LAMDA - Model specification

Iris Lorscheid
Hamburg University of Technology
Institute of Management Accounting and Simulation
iris.lorscheid@tuhh.de

August 8, 2016

Contents

1 LAMDA - Model Description

The simulation model LAMDA (*Learning Agents for Mechanism Design Analysis*) will be described along the ODD protocol [3, 4]. The ODD protocol (*Overview, Design concepts and Details*) is a standard developed for the specification of simulation models. This guides the

modeler through the description and helps to prevent the omission of important details. Furthermore, the communication of simulation models is facilitated.

The setup of the ODD protocol follows the logic *overview*, *design concepts*, and *details* [3, p.117]. The first part, *overview*, provides the most important information about the simulation model. Next, *design concepts* describe the theoretical background, modeling approaches, and concepts applied in the given simulation model [4, p.9]. Finally, more detailed (in particular *technical*) information about the simulation model is given in the third part.

1.1 Overview

This section gives an overview of the main model elements. The first subsection summarizes the *purpose* of the model. The *entities, state variables* and *scales* of the simulation model LAMDA are described afterwards. Finally, the last subsection describes the main *process* and *scheduling* of the model.

1.1.1 Purpose

The main purpose of the model is to assess the influences of varying cognitive abilities of the decision maker on the theoretically well-founded truth-inducing effect of the Groves mechanism [5, 6]. The study focuses on the effect of the combination of different information states and ways of human learning, represented by learning models for agent-based simulation. The strategies chosen by the agents are evaluated with regard to the individual payoff achievement and reporting behavior. By this, the individual success as well as the truth-inducing effect under the respective cognitive influence can be analyzed and compared. In the end, the robustness of the Groves mechanism against not perfect rationality and agent interactions should be assessed.

1.1.2 Entities, state variables and scales

For the implementation of the model, the agent concepts and processes have to be transferred into quantified variables and values. This section describes the four entities division manager agent, headquarter, groves mechanism, and observer, and their variables of the LAMDA simulation model. Furthermore the variable types and scales are described. These provide the basis for the subsequent implementation.

Table 1 provides an overview of the variables and scales of the entity **division manager agent**. For clarity, the variables are grouped in *scenario variables*, *decision variables*, and *learning model*. The scenario variables specify the given scenario by defining the capacity thresholds for the divisions (*optCapacity* and *maxCapacity*) and the value of productivity reduction (*reductProd*).

The decision variables are needed for the decision process of the division managers and change in every simulation step. In the simulation model, two division manager agents are implemented, according to the experimental setting. Their profit-maximizing choice about the productivity value report (*repProd*) is the central process in the model. The *prodValue* is the divisions' real productivity value in the given round. Based on the reports, a resource allocation (variable *resourcesAllocated*) is calculated. The resource allocation and real productivity value are the basis for the *profit* achieved by the divisions, followed by a *compensation*, given as feedback to the division manager decision by the headquarter entity, determined by the Groves mechanism. The reported productivity of the opponent (*repProdOpp*) is given as information to the agent in the end of the round.

Table 1: State variables and scales of the division manager entity

Variable	**Description**	**Scale**
Scenario variables		
optCapacity	Optimal capacity (productivity threshold I)	Integer
maxCapacity	Max. capacity (productivity threshold II)	Integer
reductProd	Reduction of productivity for resource units between threshold I and II	Double
Decision variables		
prodValue	Real productivity value (begin of round)	$[P_{min}, P_{max}]$
repProd	Reported productivity value (decision)	$[P_{min}, P_{max}]$
resourcesAllocated	Allocated resources (after report)	Integer
profit	Achieved profit after production	Double
compensation	Bonus payment by headquarter	Double
repProdOpp	Reported productivity by opponent	$[P_{min}, P_{max}]$
Learning Models		
cognitiveAbilitiy	Learning model, defined by learning algorithm and knowledge base	{ZI, LM_1, LM_2, LM_3, LM_4}
Sub-Var. ZI		
probDistrib	Probability distribution over strategy set for random action choice	{uniform}
Sub-Var. LM_1 / LM_2		
λ	Learning parameter (SV reinforcement learning model)	[0,1]
Sub-Var. LM_{2+}		
sysExplor	LM_2 + Systematic exploration	Boolean
k	Times each possible action is chosen	Integer
Sub-Var. LM_3		
repProdOpp	Opponents' reporting behavior (Fict. Play)	Boolean

Table 2: State variables and scales of the headquarter entity

Variable name	**Description**	**Scale**
Scenario variables		
divisionManagers	Division managers in the firm	Entity: division manager
resources	Available resources on the company level	Integer
optCapacity	Optimal capacity of resource units per division (productivity threshold I)	Integer
maxCapacity	Maximum capacity of resource units per division (productivity threshold II)	Integer
reductProd	Reduction of the divisions' productivity value for resource units between threshold I and II	Double
rangeProdValues	Range of possible productivity values in divisions	$[P_{min} + \epsilon, P_{max} - \epsilon]$
Decision variables		
repProds	Reported productivity values by the division manager agents	Array of double values
allocRes	Allocate resources to divisions	Array of double values
profitDiv	Achieved profit by divisions	Array of double values
payComp	Compensation payment to divisions (calculated by the Groves mechanism)	Array of double values
Incentive Scheme		
incentiveScheme	Incentive scheme by which the compensation is calculated	Groves mechanism

The variable *cognitiveAbility* is the core concept in this model. It represents the decision strategy, knowledge, and learning behavior of the division manager agent. The cognitive ability is varied by different learning models. The values of *cognitiveAbility*, thus, are sub-variables having a sub-structure determined by the learning model elements. The chosen learning models LM_{1-3} will be described more detailed in the section submodels (see below, Section 1.3.3).

Table 2 gives an overview of the state variables and scales of the **headquarter** entity. For readability, the variables are grouped in *scenario variables*, *decision variables*, and *incentive scheme*.

The scenario variables comprise again the capacity thresholds (*optCapacity* and *maxCapacity*) and productivity value reduction (*reductProd*). Those are needed for the headquarter as basis for the resource allocation decision (described in detail in submodels, see Section 1.3.3). Additionally, the headquarter knows the divisions in the firm (*divisionManagers*, here two divisions). Furthermore, the headquarter knows the number of resource units (*resources*) available to the firm. Those have to be allocated to the divisions in the most effective way.

In the decision process, the headquarter decides about the allocation of resources (*allocRes*) based on the productivity reports (*repProds*), reduced productivity value, and the given capacity thresholds. The headquarter knows the range of possible productivity values (*rangeProdValues*)

Table 3: State variables and scales of the Groves mechanism entity

Variable name	**Description**	**Scale**
Output		
payment	Payment to the agents	Double
Input		
profit i	Achieved profit by division i	Double
expProfit j	Expected profit by division j	Double
Others		
discount	Discount factor for compensation	Double
$G_{\mathrm{i}}(\cdot)$	Function - part of the Groves mechanism	set to 0
$H_{\mathrm{i}}(\cdot)$	Function - part of the Groves mechanism	set to 1

with an epsilon (ϵ), representing the information asymmetry between headquarter and division managers. The achieved profit by the divisions is known by the headquarter (*profitDiv*). The headquarter calculates the compensation payments by the *incentiveScheme* (here: Groves mechanism) and pays it to the division as bonus.

Here, the headquarter only forwards decisions, information, and payments between the division and the compensation scheme. Also the resource allocation is a fully rational decision, determined by an optimization algorithm. This is due to the setup of the laboratory experiment. Here, the headquarter was represented by a computer, calculating the optimal resource allocation and payment by the Groves mechanism functions. It is conceivable to implement the headquarter as an agent with varying cognitive abilities as well. But, the focus in this study is set on the effect of the Groves mechanism on the division manager, so that behavioral influences from the headquarter are left out. The headquarter allocates the resources deterministically, based on the reported productivity values and the known capacity thresholds. For the payment to the division managers, the Groves mechanism is used. The headquarter lacks of autonomous or self-directed behavior. Therefore, its action do not even have to be conducted by an own entity but simulated by commands in the main class of the simulation model. Nonetheless, the headquarter is implemented as entity in this simulation model. On the one hand, this makes the organizational structure in the simulation explicit. On the other hand, future extensions should be prepared and facilitated by structuring the headquarter as entity.

The **Groves mechanism** entity determines the payment to the division managers. Its variables are listed in Table 3. For the description, the division manager agent for which the payment should be calculated is indicated as agent *i*, and its opponent as agent *j*. The Groves mechanism is defined by a deterministic function over the actual achieved profit Π_{i} by agent i (in the simulation denoted by $profit_i$) , and the expected opponents' profit $\hat{\Pi}_{\mathrm{j}}$, based on the productivity report by the opponent j (in the simulation denoted by $expProfit_j$), as follows:

$$C_{\mathrm{i}} = G_{\mathrm{i}}(\hat{P}_{\mathrm{j}}) \cdot (\Pi_{\mathrm{i}} + \hat{\Pi}_{\mathrm{j}}) + H_{\mathrm{i}}(\hat{P}_{\mathrm{j}}) \tag{1}$$

Table 4: State variables and scales of the observer entity

Variable name	**Description**	**Scale**
Independent variables		
cogAbility	Cognitive ability of the division manager agents	{ZI, LM_1, LM_2, LM_3, LM_4}
Dependent variables		
delta	*Central measure I - truth-inducing effect:* Delta between reported and real productivity value	Double
compensation	*Central measure II - individual success:* Quality of decision as average compensation payment	Double
Further input variables		
incentiveScheme	Compensation scheme used by the headquarter	Groves mechanism
pMin	Minimum productivity value, specifying the division agents' strategy space	Double
pMax	Maximum productivity value, specifying the division agents' strategy space	Double
resources	Available resources on the company level	Integer
optCapacity	Optimal capacity of resource units per division (productivity threshold I)	Integer
maxCapacity	Maximum capacity of resource units per division (productivity threshold II)	Integer
reductProd	Reduction of the divisions' productivity value for resource units between threshold I and II	
Further output measures		
avgP	Average profit on the company level	Double

C_{i} - compensation payment for agent i

$G_{\mathrm{i}}(\hat{P}_{\mathrm{j}})$ - positive function based on reported productivity j, here = 1

Π_{i} - actual profit i

$\hat{\Pi}_{\mathrm{j}}$ - reported profit j

$H_{\mathrm{i}}(\hat{P}_{\mathrm{j}})$ - arbitrary function based on reported productivity j, here = 0

According to the experimental setting, the strictly positive function $G_{\mathrm{i}}(\hat{P}_{\mathrm{j}})$ is set to 1, and the arbitrary function $H_{\mathrm{i}}(\hat{P}_{\mathrm{j}})$ to 0.

