Multi Agent System for Continuously Improving Enterprise Performance

Arwa Ibrahim Ahmed
Information System Department
Princess Nourah bint Abdulrahman University
KSA , Riyadh
ariahmed@pnu.edu.sa

Abstract: - This paper examines the potential value of MAS technology to continuously improve enterprise capability to gain the highest performance level.
In terms of contribution, it describes concepts and methodologies within the field of Multi-Agent Systems (MAS) that are appropriate to work with required performance levels. As well as presenting a comprehensive review of the meaningful framework for which MAS be investigated, it also defines the technical issues, which should be addressed in order to examine the current performance of the enterprise process, and test the level of quality suggest clear strategy to gain the optimization level using multi agent technology. We also uses the banking systems as a case study to prove the merits of the proposed MAS architecture and implementation methodology.
Different categories of agents, their internal structures and functions are described. Programming solutions to MAS problems are modeled using the computation structure models since it includes data interactions among agents and the algorithm to implement the solution plan. Every solution plan should have a feedback to continuously monitoring and enhancing the target enterprise performance. Banking system is used to show the merits of using MAS to gain the required levels of performance..

Key-Words: - Multi agent system, computation structure model, Quality assessment, Process improvement.

1 Introduction
Having a global economy and increase in customer expectations in terms of cost, quality and services have put a premium on effective business process and efficient business models without weakness in any of process or activities, with proper architecture. This study presents a Multi Agent System (MAS) framework for improving enterprise performance. The new framework describes an evolutionary improvement path from an ad hoc, immature process to a mature disciplined process. It covers the practices for planning, engineering and managing enterprise performance. These key practices improve the ability of the organization to meet the goals for cost, schedule, functionality and product quality [1, 11-16]. There is now substantial evidence of the business benefits of MAS-based system and a growing understanding of the factors that contribute to a successful improvement effort [2-9].
As agents, increasingly call upon technologies to play vital roles in business field, enterprise performance should be monitored and enhanced by effective multi agent system architecture.
Multi agent system provides an effective pragmatic approach, so - called agent technology has been used to solve problems in various fields including diagnostics [2], condition monitoring [3], power system restoration [4,5], market simulation [6], network control [7], automation [8] and crisis management [9].
Multi-agent systems (MAS) are suitable for the domains that involve interactions among different people or organizations or system component with different (possibly conflicting) goals and proprietary information [1-9,11-16]
In this technology, human and business responsibilities delegated to the agent whose functions, behavior and roles carefully defined. It is autonomous, proactive, reactive and able to handle unpredictable events and change dynamically; it receives input from different sources in real-time and even taking social behavior into account. However, agent has definite advantages when function is critical to safety.

2 Problem Formulation
Towards a more systematic MAS architecture, it is important to use goals, rules, and methods to support the systematic analysis and design of

business processes. In this regard, we propose a framework that consists of three categories of agents, number of agents depend on the size and process of enterprise. Multi agent system framework has been developed to overcome the complexity, difficulty and the complicated of improving system performance.
MAS operation is described using the Computation Structure Model (CSM) [10]. CSM provides information about data interaction among different MAS agents. It also describes in details the algorithm of operating MAS agents including a feedback path required to continuously improve the enterprise performance.

3 Proposed Solution

An agent is a computer system/component situated in some environment, and is capable of autonomous action in order to meet its design objectives. It is usually internally motivated, embedded, and adaptive with inferential capability, communication ability and mobility.

3.1. MAS Framework

MAS establishes a framework for continuous process improvement The main frame work will focus on five goals: utilization of resource, trust and conflict resolution, control and compliance to procedures, interpersonal communications, problem solving , experimentation and creativity [1-9,11-16].
Architecture of the system is composed of three layers to specify the current performance and future direction, understanding, control and improvement.
Each layer governed by type of agent with different capability; environment and knowledge from agents in the other layers. The number of agent in each layer depends on the size of enterprise, process and needs.
The first layer stablish when the first type of agent which called info agent(IA) immigrate to different and homogenies environments in the organizations to collect all the enterprise process activities, information and performance. Moreover, record the result in the measurement repository as an output of the first layer.
Second layer is the backbone of the architecture is govern by Evaluation agent (EA) , it has sub agents , each agent has specific key area and specific goal depend on the key process area and delegated roles determined by the evaluation agent. EA responsible for collects the results from sub agents, called key area agent (KA), produce evaluation report as an output of the second layer, and send it to quality agent (QA).
The third layer is managed by "Quality agent" which responsible for determining the current process performance of enterprise by mapping the process , evaluate and compare the process depending on given parameters, identifying the most critical issues to improving their software quality and process, use planning, learning capabilities to produce quality report as an output of this layer, also the agent has generate results to prove that the enterprise is ready to moving to the highest performance levels.
It produces improving proposal concern with quality and process management, depending on the recommendation that received from the previous agent, the output of this layer is proposals of improvement plans, enterprise can follow selected one to move to the highest level.

