Reconstruction of an Agent-Based Simulation Model about Labor Market Policies

Master Thesis

Xi Niu

Reconstruction of an Agent-Based Simulation Model about Labor Market Policies: Master Thesis

Xi Niu

Abstract

In this master thesis an agent-based macroeconomic model used for economic policy experiments featuring a distinct geographical dimension and heterogeneous workers with respect to skill types is to be introduced and reconstructed with the help of the AOR simulation technology that was developed by the Chair of Internet Technology.

Zusammenfassung

In dieser Masterarbeit wird ein agenten-basiertes makroökonomisches Modell für die wirtschaftspolitischen Experimente mit einem eigenen geographischen Dimension und heterogenen Arbeitnehmern in Bezug auf Skill-Typen vorgestellt und mit Hilfe der am Lehrstuhl Internet-Technologie entwickelten AOR-Simulationstechnologie rekonstruiert.

Acknowledgements

I would like to express my sincere thanks to Prof. Dr. Gerd Wagner for his invaluable guidance and advice throughout the course of this thesis. Without his help this paper would not be possible.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

So far agent-based modeling (ABM) is a powerful simulation modeling technique that has been extensively developed and well used in a lot of areas, because it can provide many effective methods to facilitate the research into the complex problems of different scientific fields. In ABM, a system is modeled as a collection of independent acting units called agents. Each agent is able to perceive things and make decisions based on a set of rules. The model which will be described in this paper used ABM to explore the fields of economics. It focuses on an economy to analyse how the effects of different spatial distributions of economic policy measures depend on spatial frictions in the labor market expressed as commuting costs of workers who are employed outside their home-region. The purpose of this paper is to remodel it as a multi-agent based simulation using the Agent-Object-Relationship (AOR) simulation technology that was developed by Prof. Dr. Gerd Wagner and other team members at the Brandenburg University of Technology.

This thesis is organized as follows. Chapter 2 gives a general overview of the economic model. Chapter 3 offers a detailed analysis of the model. Chapter 4 presents the AOR simulation and the reconstructed model. Chapter 5 provides some simulation results, and Chapter 6 concludes.

Chapter 2. The model

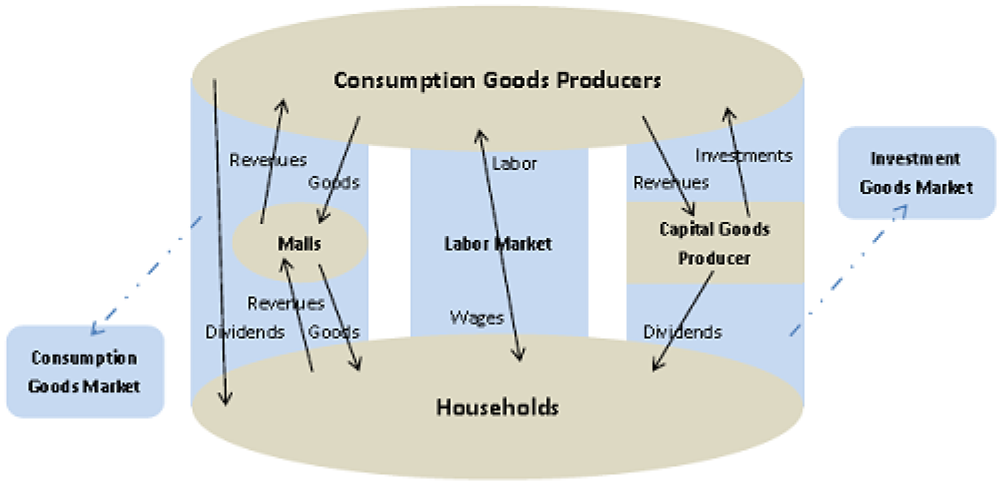
The agent-based macroeconomic model was developed by Herbert Dawid, Simon Gemkow, Philipp Harting and Michael Neugart and has been implemented in the Flexible Large-Scale Agent Modeling Environment (FLAME) developed by Simon Coakley, Mike Holcombe and others at the University of Sheffield (see http://www.flame.ac.uk for more information and references). This part gives an overview of the model.

2.1. Background to the model

The model was developed as part of a larger simulation platform for European policymaking known as EURACE. EURACE is a major project aiming at creating a complete agent-based model of the European economy for evaluating European economic policies. The three-years EURACE project started in September 2006. It involves economists and computer scientists from eight research centres in Germany, France, Italy, the UK, and Turkey, as well as the 2001 Nobel laureate in economics, Joseph Stiglitz. The EURACE model has a distinct spatial structure simulating the regional statistical units used by Eurostat. It contains various artificial markets for real commodities (that is, consumption goods, investment goods and labor) and markets for financial assets (such as loans, bonds and stocks). For a general overview of the EURACE model, see http://www.eurace.org.

The model which will be described in detail later is a simplified version based on EURACE's labor market module. Its structure can be seen as follows.

Figure 2.1. Model structure



The main purpose of the model is to investigate how the spatial skill distribution in the absence of policy intervention influences the speed of technological change, the flow of labor force and the growth of wage level in an economy. Therefore, policy implications aiming at the change of local skill distribution play an important role in the model. In order to capture the effects of different spatial distributions of policy measures, the economy is divided into two regions, in each of which a number of households live. There are five general skill levels 1 to 5, where 1 is the lowest skill level and 5 is the level with the highest skill. The general skill levels are distributed uniformly among households. According to general skill levels of households a region can be declared as one of three possible types: low skill region, medium skill region or high skill region. More specifically, in a low skill region the skill distribution is such that 80% of households have the lowest general skill level, whereas the remaining households are equally distributed across the other four levels of general skills. Analogously, a region is a medium skill or high skill region if 80% of households have general skill level 3 respectively 5.

Table 2.1. General skill distributions in the three different types of regions

Region type	Level 1	Level 2	Level 3	Level 4	Level 5
Low skill	80%	5%	5%	5%	5%
Medium skill	5%	5%	80%	5%	5%
High skill	5%	5%	5%	5%	80%

In this model, both regions are initially set to low skill regions. A policy maker intends to improve the regional skill distributions. To that end, there are two options here. Either both regions are upgraded to medium skill or all efforts are concentrated in one region thereby moving this region to high skill whereas the skill distribution in another region stays unchanged. Finally, the effects of different policy types can be compared.

- Option 1: Efforts are spread over both regions.
  - Low/Low → Medium/Medium
  - 80% skill group 3, 5% for each skill group 1, 2, 4, 5
- Option 2: Efforts are focused in one region.
  - Low/Low → Low/High
  - Region 1: 80% skill level 1, 5% skill group 2, 3, 4, 5
  - Region 2: 80% skill level 5, 5% skill group 1, 2, 3, 4

2.2. General description

The model describes an economy that contains an investment (or capital) goods, a consumption goods and a labor market. There exists a single type of product in each market, i.e., investment goods are supplied in the investment goods market, consumption goods are sold at malls in the consumption goods market and labor is considered as a commodity in the labor market. The economy is populated with numerous instances of different types of agents, which are summarized as two types of active agents and two types of passive agents in the sense that active agents can take decisions, whereas passive ones can not. Each type of agent can have different roles corresponding to its activities in the markets. The following summarizes these roles.

Table 2.2. Agent types and their market roles

Agent	Type	Market	Role
Household	Active	Comsumption Goods Market	Buyer
		Labor Market	Worker
Consumption Goods Producer (henceforth called CGP)	Active	Investment Goods Market	Buyer
		Consumption Goods Market	Seller
		Labor Market	Employer
Mall	Passive	Consumption Goods Market	Information transfer between consumption goods producers and households
Capital Goods Producer (henceforth called IGP)	Passive	Investment Goods Market	Seller

2.2.1. Markets and agents

Agents acting within markets are distributed across regions where the consumption goods market is local, the other markets are global.

The main actors in the investment goods market are IGP and CGPs. The investment goods market is global meaning that CGPs in both regions buy investment goods from the unique IGP. Investment goods are offered with infinite supply at an exogenously given price and there exists only one type of investment goods. The quality and price of supplied investment goods increase randomly over time. The amounts paid for investment goods are channeled back into the economy.

CGPs, households and malls take part in the consumption goods market. Together with labor, investment goods are used by CGPs to produce consumption goods. These goods are sold at malls to households. Malls are seen as the non-profit local market platforms. On the consumption goods market CGPs act globally in the sense that all CGPs store and offer their products at every regional mall, but households act locally because every household comes to the mall in his region to buy goods at posted prices. CGPs receive revenues from the sales. Their income is used to remunerate households in order to close the model.

CGPs and households play in the labor market. A search-and-matching process is used to represent the interaction between CGPs and households in this market. The CGP needs more laborers in order to expand its production scale. For this reason, it offers job vacancies based on the planned output. The household who is job seeker looks for a suitable position based on the corresponding salaries of these vacancies. Job seekers can apply for jobs in any region, but one thing must be pointed out, that is, working outside of their own region is associated with commuting costs which have to be subtracted form the wage. Thus, the labor market is global with spatial frictions determined by commuting costs.

2.2.2. Timing

In the model, the basic unit of time is a day and the activities of agents are calendar-driven and event-based. Some decisions, like consumption decisions of households, are taken weekly (suppose each week has 5 days) and others are taken monthly (suppose each month has 20 days).

Chapter 3. Analysis of the model

This chapter focuses on the analysis of the model. All the features will be listed. They are quite necessary for reconstruction of the model in the AOR simulation language. Moreover, in order to better understand the model, UML class diagram and BPMN model are applied. They are powerful tools for representing complex structures and relationships. The aim is to operationalize the abstract data of the model into easy readable graphical notations. This is a very important step, because it can improve efficiency of model reconstruction.

3.1. Data extraction

In general, there are three ingredients in this model: market, agent and activity. Markets do not act, because they have no intentions and cannot perform actions. However, they can provide some contexts for agents to act in. Agents always act within markets. They take some activities with different roles. Therefore, the analysis is concentrated on agents, which are involved in different markets and characterized by different actions. As mentioned, there are four types of agents: household, CGP, mall and IGP. Agents and their activities are discussed in depth as follows.

3.1.1. Household

The household is simultaneously taking the roles of buyers and workers. In the consumption goods market, households determine the monthly and weekly consumption budgets, and then visit a mall in order to decide and purchase the provided goods for the weekly consumption. In the labor market, if a household is employed, he receives monthly wage from his current employer. When a household is unemployed, he receives unemployment benefit from the government and sends job applications to CGPs that have vacancies. In general, the household makes some decisions with the related roles affecting the markets as follows.

1. Allocate budget on consumption and saving
2. Choice of consumption goods
3. Search for a job
4. Acquire specific skills

3.1.1.1. Allocate budget on consumption and saving

The household acting as the role of buyer (or consumer) sets once a month the consumption budget which is spent on the consumption goods market and consequently determines the remaining part which is saved.

Table 3.1. The savings decision

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
personal consumption budget	B^{cons}	consumptionBudget	The consumer decides about the budget that he will spend for consumption	-
the available liquidity	Liq^{Avail}	cashOnHand	The cash on hand that contains current income (i.e. labor income and dividends distributed by capital and consumption goods producers) and	-

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
			assets carried over from the previous period	
mean income	Inc^{Mean}	meanIncome	The mean individual (labor) income of a consumer over the last periods	-
the percentage of mean income	Φ	phi	$\Phi \leq 1$ is the percentage of the mean income such that the consumer spends all cash on hand below that level	*0.9*
marginal saving propensity	κ	savingPropensity	$0 < \kappa < 1$ is the saving propensity	*0.1*

Algorithm: There exists a critical value $\Phi * Inc^{Mean}$ of cash on hand to determine how much cash on hand will be spent for consumption in this month. When the available liquidity Liq^{Avail} is below this critical value the whole cash on hand will be spent. Thus, the consumption budget $B^{cons} = Liq^{Avail}$. In the opposite case the consumer will save a part of his cash on hand, so he sets his consumption budget according to the following consumption rule

$$B^{cons} = Liq^{Avail} - k * \left(Liq^{Avail} - \Phi * Inc^{Mean}\right) \tag{3.1}$$

3.1.1.2. Choice of consumption goods

The consumer purchases consumption goods according to his consumption budget. He splits the consumption budget into four equal shares, each of which is used for shopping per week. After determining the weekly budget, each consumer visits once a week to the mall in his region to buy goods. When visiting the mall he collects information about prices and quantities of different goods and then purchases goods according to his preference and available stocks of goods at posted prices. The model includes neither any kind of horizontal product differentiation, nor any kind of quality differentiation. Therefore, choice probabilities depend solely on prices.

Consumers make their purchasing decisions based on the prices of different goods using a stochastic rule as described in a standard logit model. In the marketing literature it is standard to describe individual consumption decisions. This model represents the stochastic influence of factors not explicitly modeled on consumption decisions, see [Guadagni and Little 1983].

