Project no.
035086
Project acronym
EURACE
Project title

An Agent-Based software platform for European economic policy design with heterogeneous interacting agents: new insights from a bottom up approach to economic modelling and simulation

Instrument: STREP

Thematic Priority: IST FET PROACTIVE INITIATIVE "SIMULATING EMERGENT PROPERTIES IN COMPLEX SYSTEMS

D5.3: Software module for the agent based models of goods, labour and credit markets
Due date of deliverable:
31/08/2008
Actual submission date:

Start date of project: September 1st 2006 Duration: 36 months

Organisation name of lead contractor for this deliverable
University of Sheffield - USFD

Revision 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contents

1 Introduction **1**

2 General Model Implementation **1**

3 Goods and Labour Market Implementation **2**

3.1 Description . . . 2

3.1.1 Firm Agent in the Labour and Goods Market . . . 2

3.1.2 Household Agent in the Labour and Goods Market . . . 5

3.2 Messages being used . . . 7

4 Credit Market Implementation **7**

4.1 Description . . . 7

4.1.1 Firm Agent in the Credit Market . . . 7

4.1.2 Bank Agent in the Credit Market . . . 11

4.2 Messages being Used . . . 13

4.3 Implementation Results . . . 13

5 Testing **14**

5.1 Unit Testing . . . 14

5.2 Integration Testing . . . 15

Abstract

This report presents the deliverable D5.3 accounting for the software descriptions of the models for the markets - goods and labour and credit markets. This deliverable acts as part of the work package 5 which comprises of agent based computational models of goods, labour and credit markets required for the project EURACE. Also included are sections on general model implementation, as a lead up to the economic models, and model testing strategies, to make sure the models are valid.

1 Introduction

This document contains definitions of the labour and goods market isolated model and the credit market isolated model. The definitions are not how the models work but how they are designed using extended state machines (X-machines) and how this fits the implementation using FLAME. Section 2 describes the model definition changes of adding states between functions rather than giving order to functions via dependencies. Section 5 describes strategies for testing models that include the current use of unit testing (testing individual functions) and future ideas for integration testing (testing groups of functions). Also shown are the use of tools to create test sets for integration testing and finding assertions on agent variables after simulations have been executed.

2 General Model Implementation

Models are constructed from agents that have functions. These functions must be executed in a correct order for the model to run correctly. Formerly this order was defined in the model definition XMML by dependencies between functions. These dependencies could be either internal, within individual agents, or communication, dependent on messages from other agents. This was enough information to be able to order functions and plan communication synchronisation points to work in parallel.

The economic models eventually started to run only certain functions at certain times, for example, weekly or monthly. Because each function was still being executed this required a condition at the start of every function and soon became a hindrance. Even though a series of functions used the same time series they all had to include the same condition. This was also true of other conditions, for example only households that were unemployed need execute functions involved with the labour market.

This was solved by defining the model in the XMML as a state machine. Where states are defined before and after functions. Each state can have many incoming functions and many outgoing functions. Only the unique start state has no incoming functions, and end states have no outgoing functions. This then allows branches of functions where the condition for all the functions can be defined at the start of the branch. An example of this is shown by the similar models in Figures 1 and 2.

The model can now be recognised as a state machine but with the restriction that once a state has been entered by an agent it cannot reenter the same state. This provides synchronisation of agents in parallel during execution. Also input to functions are sets of inputs (messages) which can be empty. This is the level of detail required by FLAME to plan the communication synchronisation in parallel.

Each function can then be defined by the parameters as shown in Table 1, where M_{pre} is the pre-condition of the memory and M_{post} is the post-condition of the memory.

Current State	Input	M_{pre}	Function	M_{post}	Output	Next State

Table 1: Function parameters

In XMML this is currently written as below where M_{pre} is defined as *condition* and M_{post} is written as source code within a function with the same name as the function name.

```
<function>
 <name>Function_name</name>
 <description>A description of the function</description>
 <currentState>current_state</currentState>
```

```
  <nextState>next_state</nextState>
  <condition>condition</condition>
  <inputs>inputs</inputs>
  <outputs>outputs</outputs>
</function>
```

3 Goods and Labour Market Implementation

The goods and labour market is based on the model presented by the Bielefeld unit. Detailed description of the model can be found in D5.1. Here is a brief overview of the implementation of the model.

3.1 Description

In order to parse the model and test it for results the labour and goods market has to be integrated with various models of the credit, government and financial management markets. Here we concentrate on the essence of the individual functions involved with the labour and goods market only. The two main agents involved in this process are namely - Firms and Households. The state graph in Figure 3 describes how these functions are interacting with each other.

3.1.1 Firm Agent in the Labour and Goods Market

See Table 2.

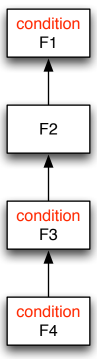

Figure 1: Dependency graph model

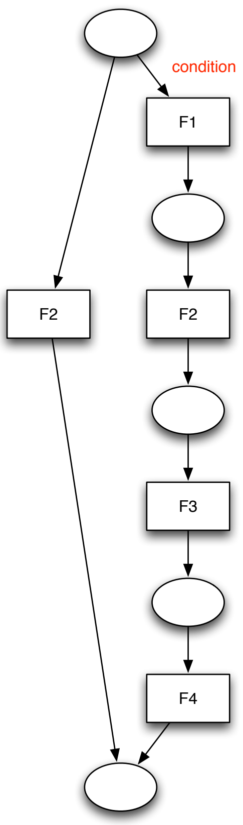

Figure 2: State graph model

Table 2: State Transition Table of the Firm involved in Labour Market.