The central management entity in the simulation is the main class **observer**[1]. This entity schedules the agent actions and controls the course of the simulation. By this, the simulation experiments are controlled. The input values and observed output measures for each simulation round are written in an output file providing the basis for the simulation data analysis. Consequently, the observer entity contains a comprehensive list of all global input and output variables of the simulation model and its entities (see Table 4).

[1]The terminology *observer* for the central entity was chosen based on the terminology used in NetLogo (see [12]).

The core relationship in a model is described by the relation between *independent* and *dependent* variables. In this model, this is the effect of cognitive ability (*cogAbility*) on the two variables truthful reporting behavior and individual success. The truthful reporting behavior is evaluated by calculating the difference between true and reported productivity (denoted in the simulation model by *delta*). The average payment of the *compensation* indicates the individual success. These core relationships are analyzed to investigate the robustness of the compensation scheme (Groves mechanism) against varying cognitive abilities.

In addition to these variables, further input variables specify the model scenario: The *incentiveScheme* indicates the mechanism by which the compensation payment for the agent is calculated. Here, only the Groves mechanism is considered. The input parameters *pMin* and *pMax* specify the productivity scale of the agents. In this simulation model, the productivity scale is the same for both division manager agents. Furthermore, the number of resources available to the company are given (*resources*), the capacity thresholds (*optCapacity* and *maxCapacity*), as well as the productivity reduction (*reductProd*). The additional output measure *avgP* measures the profit achieved on the company level. All output values are measured and written in the output-file for each division manager agent separately.

1.1.3 Process Overview and Scheduling

The process and scheduling of the simulation model LAMDA can be described by one main process. Based on the course of the experiment, an excerpt of the budgeting process is implemented. An overview of the process implemented in the simulation model is given in Figure 1. This figure illustrates the process between the entities Groves mechanism, headquarter, division manager, and observer in a flow chart. To complement the picture, the figure shows the not implemented part of the budgeting process between productivity report and resource allocation. This part is depicted in dotted lines.

The implemented process consists of various stages defining the course of *one simulation run*. First, a real productivity value for one run and each division manager agent is calculated randomly[2]. Based on this, the decision process of each division manager agent takes place. The division manager agent perceives its real productivity value and decides about its productivity report to the headquarter. The subsequent negotiation process is left out in the simulation model. Next, the headquarter allocates the resources to the divisions, based on the reported productivity values and known capacity thresholds in the divisions. Afterwards, the divisions implement the allocated resources. The achieved profit provides (next to the reported productivity values) a basis for the compensation calculation. For this, the headquarter entity invokes a method in the Groves mechanism entity. Finally, the calculated payment is given to the division manager agents, which process the compensation received.

Underlying the main process, several *sub-processes* such as the decision and learning processes of the division manager agent take place. Those are only listed here and described in detail in Section 1.3.3 (sub-models). The sub-processes, for referencing denoted and numbered by *P#*, are (P1) definition of the real productivity value, (P2) decision process for a productivity report by the division manager agents, (P3) resource allocation by the headquarter, (P4) production and achieved profit by the divisions, (P5) compensation calculation by the Groves mechanism, and (P6) bonus processing by the division manager agent.

[2]The real productivity value is chosen randomly with a uniform distribution over the set of possible productivity values between P_{min} and P_{max}.

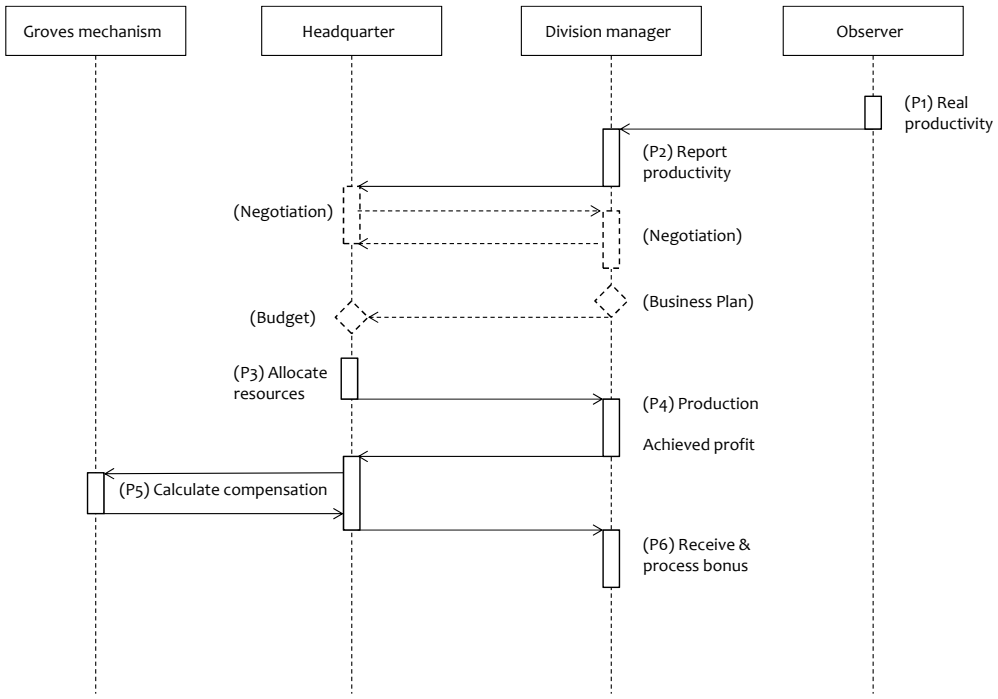

Figure 1: Overview of one round in the simulation model, source: own work.

1.2 Design Concepts

This section gives an overview of the main context and modeling approaches underlying the simulation model. The concepts should serve as a checklist for the modeler about all relevant modeling decisions and provide a way to communicate these [4, p.10].

1.2.1 Basic Principles

The simulation model LAMDA builds on two fields of research: (1) *ABS* as assessment tool for the application of (2) *mechanism design theory* (see Figure 2). The aim is to make use of ABS for the analysis of mechanisms under the complication that the decision maker has no perfect rational cognitive abilities. For modeling (and varying) these capabilities, learning agents are used. Therefore, the agents are equipped with learning algorithms applied in ACE. To give an example, LAMDA analyzes the effect of incentive schemes (here the Groves mechanism) in management control.

1.2.2 Emergence

This study focuses on the micro learning behavior and success of the division manager. Nevertheless, there are results possible on the emergent level resulting from coordinated behavior between the agents. Therefore, the behavior of both division manager agents is observed. Furthermore,

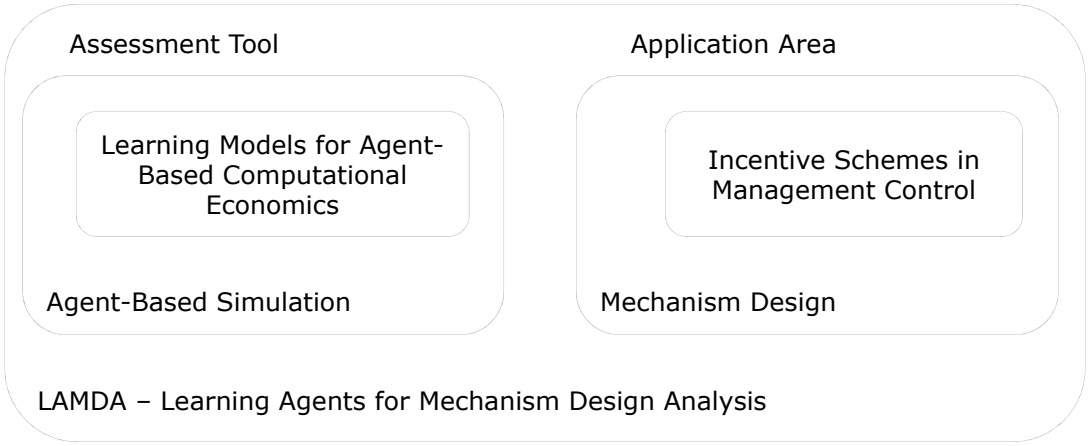

Figure 2: (Main) Fields underlying the model design of LAMDA

the overall company profit is measured to analyze the organization success in comparison to the individual payment and reporting behavior.

1.2.3 Adaptation

The learning behavior of the division manager agents is a central element in this model. A detailed consideration of their adaptive behavior is given in Section 1.3.3 (submodels). The headquarter agent only behaves deterministically.

1.2.4 Objectives

The division manager agents' objective is profit maximization. They strive for maximizing the compensation given from the headquarter. Therefore, the division manager agents explore their reporting behavior to achieve the maximum payment. The objective of the headquarter agent is to maximize the overall company profit. Therefore, the headquarter allocates the resource in the most efficient way, based on the reported productivity values from the divisions.

1.2.5 Predicting

Depending on the learning model, the division manager agents may predict the future reporting behavior of the opponent. Therefore, they observe the reports of the other division manager agent (post-report, in the end of the round) and build beliefs about the probable next reported value. Therewith, they may adapt their own decision to expected circumstances. The headquarter agent does not predict.

