next order for location<br>Initiate transaction<br>Communica-tion with booking platform<br>Authentification failed<br>Confirmation by booking platform<br>Client not available<br>Finished<br>Loading/unloading request by client<br>Authentifi-cation<br>Authenti-fication retry<br>Contacting client<br>Technical Problem<br>Successfull authenti-fication<br>Order cancelled/delayed by booking Agent<br>Loading/ Unloading<br>Final Post-delivery-assessment: Learning, ETA-update<br>Order finished<br>Planned maintenance<br>Service finished<br>Service/ recharge necessary<br>Service/ Maintenance/ Charging of Batteries<br>Driving to service station<br>Arrival at service station |
| Planning (4.) | Mathematical model of the single-vehicle routing problem with pickups and deliveries and time windows | Optimization model<br>Formulation as a Vehicle Routing Problem with Profits, Simultaneous Pickup and Delivery and Time Windows (VRP-PSPDTW)<br>$min\ z = \sum_{i \in V_0} \sum_{j \in V_0} c_t d_{ij} x_{ij} - \sum_{i \in V} p_i y_i$ (o1)<br>With:<br>$V_0 = V \cup \{1\}$ : Set of customers plus depot<br>$c_t$ : transportation cost per unit distance<br>$d_{ij}$ : distance between customer $i$ and $j$<br>$x_{ij}$ : binary arc variable, $x_{ij} \in \{0,1\}$<br>$x_{ij} = \begin{cases} 1, & \text{if there is a route between } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$<br>$p_i$ : profit collected by visiting customer $i$<br>$y_i$ : binary vertex variable, $y_i \in \{0,1\}$<br>$y_i = \begin{cases} 1, & \text{if customer } i \text{ is visited} \\ 0, & \text{otherwise} \end{cases}$<br>Constraint of route continuity, ensuring the vehicle leaves a customer after arrival:<br>$\sum_{i \in V_0} x_{ih} = \sum_{j \in V_0} x_{hj} \quad \forall h \in V$ (c1)<br>Constraint concerning a customer having no incoming/outgoing arcs if it is visited:<br>$\sum_{j \in V_0} x_{ij} = y_i$ (c2)<br>Constraint guaranteeing the vehicle to serve each customer at most once:<br>$\sum_{i \in V_0} x_{ij} \leq 1 \quad \forall j \in V$ (c3)<br>Constraints forcing the vehicle to start at the depot, leaves for another customer after arriving at the first one and terminate at the depot:<br>$\sum_{j \in V_0} x_{0j} = 1$ (c4)<br>$\sum_{j \in V} x_{1j} = \sum_{i \in V} x_{i1}$ (c5)<br>$\sum_{i \in V_0} x_{i0} = 1$ (c6)<br>Initial load constraint:<br>$L_1 = \sum_{i \in V_0} \sum_{j \in V} D_j x_{ij}$ (c7)<br>With:<br>$L_1$ : Load when leaving the depot<br>$D_j$ : Freight to deliver to customer $j$<br>Vehicle load after the first customer:<br>$L_j \geq L_1 - D_j + P_j - M(1 - x_{1j}) \quad \forall j \in V$ (c8)<br>With:<br>$L_j$ : Load of vehicle *after* leaving customer $j$<br>$P_j$ : Pickup demand of customer $j$<br>$M$ : arbitray big number, associated with the big-M-method<br>Vehicle load en-route:<br>$L_j \geq L_i - D_j + P_j - M(1 - x_{ij}) \quad \forall i \in V; \quad \forall j \in V; \quad i \neq j$ (c9)<br>Initial capacity constraint:<br>$L_1 \leq C$ (c10)<br>With:<br>$C$ : Vehicle capacity<br>En-route capacity constraint:<br>$L_j \leq C + M\left(1 - \sum_{i \in V_0} x_{ij}\right) \quad \forall j \in V$ (c11)<br>Constraints ensuring a sufficient battery level:<br>$0 \leq l_j \leq Q - hd_{ij}x_{ij} \quad \forall i \in V_0; \quad \forall j \in V_0; \quad i \neq j$ (c12)<br>$0 \leq l_j \leq l_i - hd_{ij}x_{ij} + Q(1 - x_{ij}) \quad \forall i \in V; \quad j \in V_0; \quad i \neq j$ (c13)<br>With:<br>$l_i$ : battery level when arriving at customer $i$<br>$Q$ : battery capacity<br>$h$ : battery consumption rate per unit distance<br>Time window constraint at customer (hard constraint)<br>$a_i \leq s_i \leq b_i \quad \forall i \in V_0; \quad \forall j \in V_0$ (c14)<br>With:<br>$s_i$ : Service begin time at customer $i$<br>$s_i = 0$ if customer $i$ is not served<br>$a_i$ : lower limit of time window at customer $i$<br>$b_i$ : upper limit of time window at customer $i$<br>Time of service of the next customer<br>$s_j \geq s_i + t_i + t_{ij} - M(1 - x_{ij})$ (c15)<br>With:<br>$t_i$ : Service time at customer $i$<br>$t_{ij}$ : Travel time from customer $i$ to customer $j$ |
| … | … | … |