Function Name	Current State	Next State	Condition	Inputs	Outputs
Firm _calculate _specific _skills _and _wage _offer	Start _Firm _Labour _Role	011			
Firm _send _vacancies	011	0	a.no_employees LT a.employees_needed		vacancies
Firm _send _redundancies	011	03c	a.no_employees GT a.employees_needed		firing
Firm _idle	011	03c	a.no_employees EQ a.employees_needed		
Firm _read _job _applications _send _job _offer _or _rejection	03	04		job _application	job_offer, application_rejection
Firm _read _job _responses	04	05a		job_acceptance	
Firm _read _job _quitting	05a	05b		quitting	
Firm _read _job _quitting	00b	09c		quitting	
Firm _read _job _quitting	03c	03d		quitting	
Firm _start _labour _market	03d	06	a.no_employees LT a.employees_needed		
Firm _finish _labour _market _first _round	03d	09a	a.no_employees LT a.employees_needed		
Firm _finish _labour _market _first _round	05b	09a	a.no_employees EQ a.employees_needed		
Firm _update _wage _offer	05b	06	a.no_employees LT a.employees_needed		
Firm _send _vacancies _2	06	07			vacancies2
Firm _read _job _applications _send _job _offer _or _rejection _2	07	08		job_application2	job_offer2, application_rejection2
Firm _read _job _responses _2	08	09a		job_acceptance2	
Firm _read _job _quitting _2	09a	09b		quitting2	
Firm _read _job _quitting _2	09c	Start_Firm_Seller_Role		quitting2	
Firm _update _wage _offer _2	09b	10	a.no_employees LT a.employees_needed		
Firm _idle	09b	10	a.no_employees LT a.employees_needed		
Firm _compute _mean _wage _specific _skills	10	End_Firm_Labour_Role			

The description of the functions is stated below:

Firm_send_vacancies. If additional workers are needed the firm sends vacancies messages especially the different wage offers for the different general skill groups.

Firm_send_redundancies. If the firm wants to decrease the workforce it sends redundancies.

Firm_idle. Firm does nothing.

Firm_read_job_applications_send_job_offer_or_rejection. Firm reads the application, ranks the applicants according to their general and specific skills and sends as many job offers to the first ranked applicants as the firm has vacancies to fill. The other applicants are refused.

Firm_read_job_responses. The firm reads the responses to their job offers and updates the number of employees and the number of vacancies.

Firm_read_job_quitting. The firm reads quitting messages and updates the number of employees and the number of vacancies.

Firm_start_labour_market. Dummy function if a firm has to enter the Labour Market in the second round after receiving quitting.

Firm_update_wage_offer. The firm increases the wage offer if there are vacancies left.

Firm_send_vacancies_2. If additional workers are needed the firm sends vacancies messages especially the different wage offers for the different general skill groups.

Firm_read_job_applications_send_job_offer_or_rejection_2. Firm reads the application, ranks the applicants according to their general and specific skills and sends as many job offers to the first ranked applicants as the firm has vacancies to fill. The other applicants are refused.

Firm_read_job_responses_2. The firm reads the responses to their job offers and updates the number of employees and the number of vacancies.

Firm_read_job_quitting_2. The firm reads quitting messages and updates the number of employees and the number of vacancies.

Firm_update_wage_offer_2. The firm increases the wage offer if there are vacancies left.

3.1.2 Household Agent in the Labour and Goods Market

See Table 3.

Table 3: State Transition Table of the Household involved in Labour Market.

Function Name	Current State	Next State	Condition	Inputs	Outputs
Household _read _firing _messages	Start _Household _Labour _Role	01d	a.employee _firm _id NEQ -1	firing	
Household _idle	01d	01a	a.employee _firm _id EQ -1		
Household _idle	Start _Household _Labour _Role	01a	a.employee _firm _id EQ -1		
Household _UNEMPLOYED _read _job _vacancies _and _send _applications	01a	01		vacancies	job _application
Household _on _the _job _search _decision	01d	01b	a.employee _firm _id NEQ -1		
Household _OTJS _read _job _vacancies _and _send _applications	01b	01	a.on_the_job_search EQ 1	vacancies	job_application
Household _idle	01b	06	a.on_the_job_search NEQ 1		
Household _read _job _offers _send _response	01	02		job_offer	quitting, job_acceptance
Household _finish _labour _market	02	06	(a.employee_firm_id NEQ -1) AND (a.on_the_job_search NEQ 1)		
Household _read _application _rejection _update _wage _reservation	02	03	a.employee_firm_id EQ -1	application _rejection	
Household _OTJS _read _job _vacancies _and _send _applications _2	02	04	a.on_the_job_search EQ 1	vacancies2	job_application2
Household _UNEMPLOYED _read _job _vacancies _and _send _applications _2	03	04		vacancies2	job_application2
Household _read _job _offers _send _response _2	04	05		job_offer2	job_acceptance2, quitting2
Household _read _application _rejection _update _wage _reservation _2	05	06	a.employee_firm_id EQ -1	application _rejection2	
Household _idle	05	06	a.employee_firm_id NEQ -1		

Household_read_firing_messages. The household checks whether is is fired or not.

Household_idle. Household does nothing.

Household_UNEMPLOYED_read_job_vacancies_and_send_applications. Household reads vacancies messages and sends applications.

Household_on_the_job_search_decision. Household decides whether to search on the job or not.

Household_OTJS_read_job_vacancies_and_send_applications. Household searches on the job. Reads vacancies messages and sends applications.

Household_read_job_offers_send_response. Household reads the job offers and ranks them according to the wage offer.

Household_read_application_rejection_update_wage_reservation_2. Household reads the application rejections and decreases the reservation wage.

3.2 Messages being used

See Table 4.

4 Credit Market Implementation

This model was adapted from the proposed model of the credit market by the Ancona Unit. Here we present a description of how the model was implemented.

4.1 Description

The credit market involves the interaction of the credit function with the financial management functions of the Firm and Bank agent. The state graph in Figure 4 shows the flow of activity in the model.

4.1.1 Firm Agent in the Credit Market

See Table 5.