1.2.6 Sensing

The division manager agent senses its real productivity value, the resources allocated to its division, the reported productivity value of the opponent, and the compensation value given by the headquarter. The headquarter agent senses the reported productivity values by the division managers, the profit achieved by the divisions, and the compensation for each division, given from the Groves mechanism.

1.2.7 Interaction

In this model, the division manager agents may not contact each other by sending messages or negotiating about the next productivity reports. However, they may interact with each other indirectly. The division manager agents share resource units and may observe the reported productivity value of each other. Furthermore, their payment depends on the reported productivity of the opponent. By anticipation of each other's reports, a coordination of decisions may take place without communicating with each other explicitly.

By an extension to this model, communication between the division manager agents may be implemented. As the effect of communication on truthful reporting under the Groves mechanism is not in the focus of this study, this issue is not implemented here.

The division manager agents and the headquarter agent on the other hand do communicate with each other. The division managers send their reports to the headquarter, and the headquarter responds with information about the resource allocation. After implementing the resources, information about the profit of each division is given to the headquarter. Based on this, the headquarter calculates the compensation by the Groves mechanism and forwards it to the division manager agents. A method in the observer entity (which is the main class in the model) controls the information flow and communication.

The LAMDA simulation model does not implement a negotiation process between headquarter and division managers.

1.2.8 Stochasticity

The following parts of the simulation model contain stochastic elements. First, the real productivity values for every division manager agent and round are set randomly with a uniform probability for each strategy. This process does not contain any path dependency as the selection is started for each round and agent with the initial conditions. Second, the division manager actions are chosen stochastically within the decision process (for details about the process and selection probabilities see Section 1.3.3, sub-models).

The decision of the headquarter agent and the calculation within the Groves mechanism, however, is deterministic.

1.2.9 Collectives

The two division manager agents and the headquarter in this simulation model belong to an organization. In this model, the group level is no emergent phenomenon but modeled explicitly. However, the organization is not implemented as entity with own attributes and methods, but sets the frame in which the implemented budgeting process takes place.

1.2.10 Observation

Within each simulation run, a report of the simulation behavior is written in a .csv file. This protocol stores the parameter values of all state variables (see above, Section 1.1.2). By output measures, the success of resource allocation and compensation scheme may be observed on the individual and the organizational level. Therewith, a statistical analysis may be conducted after the simulation experiment.

1.3 Details

After describing the concepts and modeling approaches behind the model, this section provides more technical details. This should facilitate later replications of the model. This part describes the initial conditions of the model indicating the variable values for the simulation runs. Furthermore, the simulation model input data are described, by which dynamic environmental circumstances may be determined. Finally, the sub-model section provides insights into the course of sub-processes within the main process of the model.

1.3.1 Initialization

The values by which the variables are initialized for one simulation run depend on the experimental design of the simulation model.

1.3.2 Input Data

The model does not use input data to represent processes over time.

1.3.3 Submodels

This section provides insights into the definition and execution of all processes within a simulation run. Describing the processes in such detail fosters the understanding of the simulation model. Moreover, all model elements and their verification may be reconstructed by the reader. This increases the reliability of the simulation results and decreases the black box effect of simulation models as perceived by many.

The description is organized along the sequence of the sub-processes within the main process, as described in Section 1.1.3, process overview.

P1 - Definition of the real productivity value Algorithm 1 describes the definition process of a real productivity value for one division manager agent in one simulation round. For each division manager agent, the observer entity chooses a *real productivity value* at the beginning of a simulation run. The set of possible productivity values (P) is defined by increments of 0.1 between P_{min} and P_{max}. From this set, one value is chosen randomly with a uniform distributed probability for all possible productivity values. By this, every value is chosen approximately for the same number of rounds.

Alternatively to random choice, the real productivity value may be set systematically. The real productivity value may be initialized in turns by each possible value. By this, each value would be chosen by the exact same number. In this simulation model, however, the real productivity is defined randomly according to the setting of the laboratory experiment.

Algorithm 1: Observer: Set real productivity value (P1)

Input: Productivity scale (P_{min}: double, P_{max}: double)
Output: Random productivity value (prodValue: double)
random $\leftarrow$ random number in (length of productivity scale - 1);
productivity value = (random $\cdot$ 0.1) + P_{min};
return *productivity value*

P2 - Decision process for the productivity report This decision process is a central element in the LAMDA simulation model. Here, the division manager agent decides about the productivity value to report to the headquarter. The agent may learn about the value of actions. For choosing a strategy, the agent uses the planning strategy given from the respective learning model. Those are described in the following.

Algorithms 2 to 4 show the planning processes in pseudo code. To identify the sub-processes, they are denoted by *P2.x*, with x as placeholder for the learning model shortening.

Zero intelligent agents with learning model *ZI* act arbitrary without any reasoning or adaption. Accordingly, the decision process follows the same procedure as Algorithm 1. The reported productivity value is chosen randomly out of the set of possible productivity values, independently from the real productivity value observed by the agent.

Algorithm 2 describes the decision process by learning model LM_1. The knowledge base of this model comprises only simple condition-action rules. The agent knows the set of possible productivity values to report. A strength value indicates the value of the reports. In this simple model, the strategy is only to choose a productivity value for the report, without relation to the true productivity of the division. Over time, the agent adapts the strength values in the knowledge base by the SV learning algorithm (for a description of the adaption process see below). The strength of a strategy indicates the experience of the agent with this reported value implicitly.

By Algorithm 2, the process of a *roulette wheel selection* is described. This is a common mechanism for random selection based on a given value or probability. This algorithm is widely applied in optimization techniques and genetic algorithms (see, e.g., [11] and [7]). The concept as applied in the LAMDA model is described in the following.

Each rule in the knowledge base of the agent is assigned a value $area_i$ between 0 and 1. The size of an area indicates the *probability* of a rule to be chosen. This corresponds to its strength value and is calculated by:

$$probability_i = \frac{strength_i}{\sum_{i=1}^{N} strength_i} \tag{2}$$

Based on this probability value, the *area* for each rule in the scale between 0 and 1 can be determined. For the implementation of roulette wheel selection, a *threshold* for each rule is calculated, indicating the upper limit of the areas. The threshold results from the sum of probabilities. For roulette wheel selection, a *random number* r between 0 and 1 is chosen. This number determines the winning rule, which is the one to which the value can be attributed. The random number lies in the 'area' of the rule, in analogy to the ball landing on a part of the wheel after spinning it in the roulette casino game.

Table 5 shows an example to illustrate the process. This example assumes three rules, rule_1 - rule_3, with the strength values 2.0, 4.0, and 2.0. According to Equation 2, the strength values result in area sizes (probabilities) of 0.25, 0.50, and 0.25. Thus, the areas of the rules on the

Algorithm 2: Division agent: Decide about productivity value report (P2.LM_1 and P2.LM_2)

Input: Observation: No observation for LM_1, real productivity value for LM_2 (prodValue: double)

Output: Decision: Reported productivity value (repProd: double)

Choose action of rule_i with probability proportional to strength_i **return** *action of* $rule_i$*;*

Table 5: Example for roulette wheel selection

	$\mathbf{strength_i}$	$\mathbf{area\text{-}size_i}$	$\mathbf{area_i}$	$\mathbf{threshold_i}$
$rule_1$	2.0	0.25	[0.00, 0.25]	0.25
$rule_2$	4.0	0.50	]0.25, 0.75]	0.75
$rule_3$	2.0	0.25	]0.75, 1.00]	1.00

Algorithm 3: Division agent: Decide about productivity value report ($P2.LM_{2+}$)

Input: Observation: Real productivity value (prodValue: double)
Input parameter k: int
Output: Decision: Reported productivity value (repProd: double)
while *rule set has more elements* **do**
  choose next $rule_i$;
  if *counter for* $rule_i \leq k$ **then**
    counter ++;
    return *action of* $rule_i$
  end
end
return choose productivity value report stochastically (Algorithm 2);

roulette wheel are the intervals [0.00, 0.25],]0.25, 0.75], and]0.75, 1.00]. For implementing roulette wheel selection, the random number will be compared with the thresholds 0.25, 0.75, and 1.00.

Algorithm 2 shows the algorithm as it is implemented in the simulation model LAMDA. This algorithm is used for the decision process of learning models LM_1 and LM_2. The same course of decision process applies for both models, with the difference of a more specific agent knowledge in LM_2. In contrast to LM_1, here the agent builds up experience for every observed real productivity value. Therewith, the agent remembers and learns about the value of productivity reports in relation to its actual productivity. The explicit experience collection of LM_2 is represented by a more extensive knowledge base, containing observed real productivity values.

LM_{2+} defines systematic exploration as additional cognitive feature to LM_2 (Algorithm 3). In this model, all strategies are chosen k times. By this, the division manager explores the effect of all productivity reports systematically. Assuming a value of k=3 and a set of 9 productivity values, the exploration phase would last 27 simulation rounds. Comparing LM_{2+} with LM_2, the exploration phase lasts longer accordingly (in the example $k \cdot \text{number of strategies}^2 = 3 \cdot 9^2 = 243$ rounds), because the agent explores the productivity reports for all observed real productivity values. After the exploration phase, the division manager agent chooses the productivity value stochastically, according to learning models LM_1 and LM_2.

As a variation, a systematic exploration of over-, underestimation, and truthful reporting is conceivable. This would fit to the intention of the invented training rounds in the experiment of [1]. Therefore, the definition of a distance for over- and underestimation would be necessary. Moreover, a strategy for minimum and maximum values had to be defined. For simplification and to overcome the dependencies to selected distance values, a systematic exploration phase for the whole strategy set was chosen for this model. In the end, the analysis of over- and underestimation is covered by this.

In the third learning model, LM_3, the division manager agent anticipates the reporting

behavior of its opponent and takes this into account in the decision for the own productivity report. For this, the fictitious learning algorithm is chosen. Here, the agent knows the best response to the behavioral alternatives of the opponent. Therefore, the agent needs complete knowledge about the feedback function. The prediction of the opponents' behavior is used as additional knowledge in the decision process for the productivity report.