Table 3.2. Selection of consumption goods

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the selection probability	$Prob$	selectionProbability	The consumer decides which consumption good to buy on the basis of the selection probability of every consumption good sampled by him	-
available stocks of goods	G_{week}	availableProducts	A list of available products at the attended mall will be created in week (of period)	-
the price of the consumption good	p_i	productSalesPrice	The price of the consumption good i	-
the value of the consumption good	$v(p_i)$	consumptionValue	A function whose parameter is p_i determines the subjective value	$-ln(p_i)$

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
			of the consumption good i for consumer	
the intensity of choice by consumer	λ^{cons}	intensityOfProductChoice	The intensity of choice by consumer	*8.5*

Algorithm: The decision of a consumer which consumption good to buy is random, where purchasing probabilities are based on the values he attaches to the different choices he is aware of. The consumer selects one consumption good i ∊ G_{week}, where the selection probability reads

$$Prob = \frac{Exp\left[\lambda^{cons} * v\left(p_i\right)\right]}{\sum_{i \in G_{week}} Exp\left[\lambda^{cons} * v\left(p_i\right)\right]} \tag{3.2}$$

Once the consumer has selected a consumption good he tries to spend the whole weekly budget for that consumption good if the stock at the mall is sufficiently large. In case the consumer cannot spend all his budget on the consumption good selected first, he has a single opportunity to select another good. If the budget is then not completely spent, the remaining amount is rolled over to the following week.

3.1.1.3. Search for a job

On the labor market households who are job seekers search for jobs (there are the unemployed plus a certain fraction of on-the-job searchers). They see posted vacancies and apply to the ones if the wage offers exceed the current reservation wage of the job seeker. After applying they receive zero, one or more job offers and rank these offers with respect to the wage offer. In case the offered position is outside the home region of the job seeker, commuting costs are subtracted from the offered wage. If two or more wage offers are equal then these are ordered randomly. Job seekers accept at most one job with the highest offered wage and update their reservation wage which is the new wage. If job seekers are still unemployed they decrease their reservation wage.

3.1.1.4. Acquire specific skills

The household is characterized by a general skill level and specific skills. His general skill level is determined by outside factors like government and economic policy. There exist five general skill levels, described by different values ranging from 1 to 5, i.e. {1, 2, 3, 4, 5}. 1 is the lowest general skill level and 5 is the highest. The specific skills of workers are acquired on the job. The acquisition of specific skills is faster for higher general skill levels.

Table 3.3. Specific skills (of workers) decision

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the average quality of the capital stock	A_i	averageQualityOfCapitalStock	The average quality of the investment goods employed by CGP i	-
general skill level	b^{gen}	generalSkillLevel	Every worker has a level of general skills $b^{gen} \in \{1, \ldots, b^{gen}_{\max}\}$	*[1…5]*
specific skill level	b_t	specificSkillLevel	Every worker has a level of specific skills in period t	-
increasing in the general skill level of the worker	$\chi(b^{gen})$	chi	A function whose parameter is b^{gen} governs the speed of specific skill improvement	-

Algorithm: While being employed each worker adjusts his specific skills to the average quality of the capital stock of his employer. The adjustment speed $\chi(b^{gen})$ depends positively on the general skill level of the worker.

$$b_{t+1} = b_t + \chi(b^{gen}) * (A_i - b_t) \tag{3.3}$$

where the formula of the function $\chi(b^{gen})$ is

$$\chi(b^{gen}) = 1 - 0.5^{1/(20+0.25*(b^{gen}-1)*(4-20))} \tag{3.4}$$

Brief interpretation: There are 5 general skill groups 1 to 5, where 1 is the lowest skill group and 5 is the group with the highest skills. A worker from skill group 1 needs 20 months to close half of the gap between his specific skills and the technology of his employer, where a worker of skill group 5 needs only one fifth of that time, namely 4 months. Therefore, the higher the general skill level of a worker, the faster he acquires the specific skills associated with a given job.

3.1.2. Consumption goods producer (CGP)

The CGP plays the roles of buyer, seller and employer and makes a large number of decisions to influence the markets. At the consumption goods market, the CGP computes the planned production quantity and determines the required input factors for producing the planned output. After that, it produces and distributes the output among the malls. CGPs are active on the labor market after the production planning but before the production takes place. They lay off workers if number of employees is bigger than the actual labor demand and employ workers if number of employees is smaller than the actual labor demand. Overall, the CGP operates the sequence of events in the following way:

1. Product stock (optimal inventory) decision
2. Production inputs (labor and capital) decision
3. Investment (in investment goods) decision
4. Employment (hiring and firing) decision
5. Production (quantity) decision
6. Pricing decision (which price to set)
7. Dividend payment decision

3.1.2.1. Product stock (optimal inventory) decision

The operating cycle starts with product stock decision. The CGP keeps a stock of its products at every regional mall. It decides once a month whether the inventories at different malls need to be refilled in order to try to avoid the shortage of supplied goods and maximize the expected profit. To that end the CGP checks the current stock level reported by each mall it serves and determines an optimal stock level for each mall using a standard managerial method, which is based on a solution to the "newsvendor problem", faced by a newsvendor trying to decide how many newspapers to stock on a newsstand before observing demand, trying to avoid both overage and underage costs if he orders too much or if he orders too little, see [Hillier and Lieberman 1986].

Table 3.4. Quantity choice

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the optimal stock level	Y	optimalStockLevel	The CGP replenishes its stock at each mall	-

Variable/ Parameter	**Symbol**	**Name (in the sim model)**	**Description**	**Value**
			in every period up to a given optimal stock level	
the price of the consumption good	p_i	productSalesPrice	The price of the consumption good i	-
the unit cost of production	c_{t-1}	unitCostOfProduction	The unit cost of production in period t - 1 (the previous period)	-
holding cost	C^{inv}	holdingCost	Holding cost per unit remaining at the mall for one period	*0.1*
monthly discount factor	ρ	rho	The discount rate which takes into account the time value of money	*0.95*
the current stock level	SL	currentStockLevel	The level of the stock which is checked at each mall	-
the desired replenishment quantity	D_r	replenishmentQuantity	The desired replenishment quantity at the mall in region r	-
the sum of the orders	D^{plan}	sumOfOrders	The sum of the planned delivery volumes for the malls	-
the planned output	Q^{plan}	plannedOutput	The planned output that is used for the determination of the input factor needs	-
a linear combination	ξ	xi	For combining the planned current demand with weight xi and the historic demand with weight (1 - xi)	*0.5*

Algorithm: In order to determine the optimal stock level the CGP estimates the demand distribution based on demands reported by the mall in the previous T months. Because the estimated demand distribution is not clearly spelled out in the original model, suppose now that the *D* demand follows a uniform distribution (continuous) between D_{min} and D_{max} among the last sales. The value of the optimal stock level satisfies the equation

$$\Phi^{plan}(Y) = \frac{p_i - (1-\rho) * c_{t-1}}{p_i + C^{inv}} \tag{3.5}$$

Here $\Phi^{plan}(Y)$ denotes the cumulative distribution function (CDF) of the estimated demand distribution.

The CGP applies an optimal inventory policy to determine whether and how much to replenish inventory. The determination of the desired replenishment quantity to each mall is the difference between the optimal stock level and the current mall stock. The optimal inventory policy is the following:

- If the current stock level is greater than or equal to the optimal stock level $SL \geq Y$, the CGP does not need to replenish inventory D_r=0.
- If the current stock level is less than the optimal stock level $SL < Y$, the CGP needs to replenish inventory $D_r = Y - SL$.

The sum of the planned delivery volumes for all malls becomes

$$D^{plan} = \sum_{r=1}^{R} D_r \tag{3.6}$$

where *R* denotes the number of regions. As previously mentioned, this economy consists of *R=2* regions.

However, in order to smooth the simulation and to avoid excessive oscillations, the final planned production quantity is not simply the sum of all planned mall deliveries D^{plan}, but a linear combination of D^{plan} and a mean of the last *T=4* production volumes.

$$Q^{plan} = \xi * D^{plan} + (1-\xi) * \frac{1}{T} * \sum_{k=t-T}^{t-1} Q_k \tag{3.7}$$

3.1.2.2. Production inputs (labor and capital) decision

After completing the planned output, the CGP computes the required factor inputs. In this model for producing the homogenous consumption good two input factors are used, i.e. labor and capital. In order to realize a capital-to-labor ratio, the standard rule for Constant Elasticity of Substitution (CES) production functions is applied.

Table 3.5. Factor demand

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the planned output	Q^{plan}	plannedOutput	The planned output that is used for the determination of the input factor needs	-
labor intensity of production	α	alpha	0 < α, β and $\alpha + \beta = 1$ α and β are the output elasticities of labor and capital	*0.662*
capital intensity of production	β	beta		*0.338*
the average quality of the capital stock	A_i	averageQualityOfCapitalStock	The average quality of the investment goods employed by CGP i	-
the average level of specific skills of employees	B	averageSpecificSkillLevel	The average level of specific skills of employees	-
the planned labor input	L^{plan}	plannedLaborInput	The planned labor force is directly related to the planned production quantity	-
the planned capital input	K^{plan}	grossInvestment	The planned capital stock is directly related to the planned production quantity	-
the price of the investment good	p^{inv}	investmentSalesPrice	The price of the investment good	-
The average wage of employees	w^e	laborPrice	The average wage of employees	-
the gross investment	K_{t-1}	grossInvestment	The stock of machines etc. in period t - 1 (the previous period)	-
the depreciation rate of capital	δ	delta	The depreciation rate of capital	*0.01*

Algorithm: Based on the planned output the corresponding demand for capital and labor are determined.

$$K^{PLAN} = \frac{(\beta * w^e)^{\alpha} * Q^{plan}}{(\alpha * p^{inv})^{\alpha} * \min[A_i, B]}$$
$$L^{PLAN} = \frac{(\alpha * p^{inv})^{\beta} * Q^{plan}}{(\beta * w^e)^{\beta} * \min[A_i, B]} \tag{3.8}$$

Two cases have to be considered for the factor demand determination: If $K^{PLAN} \geq (1-\delta) * K_{t-1}$, the desired capital and labor stocks read $K^{PLAN} = K^{plan}$ and $L^{PLAN} = L^{plan}$. Otherwise,

$$K^{plan} = (1-\delta) * K_{t-1}$$
$$L^{plan} = \left(\frac{Q^{plan}}{((1-\delta) * K_{t-1})^{\beta} * \min[A_i, B]} \right)^{\frac{1}{\alpha}} \tag{3.9}$$

3.1.2.3. Investment (in investment goods) decision

The existing capital stock of the CGP depreciates over time. Once there is a positive demand for investment goods, the CGP purchases the needed amount from the IGP thereby upgrading its capital stock.

Table 3.6. Investment demand

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the gross investment	K_t	grossInvestment	The stock of machines etc. needed for production	-
the new investment	I	newInvestment	The CGP needs more investments in order to expand its production scale	-
the depreciation rate of capital	δ	delta	The depreciation rate of capital	*0.01*

Algorithm: The capital stock of the CGP is updated as old capital is replaced by new investments.

$$K_{t+1} = (1-\delta) * K_t + I \tag{3.10}$$

3.1.2.4. Employment (hiring and firing) decision

After determining the required labor force during the calculation of planned production inputs, the CGP compares it to the existing labor force, and then decides to post vacancies or to dismiss workers depending on the difference between the required labor force and the existing labor force. In case a CGP has to downsize the labor force, it fires workers with the lowest general skill levels until the needed number of workers is reached.

In another case, if a CGP has a positive demand for labor, vacancies are posted together with a wage offer. The incoming applications are ranked with respect to the general skill level. More specifically, applicants with higher general skill levels are ranked higher. If there exist two or more applicants who have the same general skill level, they are ranked by chance. The CGP sends as many job offers as it has vacancies to the highest ranked applicants. If the CGP then receives job acceptances from the applicants, it updates the number of employees and the number of vacancies. Otherwise there are still some vacancies and the CGP increases the offered wage.

3.1.2.5. Production (quantity) decision

After the two input factors are completed, the CGP starts with the actual production cycle. The production technology is represented by a Cobb-Douglas type production function with complementarities between the quality of investment goods and the specific skills of employees for using that type of technology. In economics, the Cobb-Douglas functional form of production functions is widely used to represent the relationship of an output to inputs.

Table 3.7. Production

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the quantity of production	Q	producedQuantity	The quantity of production	-
labor intensity of production	α	alpha	0 < α, β and α + β = 1 α and β are the output elasticities of labor and capital	*0.662*
capital intensity of production	β	beta		*0.338*
the average quality of the capital stock	A_i	averageQualityOfCapitalStock	The average quality of the investment goods employed by CGP i	-
the average level of specific skills of employees	B	averageSpecificSkillLevel	The average level of specific skills of employees	-
the actual labor input	L	laborInput	The actual labor force	-
the actual capital input	K	grossInvestment	The actual capital stock	-
the desired replenishment quantity	D_r	replenishmentQuantity	The desired replenishment quantity at the mall in region r	-
the sum of the orders	D^{plan}	sumOfOrders	The sum of the planned delivery volumes for the malls	-

Algorithm: The production quantity of a CGP is given by

$$Q = \min\left[B, A_i\right] * L^{\alpha} * K^{\beta} \tag{3.11}$$

where,

$\min\left[B, A_i\right]$ denotes the factor productivity which is given by the minimum of B and A_i.

After finishing the production, the CGP distributes the output among the malls without costs. Because the realized production volume does not necessarily correspond to the planned output, the CGP determines the actual delivery quantities proportionally to the intended quantities:

$$Q_r = \frac{D_r}{D^{plan}} * Q \tag{3.12}$$

3.1.2.6. Pricing decision (which price to set)

The price of the consumption good produced by the CGP changes with the unit cost in production.