Table 4: Messages involved in the labour and goods market implementation.

Name	Variables being sent	Description
vacancies	firm_id, region_id, firm_vacancies, skill_group, firm_wage_offer	Send by firms. Includes the id, the region id the number of vacancies and the wage offer for the according general skill level.
vacancies2	firm_id, region_id, firm_vacancies, skill_group, firm_wage_offer	Send by firms. Includes the id, the region id the number of vacancies and the wage offer for the according general skill level.
firing	firm_id, worker_id	Send by firms. Includes the id and the id of the dismissed employee.
job_application	worker_id, firm_id, region_id, general_skill, specific_skill	Send by households to apply for a job. Includes the id, firm id, region id the general as well as the specific skills.
job_application2	worker_id, firm_id, region_id, general_skill, specific_skill	Send by households to apply for a job. Includes the id, firm id, region id the general as well as the specific skills.
job_offer	firm_id,worker_id, region_id, wage_offer	Send by firms to make a job offer for a household. Includes the id, worker id and the corresponding wage offer.
job_offer2	firm_id,worker_id, region_id, wage_offer	Send by firms to make a job offer for a household. Includes the id, worker id and the corresponding wage offer.
job_acceptance	worker_id, firm_id, region_id, general_skill, specific_skill	Send by households to accept the job offer. Includes the id, firm id, region id, the general as well as the specific skills.
job_acceptance2	worker_id, firm_id, region_id, general_skill, specific_skill	Send by households to accept the job offer. Includes the id, firm id, region id, the general as well as the specific skills.
application_rejection	firm_id, worker_id	Send by firms. Includes the id and the id of the refused applicant.
application_rejection2	firm_id, worker_id	Send by firms. Includes the id and the id of the refused applicant.
quitting	worker_id, firm_id	Send by households to quit the current job. Includes the id, firm id.
quitting2	worker_id, firm_id	Send by households to quit the current job. Includes the id, firm id.

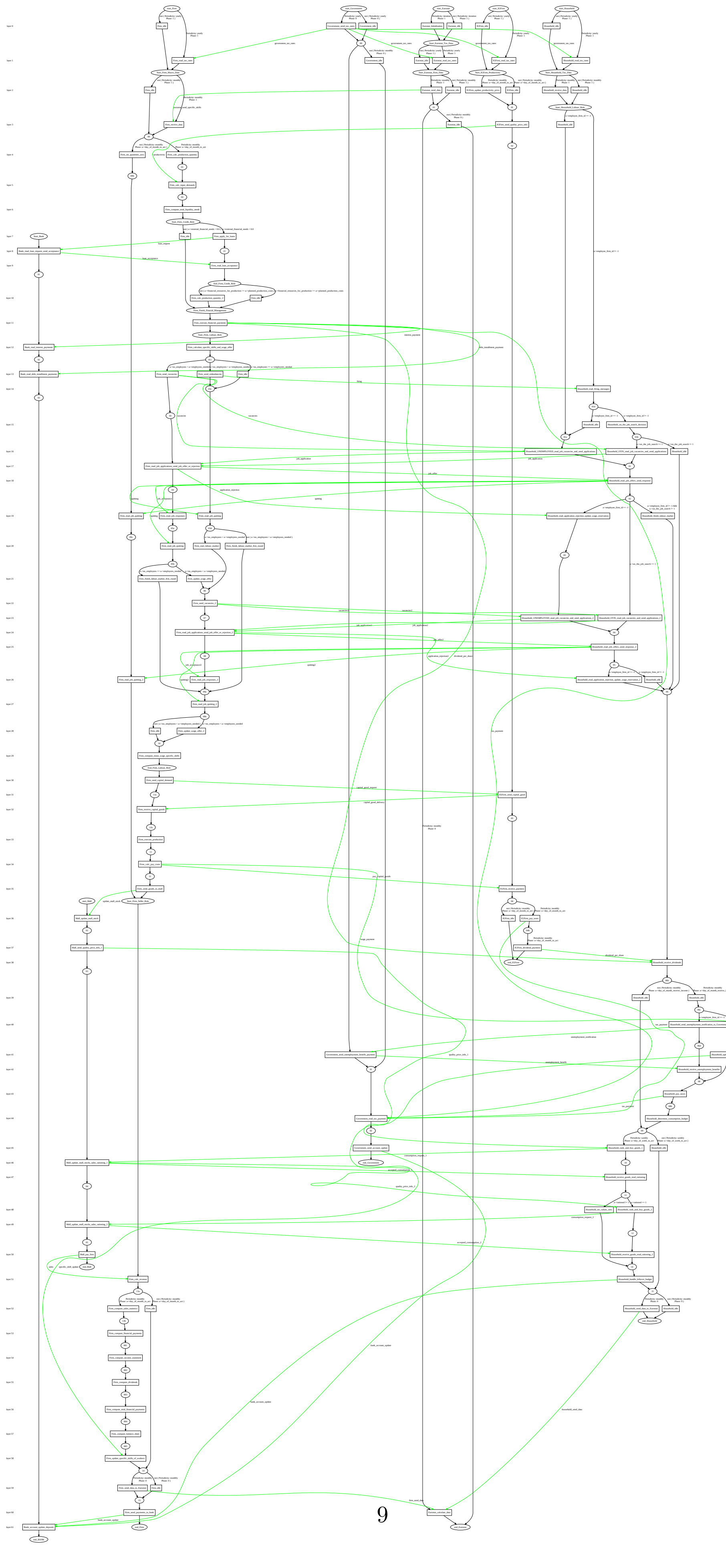

Figure 3: State graph of the Labour Market Model.

Table 5: Functions being performed by the Firm involved in Credit Market.