For prediction, the division manager agent uses his knowledge about all observed productivity reports by the opponent in past simulation rounds. Based on the frequency of occurrence, the agent calculates the probability for the next report. Accordingly, a roulette wheel selection may be used again. For this, the number of occurrences provides the strength value by which the probability values and value ranges are calculated. Algorithm 4 provides an overview.

This prediction method does not enable the agent to learn about more complex behavioral pattern such as the character of report sequences. This can be achieved by a more complex concept such as the event board described in [9]. Here, the agent memorizes not only the number of occurrences but also the sequences of events. By this, typical decision orders may be revealed.

Algorithm 4: Division agent: Decide about productivity value report ($P2.LM_3$)

Input: Observation: Real productivity value (prodValue: double)
Output: Decision: Reported productivity value (repProd: double)
Choose $report_i$ with probability proportional to the number of $occurences_i$ **return** choose best response to prediction;

P3 - Allocate resources Based on the reported productivity values by the division manager agents, the headquarter allocates the resources available to the company. The productivity is not the only criteria for the most efficient distribution of resources, as the divisions are not able to implement an unlimited number of resource units to their full productivity. The efficiency of divisions depends on their capacities. Thus, the headquarter considers also the *capacity thresholds* (see above, Section 1.1.2) within the divisions. By this, the coordination problem in multi-divisional firms is represented.

For including this coordination problem in the simulation model LAMDA, an algorithm for resource allocation was designed and implemented (based on the setting in the laboratory experiment by [1]). By this, coordination scenarios may be described on an abstract, yet applicable level for many situations. The existence of limited capacities influencing the effectiveness of implementations can be observed frequently, such as the limited capacity of human employees in teams, or of (human or technical) stations in production lines. As the algorithm for effective resource allocation in such scenarios is complex, the following description is divided in three Algorithms 5, 6, and 7, according to the implemented methods in the program code. This algorithm is designed for only two divisions. The application of this algorithm for more divisions would require an extension.

In these algorithms, the following abbreviations and terminologies are used. A division manager agent is named with *DM*. The two agents implemented in the simulation model are shortened with DM 1 and DM 2. The resources allocated to the division agent with the higher reported productivity value are named with $resources_{DM\ report(+)}$, the resources allocated to the division with lower productivity denoted by $resources_{DM\ report(-)}$ accordingly. The other variables are compliant with the variables described in Section 1.1.2. The expression $resources_{DM\ 1} \leftarrow resources/2$ means that half of the resources available to the company are allocated to division

manager agent 1. Expression $resources_{DM\ report(+)} \leftarrow optCapacity$ indicates that the division with the higher reported productivity receives resource units to the amount of the optimal capacity of the divisions.

Algorithm 5: Headquarter: Resource allocation (P3)

Input: Observation: Reported productivity values by division managers (DM) ($report_1$: double, $report_2$: double)
Output: Decision: Resource allocation for division manager (DM) 1 and 2 (allocRes: int[])

```
double difference ← |report1 - report2|;
if difference == 0 then
    return resources_DM 1 ← resources/2 ;
    return resources_DM 2 ← resources/2 ;
    if (resources % 2) != 0 then
        allocate rest to random division manager;
    end
else
    resources_DM report(+) ← optCapacity ;
    if report_DM report(-) > ( report_DM report(+) - reductProd ) then
        return resources ← Algorithm 6 (report1: double, report2: double) ;
    else
        return resources ← Algorithm 7 (report1: double, report2: double);
    end
end
```

Algorithm 5 distinguishes between the following cases. In the case of *equal productivity reports*, the resources are divided homogeneously between the divisions. Because the parameter *difference* is a double value, the equality is checked by an error of .001. If the difference of the reports is less than .001, they are evaluated as equal. The reports are double values with increments of .1 in the default setting.

If one division reports a *higher productivity* value than the other, the distribution of resources depends on the level of difference between the productivity values. The divisions may only use a reduced productivity for resources between the *optimal capacity* and *maximum capacity*.[3] This challenges the resource allocation process, as the best decision is not only to allocate the resources to the division with the higher reported productivity. The reduced productivity for resources above the optimal capacity has to be taken into account as well. This is solved by the algorithms as follows.

First, the resources up to the optimal capacity are given to the division with the higher reported productivity value. For the resource units above this threshold, two basic cases are distinguished:

1. The lower reported productivity *is higher than the reduced productivity of the division with the higher reported value.* In this case, it is more efficient to allocate the remaining resources up to the *optimal capacity* to the division with the *lower* value, and the rest (if existent) to the other division with the higher report (see Algorithm 6).

[3] In this simulation model, the same capacity thresholds and productivity reductions are assumed for both divisions (according to the experimental setting). In case of different levels within divisions, the algorithm would have to be extended.

Algorithm 6: Headquarter: Resource allocation (P3.1)

Input: Observation: Reported productivity values by division managers (DM) ($report_1$: double, $report_2$: double)
Output: Decision: Resource allocation for division manager (DM) 1 and 2 (allocRes: int[])

```
if resources ≤ optCapacity then
    return resources_DM report(-) ← 0;
    return resources_DM report(+) ← optCapacity;
else
    if ( resources - optCapacity ) ≤ optCapacity then
        return resources_DM report(-) ← resources - optCapacity;
        return resources_DM report(+) ← optCapacity ;
    else
        if (resources - optCapacity) ≤ maxCapacity then
            return resources_DM report(-) ← optCapacity;
            return resources_DM report(+) ← resources - optCapacity;
        else
            if ( (resources - optCapacity ) > maxCapacity ) & ( resources ≤ ( 2 ·
              maxCapacity ) ) then
                rest = resources - maxCapacity - optCapacity;
                return resources_DM report(-) ← optCapacity + rest;
                return resources_DM report(+) ← maxCapacity ;
            else
                if resources > 2 · maxCapacity then
                    restShare ← resources - 2 · maxCapacity ;
                    return resources_DM report(-) ← maxCapacity + restShare/2;
                    return resources_DM report(+) ← maxCapacity+ restShare/2;
                end
            end
        end
    end
end
```

2. If the lower reported productivity *is even lower than the reduced productivity of the division with the higher reported value*, the resource units should be assigned to the division with the *higher* value up to the *maximum capacity*, and only the rest (if existent) to the other (see Algorithm 7).

In both cases, the resource units above the sum of both division capacities are divided equally (if this case occurs).

This shows the importance for the headquarter and the company success to have *true* information about the real productivity values. If the reported productivity value does not match with the real value, the company cannot solve the coordination problem in the best possible way.

P4 - Production and achieved profit In the next step, the division manager agents implement the allocated resources. Based on the *real* productivity value and capacity thresholds, the

Algorithm 7: Headquarter: Resource allocation (P3.2)

Input: Observation: Reported productivity values by division managers (DM) ($report_1$: double, $report_2$: double)
Output: Decision: Resource allocation for division manager (DM) 1 and 2 (allocRes: int[])

```
if resources ≤ maxCapacity then
    return resources_DM report(-) ← 0 ;
    return resources_DM report(+) ← resources ;
else
    if ( resources > maxCapacity ) & ( resources ≤ ( 2 · maxCapacity ) ) then
        return resources_DM report(-) ← resources - maxCapacity;
        return resources_DM report(+) ← maxCapacity;
    else
        if resources > ( 2 · maxCapacity ) then
            restShare = resources - 2 · maxCapacity;
            return resources_DM report(-) ← maxCapacity + restShare/2;
            return resources_DM report(+) ← maxCapacity + resthare/2;
        end
    end
end
```

actual achieved profit of each division is calculated. Algorithm 8 describes the implementation process and resulting profit for one division.

Depending on the relation between the number of allocated resources and the capacity thresholds, the division may implement the resources to *full*, *reduced* or *no productivity*. Those three cases are distinguished accordingly. The allocated resources up to the optimal capacity are implemented to the full productivity. Up to the maximum capacity, the division may implement the resources to a reduced productivity.

The resources above the maximum capacity are implemented without any profit. Therefore, these resource units are first only added to the profit. In the end of the algorithm, all resource units allocated and deployed by the division are deducted from the profit. The resource units represent the costs engaged for the production process. Thus, the allocation and production of the resource units above the maximum capacity cancel each other out.

It is important to note that the resource allocation decision (P3) by the headquarter depends on the *reported* productivity value, and the achieved profit in the production process (P4) on the *real* productivity value, as this is a central point in this model. This describes the coordination problem within firms and motivates the design of incentive schemes such as the Groves mechanism for truthful reporting.

P5 - Calculate compensation Given the profit achieved by the division manager agents, the headquarter calculates the compensation for the divisions. Therefore, the Groves mechanism entity is used. For the truth-inducing effect of this bonus, the calculation for each division bases on the *expected profit of the opponent* and the *actual achieved profit of the own division.*

Algorithm 9 illustrates the course of calculation by the Groves mechanism. First, the expected profit of both divisions is calculated. Afterwards, the compensation payment for the division managers is calculated. Therefore, the sum over the achieved profit and the expected profit of the opponent is calculated. A discount factor scales the payment value.

Algorithm 8: Division agent: Implement allocated resources (P4)

Input: Resources allocated from the headquarter (resourcesAllocated: int)
Productivity value (prodValue: double)
Output: Achieved profit by division (profit: double)

```
profit ← 0.0;
if allocated resources ≤ optCapacity then
    profit += ( prodValue · allocated resources);
else
    profit += ( prodValue · optCapacity);
    if allocated resources ≤ maxCapacity then
        profit += (prodValue - reductProd) · (allocated resources - optCapacity);
    else
        profit += (prodValue - reductProd) · (maxCapacity - optCapacity);
        profit += (allocated resources - maxCapacity);
    end
end
profit -= resources;
return profit
```

P6 - Receive and process bonus In the final step, the division manager agents receive and process their compensation. Depending on the chosen learning model, the bonus is processed in the agents' knowledge base. Agents with the *ZI* model do not adapt their behavior. Consequently, no update of the agents' knowledgebase takes place. In LM_1 - LM_{2+}, the strength values of the rules are updated based on the compensation payment. This will be described in the following. Agents equipped with learning model LM_3, however, do not proceed the feedback from the Groves mechanism, as they only learn about the opponents' behavior and know the best response given from the compensation scheme.