Table 3.8. Pricing

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the price of the consumption good	p_i	productSalesPrice	The price of the consumption good i	-

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
Mark-up factor	μ	markUpFactor	The difference between the cost of a product and its sales price	*0.2*
the unit cost of production	c_{t-1}	unitCostOfProduction	The unit cost of production in period t - 1 (the previous period)	-

Algorithm: The CGP sets the price of its product according to the standard rule

$$p_i = (1+\mu) * c_{t-1} \tag{3.13}$$

3.1.2.7. Dividend payment decision

At the end of every month CGPs have to check whether they are in a profitable position that households can receive dividends from them. The CGP pays dividends depending on its monthly realized profit and current balance of saving account according to a simple dividend policy.

Table 3.9. Dividend payment

Variable/ Parameter	Symbol	Name (in the sim model)	Description	Value
the sales revenue	Rev_t	productSalesRevenue	The sales revenue in period t	-
cost of production	C	costOfProduction	The production cost	-
the monthly realized profit	Pro	monthlyRealizedProfit	The monthly realized profit is the difference between the sales revenue and the production cost	Rev_t-C
the current balance of saving account	Acc	currentBalanceOfSavingAccount	The current balance of saving account	-
the dividends	Div	dividends	The CGP pays dividends to all households	-
a fixed proportion	div	div	The CGP pays a fixed proportion $div \in [0,1]$ of its profit as dividends to all households	*0.9*

Algorithm: If the monthly realized profit of a CGP is not positive, the CGP pays no dividends and the losses are entered on the current balance of saving account. In case of positive profit, the CGP pays dividends based on a simple dividend policy that defines three kinds of dividend rates depending on the current balance of saving account. The rule states

1. If the balance is negative $Acc < 0$ and the debt is on a scale above the last monthly revenue $|Acc| > Rev_{t-1}$, the CGP pays no dividends $Div = 0$.
2. If the balance is positive $Acc > 0$ and savings are above the monthly revenue $Acc > Rev_t$, the CGP disburses all profits as dividends $Div = Pro$.
3. In the remaining case, if the balance is between these critical levels of the above two cases, the CGP pays out a fixed proportion of profits as dividends $Div = div * Pro$.

3.1.3. Mall

The mall is modeled as a passive agent in this model, so it cannot take decisions. This agent performs the selling role of CGP in the region. It keeps and receives consumption goods produced by CGPs, then sells them and collects every product sales revenue that is reported back to the corresponding CGP.

3.1.4. Capital Goods Producer (IGP)

In this model, the IGP is unique and acts globally. It only has one role and plays the role of seller on the investment goods market. It supplies investment goods infinitely to all CGPs. The investment good as a kind of productive factor of CGP has two properties, i.e. quality and price, which are increased by simple rules.

- The quality and price of the investment good increase over time due to technological change. The price varies with the quality.
- Every month the quality is increased by 5% with probability 10% where with probability 90% there is no change of quality.

Finally, in order to close the model, the monthly revenue of the IGP is uniformly distributed to all households.

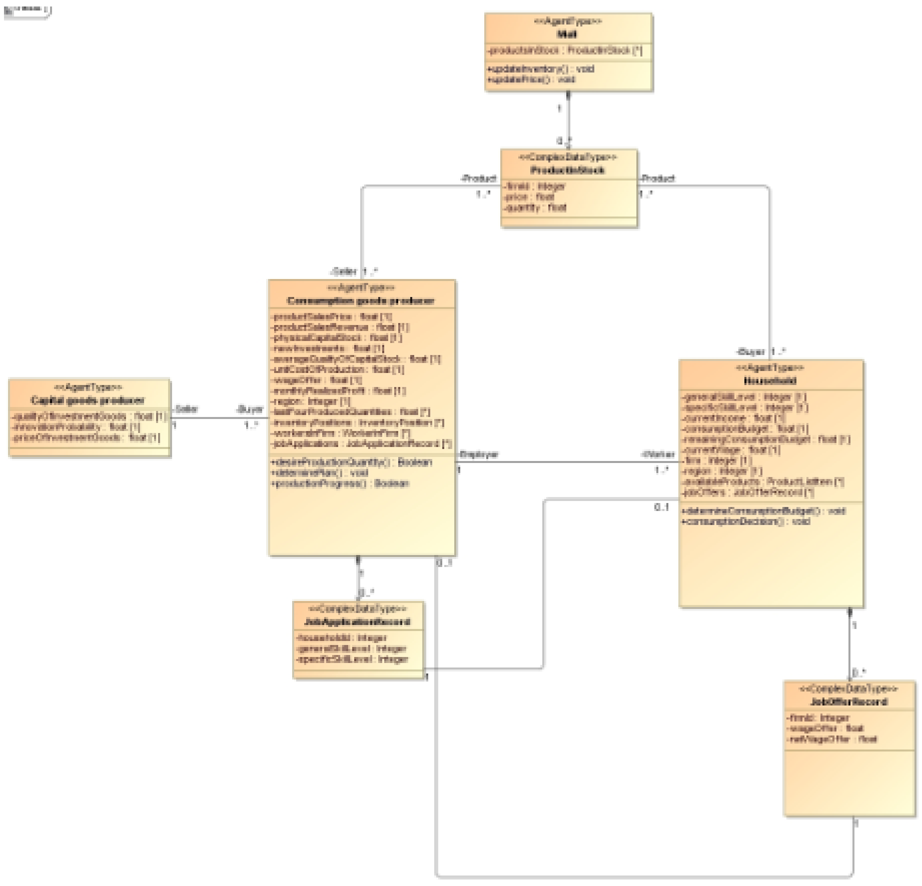
3.2. Data processing

In order to show agents and their interactions more clearly and prepare for reconstruction of the model, a UML class diagram for the model and a BPMN model for the labor market interaction process are created.

3.2.1. UML class diagram

The UML class diagram describes the static structure of the model. It is a convenient way of representing the agents of the model. Each agent class is represented by a set of attributes and methods that operate on the agent class. As mentioned, there are four types of agents in the model: household, consumption goods producer, mall and capital goods producer.

Figure 3.1. UML class diagram



<<AgentType>>
Mall
-productsInStock : ProductInStock [*]
+updateInventory() : void
+updatePrice() : void
<<ComplexDataType>>
ProductInStock
-firmId : Integer
-price : float
-quantity : float
-Product
-Seller
-Buyer
<<AgentType>>
Consumption goods producer
-productSalesPrice : float [1]
-productSalesRevenue : float [1]
-physicalCapitalStock : float [1]
-newInvestments : float [1]
-averageQualityOfCapitalStock : float [1]
-unitCostOfProduction : float [1]
-wageOffer : float [1]
-monthlyRealizedProfit : float [1]
-region : Integer [1]
-lastFourProducedQuantities : float [*]
-inventoryPositions : InventoryPosition [*]
-workersInFirm : WorkerInFirm [*]
-jobApplications : JobApplicationRecord [*]
+desireProductionQuantity() : Boolean
+determinePlan() : void
+productionProgress() : Boolean
<<AgentType>>
Capital goods producer
-qualityOfInvestmentGoods : float [1]
-innovationProbability : float [1]
-priceOfInvestmentGoods : float [1]
-Employer
-Worker
<<AgentType>>
Household
-generalSkillLevel : Integer [1]
-specificSkillLevel : Integer [1]
-currentIncome : float [1]
-consumptionBudget : float [1]
-remainingConsumptionBudget : float [1]
-currentWage : float [1]
-firm : Integer [1]
-region : Integer [1]
-availableProducts : ProductListItem [*]
-jobOffers : JobOfferRecord [*]
+determineConsumptionBudget() : void
+consumptionDecision() : void
<<ComplexDataType>>
JobApplicationRecord
-householdId : Integer
-generalSkillLevel : Integer
-specificSkillLevel : Integer
<<ComplexDataType>>
JobOfferRecord
-firmId : Integer
-wageOffer : float
-netWageOffer : float

3.2.2. BPMN model

The Business Process Modeling Notation (BPMN) is a graphical notation for describing various kinds of processes. The main notational elements in BPMN are *FlowObjects*, that are contained in *Pools* and connected via *Sequence*- and *MessageFlows*. They subdivide in *Events*, atomic and composite *Activities* and *Gateways* for forking and joining. *SequenceFlows* describe the sequence in which the several *FlowObjects* have to be completed, while *MessageFlows* describe the exchange of messages between *Pools*. Thus, BPMN combines the definition of local workflows and the interaction between them.

3.2.2.1. Labor market interaction process

According to the procedures described in the previous sections CGPs check once a month whether to post vacancies. Job seekers search for jobs. The matching algorithm between vacancies and job seekers can be summarized as follows:

Step 1: CGPs determine once a month their planned production output and decide to post vacancies including wage offers or to fire workers.

Step 2: Job seekers look at the vacancies, rank them according to the wage offers, and then send job applications in terms of their reservation wage.

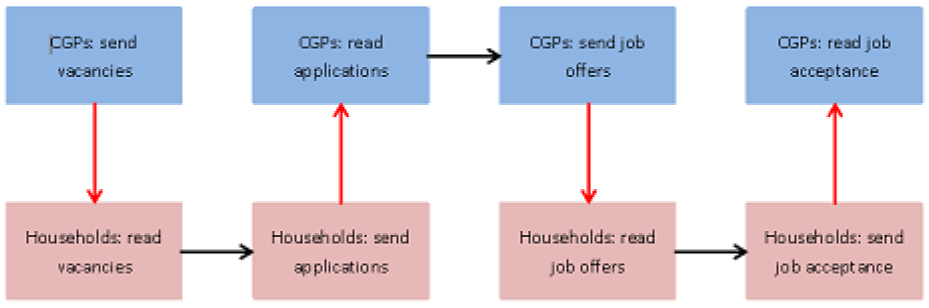
Step 3: CGPs receive applications, rank the applicants, and send job offers to as many applicants as they have vacancies to fill. An applicant with higher general skill level is more likely to receive a job offer.

Step 4: Job seekers receive job offers, then send respectively a offer acceptance. A job seeker who remains unemployed lowers his reservation wage.

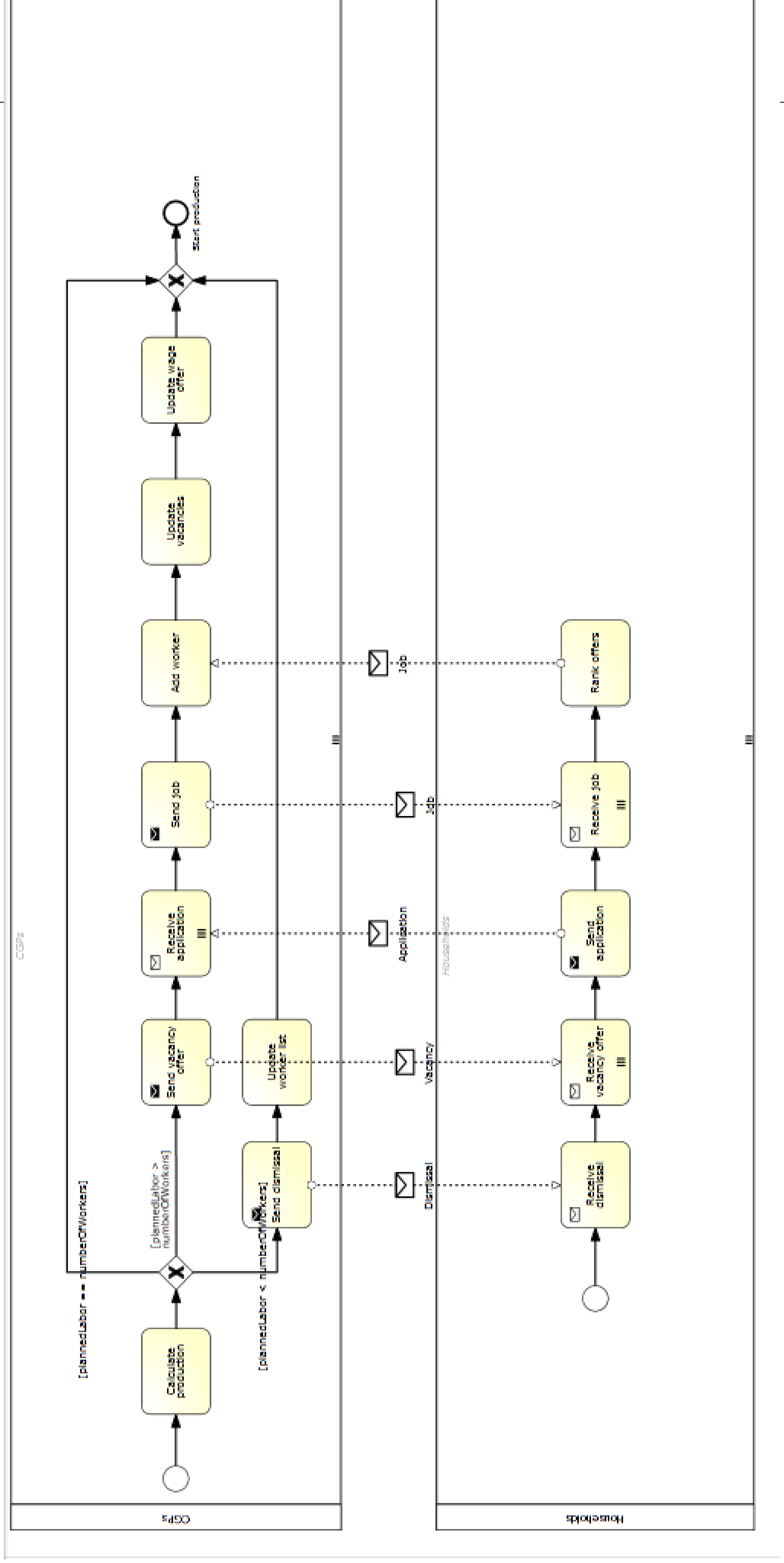
Step 5: CGPs receive offer acceptances, then update their wage offer depending on how many vacancies are left unfilled.