Function Name	State From	State to	Condition on Function	Inputs	Outputs
Firm _ask _loan	Start _Firm _Credit _Role	Firm _Credit _02	a.external _financial _needs GT 0.0		loan _request
Firm _get _loan	Firm _Credit _02	Firm _End _Credit _Role		loan _conditions (a.id EQ m.firm_id	loan _acceptance

4.1.2 Bank Agent in the Credit Market

See Table 6.

Table 6: Functions being performed by the Bank involved in Credit Market.

Function Name	State From	State to	Condition on Function	Inputs	Outputs
Bank _decide _credit _conditions	Bank _start _credit _market _role	Bank_02		loan_request (a.id EQ m.bank_id)	loan _conditions
Bank _give _loan	Bank_02	Bank_03		loan _acceptance (a.id EQ m.bank _id)	
Bank _receive _instalment	Bank_03	Bank_04		instalment (a.id EQ m.bank_id) bankruptcy (a.id EQ m.bank_id)	
Bank _account _update _deposits	Bank_04	Bank_05		bank_account _update (a.id EQ m.bank_id)	central _bank _account _update
Bank _accounting	Bank_05	end _Bank _cycle	monthly (a.day _of _month _to _act)		
Bank_idle	Bank_05	end _Bank _cycle	not (monthly (a.day _of _month _to _act))		

4.2 Messages being Used

See Table 7.

Table 7: Messages involved in the credit market implementation.

Name	Variables being sent	Description
loan_request	firm_id, bank_id, equity, total_debt, external_financial_needs	Message added by firm to demand credit with bank_id, with financial info of applying firm.
loan_conditions	firm_id, bank_id, proposed_interest_rate, amount_offered_credit, value_at_risk	Message added by bank to offer credit, contains the interest rate, the amount of offered credit, and the value_at_risk.
loan_acceptance	bank_id, credit_amount_taken, loan_total_var	Message added by firm to accept a loan with bank_id, for the amount credit taken and VAR. The bank does not need to know the firm_id.
instalment	bank_id, instalment_amount, interest_amount, var_per_instalment	Message added by firm pays instalment and interest to the bank.
bankruptcy	bank_id, bad_debt, credit_refunded, residual_var	Message added by firm to bank to signal bankruptcy.
BCE_return	bce_debt, id	

4.3 Implementation Results

The implementation of the agents described above was then simulated to produce results which could later then be tested for verification of the model. The results and technical details of the credit market have also been discussed in D5.2.

The simulation consists of fifty iterations, that is 50 days. The network is composted by 100 firms and 10 banks. Households are replaced by the Dummy Agent.

Every month firms accomplish two main actions: they ask for loans and pay debt installments, inclusive of interests. The demand of credit occurs once per month. All firms, for sake of simplicity, are activated in the first day of each month. Interests are payed back every day after the loan has been taken out. The more relevant initial conditions of firms have been set as follows: *equity* = 100, *total debt=0*, *cash* = 100 and *total assets=0.*

As far as banks are concerned, they have the task to lend money and fix interest rates whenever required by firms. Then, banks collect interests and installments payed back by firms increasing, so, their equity and cash. At the end of the month, banks with positive profits will pay taxes and dividends.

Relevant bank variables have the following initial values: *cash=1000*, *total credit=0*, *equity=1000*, *debts versus BCE=0*, γ_t=0.2, γ_{t-1}=0.4, *BCE's interest 2%*, α=0.8 and $\pi_{t-1} = \pi_{t-2}$=500.

Figure (5) shows the average bank equity along the simulation time. In the first twenty days it increases thanks to interest payments. In the first day of the second month (21st iteration), banks pay taxes and, so, the equity shrinks. Then, since banks grant new loans and, consequently, new interests are payed, the everage equity increases again and so on. It is worthwhile to note that in the second month equity grows less because banks are short of liquidity and lend less money. This is because we have not considered deposits from households (which are created in another *EURACE* module); additional liquidity, disposable for new loans, comes only from firms' payment of installments relative to previously contracted debts.

In figure (6) we can see the average bank cash. After the first period it decreases abruptly because of loan demands occurring in the same time. In the following days, banks receive installments and can build up their liquidity again. The same reasoning for equity applies: we do not have deposits, so cash can only re-establish slowly and is immediately absorbed by new credit demands: that is why the average cash shows along time only small increases.

5 Testing

Testing agent-based models is difficult because the interaction of agents in a simulation is dynamic and possibly random which produces complex patterns and behaviours. Traditional software testing strategies which use the divide and conquer approach can be applied but will typically miss some undesirable possibilities due the emergent nature of agent-based systems.

Envisioned possibilities can be easily tested but when a model diverges from its intended path the relationship between variables could show a bug or even an unexpected solution. The invariants to a simulation, the relationship of variables that remain true, can be pre-written or detected during simulation runs and provide a dynamic way of testing agent-based models.

For example during a simulation run assertions can be placed on variables to make sure they fall within a certain range. If ranges are not known before hand they can be detected and the results viewed by the model creator for correctness.

The testing of individual parts of a system can still achieved by unit testing but testing the interaction of these parts, integration testing, can use techniques designed for state machines, and the use of invariant detectors.

5.1 Unit Testing

Unit testing is the testing of individual modules of a piece of software, in the case of models these are the individual agent functions. Each function has an accompanying unit test function that sets the initial agent memory and any input, calls the function, and assets that the new agent memory and outputs are of the expected values.

FLAME provides procedures to help with unit testing:

- initialise_unit_testing() – initialises FLAME for unit testing agent functions, required at the start of a testing suite.
- unittest_init_agentname_agent() – initialises the agent memory that will used in testing
- The agent memory is then updated
- Messages are sent that will be the input to the function to be tested
- The function to be tested is called
- Assertions are tested against the resulting agent memory and any output
- unittest_free_agentname_agent() – the agent memory is deleted
- free_messages() – any messages used are deleted
- clean_up() – FLAME is finalised and ready to end.