### 3.4 Implementing the Software Agents with Spade

The Spade development environment was chosen for the agent implementation because it can be used to build most of the necessary components of a software agent or at least allows the integration of various Python libraries (see Table 4 below). Spade requires a XMPP server for the inter-agent communication that needs to be accessible for all software agents. A guide for setting up your own XMPP-Server was written as part of this research project (https://roms.dfki.de/prosody.html, Pieper 2023). Spade provides classes that perform the XMPP-based communication. Spade classes also directly support the setup of finite state machines for the agent's self-regulation loops. For the collaboration between software agents, there were neither Spade classes nor Python libraries to be used out of the box. Therefore, we decided to develop a Spade extension during our research project that allows for different types of order assignments (with or without user interaction) among software agents using CNP-based auctions. Optimization models are formulated and solved with the Python package Pyomo for classical heuristics and with PyTorch, NumPy, and gymnasium for reinforcement-learning-based approaches. Reinforcement learning (without optimization) can also be supported by a combination of PyTorch, NumPy, and gymnasium. If sensors or actors need to be integrated, vertical communication using web sockets can be set up with the Python package Flask-SocketIO. To develop user interfaces for the agents in Python, we used aiohttp, jinja2, and plotly. Finally, we used Docker to virtualize the Spade agents for their easy deployment on edge devices or remote servers – and for inter-agent learning we set up a sharded NoSQL MongoDB database which allows sharing and storing data across domains and agent types in a distributed fashion. The use of sharded replica sets in Docker allows for fast, reliable data storage. With this stack of technologies, we are currently implementing the software agents for our use case. A demonstrator for our MAS consisting of a booking platform, booking agents, bordero agents, PDR agents, and a small-scale robot will be presented at the upcoming Hanover Fair 2024.

**Table 4.** Spade classes and Python libraries to develop the different agent components.

| **Software agent component** | **Python** | **Spade** | **Python libary** | **Individual solution** |
|---|---|---|---|---|
| Self-regulation (see 1.in Figure 1) | | x | | |
| Learning (3.) | | | x (PyTorch, NumPy, gymnasium for reinforcement-learning) | |
| Planning (4.) | | | x (Pyomo for classical optimization & PyTorch, NumPy, gymnasium for reinforcement-learning based optimization) | |
| Collaboration (5.) | | | | x |

| | | | | |
|---|---|---|---|---|
| Knowledge base (6.) | x | | x | |
| Communication with environment (7.) | | x (XMPP) | | |
| UI (8./13.) | | x (for agent status) | x (aiohttp, jinja2, plotly) | |
| Sensor and actor integration (9./10.) | | | x (Flask-SocketIO) | |

## 4 Conclusion

In this paper, we present a procedure for the development of a MAS in the ongoing research project Gaia-X 4 ROMS. This procedure is similar to the available methodologies for designing agents (e.g. AAII and Prometheus) in its main steps: first building an overview of the different types of software agents, then building an internal model for each type of software agent. Our procedure uses the reference architecture of Wahlster (2017) for defining the internal model of a software agent. Our procedure helps to draw lines between an agent's models and their implementation with Python, the Python-based development environment Spade, and additional Python libraries. In general, our procedure helps to understand the different components of an agent and how to implement them with Spade. All components of a software agent that we needed for our use case (self-regulation, knowledge base, planning, learning, communication, UI integration, and actor/sensor integration) can indeed be implemented using Python, Spade or a Python library – except for the mechanisms between agents to assign tasks. Therefore, we developed a Spade extension for integrating CNP-based auctions in our MAS. Other projects that want to follow our procedure and want to use Spade, also must come up with such an extension for task assignments. Additionally, our procedure proposes modelling languages (e.g., finite state machines, mathematical model, UML component model, etc.) for the different parts of the software agent except for the learning model. This is due to the limited focus on learning functionalities in the software agents in the Gaia-X 4 ROMS project. Other projects that want to follow our procedure would need to come up with their own suggestions for how to model their agents' learning mechanisms before implementing them. In the future we want refine our procedure accordingly – and also want to publish our Spade extension for CNP-based auctions. Before doing so, we aim to complete the implementation of the six agent types.

**Acknowledgments.** This paper was written as part of the project Gaia-X 4 ROMS - Support and Remote Operation of Automated and Networked Mobility Services (FKZ: 19S21005C). The joint project is funded by the German Federal Ministry of Economics and Climate Protection (BMWK). The authors are responsible for the content of this article.

## References

1. Gath, M.:. Optimizing Transport Logistics Processes with Multiagent Planning and Control. Springer Vieweg, Wiesbaden, Germany (2016).
2. Gregor, S., Chandra Kruse, L., Seidel, S.: Research Perspectives: The Anatomy of a Design Principle. Journal of the Association for Information Systems 21(6) (2020)
3. Heinbach, C., Gösling, H., Meier, P., Thomas, O.: Smart Managed Freight Fleet: Ein automatisiertes und vernetztes Flottenmanagement in einem föderierten Datenökosystem. HMD Praxis der Wirtschaftsinformatik 60, 193–213 (2022)
4. IBM Corporation: An Architectural Blueprint for Autonomic Computing. IBM, Hawthorne, USA (2006)
5. Maecker, D., Gösling, H., Heinbach, C., Kammler, F.: Exploring Multi-Agent Systems for Intermodal Freight Fleets: Literature-based Justification of a New Concept. Wirtschaftsinformatik 2023 Proceedings, Paderborn, Germany (2023)
6. Palanca, J.: Spade documentation, https://spade-mas.readthedocs.io/en/latest/, last accessed 2024/02/19
7. Pieper, T.: How to Setup a SPADE-Ready Prosody Server on Ubuntu / Raspberry PI, https://roms.dfki.de/prosody.html, last accessed 2024/02/19
8. Wahlster, W.: Künstliche Intelligenz als Grundlage autonomer Systeme. Informatik-Spektrum 40(5), 409–418 (2017)
9. Wooldridge, M.: An Introduction to Multiagent Systems. 2nd edn. John Wiley & Sons, West Sussex, UK (2009)