The search-and-matching algorithm is illustrated below.

Figure 3.2. Sequence of events in the labor market


CGPs: send vacancies
CGPs: read applications
CGPs: send job offers
CGPs: read job acceptance
Households: read vacancies
Households: send applications
Households: read job offers
Households: send job acceptance

The model consists of two pools: CGPs and households. The process model based on the search-and-matching process is used to represent the interaction between CGPs and households in the labor market.


CGPs
Households
Calculate production
[plannedLabor == numberOfWorkers]
[plannedLabor > numberOfWorkers]
[plannedLabor < numberOfWorkers]
Send vacancy offer
Send dismissal
Update worker list
Receive application
Send job
Add worker
Update vacancies
Update wage offer
Start production
Dismissal
Vacancy
Application
Job
Job
Receive dismissal
Receive vacancy offer
Send application
Receive job
Rank offers

Chapter 4. Reconstruction of the model in the AOR simulation language

This chapter starts with a description of Agent-Object-Relationship simulation. After this, the reconstructed simulation scenario of the original model will be introduced.

4.1. AOR simulation

This section introduces an important agent-based approach, called Agent-Object-Relationship (AOR) simulation. The model will be reconstructed with the help of it. In this approach, the simulation system includes a set of interacting agents and a simulation environment where all agents exist. Every agent is an independent entity with activities. It can act with other agents and with the simulation environment.

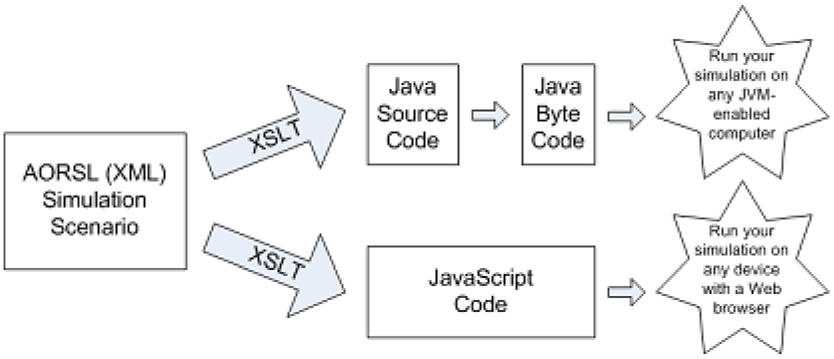
4.1.1. Simulation language

AOR simulations use a high-level declarative language, called Agent-Object-Relationship Simulation Language (AORSL). The language can be processed directly with AOR-JavaSim. It is an XML-based language that uses an XML Schema definition to enable easy validating and parsing of AORSL documents. Using XML syntax also has the advantage that AORSL files can be created, edited and viewed with a lot of free software.

4.1.2. Simulation scenario and simulation model

A simulation scenario is expressed with the help of AORSL. In the simulation stage the scenario is first stored in XML tagged data files and then translated to program code (either Java or JavaScript) and finally executed with a JVM or in a Web browser.

Figure 4.1. Code Generation


AORSL (XML) Simulation Scenario
XSLT
XSLT
Java Source Code
Java Byte Code
Run your simulation on any JVM-enabled computer
JavaScript Code
Run your simulation on any device with a Web browser

A simulation scenario consists of a simulation model, an initial state definition and a user interface definition, including a statistics UI and an animation UI. A simulation model consists of an optional space model, a set of entity type definitions, including different categories of event, message, object and agent types and a set of rules, which define causality laws governing the environment's state changes and the causation of follow-up events.

Both the behavior of the environment and the behavior of agents are modeled with the help of rules. Rules are defined as follows,

A rule is a 6-tuple < *WHEN, FOR, DO, IF, THEN, ELSE* > where

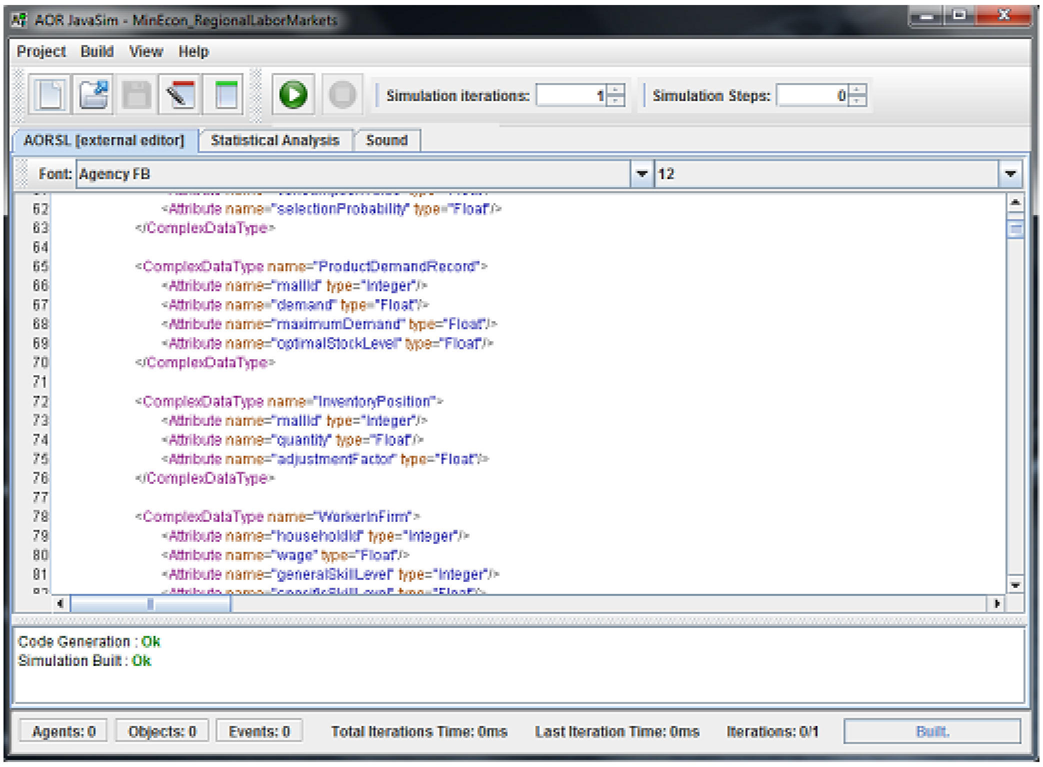
- *WHEN* is an event expression specifying the type of event that triggers the rule

- *FOR* is a set of variable declarations, such that each variable is bound either to a specific object or to a set of objects
- *IF* is a logical condition formula possibly containing variables
- *DO*, *THEN* and *ELSE* are execution elements consisting of two blocks:
  - *UPDATE-ENV* is an expression specifying an update of the environment state
  - *SCHEDULE-EVT* specifies a list of resulting events that will be scheduled

4.1.3. AOR-JavaSim

AOR-JavaSim is a java software for generating and running AOR simulations. The software generates executable Java code from a given simulation scenario and compiles this code.

Figure 4.2. Interface of AOR-JavaSim



Simulation iterations: 1
Simulation Steps: 0
Font: Agency FB
12

```
            <Attribute name="selectionProbability" type="Float"/>
        </ComplexDataType>

        <ComplexDataType name="ProductDemandRecord">
            <Attribute name="mallId" type="Integer"/>
            <Attribute name="demand" type="Float"/>
            <Attribute name="maximumDemand" type="Float"/>
            <Attribute name="optimalStockLevel" type="Float"/>
        </ComplexDataType>

        <ComplexDataType name="InventoryPosition">
            <Attribute name="mallId" type="Integer"/>
            <Attribute name="quantity" type="Float"/>
            <Attribute name="adjustmentFactor" type="Float"/>
        </ComplexDataType>

        <ComplexDataType name="WorkerInFirm">
            <Attribute name="householdId" type="Integer"/>
            <Attribute name="wage" type="Float"/>
            <Attribute name="generalSkillLevel" type="Integer"/>
```

```
Code Generation : Ok
Simulation Built : Ok
```

Agents: 0 Objects: 0 Events: 0 Total Iterations Time: 0ms Last Iteration Time: 0ms Iterations: 0/1 Built.

4.2. The scenario specification

This section shows the reconstruction of the original model as an AOR simulation. As previously mentioned, in the original model for the policy experiments there are three types of policies. In order to show possible consequences of different policy measures, different scenarios need to be created. More specifically, each type of policy corresponds to two different scenarios characterized by the level of commuting costs. Thereinto, in the one scenario the commuting costs are set to zero and the other scenario where the commuting costs are 0.05 is to be considered. Thus, different scenarios can be generated by specifying alternative sets of values for the exogenous variables.

4.2.1. MinEcon_RegionalLaborMarkets.xml

This section now shows the simulation file in AORSL. Each single run of the simulation scenario represents 5000 steps, each of which is seen as a day. The scenario model includes eleven variables of statistics, seven complex data types, ten message types, ten action event types, four agent types, several environment rules and so on. The subsequent descriptions will be focused on the agent types and the environment rules. The explanations of the others can be seen in Appendix.

4.2.1.1. Agents specification

As mentioned, there are four types of agents necessary in this AOR model. All agents in the simulation are instances of these agent types. To be more specific, there are two regions, each of which hosts 200 households, 5 consumption goods producers and a regional market denoted as mall. There is a single capital goods producer. Every agent in the simulation has several reaction rules that are used to define the behavior of the agent.

4.2.1.1.1. CapitalGoodsProducer

The rule *CalculateEqualShare_Rule* applies when the periodic time event *CalculateEqualShare* occurs. This event simulates at the end of every month (occurrenceTime="24") and occurs at every 24 steps in the simulation. This rule is used to calculate the equal shares which will be soon distributed to all households. The *equalShare* property holds an equal share for a household is determined by the amount of households. There are 400 instances of the agent type *household*, so the agent divides its monthly revenues into 400 equal shares. The *equalShare* is expressed as the amount of households divided by the *investmentSalesRevenue* property of the agent. Then, the *investmentSalesRevenue* property is set to 0. This event causes a thing, the agent does the *PayEqualShare* action event to pay the equal shares to another agent *household*.

Table 4.1. Reaction Rule: CalculateEqualShare_Rule

Triggering Event	CalculateEqualShare
Declaration	-
Condition	-
Resulting Actions	PayEqualShare • share = equalShare
Resulting Messages	-
State Effects	• equalShare = investmentSalesRevenue / 400 • investmentSalesRevenue = 0

4.2.1.1.2. Mall

The rule *AtStartOfMonthCheckStockLevel_Rule* is used when the periodic time event *AtStartOfMonthCheckStockLevel* occurs. This event simulates at the beginning of every month (occurrenceTime="2") and occurs at every 24 steps during the running of the simulation. This rule is used to check the current inventories of consumption goods. The result of the reaction rule is the creation of a new outgoing message to report the current stock level of every product to the corresponding CGP. The value of the *quantity* property of the message is obtained from the *productsInStock* list of the agent.

Table 4.2. Reaction Rule: AtStartOfMonthCheckStockLevel_Rule

Triggering Event	AtStartOfMonthCheckStockLevel
Declaration	ProductInStock p : productsInStock
Condition	-
Resulting Actions	-
Resulting Messages	TellCurrentStockLevel • receiverIdRef = p.firmId

	• quantity = p.quantity
State Effects	-

The rule *DeliverProduct_Rule* applies in case a message of type *DeliverProduct* is received by the agent. This will increase the inventory of the target product by the *quantity* property of the message. To that end, the *updateInventory* function of the agent is called. It has two parameters: *firmId* and *quantity*. *firmId* holds the "identity" of the sender of the message and *quantity* holds the *quantity* property of the message. The function is used to update the value of the *quantity* property of the selected record of the *productsInStock* list.

Table 4.3. Reaction Rule: DeliverProduct_Rule

Triggering Event	DeliverProduct
Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	updateInventory(firmId, quantity) • firmId = InMessageEvent.senderIdRef • quantity = DeliverProduct.quantity

4.2.1.1.3. ConsumptionGoodsProducer

The rule *TellCurrentStockLevel_Rule* applies when a message of type *TellCurrentStockLevel* is received. The agent will check whether the stock it keeps at the mall has to be refilled based on the given condition. If statement needs to be used. If the *quantity* property of the message is greater than or equal to the *optimalStockLevel*, the desired replenishment quantity of the product is set to zero. In the opposite case, the desired replenishment quantity is obtained by deducting the *quantity* property of the message from the *optimalStockLevel*. When the desired replenishment quantity is fixed, a new record which has three attributes (that is, *mallId*, *quantity* and *adjustmentFactor*) will be added to the bottom of the *inventoryPositions* list of the agent.