Unit testing treats the function to be tested as a black box, inputs are given and only the outputs are tested, as shown in Figure 7.

5.2 Integration Testing

Integration testing is the testing of combinations of individual modules of a piece of software. Modules that have been unit tested are aggregated into groups and each group is tested as a whole system. Generating the groups requires a specific coverage of testing combinations, a test set.

The W-method proposed by T. Chow [1] provides a complete test set of sequences through a state machine. Because models have been defined as state machines this technique can be used.

An implementation of the W-method is available through a program called statechum [4, 3] (http://statechum.sourceforge.net/) which accepts as input graphml (http://graphml.graphdrawing.org/), an XML standard for describing graphs. Each agent as a separate state machine when fed into the W-method will produce a test set with a specified coverage. An example test set for the Firm agent in the labour and goods markets is as follows:

```
#1 [Firm_idle, Firm_receive_data, Firm_calc_production_quantity]
#2 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_idle, Firm_execute_financial_payments,
    Firm_calculate_specific_skills_and_wage_offer, Firm_idle, Firm_read_job_quitting,
    Firm_finish_labour_market_first_round, Firm_read_job_quitting_2, Firm_idle,
    Firm_compute_mean_wage_specific_skills, Firm_send_capital_demand,
    Firm_receive_capital_goods, Firm_execute_production, Firm_calc_pay_costs,
    Firm_send_goods_to_mall, Firm_calc_revenue]
#3 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_idle, Firm_execute_financial_payments,
    Firm_calculate_specific_skills_and_wage_offer, Firm_idle, Firm_read_job_quitting,
    Firm_finish_labour_market_first_round, Firm_read_job_quitting_2,
    Firm_update_wage_offer_2, Firm_compute_mean_wage_specific_skills]
#4 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_idle, Firm_execute_financial_payments,
    Firm_calculate_specific_skills_and_wage_offer, Firm_idle, Firm_read_job_quitting,
    Firm_start_labour_market, Firm_send_vacancies_2,
    Firm_read_job_applications_send_job_offer_or_rejection_2, Firm_read_job_responses_2,
    Firm_read_job_quitting_2, Firm_idle]
#5 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_idle, Firm_execute_financial_payments,
    Firm_calculate_specific_skills_and_wage_offer, Firm_send_vacancies,
    Firm_read_job_applications_send_job_offer_or_rejection, Firm_read_job_responses,
    Firm_read_job_quitting, Firm_finish_labour_market_first_round, Firm_read_job_quitting_2,
    Firm_idle]
#6 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_idle, Firm_execute_financial_payments,
    Firm_calculate_specific_skills_and_wage_offer, Firm_send_vacancies,
    Firm_read_job_applications_send_job_offer_or_rejection, Firm_read_job_responses,
    Firm_read_job_quitting, Firm_update_wage_offer, Firm_send_vacancies_2]
#7 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_apply_for_loans, Firm_read_loan_acceptance,
    Firm_idle, Firm_execute_financial_payments]
#8 [Firm_idle, Firm_idle, Firm_calc_production_quantity, Firm_calc_input_demands,
    Firm_compute_total_liquidity_needs, Firm_apply_for_loans, Firm_read_loan_acceptance,
    Firm_calc_production_quantity_2, Firm_execute_financial_payments]
#9 [Firm_idle, Firm_idle, Firm_set_quantities_zero, Firm_read_job_quitting,
    Firm_read_job_quitting_2, Firm_calc_revenue, Firm_idle, Firm_send_data_to_Eurostat,
    Firm_send_payments_to_bank]
#10 [Firm_idle, Firm_idle, Firm_set_quantities_zero, Firm_read_job_quitting,
     Firm_read_job_quitting_2, Firm_calc_revenue, Firm_compute_sales_statistics,
     Firm_compute_financial_payments, Firm_compute_income_statement, Firm_compute_dividends,
     Firm_compute_total_financial_payments, Firm_compute_balance_sheet,
     Firm_update_specific_skills_of_workers, Firm_idle]
#11 [Firm_idle, Firm_idle, Firm_set_quantities_zero, Firm_read_job_quitting,
     Firm_read_job_quitting_2, Firm_calc_revenue, Firm_compute_sales_statistics,
     Firm_compute_financial_payments, Firm_compute_income_statement, Firm_compute_dividends,
     Firm_compute_total_financial_payments, Firm_compute_balance_sheet,
     Firm_update_specific_skills_of_workers, Firm_send_data_to_Eurostat]
#12 [Firm_read_tax_rates, Firm_receive_data]
#13 [Firm_read_tax_rates, Firm_idle]
```

Anticipating the results of individual functions is straight forward. They are coded for their specific purpose with required results and are therefore easy to test. With the introduction of communication and interchange of information, anticipating the results from a group of functions is not so easy, this being one of the reasons for using agent-based modelling. The consequence of not being able to predict the results is that first the results have to be created and then tested for what was roughly expected.

One way to analyse the output of groups of functions is to use an invariant detector. A program of this type reports any likely invariants, properties that hold at certain points. This could then be analysed by the modeller and possibly used as assertions in future test runs.