Algorithm 10 shows the strength update for the agents' behavioral rules. In this description, t denotes the current time step of the simulation run, and $t+1$ the subsequent simulation step. By the learning algorithm, the strength value of the next step $t+1$ is calculated for all rules in the knowledge base. In Algorithm 10, $\text{rule}_{\text{sel.}}$ denotes the rule that determined the last report of the division manager. First, this rule is updated. Afterwards the strength of the other, not chosen rules are calculated, named with rule_{i}.

For the adaption process, learning models LM_1 - LM_{2+} apply the SV learning algorithm. As given from the SV learning algorithm [10], learning parameter λ determines the influence of exploration and exploitation on the new strength value $strength_{t+1}$. According to this, the sum of (1-λ) of the existing rule strength and λ of the compensation payment determine the updated strength value.

Next, the resulting difference between new and previous strength values of this rule is shared equally over the other rules in the knowledge base. Therefore, an *updateShare* is calculated by this difference divided by the number of rules - 1 (for the selected rule). This share is deducted from the previous strength values of all other rules in the knowledge base.

Algorithm 9: Headquarter with Groves mechanism: Calculate compensation (P5)

Input: Reported productivity values (repProds: double[])
Achieved profit by divisions (profitDiv: double[])
Output: Compensation payment (payComp: double[])
for *all division manager agents* **do**
  $\text{expProfit}_{\text{DM 1}} \leftarrow \text{profit}_{\text{DM 1}}$ with *reported* productivity (Algorithm 8 ($\text{repProd}_{\text{DM 1}}$: double));
  $\text{expProfit}_{\text{DM 2}} \leftarrow \text{profit}_{\text{DM 2}}$ with *reported* productivity (Algorithm 8 ($\text{repProd}_{\text{DM 2}}$: double));
end
$\text{payment}_{\text{DM 1}} \leftarrow \text{discount} \cdot (\text{profit}_{\text{DM 1}} + \text{expProfit}_{\text{DM 2}})$;
$\text{payment}_{\text{DM 2}} \leftarrow \text{discount} \cdot (\text{profit}_{\text{DM 2}} + \text{expProfit}_{\text{DM 1}})$;
return payment;

Algorithm 10: Division manager: Receive and process bonus in LM_1 - LM_3 (P6)

Input: Payment by compensation scheme (compensation: double)
Output: Updated strength values in knowledge base (strengths: double[])
choose $\text{rule}_{\text{sel.}}$ that determined the last decision;
strength_{t+1} ($\text{rule}_{\text{sel.}}$) $\leftarrow$ (1 - λ) $\cdot$ strength_{t} ($\text{rule}_{\text{sel.}}$) + $\lambda \cdot \text{compensation}_{t}$;
update $\leftarrow$ strength_{t+1} ($\text{rule}_{\text{sel.}}$) - stregrimm2006ngth$_{t}$ ($\text{rule}_{\text{sel.}}$) ;
updateShare $\leftarrow$ 1 / (number of rules in knowledge base - 1);
while *rule subset (without* $\text{rule}_{sel.}$ *) has more elements* **do**
  choose rule_{i};
  strength_{t+1} (rule_{i}) $\leftarrow$ strength_{t} (rule_{i}) - updateShare ;
end

2 LAMDA Simulation Framework

For the LAMDA simulation model, the programming language Java was chosen. For the implementation of ABS models, the programming language NetLogo [13] and the library Repast [2] are possible alternatives, and commonly used in the social simulation community. Netlogo and Repast support the implementation of ABS in particular by libraries for steering the simulation experiments and creating graphical outputs. In the LAMDA model, however, spatial issues do not play a role. Furthermore, a workbench providing a certain level of flexibility for future use was aimed. Therefore, the object-oriented language Java was chosen for the given simulation model. Also, the objective-oriented character of Java with the concepts *class* and *object* matches well with the concepts *agent type* and *(individual) agents* in ABS.

This section introduces to the Java class structure as implemented in the program code. Therefore, UML diagrams and figures provide an overview of the structure and main methods.

An overview of the package structure of the LAMDA simulation framework is given in Figure 3. The package *gui* contains the interface classes for setting the simulation experiment parameter. Here, the user may specify the experimental setup. The main class of the simulation model as well as a central management entity are part of the *main* package. A management entity instantiates one *observer* object per experimental design point (= parameter combination). The classes in the observer package schedule one simulation experiment and manage the simulation output for this. Therefore, the observer class instantiate the *agents* as well as the *mechanism*

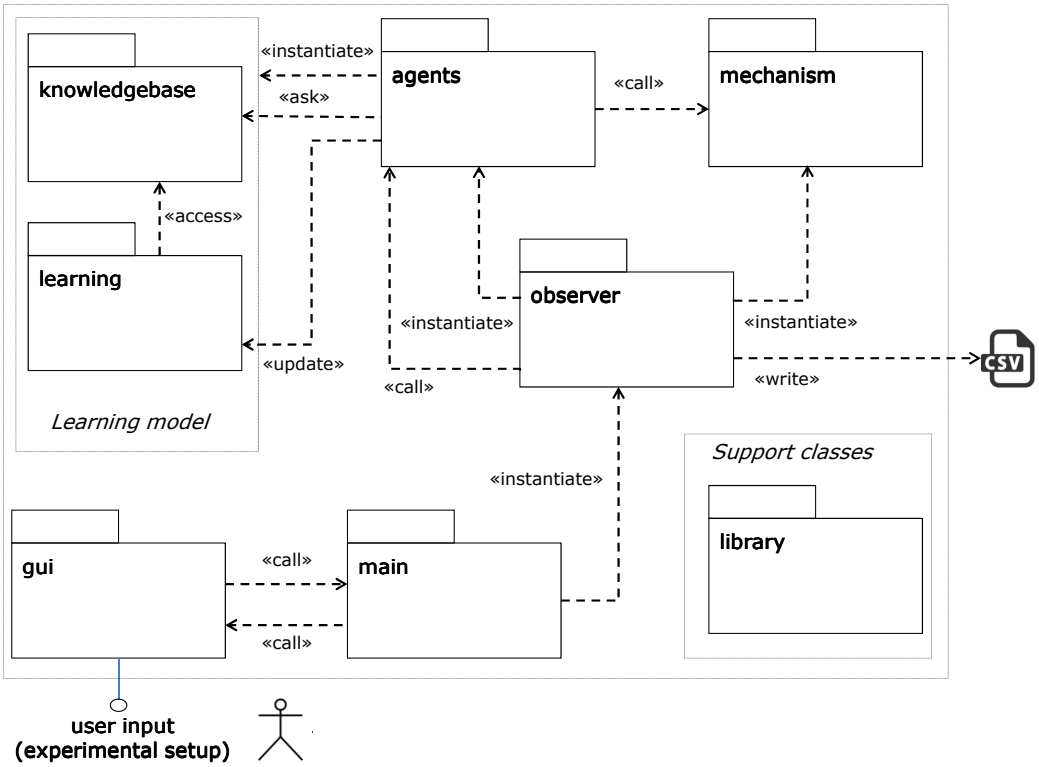

Figure 3: Package structure of the LAMDA simulation workbench

with the parameter values as specified for the respective simulation experiment. The package *agents* contain the agent classes. The agents instantiate their *knowledgebase* and *learning algorithm* as given from the observer class and form their learning models. Package *library*, finally, is a collection of support classes for some standard operations in the simulation model.

In the following, the classes behind the package structure are described in more detail. Please note that the following sections describe no extensive documentation of the program code, but an explanation of the most important setup elements.

2.1 Simulation Experiment Environment

For running a simulation experiment, the user may use a GUI. After the simulation started, the simulation parameters can be set in window *"LAMDA - Design of Experiments"* (see Figure 4). Here, four categories of factors are listed: (1) learning models, (2) scenario parameters, (3) opponent parameters, and (4) the experimental parameters.

First, the user of the simulation model may choose one or several learning models (ZI, LM_1, LM_2, LM_{2+}, or LM_3. If several learning models are chosen, the program runs one simulation experiment for each model. This means that the division manager agents of the model are equipped with *the same* model within one simulation experiment. For equipping the agents with different models, an adaptation of this version would be necessary. This could be achieved by creating an observer class for mixed learning models (for more details see below).

Second, the scenario parameters can be set by the user. Those are the state variables and

scales for the attributes of the given coordination scenario (see Section 1.1.2).[4]

For the division manager agents, the scale and increment for the reported productivity values can be either set differently from, or according to the real productivity values (see the input fields *"Productivity scale for reported values"* and *"Increment for reported values"*). In the default setting, the value range of the reported productivity is set and varied with the scale for the real productivity (indicated by *"<varied with prodScale>"* and *"<varied with increment for real values>"*). If the user chooses values instead of this default setting, design points are added to the simulation experiment with variations for the reported productivity value independent from the real. Again, the indicated scales are the same for both division manager agents.

However, the simulation toolkit allows a variation of some factors between the division managers in area (3), which are the opponent parameters. In the default setting, the scales and increments for the real and reported productivity values are set for the agents in area (2). If the user sets some values in the fields of area (3) instead (with the same syntax as described before), the real and reported productivity values for both agents are varied independently from each other, as indicated in the fields.