Table 4.4. Reaction Rule: TellCurrentStockLevel_Rule

Triggering Event	TellCurrentStockLevel
Declaration	-
Condition	TellCurrentStockLevel.quantity >= demandRecordFromListWithId(InMessageEvent.senderIdRef).optimalStockLevel
Resulting Actions	-
Resulting Messages	-
State Effects	inventoryPositions • new InventoryPosition() • mallId = InMessageEvent.senderIdRef • quantity = 0 • adjustmentFactor = 0
Condition	TellCurrentStockLevel.quantity < demandRecordFromListWithId(InMessageEvent.senderIdRef).optimalStockLevel
Resulting Actions	-

Resulting Messages	-
State Effects	inventoryPositions • new InventoryPosition() • mallId = InMessageEvent.senderIdRef • quantity = demandRecordFromListWithId(InMessageEvent.senderIdRef).optimalStockLevel - TellCurrentStockLevel.quantity • adjustmentFactor = 0

The rule *MakeProductionPlan_Rule* is used when the periodic time event *MakeProductionPlan* occurs. This event occurs when the agent starts making its production plan (occurrenceTime="4") and is repeated every 24 steps. This rule is used to determine the demand of two input factors (labor and capital) for production. For that purpose, the *determineProductionPlan* function of the agent is called. This function is constructed based on CES production functions that have been mentioned above. It has five arguments: *laborPrice*, *investmentPrice*, *plannedProductionQuantity*, *averageSpecificSkillLevel* and *averageCapitalStockQuality*. *laborPrice* holds the average wage of employees of the agent. *investmentPrice* holds the price of the investment good. *plannedProductionQuantity* holds the planned output based on the desired replenishment quantity of the product for each mall. *averageSpecificSkillLevel* denotes the average specific skill level of workers and *averageCapitalStockQuality* denotes the average quality of the capital stock of the agent. The event *MakeProductionPlan* causes several things. First, the agent does the *BuyNewInvestment* action event to purchase the needed amount of investments from the IGP. This event triggers if the *newInvestment* property of the agent is greater than zero. Second, the agent does the *DismissWorker* action event to downsize the labor force. Finally, the agent creates an action event *PostVacancyInformation*, if the value of the *laborSupplyQuantity* property of the agent is larger than zero.

Table 4.5. Reaction Rule: MakeProductionPlan_Rule

Triggering Event	MakeProductionPlan
Declaration	-
Condition	-
Resulting Actions	BuyNewInvestment • newInvestment > 0 DismissWorker • downsizingIncumbentWorkforce() PostVacancyInformation • laborSupplyQuantity > 0 • delay = 3 • firmId = id • wageOffer = wageOffer * determineAverageSpecificSkillLevel()
Resulting Messages	-
State Effects	determineProductionPlan(laborPrice, investmentPrice, plannedProductionQuantity, averageSpecificSkillLevel, averageCapitalStockQuality) • laborPrice = determineLaborCost() / workersInFirm.size

	• investmentPrice = Global.investmentSalesPrice • plannedProductionQuantity = desireProductionQuantity() • averageSpecificSkillLevel = determineAverageSpecificSkillLevel() • averageCapitalStockQuality = averageQualityOfCapitalStock

The rule *TellSalesRevenue_Rule* applies in case a message of type *TellSalesRevenue* is received by the agent. This will increase the *productSalesRevenue* property and *productSalesQuantity* property of the agent by the corresponding *revenue* property and *quantity* property of the message. There is nothing else to do in this case.

Table 4.6. Reaction Rule: TellSalesRevenue_Rule

Triggering Event	TellSalesRevenue
Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	• productSalesRevenue = productSalesRevenue + TellSalesRevenue.revenue • productSalesQuantity = productSalesQuantity + TellSalesRevenue.quantity • updateDemand(mallId, demand) • mallId = InMessageEvent.senderIdRef • demand = TellSalesRevenue.quantity

The rule *TellVacancy_Rule* applies in case a message of type *TellVacancy* is received. The agent receives informations from the applicants about their respective general as well as specific skill levels, and then adds these informations to the bottom of the *jobApplications* list of the agent.

Table 4.7. Reaction Rule: TellVacancy_Rule

Triggering Event	TellVacancy
Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	jobApplications • new JobApplicationRecord() • householdId = InMessageEvent.senderIdRef • generalSkillLevel = TellVacancy.generalSkillLevel • specificSkillLevel = TellVacancy.specificSkillLevel

The rule *InFirstIterationRankApplicant_Rule* is used when the periodic time event *InFirstIterationRankApplicant* occurs. This event occurs when the agent ranks the applicants in the first round (occurrenceTime="9") and is repeated every 24 steps. It applies only when the size of the *jobApplications* list is greater than zero. This rule is used to choose the best applicants for the positions that are needed. Thus, the *determineJobOffer* function of the agent is called. It sorts and updates the *jobApplications* list. This event causes a thing, the agent does the *InFirstIterationOfferJob* action event to send job offers to the highest ranked applicants.

Table 4.8. Reaction Rule: InFirstIterationRankApplicant_Rule

Triggering Event	InFirstIterationRankApplicant
Declaration	-
Condition	jobApplications.size > 0
Resulting Actions	InFirstIterationOfferJob
Resulting Messages	-
State Effects	determineJobOffer()

The rule *AcceptJob_Rule* applies in case a message of type *AcceptJob* is received by the agent. This rule is used to update the *laborSupplyQuantity* property and the *workersInFirm* list of the agent. The value of the *laborSupplyQuantity* property minus one. Moreover, the informations which contains the message sender, the *wage* property of the message, the *generalSkillLevel* property of the message and the *specificSkillLevel* property of the message will be added to the bottom of the *workersInFirm* list of the agent.

Table 4.9. Reaction Rule: AcceptJob_Rule

Triggering Event	AcceptJob
Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	• laborSupplyQuantity = laborSupplyQuantity - 1 • workersInFirm • new WorkerInFirm() • householdId = InMessageEvent.senderIdRef • wage = AcceptJob.wage • generalSkillLevel = AcceptJob.generalSkillLevel • specificSkillLevel = AcceptJob.specificSkillLevel

The rule *ResignJob_Rule* applies in case a message of type *ResignJob* is received. The agent receives resignation from its employee and drops him from the list of workers. Specifically, the value of the *laborSupplyQuantity* property of the agent is increased by one. In order to delete a record from the *workersInFirm* list, the *deleteWorkerRecordFromList* function of the agent is called with a parameter *householdId* which holds the message sender. It first determines the record whose *householdId* property is equal to the "identity" of the sender of the message, and then delete this record from the *workersInFirm* list.

Table 4.10. Reaction Rule: ResignJob_Rule

Triggering Event	ResignJob
Declaration	-
Condition	-
Resulting Actions	-

Resulting Messages	-
State Effects	• laborSupplyQuantity = laborSupplyQuantity + 1 • deleteWorkerRecordFromList(householdId) • householdId = InMessageEvent.senderIdRef

The rule *StartSecondIterationLaborSupply_Rule* carries out when the periodic time event *StartSecondIterationLaborSupply* occurs. This event happens at the beginning of the second iteration of hiring activity (occurrenceTime="15") and is also repeated every 24 steps. It is used only when the *laborSupplyQuantity* property of the agent is greater than zero. This event causes a thing, the agent does the *PostVacancyInformation* action event to post vacancies for job seekers.

Table 4.11. Reaction Rule: StartSecondIterationLaborSupply_Rule

Triggering Event	StartSecondIterationLaborSupply
Declaration	-
Condition	laborSupplyQuantity > 0
Resulting Actions	PostVacancyInformation • firmId = id • wageOffer = wageOffer * determineAverageSpecificSkillLevel()
Resulting Messages	-
State Effects	-

The rule *InSecondIterationRankApplicant_Rule* is used when the periodic time event *InSecondIterationRankApplicant* occurs. This event occurs when the agent ranks the applicants in the second round (occurrenceTime="18") and is repeated every 24 steps. The content of this rule is the same as that of the *InFirstIterationRankApplicant_Rule* rule.

Table 4.12. Reaction Rule: InSecondIterationRankApplicant_Rule

Triggering Event	InSecondIterationRankApplicant
Declaration	-
Condition	jobApplications.size > 0
Resulting Actions	InFirstIterationOfferJob
Resulting Messages	-
State Effects	determineJobOffer()

The rule *StartOfProduction_Rule* applies when the periodic time event *StartOfProduction* occurs. This event simulates at the end of every month (occurrenceTime="23") and occurs at every 24 steps in the simulation. This rule is used to calculate the produced quantities. To do this the *productionProgress* function of the agent is called. The function is created based on Cobb-Douglas type production function which has been mentioned above. The value of the *producedQuantity* property of the agent is updated by using this function. The event *StartOfProduction* causes several things. First, the agent creates an action event *DistributeProduct* to deliver the produced quantities. Second, the agent does the *PayWage* action event to pay wages to employees. Finally, the agent does the *IncreaseSpecificSkillLevel* action event to improve the specific skill levels of workers.

Table 4.13. Reaction Rule: StartOfProduction_Rule

Triggering Event	StartOfProduction
Declaration	-

Condition	-
Resulting Actions	DistributeProduct PayWage IncreaseSpecificSkillLevel
Resulting Messages	-
State Effects	productionProgress(averageSpecificSkillLevel, averageCapitalStockQuality, labor, investment) • averageSpecificSkillLevel = determineAverageSpecificSkillLevel() • averageCapitalStockQuality = averageQualityOfCapitalStock • labor = workersInFirm.size • investment = grossInvestment

The rule *CalculateDividend_Rule* is used when the periodic time event *CalculateDividend* occurs. This event simulates at the end of every month (occurrenceTime="24") and occurs at every 24 steps in the simulation. This rule is used to calculate the dividends. The *determineDividend* function is called to determine the dividends based on the algorithm of dividend payment decision of the agent. The event *CalculateDividend* causes two things. First, the agent creates an action event *PayDividend*, if the *equalDividend* property of the agent is greater than zero. Second, the agent does the *SetNewPrice* action event to adjust the product price of the agent. This event has one attribute which records the new price.

Table 4.14. Reaction Rule: CalculateDividend_Rule

Triggering Event	CalculateDividend
Declaration	-
Condition	-
Resulting Actions	PayDividend • equalDividend > 0 • dividend = equalDividend SetNewPrice • price = productSalesPrice
Resulting Messages	-
State Effects	determineDividend()

4.2.1.1.4. household

The rule *AtStartOfMonthDetermineConsumptionBudget_Rule* is used when the periodic time event *AtStartOfMonthDetermineConsumptionBudget* occurs. This event occurs when the agent intends to set the budget which will be spent on consumption (occurrenceTime="2") and is repeated every 24 steps. This rule is used to determine how much to spend and how much to save based on the personal income of the agent. For that purpose, the *determineConsumptionBudget* function of the agent is called.

Table 4.15. Reaction Rule: AtStartOfMonthDetermineConsumptionBudget_Rule

Triggering Event	AtStartOfMonthDetermineConsumptionBudget

Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	determineConsumptionBudget()

The rule *TellDismissal_Rule* applies in case a message of type *TellDismissal* is received by the agent. This will change several properties of the agent. First, the *lastFirm* property is updated to the *firm* property of the agent. Second, the *firm* property is set to 100. Finally, the value of the *jobSeeker* property of the agent is set true.

Table 4.16. Reaction Rule: TellDismissal_Rule

Triggering Event	TellDismissal
Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	• lastFirm = firm • firm = 100 • jobSeeker = true

The rule *JobOffer_Rule* applies when a message of type *JobOffer* is received by the agent. A new record which has three attributes (such as *firmId*, *wageOffer* and *netWageOffer*) will be added to the bottom of the *jobOffers* list of the agent. *firmId* holds the "identity" of the sender of the message and *wageOffer* holds the *wageOffer* property of the message. *netWageOffer* is determined by if statement. More specifically, if the *region* property of the message is equal to the *region* property of the agent, the value of *netWageOffer* is the equivalent of the value of *wageOffer*. In the opposite case, *netWageOffer* is measured by subtracting the commuting costs from the *wageOffer* property of the message.

Table 4.17. Reaction Rule: JobOffer_Rule

Triggering Event	JobOffer
Declaration	-
Condition	JobOffer.region == region
Resulting Actions	-
Resulting Messages	-
State Effects	jobOffers • new JobOfferRecord() • firmId = InMessageEvent.senderIdRef • wageOffer = JobOffer.wageOffer • netWageOffer = JobOffer.wageOffer
Condition	JobOffer.region != region
Resulting Actions	-
Resulting Messages	-
State Effects	jobOffers

	• new JobOfferRecord() • firmId = InMessageEvent.senderIdRef • wageOffer = JobOffer.wageOffer • netWageOffer = JobOffer.wageOffer - Global.comm

The rule *InFirstIterationAcceptJob_Rule* is used when the periodic time event *InFirstIterationAcceptJob* occurs. This event occurs in the first round of labor supply when the agent intends to accept a new job (occurrenceTime="12") and is repeated every 24 steps. It applies only when the size of the *jobOffers* list is greater than zero. The *determineJobAcceptance* function of the agent is called. It is used to sort the *jobOffers* list and select the highest ranked record. This event causes several things. First, the agent creates a new outgoing message *AcceptJob* to accept the new position. The message which has three attributes (that is, *wage*, *generalSkillLevel* and *specificSkillLevel*) will be sent to the CGP which provides this position. The *wage* property holds the *currentWage* property of the agent. The *generalSkillLevel* property holds the *generalSkillLevel* property of the agent and the *specificSkillLevel* property holds the *specificSkillLevel* property of the agent. The next result of the reaction rule is the creation of another new outgoing message *ResignJob* to resign the old position, if the value of the *lastFirm* property of the agent is not equal to 100.