The Daikon program is an invariant detector [2] (http://groups.csail.mit.edu/pag/daikon/). After feeding Daikon the results from fifty iterations from the labour and goods market model, the invariants of each agents variables are produced. The examples given below show invariants of variables where: they keep a certain value; are one of a limited list of values; or are large or smaller than a certain value. These are only three types of invariants but Daikon can be told to search for many more, for example the relationship between two or more variables.

```
===========================================================================
Bank:::OBJECT
id == 1233
region_id == 2
gov_id == 1232
last_credit_id == 0
amount_credit_offer == 0.0
total_deposits one of { 0.0, 1808.8 }
total_loan_supply one of { 0.0, 1627.92 }
===========================================================================
Eurostat:::OBJECT
id == 1236
region_id == 2
num_households == 0
unemployment_rate == 0.0
average_s_skill == 1.0
no_firms == 0
gdp == 0.0
===========================================================================
Firm:::OBJECT
id >= 1
region_id one of { 1, 2 }
gov_id == 1232
day_of_month_to_act == 0
payment_account == 50.0
wage_offer == 1.0
average_g_skill == 0.0
ebit == 0.0
tax_rate_corporate == 0.25
```

```
earnings_per_share == 0.01
current_shares_outstanding == 1200
total_value_capital_stock == 200.0
planned_value_capital_stock == 2.0
bank_id == 1233
mean_specific_skills == 0.8
out_of_stock_costs one of { 0.0, 1.0 }
============================================================================
Government:::OBJECT
id == 1232
bank_id == 1233
payment_account == 100.0
tax_revenues == 0.0
unemployment_benefit_payment == 0.8
tax_rate_corporate == 0.25
num_unemployed == 0
============================================================================
Household:::OBJECT
region_id one of { 1, 2 }
gov_id == 1232
bank_id == 1233
day_of_month_to_act == 0
payment_account == 0.8
wage == 0.0
wage_reservation == 1.0
general_skill >= 1
number_applications == 15
last_labour_income == 0.0
employee_firm_id == -1
employer_region_id == 0
week_of_month one of { 3, 4 }
mall_completely_sold_out one of { 0, 1 }
day_of_week_to_act >= 0
tax_rate_hh_capital == 0.25
============================================================================
IGFirm:::OBJECT
id == 31
region_id == 2
gov_id == 1232
bank_id == 1233
day_of_month_to_act == 0
payment_account == 0.0
productivity == 1.0
capital_good_price == 100.0
tax_rate_corporate == 0.25
tax_payment == 0.0
outstanding_shares == 1200
============================================================================
Mall:::OBJECT
id one of { 1234, 1235 }
region_id one of { 1, 2 }