Finally, the experimental parameters are set in area (4). Here, the number of ticks per simulation run, as well as the number of runs per parameter combination can be specified. As in the other input fields, several factor levels can be defined by writing them in the field, so that the simulation toolkit allows experiments with varying ticks and runs, for analyzing their influence on the output.

The definition of the factor levels, and thus the input for this GUI, is part of the *design of experiments*. Here, the parameter combinations for the simulation runs as well as the number of runs and ticks are specified in a systematic way.

A UML diagram of the main components behind the GUI is given in Figure 5. In the main class *LamdaMain* an object of the *Window* class is instantiated. By *java.awt.event.ActionListener*, the program waits for the input of the user. Based on the indicated parameter combinations, the class *LamdaScheduler* organizes the simulation experiment. Therefore, this class prepares the simulation output (*FileWriter outputCSV*) as well as the observer objects.

LamdaScheduler initializes one observer object for each parameter combination. The observer objects, organize the simulation run schedule and simulation output. Also, the number of runs and ticks per simulation run as indicated by the experimental parameters are organized here.

As the learning model has the biggest impact on the simulation objects and schedule, one observer class for each learning model is defined as sub-classes to *Observer.java*. The sub-classes are accordingly *ObserverZI, ObserverLM1, ObserverLM2, ObserverLM2plus,* and *ObserverLM3* (see UML diagram in Figure 6). Here, the learning models are initialized (method *initializeLA()*), as well as the agents (*createHeadquarter()* and *createDivManagers()*). The method *runSimulationExperiment()* organizes the schedule of runs and ticks, while *round()* manages the process of one simulation tick. The method *realProductivity()* returns a random productivity value, which is evoked once per round for each division manager agent. By the method *resetRun()*, the objects and parameters of the simulation experiment are reset. Finally, the methods *observeReportingBehavior()* and *printOutput()* manage the simulation output.

Figure 7 illustrates the agent classes. The *Headquarter* agent knows about the scenario variables, which are the number of resources available to the firm (*resources*), the productivity reduction for resources above the optimal capacity (*reductProd*), and the optimal and maximum capacities of the divisions (*optCapacity, maxCapacity*). Based on the reported productivity values by the divisions (*repProd*), the *Headquarter* allocates the resources (*allocRes*). The divi-

[4]The factor levels for these parameters can be written in the field with the system decimal separator and a blank (space) as separator.

sions use the resources, which results in profits being observable by the *Headquarter* (*profitDiv*). Furthermore, the *Headquarter* has an object of the Groves mechanism (*gm*) and its parameter *discount*, to calculate the *compensation* payments to the divisions.

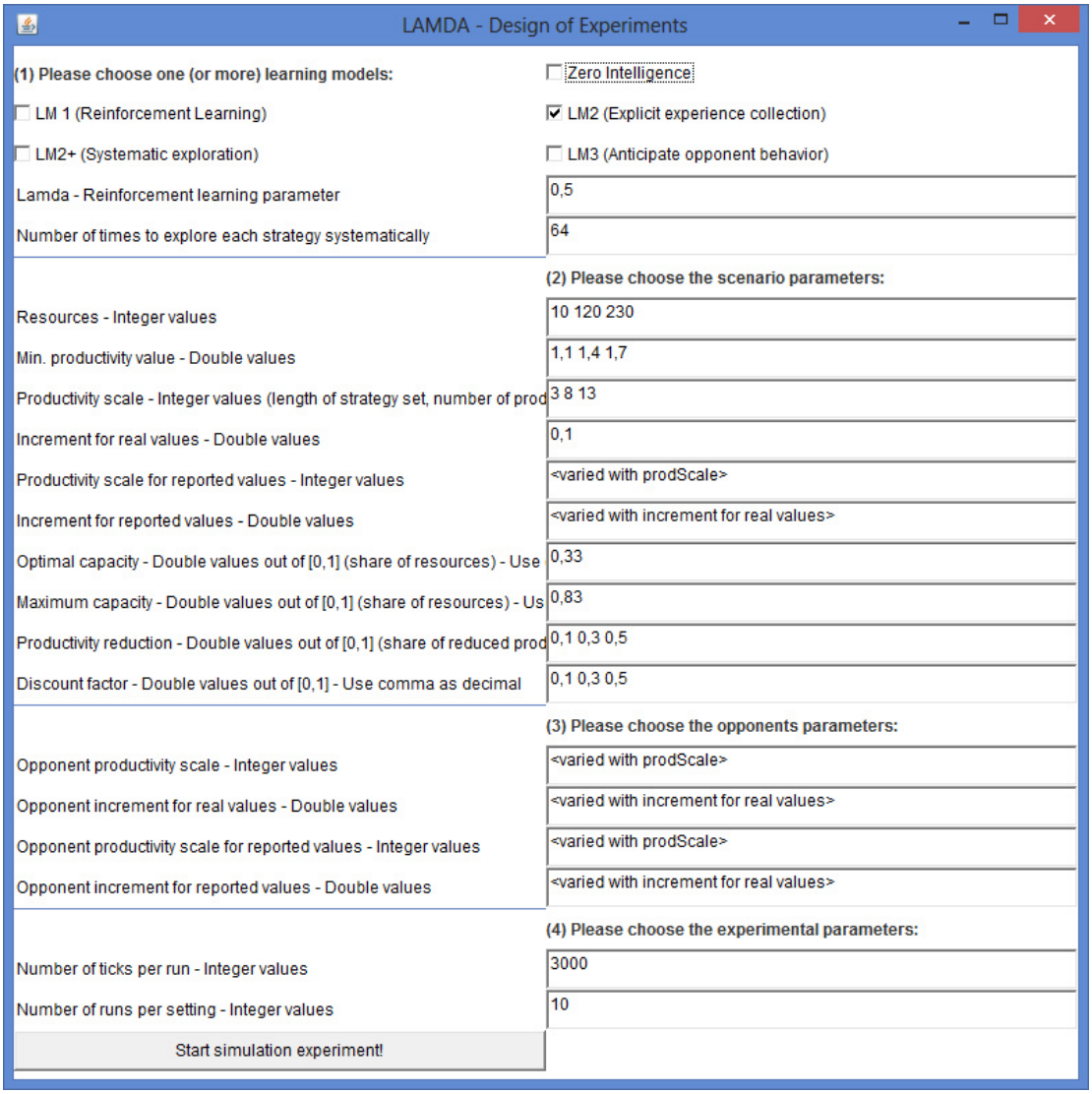

Figure 4: Graphical user interface (GUI) of the LAMDA simulation workbench

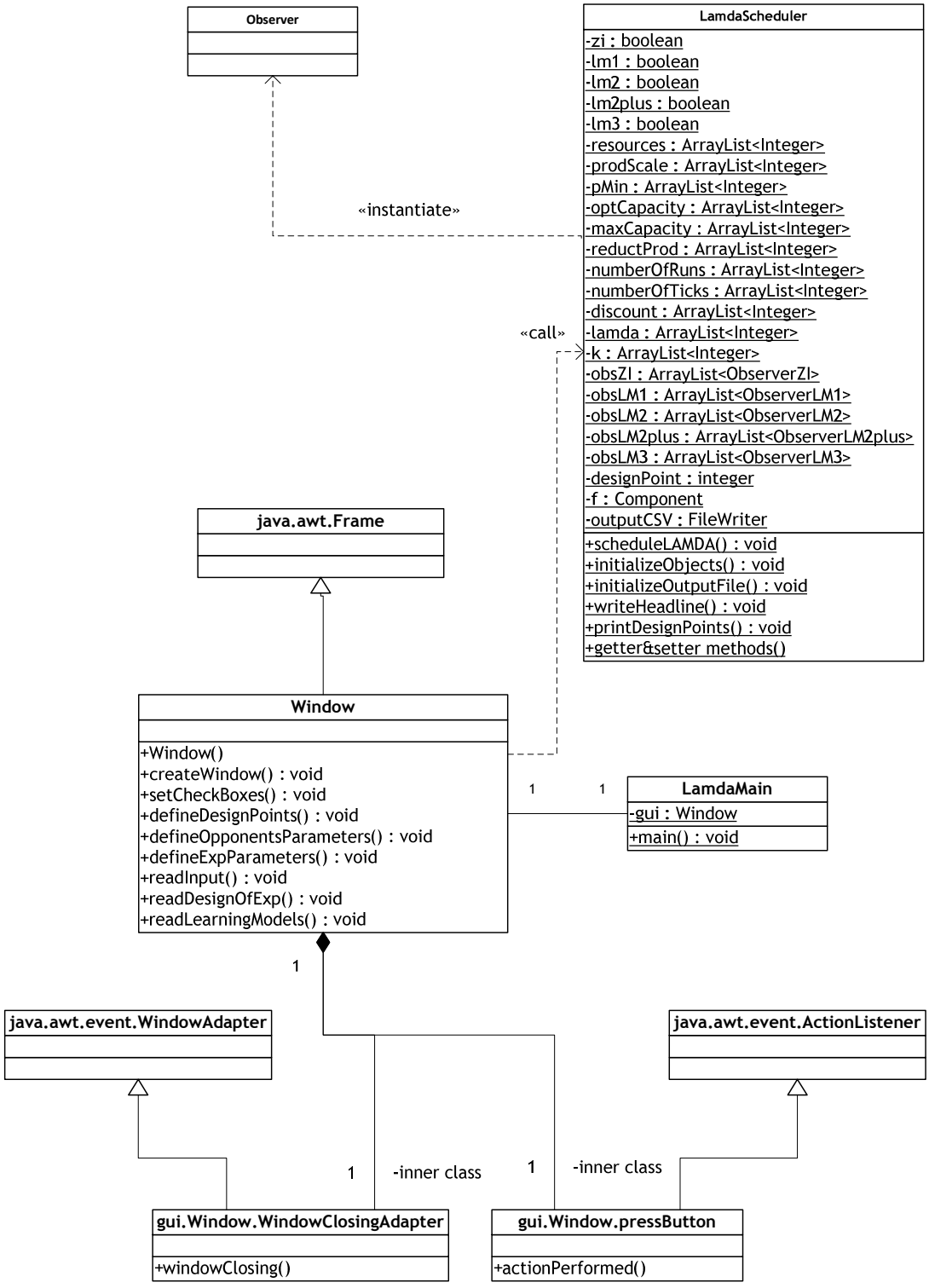

Figure 5: UML diagram for the packages *lamda.gui* and *lamda.main*.