Table 4.18. Reaction Rule: InFirstIterationAcceptJob_Rule

Triggering Event	InFirstIterationAcceptJob
Declaration	-
Condition	jobOffers.size > 0
Resulting Actions	-
Resulting Messages	AcceptJob • receiverIdRef = firm • wage = currentWage • generalSkillLevel = generalSkillLevel • specificSkillLevel = specificSkillLevel ResignJob • lastFirm != 100 • receiverIdRef = lastFirm
State Effects	determineJobAcceptance()

The rule *InSecondIterationAcceptJob_Rule* is used when the periodic time event *InSecondIterationAcceptJob* occurs. This event occurs in the second round of labor supply when the agent intends to accept a new job (occurrenceTime="21") and is repeated every 24 steps. The content of this rule is the same as that of the *InFirstIterationAcceptJob_Rule* rule.

Table 4.19. Reaction Rule: InSecondIterationAcceptJob_Rule

Triggering Event	InSecondIterationAcceptJob
Declaration	-
Condition	jobOffers.size > 0
Resulting Actions	-
Resulting Messages	AcceptJob

	• receiverIdRef = firm • wage = currentWage • generalSkillLevel = generalSkillLevel • specificSkillLevel = specificSkillLevel ResignJob • lastFirm != 100 • receiverIdRef = lastFirm
State Effects	determineJobAcceptance()

The rule *TellWage_Rule* applies in case a message of type *TellWage* is received. The agent receives the wage for the full month from his employer. If statement is used. More specifically, if the *region* property of the message is equal to the *region* property of the agent, the *laborIncome* property of the agent is the equivalent of the *wage* property of the message. In the opposite case, the *laborIncome* property is measured by subtracting the commuting costs from the *wage* property of the message. When the *laborIncome* property is fixed, the value of the *currentIncome* property of the agent is increased by the value of *laborIncome*.

Table 4.20. Reaction Rule: TellWage_Rule

Triggering Event	TellWage
Declaration	-
Condition	TellWage.region == region
Resulting Actions	-
Resulting Messages	-
State Effects	• laborIncome = TellWage.wage • currentIncome = currentIncome + laborIncome
Condition	TellWage.region != region
Resulting Actions	-
Resulting Messages	-
State Effects	• laborIncome = TellWage.wage - Global.comm • currentIncome = currentIncome + laborIncome

The rule *TellSpecificSkillLevel_Rule* applies when a message of type *TellSpecificSkillLevel* is received by the agent. This will change the specific skill level of the agent. Thus, the *specificSkillLevel* property of the agent is updated to the *specificSkillLevel* property of the message. There is nothing else to do in this case.

Table 4.21. Reaction Rule: TellSpecificSkillLevel_Rule

Triggering Event	TellSpecificSkillLevel
Declaration	-
Condition	-
Resulting Actions	-
Resulting Messages	-
State Effects	• specificSkillLevel = TellSpecificSkillLevel.specificSkillLevel

4.2.1.2. EnvironmentRules

The rule *Create_InitialProductsInStock_Rule* is carried out, when an exogenous event *Init* occurs. This event happens at the beginning of the simulation (occurrenceTime="1") and occurs only once. It is used to create some values of initial state. Back to this rule, at first a variable *m* of a *Mall* type and a variable *f* of a *ConsumptionGoodsProducer* type are declared. Then the new record which has three attributes (that is, *firmId*, *price* and *quantity*) will be added to the bottom of the empty list *productsInStock* of the variable *m*. Meanwhile, the new record which has four attributes (that is, *mallId*, *demand*, *maximumDemand* and *optimalStockLevel*) will be added to the bottom of the empty list *productDemands* of the variable *f*.

Table 4.22. Environment Rule: Create_InitialProductsInStock_Rule

Triggering Event	Init
Declaration	Mall m ConsumptionGoodsProducer f
Condition	-
Resulting Messages	-
State Effects	m • productsInStock • new ProductInStock() • firmId = f.id • price = f.productSalesPrice • quantity = 8 f • productDemands • new ProductDemandRecord() • mallId = m.id • demand = 0 • maximumDemand = 8 • optimalStockLevel = 0

The rule *Create_InitialUnemployedWorkerAsJobSeeker_Rule* is also used when the exogenous event *Init* occurs. A variable *h* of a *Household* type is declared. The *jobSeeker* property of the variable *h* is set true only when the value of the *firm* property of the variable *h* is equal to 100.

Table 4.23. Environment Rule: Create_InitialUnemployedWorkerAsJobSeeker_Rule

Triggering Event	Init
Declaration	Household h
Condition	h.firm == 100
Resulting Messages	-
State Effects	h

	• jobSeeker = true

The rule *Create_InitialWorkersInFirm_Rule* is also used when the exogenous event *Init* occurs. A variable *f* of a *ConsumptionGoodsProducer* type and a variable *h* of a *Household* type are declared. After that, the new record will be added to the bottom of the empty list *workersInFirm* of the variable *f* only when the "identity" of the variable *f* is equal to the *firm* property of the variable *h*.

Table 4.24. Environment Rule: Create_InitialWorkersInFirm_Rule

Triggering Event	Init
Declaration	ConsumptionGoodsProducer f Household h
Condition	f.id == h.firm
Resulting Messages	-
State Effects	f • workersInFirm • new WorkerInFirm() • householdId = h.id • wage = h.currentWage • generalSkillLevel = h.generalSkillLevel • specificSkillLevel = h.specificSkillLevel

The rule *AtStartOfMonthDetermineEmployedWorkerAsJobSeeker_Rule* applies when an exogenous event *StartOfMonth* occurs. This event simulates at the beginning of every month (occurrenceTime="2") and occurs at every 24 steps during the running of the simulation. The household decides whether to search on the job or not. A variable *h* of a *Household* type is declared. The *jobSeeker* property of the variable *h* is set true, when the value of the *firm* property of the variable *h* is not equal to 100 and the global function returns true.

Table 4.25. Environment Rule: AtStartOfMonthDetermineEmployedWorkerAsJobSeeker_Rule

Triggering Event	StartOfMonth
Declaration	Household h
Condition	h.firm != 100 and Global.wouldBeJobSeeker()
Resulting Messages	-
State Effects	h • jobSeeker = true

The rule *AtWeeklyIndividualConsumption_Rule* is used when an exogenous event *AtWeeklyIndividualConsumption* occurs. This event triggers the consumption activities of households (occurrenceTime="3"). It is repeated every 6 steps, namely on a weekly basis. A variable *m* of a *Mall* type and a variable *h* of a *Household* type are declared. For this rule to take place, a condition must be satisfied. The "identity" of the variable *m* must be equal to the *region* property of the variable *h*. Then the *consumptionDecision* function of the variable *h* is called. It is used to choose products from the *productsInStock* list of the variable *m* according to the algorithm of choice of consumption goods of the agent *household*. This rule may result in creating twice messages of type *TellSalesRevenue*. First, a message is sent to an agent whose "identity" is equal to the value of the *selectFirstProductId* property of the variable *h* by the variable *m*, when the *selectFirstProductId* property is not equal to 0. Second, the variable *m* needs to send a message to an agent whose "identity" is equal to the *selectSecondProductId* property of the variable *h* only when the *selectSecondProductId* property is not equal to 0.

Table 4.26. Environment Rule: AtWeeklyIndividualConsumption_Rule

Triggering Event	AtWeeklyIndividualConsumption
Declaration	Mall m Household h
Condition	m.id == h.region
Resulting Messages	TellSalesRevenue • h.selectFirstProductId != 0 • senderIdRef = m.id • receiverIdRef = h.selectFirstProductId • revenue = h.spendBudgetForFirstProduct • quantity = h.purchaseQuantityForFirstProduct TellSalesRevenue • h.selectSecondProductId != 0 • senderIdRef = m.id • receiverIdRef = h.selectSecondProductId • revenue = h.spendBudgetForSecondProduct • quantity = h.purchaseQuantityForSecondProduct
State Effects	h • consumptionDecision(productsCollection) • productsCollection = m.productsInStock

The rule *ConsumptionGoodsProducerBuyNewInvestment_Rule* is used when an action event *BuyNewInvestment* is perceived. Two variables are declared. One is the variable *i* of a *CapitalGoodsProducer* type. The other is the actor of the event: the variable *f* of a *ConsumptionGoodsProducer* type. The states of the two variables will change. The *investmentSalesRevenue* property of the variable *i* is increased by the *newInvestment* property of the variable *f*. Then the *grossInvestment*, *physicalCapitalStock*, *totalQualityOfCapitalStock* and *averageQualityOfCapitalStock* property of the variable *f* will be updated.

Table 4.27. Environment Rule: ConsumptionGoodsProducerBuyNewInvestment_Rule

Triggering Event	BuyNewInvestment
Declaration	CapitalGoodsProducer i • objectIdRef = 11 ConsumptionGoodsProducer f • objectRef = BuyNewInvestment.actor

Condition	-
Resulting Messages	-
State Effects	i • investmentSalesRevenue = i.investmentSalesRevenue + f.newInvestment f • grossInvestment = f.grossInvestment + f.newInvestment • physicalCapitalStock = f.physicalCapitalStock + f.newInvestment / Global.investmentSalesPrice • totalQualityOfCapitalStock = f.totalQualityOfCapitalStock + f.newInvestment / Global.investmentSalesPrice * Global.qualityOfInvestment • averageQualityOfCapitalStock = f.totalQualityOfCapitalStock / f.physicalCapitalStock

The rule *ConsumptionGoodsProducerDismissWorker_Rule* applies when an action event *DismissWorker* is perceived by a declared variable *f* of a *ConsumptionGoodsProducer* type. Afterwards, a variable *w* of a *WorkerInFirm* type which denotes the *dismissalsList* list of the variable *f* is also declared. After that, the *laborSupplyQuantity* property of the variable *f* is increased by one and the *deleteWorkerRecordFromList* function is called to remove a record from the *workersInFirm* list. This rule only results in creating a new *TellDismissal* message.

Table 4.28. Environment Rule: ConsumptionGoodsProducerDismissWorker_Rule

Triggering Event	DismissWorker
Declaration	ConsumptionGoodsProducer f • objectRef = DismissWorker.actor WorkerInFirm w : f.dismissalsList
Condition	-
Resulting Messages	TellDismissal • senderIdRef = DismissWorker.actorIdRef • receiverIdRef = w.householdId
State Effects	f • laborSupplyQuantity = f.laborSupplyQuantity + 1 • deleteWorkerRecordFromList(householdId) • householdId = w.householdId

The rule *ConsumptionGoodsProducerPostVacancyInformation_Rule* is used when an action event *PostVacancyInformation* is perceived. A variable *h* of a *Household* type is declared. To make this rule effective, several conditions must be satisfied simultaneously, i.e. the *firmId* property of the event is not equal to the *firm* property of the variable *h*, the *wageOffer* property of the event is greater than or equal to the *currentWage* property of the variable *h* and the *jobSeeker* property of the variable *h* returns true. This rule only results in creating a new *TellVacancy* message.

Table 4.29. Environment Rule: ConsumptionGoodsProducerPostVacancyInformation_Rule

Triggering Event	PostVacancyInformation
Declaration	Household h
Condition	PostVacancyInformation.firmId != h.firm and PostVacancyInformation.wageOffer >= h.currentWage and h.jobSeeker = true
Resulting Messages	TellVacancy • senderIdRef = h.id • receiverIdRef = PostVacancyInformation.actorIdRef • generalSkillLevel = h.generalSkillLevel • specificSkillLevel = h.specificSkillLevel
State Effects	-

The rule *InFirstIterationConsumptionGoodsProducerOfferJob_Rule* applies when an action event *InFirstIterationOfferJob* is perceived by a declared variable *f* of a *ConsumptionGoodsProducer* type. Then a variable *a* of a *JobApplicationRecord* type which denotes the *jobApplications* list of the variable *f* is also declared. This rule only results in creating a new *JobOffer* message.

Table 4.30. Environment Rule: InFirstIterationConsumptionGoodsProducerOfferJob_Rule

Triggering Event	InFirstIterationOfferJob
Declaration	ConsumptionGoodsProducer f • objectRef = InFirstIterationOfferJob.actor JobApplicationRecord a : f.jobApplications
Condition	-
Resulting Messages	JobOffer • senderIdRef = InFirstIterationOfferJob.actorIdRef • receiverIdRef = a.householdId • wageOffer = f.wageOffer * f.determineAverageSpecificSkillLevel() • region = f.region
State Effects	-

The rule *EndFirstIterationConsumptionGoodsProducerClearJobApplications_Rule* applies when an exogenous event *EndFirstIterationLaborSupply* occurs. This event occurs when the first round of hiring activity ends (occurrenceTime="14"). It is repeated every 24 steps, namely on a monthly basis. A variable *f* of a *ConsumptionGoodsProducer* type is declared. Then the *clearJobApplications* function of the variable *f* is called to empty the *jobApplications* list, when the size of this list is greater than zero.

Table 4.31. Environment Rule: EndFirstIterationConsumptionGoodsProducerClearJobApplications_Rule

Triggering Event	EndFirstIterationLaborSupply

Declaration	ConsumptionGoodsProducer f
Condition	-
Resulting Messages	-
State Effects	f • clearJobApplications()

The rule *EndFirstIterationConsumptionGoodsProducerRaiseWageOffer_Rule* applies when an exogenous event *EndFirstIterationLaborSupply* occurs. A variable *f* of a *ConsumptionGoodsProducer* type is declared. Then the *wageOffer* property of the variable *f* will increase by 2%, if the *laborSupplyQuantity* property of the variable *f* is greater than five.