gov_id == 0
total_supply == 0.0
```

This technique can be utilised for tracking changes of variables between functions and not just between iterations giving more testing coverage of groups of functions.

References

[1] TS Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, March 1978.

[2] MD Ernst, JH Perkins, PJ Guo, S McCamant, C Pacheco, MS Tschantz, and C Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35–45, December 2007.

[3] N Walkinshaw and K Bogdanov. Inferring finite-state models with temporal constraints. *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08), L'Aquila, Italy*, 2008.

[4] N Walkinshaw, K Bogdanov, M Holcombe, and S Salahuddin. Reverse engineering state machines by interactive grammar inference. *14th IEEE Working Conference on Reverse Engineering (WCRE'07), Vancouver, Canada*, October 2007.

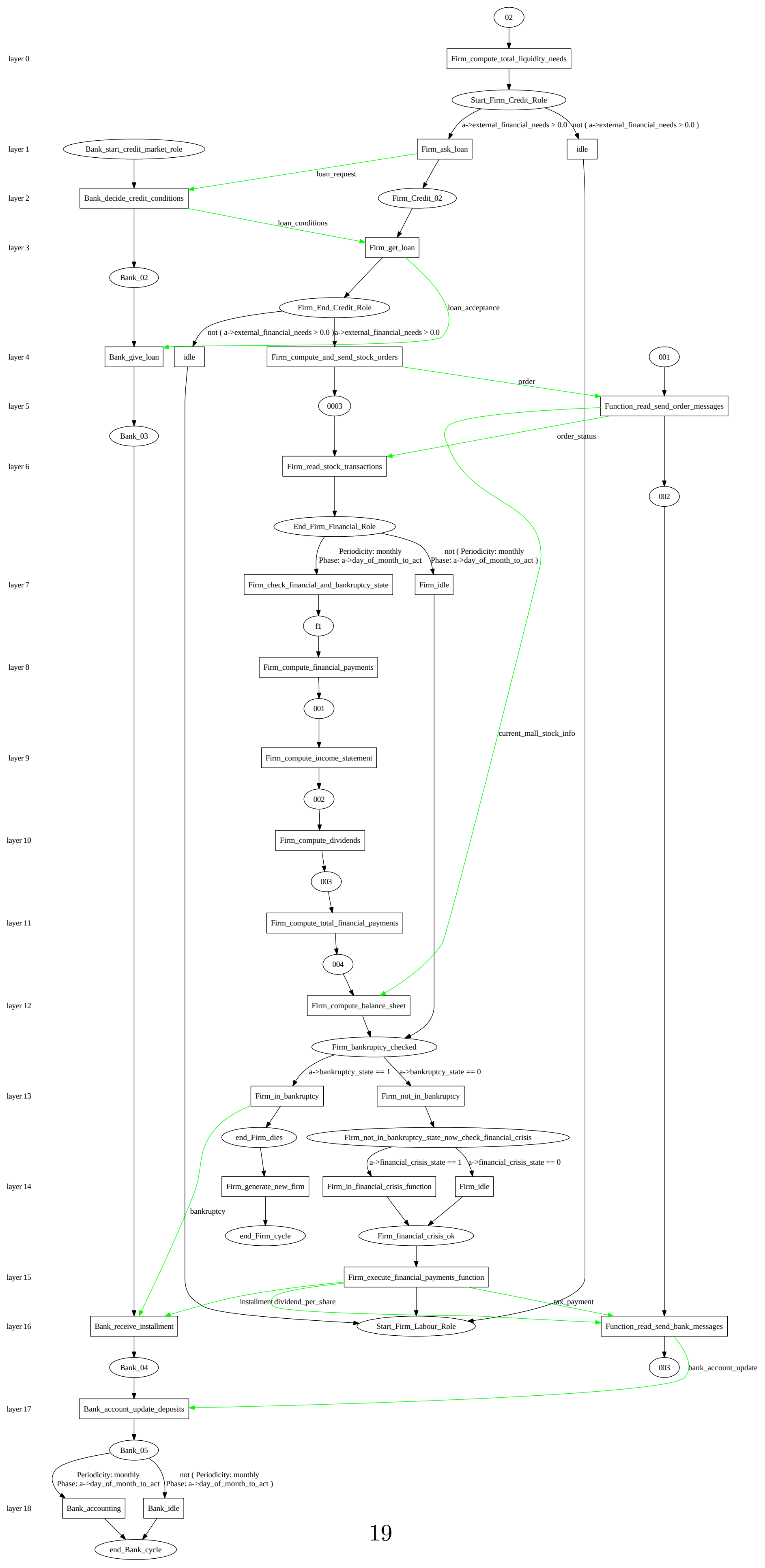
02
layer 0
Firm_compute_total_liquidity_needs
Start_Firm_Credit_Role
a->external_financial_needs > 0.0
not (a->external_financial_needs > 0.0)
layer 1
Bank_start_credit_market_role
Firm_ask_loan
idle
loan_request
layer 2
Bank_decide_credit_conditions
Firm_Credit_02
loan_conditions
layer 3
Firm_get_loan
Bank_02
Firm_End_Credit_Role
loan_acceptance
not (a->external_financial_needs > 0.0)
a->external_financial_needs > 0.0
layer 4
Bank_give_loan
idle
Firm_compute_and_send_stock_orders
001
order
layer 5
0003
Function_read_send_order_messages
Bank_03
order_status
layer 6
Firm_read_stock_transactions
002
End_Firm_Financial_Role
Periodicity: monthly
Phase: a->day_of_month_to_act
not (Periodicity: monthly
Phase: a->day_of_month_to_act)
layer 7
Firm_check_financial_and_bankruptcy_state
Firm_idle
f1
layer 8
Firm_compute_financial_payments
001
current_mall_stock_info
layer 9
Firm_compute_income_statement
002
layer 10
Firm_compute_dividends
003
layer 11
Firm_compute_total_financial_payments
004
layer 12
Firm_compute_balance_sheet
Firm_bankruptcy_checked
a->bankruptcy_state == 1
a->bankruptcy_state == 0
layer 13
Firm_in_bankruptcy
Firm_not_in_bankruptcy
end_Firm_dies
Firm_not_in_bankruptcy_state_now_check_financial_crisis
a->financial_crisis_state == 1
a->financial_crisis_state == 0
layer 14
Firm_generate_new_firm
Firm_in_financial_crisis_function
Firm_idle
bankruptcy
end_Firm_cycle
Firm_financial_crisis_ok
layer 15
Firm_execute_financial_payments_function
installment
dividend_per_share
tax_payment
layer 16
Bank_receive_installment
Start_Firm_Labour_Role
Function_read_send_bank_messages
Bank_04
003
bank_account_update
layer 17
Bank_account_update_deposits
Bank_05
Periodicity: monthly
Phase: a->day_of_month_to_act
not (Periodicity: monthly
Phase: a->day_of_month_to_act)
layer 18
Bank_accounting
Bank_idle
end_Bank_cycle