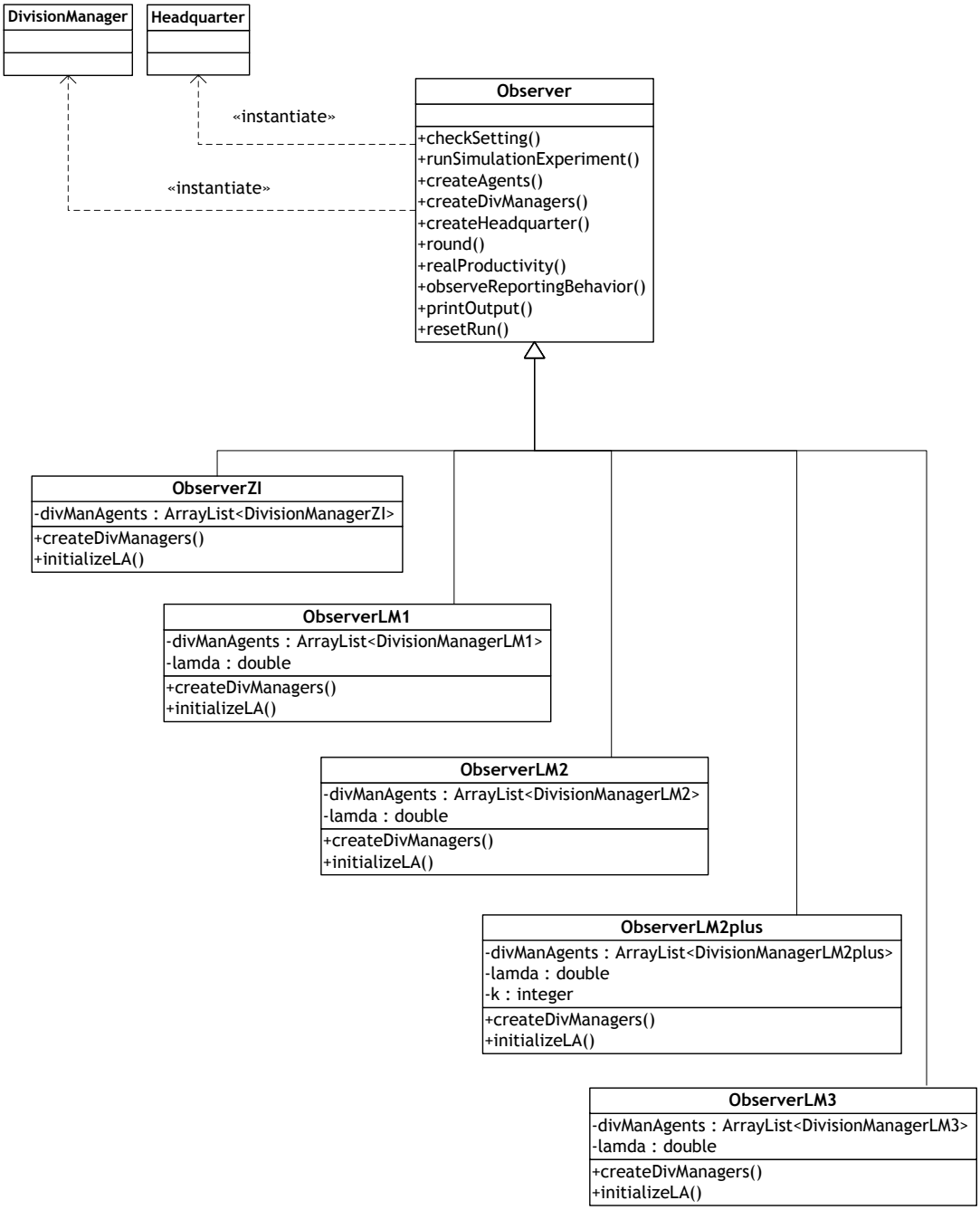

Figure 6: UML diagram for the *Observer* package.

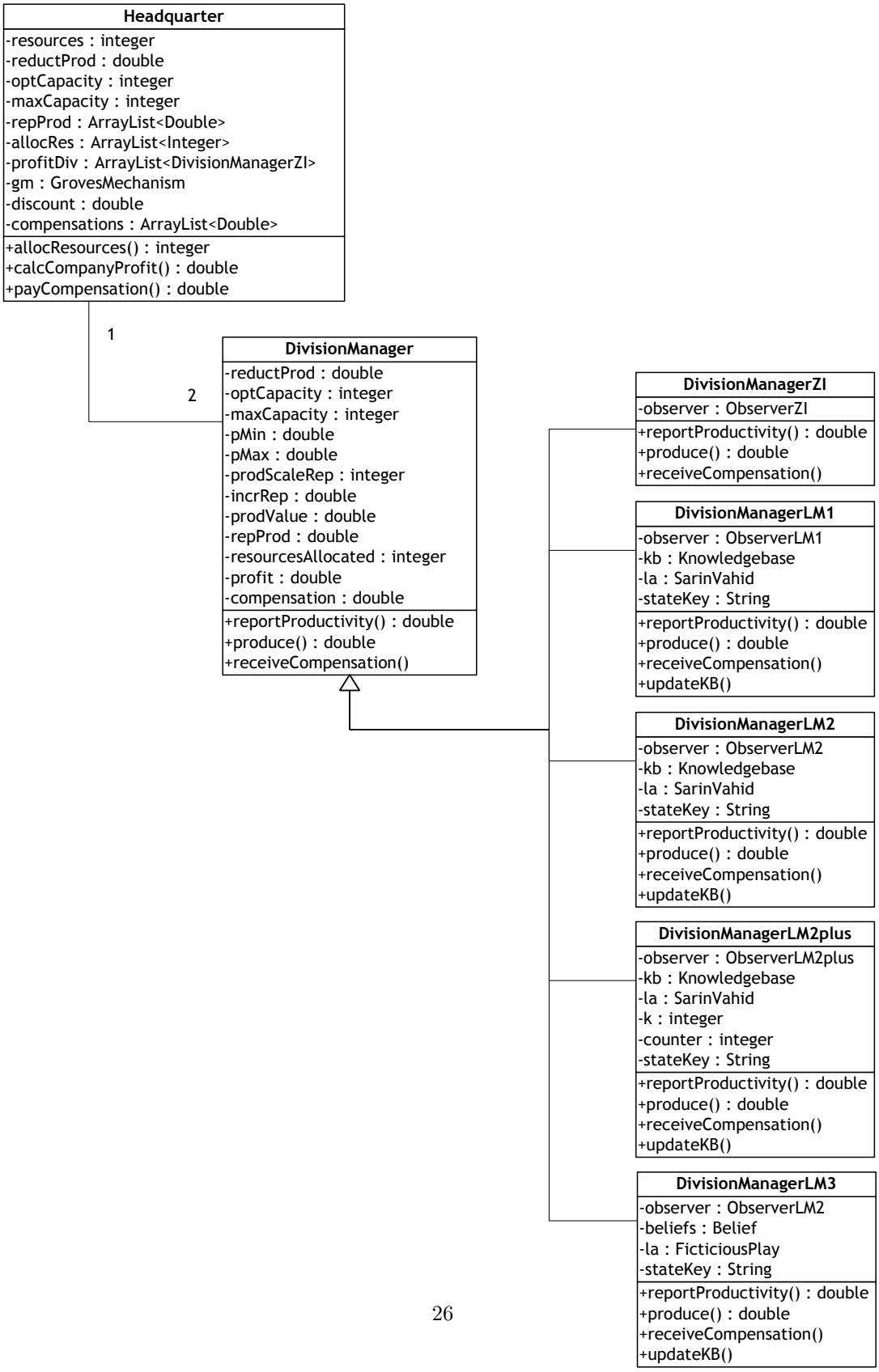

Figure 7: UML diagram for the *Agent* package.

2.2 Agent Architecture and Learning Model

According to the setup of the LAMDA learning models as described in the model section, the agent architecture is implemented in the two packages *knowledgebase* and *learning*. The *knowledgebase* describes the structure of the cognitive model of the agent, while the learning classes describe how the knowledge is adapted over time.

This section describes the knowledgebase first. Next, an overview of the learning package is given. Along the description, this section will consider what steps need to be taken for other models using this test bed.

The implementation of the LAMDA agent architecture is designed according to the general procedure of perception, *decision*, and *action*, where the perception indicates the given *state* of the agent. This model enables the agent to behave autonomously based on individual experiences. The structure of the cognitive model consists of the main components *Knowledgebase*, *State*, and *Strategy* (see Figure 8). One *Knowledgebase* contains *m State* objects, whereas one *State* has *n Strategy* objects. By the knowledge base, the agent builds a *symbolic model* of the environment. This goes along with the agent architecture of symbolic-reasoning agents in AI [14].

Figure 9 describes the control flow of the agents' decision process in the course of the simulation. The structure of the cognitive model is described along this process in the following.