Table 4.32. Environment Rule: EndFirstIterationConsumptionGoodsProducerRaiseWageOffer_Rule

Triggering Event	EndFirstIterationLaborSupply
Declaration	ConsumptionGoodsProducer f
Condition	f.laborSupplyQuantity > 5
Resulting Messages	-
State Effects	f • wageOffer = (1 + 0.02) * f.wageOffer

The rule *EndFirstIterationJobSeekerReduceReservationWage_Rule* is used when an exogenous event *EndFirstIterationLaborSupply* occurs. A variable *h* of a *Household* type is declared. This rule applies only when the *firm* property of the variable *h* is equal to 100 and the *currentWage* property after decreasing by 2% is greater than or equal to one. Then the *currentWage* property of the variable *h* is reduced by two percent.

Table 4.33. Environment Rule: EndFirstIterationJobSeekerReduceReservationWage_Rule

Triggering Event	EndFirstIterationLaborSupply
Declaration	Household h
Condition	h.firm == 100 and (1 - 0.02) * h.currentWage >= 1
Resulting Messages	-
State Effects	h • currentWage = (1 - 0.02) * h.currentWage

The rule *EndSecondIterationConsumptionGoodsProducerClearJobApplications_Rule* applies when an exogenous event *EndSecondIterationLaborSupply* occurs. This event happens at the end of the second iteration of hiring activity (occurrenceTime="23"). It is also repeated every 24 steps. The content of this rule is the same as that of the *EndFirstIterationConsumptionGoodsProducerClearJobApplications_Rule* rule.

Table 4.34. Environment Rule: EndSecondIterationConsumptionGoodsProducerClearJobApplications_Rule

Triggering Event	EndSecondIterationLaborSupply
Declaration	ConsumptionGoodsProducer f
Condition	f.jobApplications.size > 0
Resulting Messages	-

State Effects	f • clearJobApplications()

The rule *ConsumptionGoodsProducerDistributeProduct_Rule* is used when an action event *DistributeProduct* is perceived by a declared variable *f* of a *ConsumptionGoodsProducer* type. After the production the output is distributed among the malls. Then a variable *o* of a *InventoryPosition* type which denotes the *inventoryPositions* list of the variable *f* is also declared. This rule only results in creating a new *DeliverProduct* message. The message containe the delivery volume for an individual mall.

Table 4.35. Environment Rule: ConsumptionGoodsProducerDistributeProduct_Rule

Triggering Event	DistributeProduct
Declaration	ConsumptionGoodsProducer f • objectRef = DistributeProduct.actor InventoryPosition o : f.inventoryPositions
Condition	-
Resulting Messages	DeliverProduct • senderIdRef = DistributeProduct.actorIdRef • receiverIdRef = o.mallId • quantity = o.adjustmentFactor * f.producedQuantity
State Effects	-

The rule *ConsumptionGoodsProducerPayWage_Rule* is used when an action event *PayWage* is perceived by a declared variable *f* of a *ConsumptionGoodsProducer* type. Then a variable *w* of a *WorkerInFirm* type which denotes the *workersInFirm* list of the variable *f* is also declared. This rule only results in creating a new *TellWage* message.

Table 4.36. Environment Rule: ConsumptionGoodsProducerPayWage_Rule

Triggering Event	PayWage
Declaration	ConsumptionGoodsProducer f • objectRef = PayWage.actor WorkerInFirm w : f.workersInFirm
Condition	-
Resulting Messages	TellWage • senderIdRef = PayWage.actorIdRef • receiverIdRef = w.householdId • wage = w.wage • region = f.region
State Effects	-

The rule *ConsumptionGoodsProducerIncreaseSpecificSkillLevel_Rule* is used when an action event *IncreaseSpecificSkillLevel* is perceived by a declared variable *f* of a *ConsumptionGoodsProducer* type.

Then a variable *w* of a *WorkerInFirm* type which denotes the *workersInFirm* list of the variable *f* is also declared. After that, the *updateSpecificSkillLevel* function is called. This rule only results in creating a new *TellSpecificSkillLevel* message.

Table 4.37. Environment Rule: ConsumptionGoodsProducerIncreaseSpecificSkillLevel_Rule

Triggering Event	IncreaseSpecificSkillLevel
Declaration	ConsumptionGoodsProducer f • objectRef = IncreaseSpecificSkillLevel.actor WorkerInFirm w : f.workersInFirm
Condition	-
Resulting Messages	TellSpecificSkillLevel • senderIdRef = IncreaseSpecificSkillLevel.actorIdRef • receiverIdRef = w.householdId • specificSkillLevel = w.specificSkillLevel
State Effects	f • updateSpecificSkillLevel(householdId, specificSkillLevel) • householdId = w.householdId • specificSkillLevel = w.specificSkillLevel + (1 - Math.pow((0.5), (1 / (20 + 0.25 * (w.generalSkillLevel - 1) * (4 - 20))))) * (f.averageQualityOfCapitalStock - w.specificSkillLevel)

The rule *EndOfMonth_Rule* applies when an exogenous event *EndOfMonth* occurs. This event simulates at the end of every month (occurrenceTime="25") and occurs at every 24 steps in the simulation. This rule is used to update some values of global variables at the end of each period.

Table 4.38. Environment Rule: EndOfMonth_Rule

Triggering Event	EndOfMonth
Declaration	-
Condition	-
Resulting Messages	-
State Effects	• Global.period = Global.period + 1 • Global.innovationProbability = Global.wouldInnovate() • Global.qualityOfInvestment = (1 + Global.innovationProbability) * Global.qualityOfInvestment • Global.investmentSalesPrice = (1 + Global.innovationProbability) * Global.investmentSalesPrice

The rule *AtEndOfMonthConsumptionGoodsProducerClearInventoryPositions_Rule* applies when an exogenous event *EndOfMonth* occurs. A variable *f* of a *ConsumptionGoodsProducer* type is declared. Then the *grossInvestment* property of the variable *f* is reduced by one percent. After that, three functions are called.

Table 4.39. Environment Rule: AtEndOfMonthConsumptionGoodsProducerClearInventoryPositions_Rule

Triggering Event	EndOfMonth
Declaration	ConsumptionGoodsProducer f
Condition	-
Resulting Messages	-
State Effects	f • grossInvestment = (1 - 0.01) * f.grossInvestment • updateOptimalStockLevel() • clearInventoryPositions() • updateLastFourProducedQuantities()

The rule *AtEndOfMonthPayWage_Rule* applies when an exogenous event *EndOfMonth* occurs. A variable *h* of a *Household* type is declared. Then the *currentIncome* property of the variable *h* is increased by one, when the *firm* property of the variable *h* is equal to 100.

Table 4.40. Environment Rule: AtEndOfMonthPayWage_Rule

Triggering Event	EndOfMonth
Declaration	Household h
Condition	h.firm == 100
Resulting Messages	-
State Effects	h • currentIncome = h.currentIncome + 1

The rule *AtEndOfMonthDetermineEmployedWorkerNotAsJobSeeker_Rule* applies when an exogenous event *EndOfMonth* occurs. A variable *h* of a *Household* type is declared. Then the *jobSeeker* property of the variable *h* is set true, when the *firm* property of the variable *h* is not equal to 100 and the value of the *jobSeeker* property returns true.

Table 4.41. Environment Rule: AtEndOfMonthDetermineEmployedWorkerNotAsJobSeeker_Rule

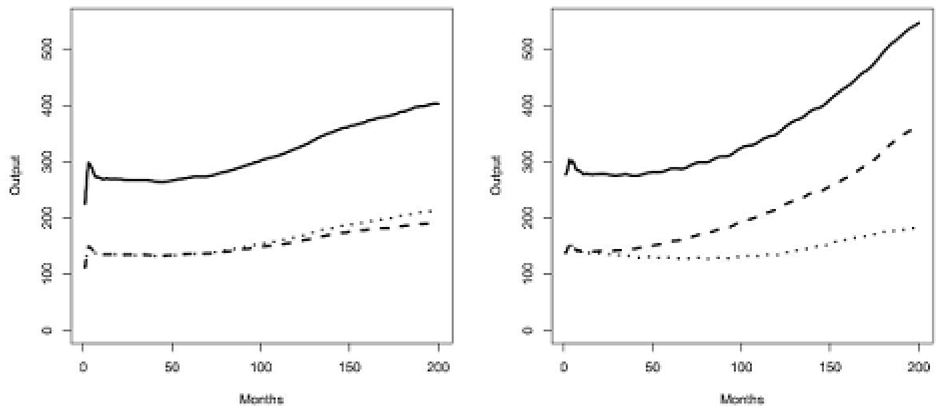
Triggering Event	EndOfMonth
Declaration	Household h
Condition	h.firm != 100 and h.jobSeeker = true
Resulting Messages	-
State Effects	h • jobSeeker = false

Chapter 5. Simulation results

This chapter shows the simulation results with the help of graphical statistics which appear at simulation runtime on the panel in AOR simulator. In the simulation presented here there are six scenarios, each of which corresponds to one set of simulation results. In order to confirm that the model has been reconstructed successfully, the results will be compared with those of the original model. But it is worth note that because of some deficiency of the original model, for example, there exist some fuzzy rules, these uncertain factors will cause that the reconstructed results can deviate from the actual results of the original model.

In the paper of the original model, the low-high scenario is taken as an example to publish the simulation results. In view of this situation, the reconstructed results for that scenario even taking into account costs from commuting are also shown.

Figure 5.1. Runs for zero (left panel) and low (0.05) (right panel) commuting costs; total outputs (solid line), output in the low skill region (dotted line), output in the high skill region (dashed line)



In the low-high scenario, in case of no commuting costs both regions produce about the same amount. Whereas with low (0.05) commuting costs the output of the high skill region is larger than that of the low skill region. Clearly, CGPs in the high skill region take advantage of the more skilled workforce which could lead to increased production to a larger degree than CGPs in the low region.

Figure 5.2. Output from the AOR model

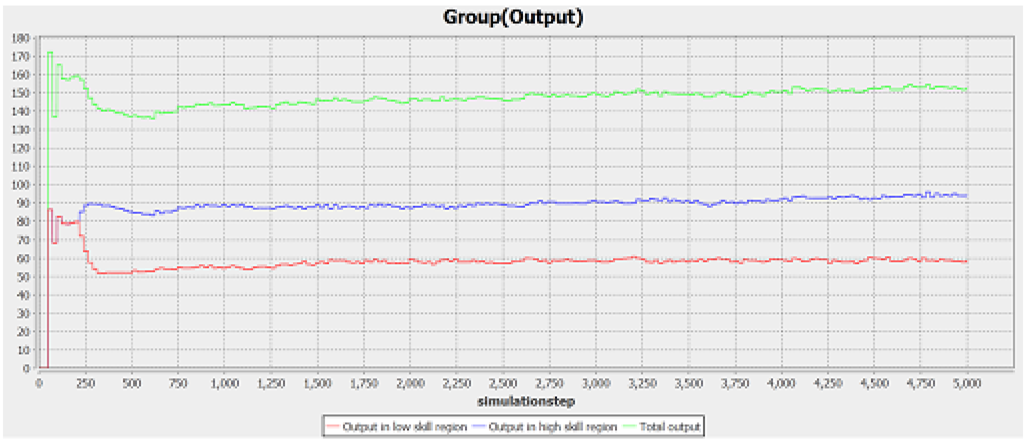
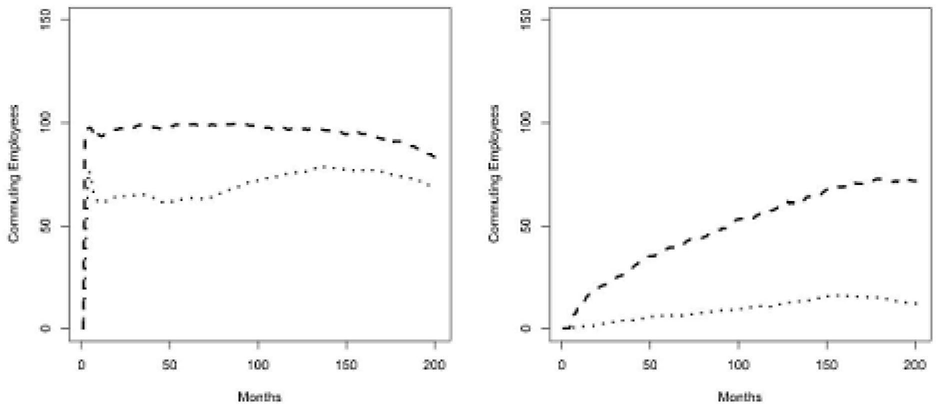


Figure 5.3. Runs for zero (left panel) and low (0.05) (right panel) commuting costs; number of commuters from low to high skill region (dotted line), number of commuters from high to low skill region (dashed line)



The both panels give insight into a large number of high skill workers from the high skill region commute to the low skill region. In case of no commuting costs there is a large number of commuters from the low skill to the high skill region. In the opposite case, the flow from the low skill to the high skill region becomes very small.