Figure 4: State graph of the Credit Market Model.

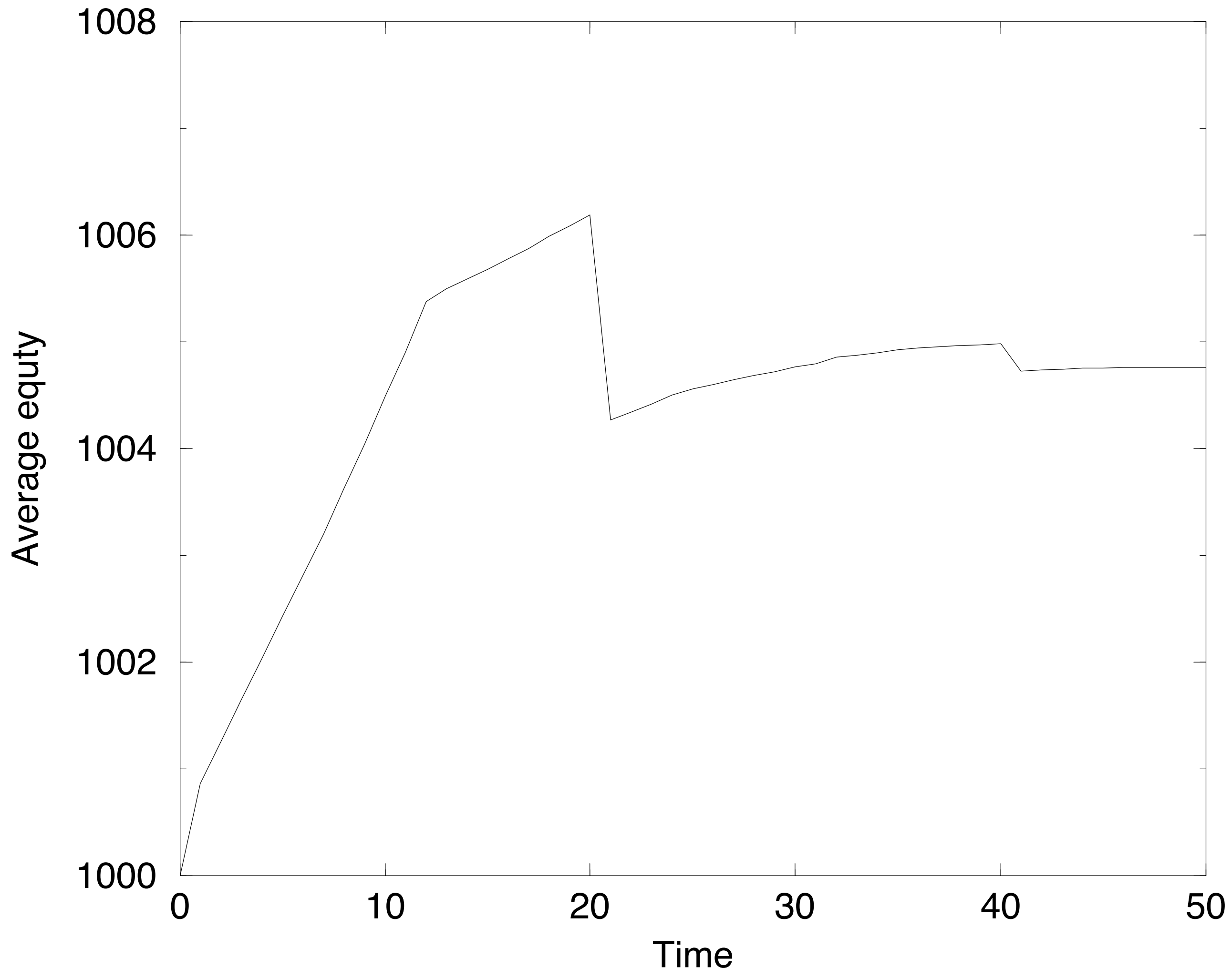

Figure 5: Time series of average bank equity. Jumps occur when taxes are payed.

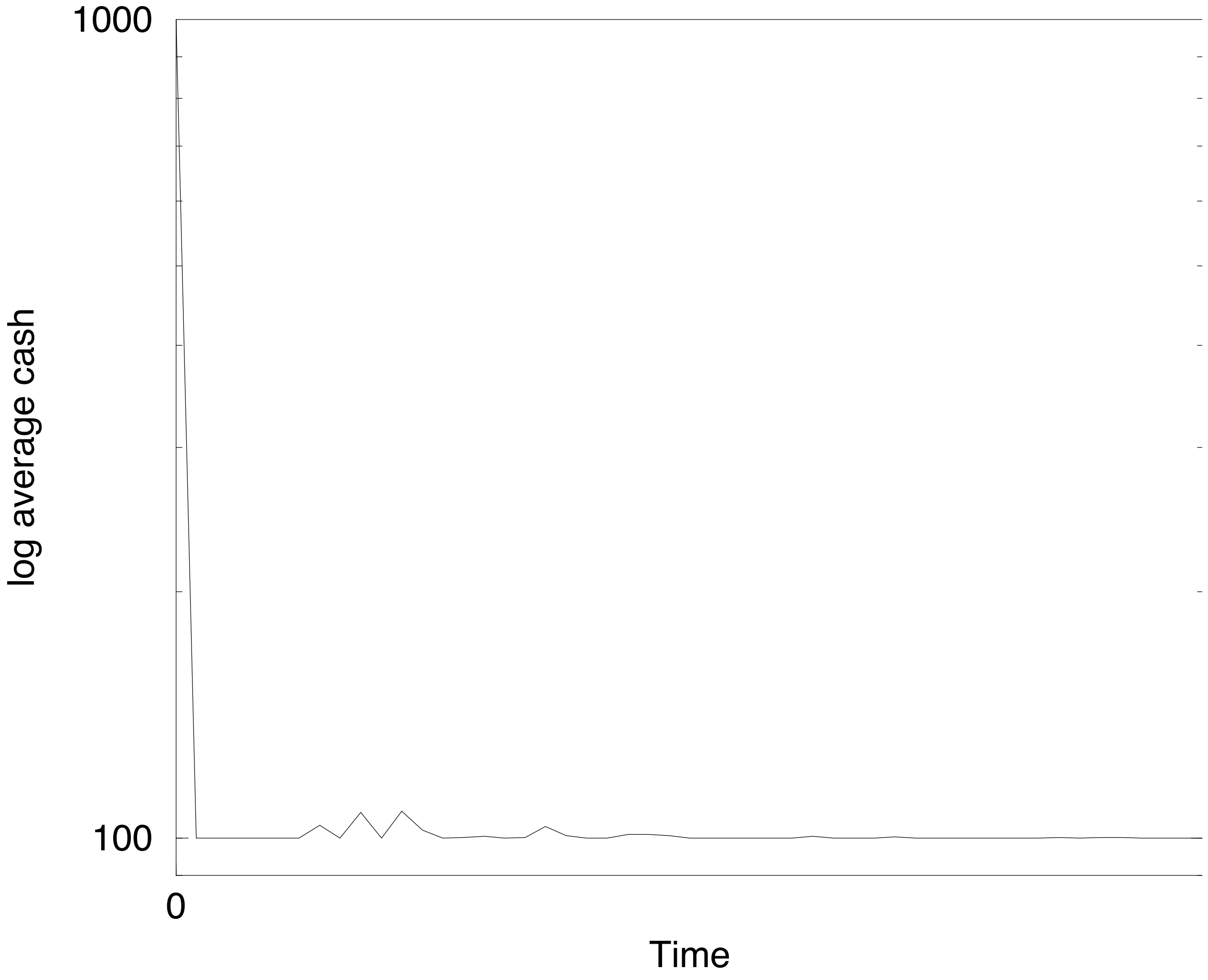

Figure 6: Time series of average bank liquidity supply.

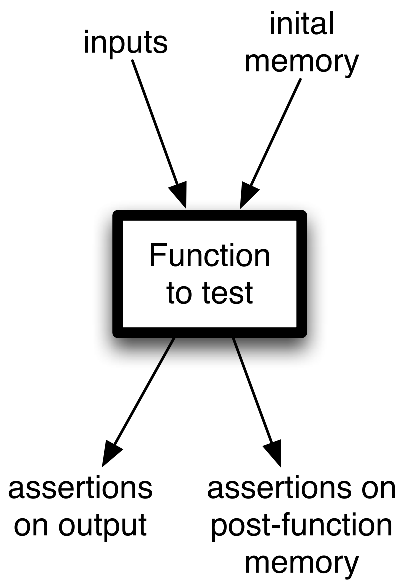

Figure 7: Unit testing