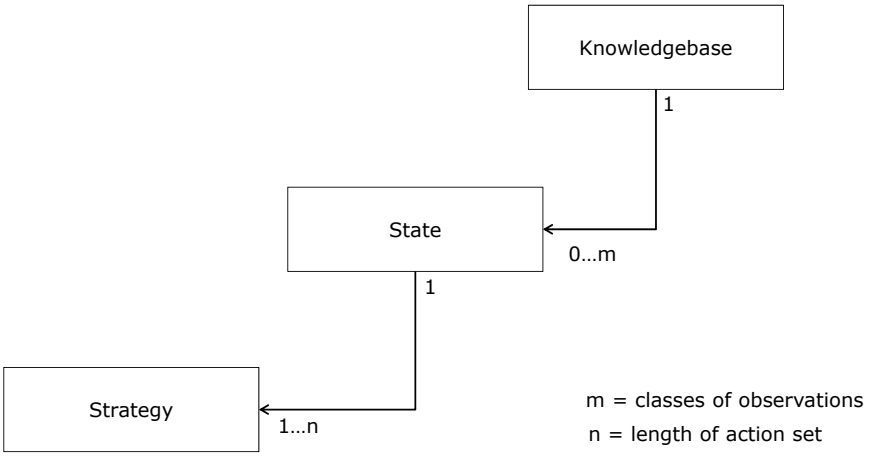

Figure 8: The simplified structure of the cognitive model implemented for LAMDA.

The central element is the *state key*, specified by the observation of the agent (1.). The observed information is transformed into a string of elements (2.). The concatenation of the information elements (separated with a comma) forms the state key, which specifies the state in which the agent is in. The state key may contain wildcards (symbol "#").

The requirement for the application of this model is that the information is given to the agent in a fixed order, so that a position within the state key always refers to the same content. The information elements can be a text or a number. In the given model, the observation contains only numbers, describing true or reported productivity values. Thus, the transformation into a symbolic representation can be achieved by using the value combinations as state description. However, a transduction problem may arise in models, where more complex symbolic representation is needed, such as complex interdependencies between observations.

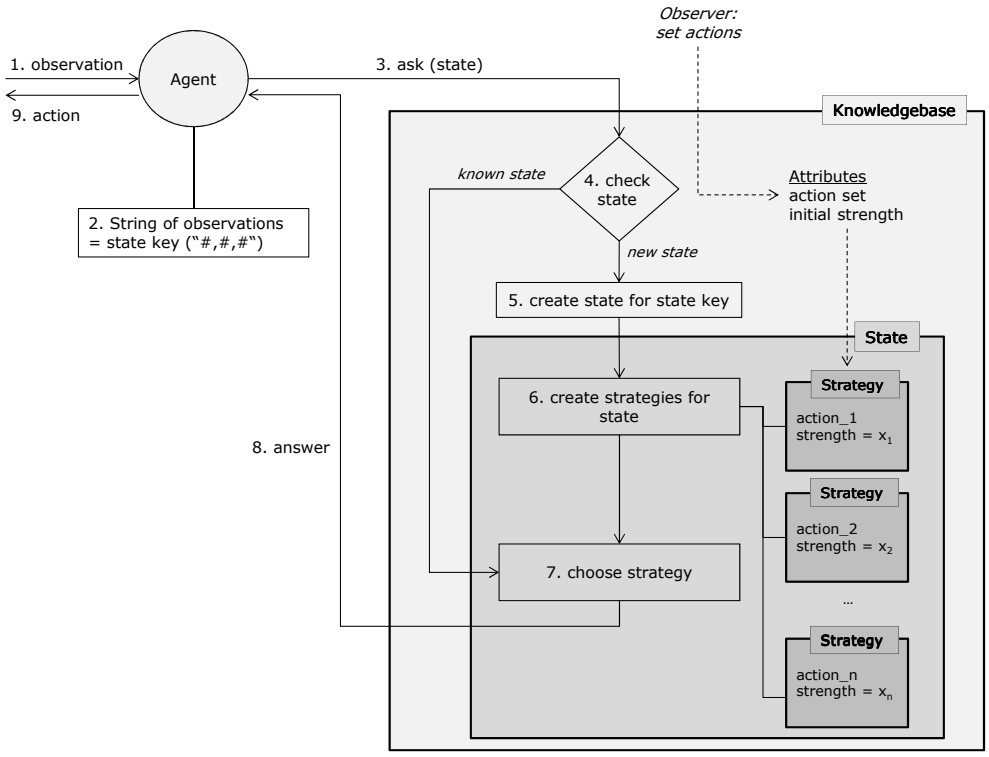

Figure 9: The application of the cognitive model in the decision making process.

The agent forwards the state key to the *Knowledgebase* object (3.). If the agent observes a new combination of information, the *Knowledgebase* creates a new *State* object (5.). Coincident with the *State* object, a set of *Strategy* objects are created (6.); for each action a_i one *Strategy* object per *State* object. The *Strategy* object contains the action as well as a strength value, indicating the value of this strategy for the given state (the initial strength value is indicated by the learning algorithm).

After creating the *State* and *Strategy* objects, the agent may choose one strategy (the selection strategy is defined by the learning algorithm) (7.). If the agent already knows the situation, a corresponding *State* object is part of the *Knowledgebase*, and a strategy is chosen immediately. The reasoning process is determined by choosing the *best experienced strategy* in the given state. The chosen *Strategy* object determines the agents next action, which is forwarded as answer from the *Knowledgebase* to the agent entity (8.). Finally, the agent performs its action in the environment (9.).

The LAMDA agent architecture has no internal mental conditions. Possible extensions may consider different internal preferences, such as setting a focus on specific environmental circumstances (i.e. certain positions of the symbolic representation). Here, the agent only reacts to environmental conditions with a unified relevance for all observations. Therewith, the *reactive* cognitive ability is in the focus of the model.

By this setup, the cognitive model provides a general model, applicable to many learning models. In particular the close relation between observation and the dynamically created *State*

objects makes the system more flexible for varying scenarios.[5].

Next to the knowledge representation, the *learning algorithm* defines the learning dynamics. The central element is the update process of the knowledge base. By the applied reinforcement learning strategies, the strength values of the strategies are updated based on the experienced value of action. How the feedback is translated into the updated strength values depends on the respective algorithm. For applying the general LAMDA model to other reinforcement learning algorithms than the applied, one has to create a subclass to *Learning.java* and implement the method *updateKB* accordingly. In the constructor, the action set as well as the initial strength values are initialized.

2.3 Mechanisms

The next central element of the LAMDA model is the mechanism implementation. The mechanism calculates the compensation (=payoff) to the agent. The package mechanism comprises the superclass *Mechanism.java* as well as the subclass *GrovesMechanism.java*. The classes contain the only method *calcCompensation*, which is a calculator of the compensation scheme, based on the parameters *achieved profit*, *expected profit opponent*, and *discount* (as defined by the Groves mechanism).

The LAMDA model shows the potential of using ABS for analyzing the effect of mechanisms under varying behavioral models. The Groves mechanism was chosen, amongst other reasons because of its theoretically truth-inducing effect. Nevertheless, the consideration of alternative mechanisms is possible with this framework.

References

[1] Arnold, M., Ponick, E., and Schenk-Mathes, H. Groves mechanism vs. profit sharing for corporate budgeting - An experimental analysis with preplay communication. *The European Accounting Review 17*, 1 (2008), 37–63.

[2] Collier, N. Repast: An extensible framework for agent simulation. *The University of Chicago's Social Science Research 36* (2003).

[3] Grimm, V., Berger, U., Bastiansen, F. Eliassen, S., Ginot, V., Giske, J., Goss-Custard, J., Grand, T., Heinz, S., Huse, G., Huth, A., Jepsen, J., Jorgensen, C., Mooij, W., Müller, B., Pe'er, G., Piou, C., Railsback, S., Robbins, A., Robbins, M., Rossmanith, E., Rüger, N., Strand, E., Souissi, S., Stillman, R., Vabo, R., Vissar, U., and DeAngelis, D. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling 198*, 1-2 (2006), 115–126.

[4] Grimm, V., Berger, U., DeAngelis, D., Polhill, J., Giske, J., and Railsback, S. The ODD protocol: A review and first update. *Ecological Modelling 221*, 23 (2010), 2760–2768.

[5] Groves, T. Incentives in teams. *Econometrica 41*, 4 (1973), 617–631.

[5] A similar approach is applied in the rule engine as implemented in the *Declarative Rulebased Agent Modelling System (DRAMS)* by Lotzmann [8]. Here, a *fact base* stores information about the current state of the world, and an inference engine controls the rule selection process. However, the approach applied here is a simplified version, which may be applied for less complex rules than in DRAMS, but underlying less restrictions at the same time.

[6] Groves, T., and Loeb, M. Incentives in a divisionalized firm. *Management Science 25*, 3 (1979), 221–230.

[7] Gupta, N., Shekhar, R., and Kalra, P. Congestion management based roulette wheel simulation for optimal capacity selection: Probabilistic transmission expansion planning. *International Journal of Electrical Power & Energy Systems 43*, 1 (2012), 1259–1266.

[8] Lotzmann, U., and Meyer, R. DRAMS - a declarative rule-based agent modelling system. In *Proceedings of 25th European Conference on Modelling and Simulation* (2011), pp. 78–83.

[9] Lotzmann, U., Möhring, M., and Troitzsch, K. Simulating normative agents. *International Journal of Agent Technologies and Systems (IJATS) 2*, 1 (2010), 31–49.

[10] Sarin, R., and Vahid, F. Payoff assessments without probabilities: A simple dynamic model of choice. *Games and Economic Behavior 28*, 2 (1999), 294–309.

[11] Sharma, D., Singh, V., and Sharma, C. GA based scheduling of FMS using roulette wheel selection process. In *Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011*, K. Deep, A. Nagar, M. Pant, and J. C. Bansal, Eds., vol. 131 of *Advances in Intelligent and Soft Computing*. Springer India, New Delhi, India, 2012, pp. 931–940.

[12] Sklar, E. Netlogo, a multi-agent simulation environment. *Artificial life 13*, 3 (2007), 303–311.

[13] Tisue, S., and Wilensky, U. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems* (2004), vol. 21, Boston, MA, pp. 16–21.

[14] Wooldridge, M. *An introduction to multiagent systems*, 2 ed. John Wiley & Sons, Inc., Chichester, United Kingdom, 2008.