Figure 5.4. Number of commuters from the AOR model

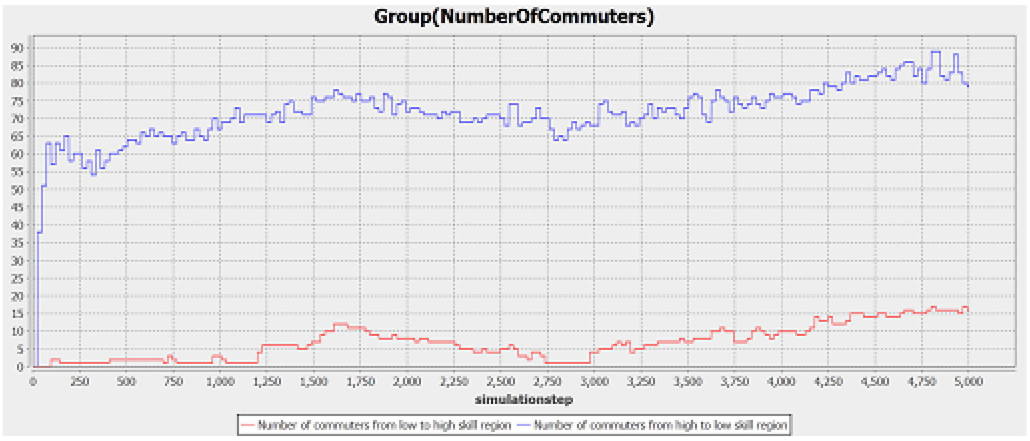
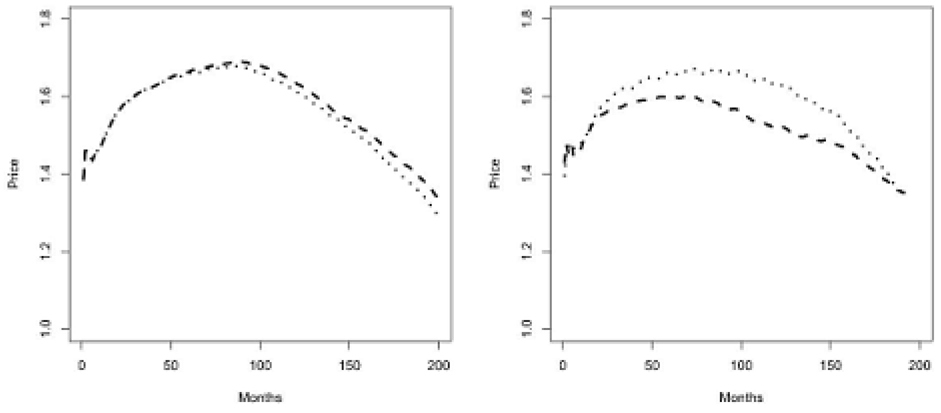


Figure 5.5. Runs for zero (left panel) and low (0.05) (right panel) commuting costs; prices in the low skill region (dashed line), prices in the high skill region (dotted line)



In the low-high scenario, there is not much difference in price between the goods produced in the two regions.

Figure 5.6. Prices from the AOR model

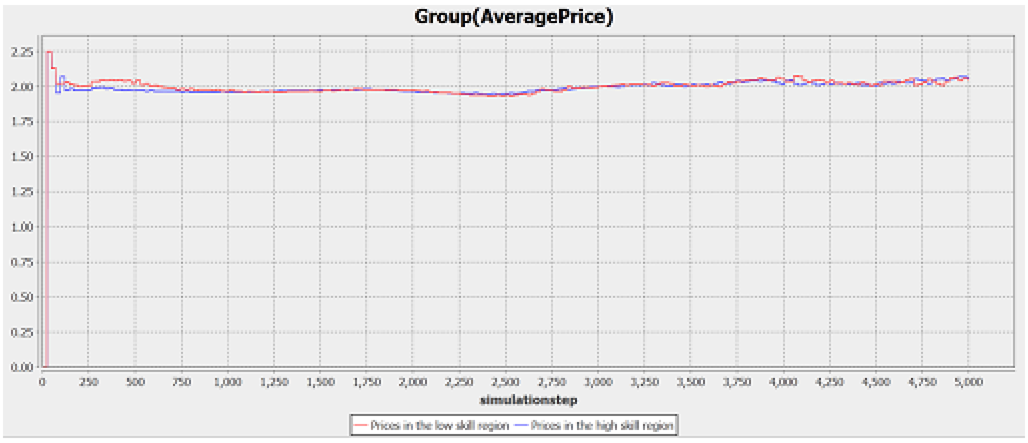
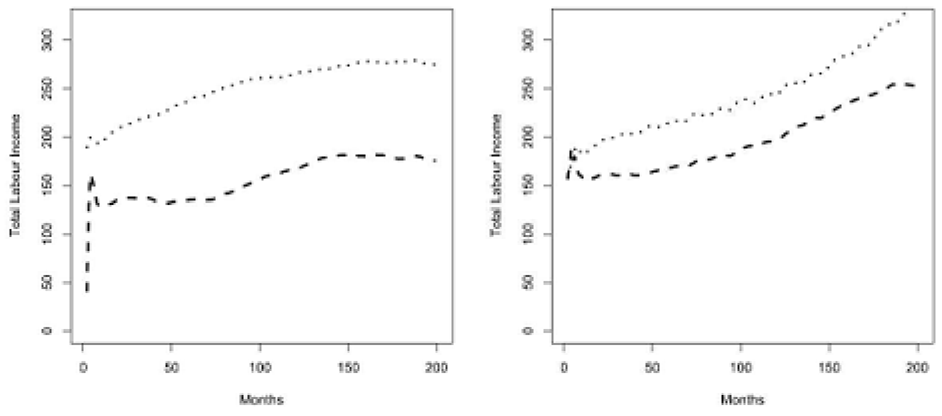
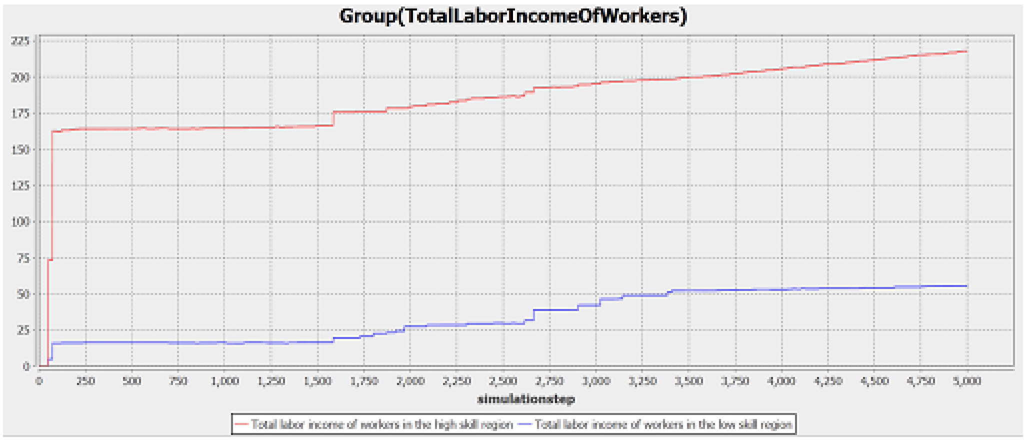


Figure 5.7. Runs for zero (left panel) and low (0.05) (right panel) commuting costs; total labor income of workers in the low skill region (dashed line) and the high skill region (dotted line)



In both scenarios the total labor income of workers is larger in the high skill region than in the low skill region. In case of low (0.05) commuting costs the difference is smaller.

Figure 5.8. Total labor income of workers from the AOR model



Chapter 6. Conclusion

In this paper an agent-based spatial macroeconomic model is chosen for reconstruction. There are two special results of this thesis: First, although the original model is very complex and contains too many varibales and parameters, it can be reconstructed very well with the help of the AOR simulation technology. Second, and most important, AOR simulation is a powerful approach regarding the study and the development of multi-agent models. In fact, this approach has many advantages. It uses a well defined metamodel and simulation language AORSL that can be executed directly with special simulation software AOR-JavaSim. This language is based on XML, so it is very easy to write and read.

Appendix A. ComplexDataType

A.1. ComplexDataTypes

This tag is used to define the new complex data type. It can have attributes. These complex data types can be used in the same way as the standard data types.

- **ProductInStock:** Represents a product, which is offered by a CGP and stored at malls. It has some attributes as follows.

  - **firmId** type: Integer

    description: Holds the "identity" of the CGP agent, which provides the product.

  - **price** type: Float

    description: Holds the price of the product.

  - **quantity** type: Float

    description: Holds the amount of the product, which is sold at malls.

- **ProductListItem:** Represents a product, which can be sampled by consumers. The attributes can be summarized as follows.

  - **firmId** type: Integer

    description: Holds the "identity" of the CGP agent, which provides the product.

  - **consumptionValue** type: Float

    description: Holds the value of the product depending on the price of the product.

  - **selectionProbability** type: Float

    description: The consumer decides which product to buy with the selection probability that depends solely on the price.

- **ProductDemandRecord:** Represents the market demand of a product in a period. The attributes are given below.

  - **mallId** type: Integer

    description: Holds the "identity" of the mall agent.

  - **demand** type: Float

    description: Holds the sales volume of the product.

  - **maximumDemand** type: Float

    description: Holds the maximum volume of sales in all periods.

  - **optimalStockLevel** type: Float

    description: Holds an optimum order point for each mall.

- **InventoryPosition:** Represents the desired replenishment quantity for a mall. The attributes are as follows.

    - **mallId** type: Integer

      description: Holds the "identity" of the mall agent.

    - **quantity** type: Float

      description: Holds the desired replenishment quantity of the product for the mall.

    - **adjustmentFactor** type: Float

      description: Holds the ration for each mall.

- **WorkerInFirm:** Represents a household, who is employed. It has some attributes as follows.

    - **householdId** type: Integer

      description: Holds the "identity" of the household agent.

    - **wage** type: Float

      description: Holds the wage of the worker.

    - **generalSkillLevel** type: Integer

      description: Holds the general skill level of the worker.

    - **specificSkillLevel** type: Float

      description: Holds the specific skill level of the worker.

- **JobApplicationRecord:** Represents a job applicant, who needs to find a suitable position. The attributes can be summarized as follows.

    - **householdId** type: Integer

      description: Holds the "identity" of the household agent.

    - **generalSkillLevel** type: Integer

      description: Holds the general skill level of the applicant.

    - **specificSkillLevel** type: Float

      description: Holds the specific skill level of the applicant.

- **JobOfferRecord:** Represents a job offer, which is sent to qualified applicants. The attributes are given below.

    - **firmId** type: Integer

      description: Holds the "identity" of the CGP agent.

    - **wageOffer** type: Float

      description: The CGP posts vacancies including the wage offer.

    - **netWageOffer** type: Float

      description: The commuting costs are deducted from the wage offer.

A.2. MessageTypes

Messages are used to communicate between the agents. Every message structure is difined seperately. There are ten message types, two of which have no attributes.

- **TellCurrentStockLevel:** Sent by malls to tell the CGP about the current stock level. This message has one attribute including quantity, which holds the amount of the product.
- **TellSalesRevenue:** Sent by malls to tell the CGP about the sales situation. This message has two attributes including revenue which holds the sales revenue, and quantity that holds the sales volume.
- **TellDismissal:** Sent by CGPs. The household checks whether is fired or not. This message has no attributes.
- **TellVacancy:** Sent by households. The CGP receives applications for vacancies. This message has two attributes which hold the general skill level and the specific skill level of the applicant.
- **JobOffer:** Sent by CGPs to qualified applicants. This message has two attributes which store the offered wage and region where the CGP is located.
- **AcceptJob:** Sent by households to accept the positions. This message has three attributes including wage, generalSkillLevel and specificSkillLevel.
- **ResignJob:** Sent by households. Because the household is offered a better job, he resigned his position. This message has no attributes.
- **DeliverProduct:** Sent by CGPs to deliver the ordered products to the mall. This message has one attribute containing quantity, that holds the delivery quantity.
- **TellWage:** Sent by CGPs. Wage is paid to the worker. This message has two attributes which store the monthly wage that an individual worker earns in the current production cycle and region where the CGP is located.
- **TellSpecificSkillLevel:** Sent by CGPs. The household checks whether his specific skill level needs to be updated or not. This message has one attribute including specificSkillLevel which holds the adjusted specific skill level.

A.3. ActionEventTypes

Action events are used by agents, which want to perform some actions. There are ten action event types, six of which have no attributes.

- **BuyNewInvestment:** If additional investments are needed the CGP performs the action of purchasing new investment goods from the IGP. This event has no attributes.
- **DismissWorker:** If the CGP wants to decrease the incumbent workforce it performs the action of downsizing. This event has no attributes.
- **PostVacancyInformation:** If additional workers are needed the CGP performs the action of posting vacancies. This event has two attributes including firmId which holds the "identity" of the CGP agent, and wageOffer that holds the offered wage for vacancies.
- **InFirstIterationOfferJob:** After ranking the applicants, the CGP performs the action of offering the positions. This event has no attributes.
- **DistributeProduct:** After production, the CGP performs the action of distributing the products. This event has no attributes.
- **PayWage:** When the CGP updates its labor force, it performs the action of paying wages. This event has no attributes.

- **IncreaseSpecificSkillLevel:** The CGP performs the action of adjusting the specific skills of its employees. This event has no attributes.
- **PayEqualShare:** The IGP performs the action of distributing its revenue to all households. This event has one attribute including share, which holds equal shares of the revenue.
- **PayDividend:** The CGP performs the action of paying dividends to all households. This event has one attribute containing dividend, that holds equal shares of the dividends.
- **SetNewPrice:** Once the CGP needs to adjust the price of its product, it performs the action of updating the price. This event has one attribute which records the new price.

Glossary

Glossar

EURACE	EUR-ACE European Agent-Based Economics