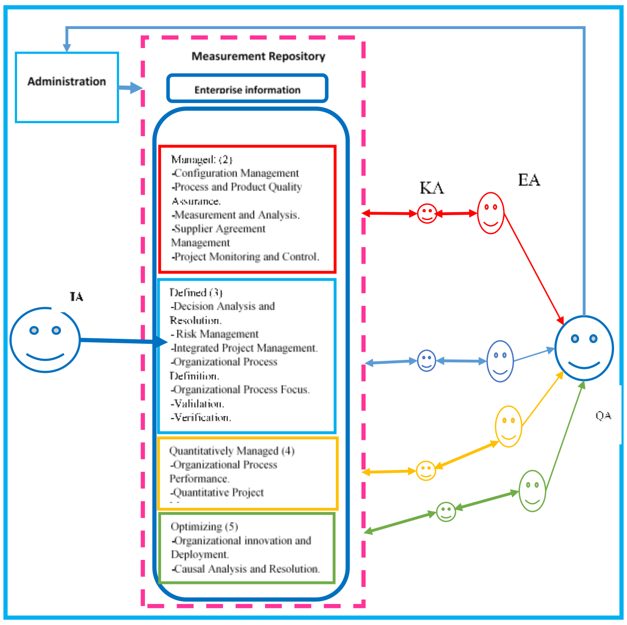

Administration
Measurement Repository
Enterprise information
Managed: (2)
-Configuration Management
-Process and Product Quality Assurance.
-Measurement and Analysis.
-Supplier Agreement Management
-Project Monitoring and Control.
Defined (3)
-Decision Analysis and Resolution.
- Risk Management
-Integrated Project Management.
-Organizational Process Definition.
-Organizational Process Focus.
-Validation.
-Verification.
Quantitatively Managed (4)
-Organizational Process Performance.
-Quantitative Project
Optimizing (5)
-Organizational innovation and Deployment.
-Causal Analysis and Resolution.
IA
KA
EA
QA

Fig.1 System architecture

3.2 Internal architecture and function of each agent category

3.2.1 Info agents (IA)

Info agents works in the first layer. The number of Info Agents depend on the size of enterprise and processes , Agent using "mobility" characteristic to migrate to different departments, sectors and environment of enterprise , however A mobile agent's primary identifying Characteristic is its ability to autonomously migrate from host to host [15].

Info agents install themselves and act as customer, manager, and employee to test the current behavior, collect all possible performance information and send it to the measurements repository.

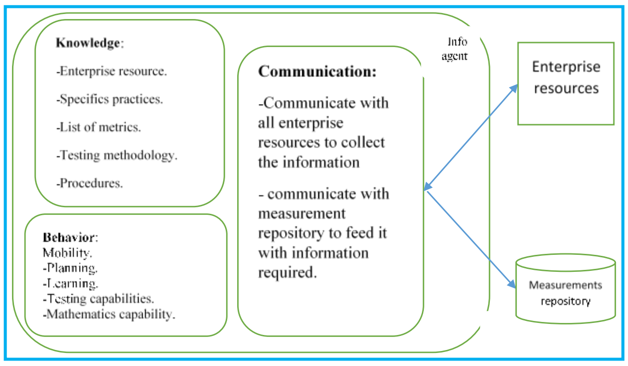


Fig.2 Internal architecture for info agent (IA)

3.2.2 Evaluation agents

Evaluation agent is the heart of the system; worked in the second layer, it is proactive with intelligent features to support the behavior to measurements of the capability performance level.

Each key process area governs by evaluation agent. However, key process area identifies a cluster of related activities that, when performed collectively, achieve a set of goals necessary for enhancing process capability. Depending on the application function, the system should contains at least four evaluation agents. These are: (Manage Evaluation Agent (MEA), Define Evaluation Agent, (DEA) Quantitatively Managed evaluation Agent (QEA) and Optimizing Evaluation Agent (OEA). Each evaluation agent has its sub-agents called Key Area process agents (KA) to perform the required measurements and evaluate the process and activities in specific key process area. (EA) distribute the goals and delegate the practices of each Key Area to (KA). Key Area agent, process the following measurements:

General Management:
- Configuration Management
- Process and Product Quality Assurance.
- Measurement and Analysis.
- Supplier Agreement Management
- Project Monitoring and Control
- Decision Analysis and Resolution.

Risk Management
- Integrated Project Management.
- Organizational Process Definition.
- Organizational Process Focus.
- Validation.
- Verification.

Quantitatively Management:
- Organizational Process Performance.
- Quantitative Project.

Optimization:
- Organizational innovation and Deployment.
- Causal Analysis and Resolution.

Evaluation agents are deferent in the internal architecture due to their different responsibilities, deferent in practices, activates, environment and functions as well as procedures that have to use it in measurement.

(EA) receives all the measurement result from (KA)-which describes its unique characteristics that must basically be present to satisfy the particular process area and coordi*nate between agents to prepare the evaluation report. The evaluation report should contains all the measurements results for key process areas and enterprise practices.

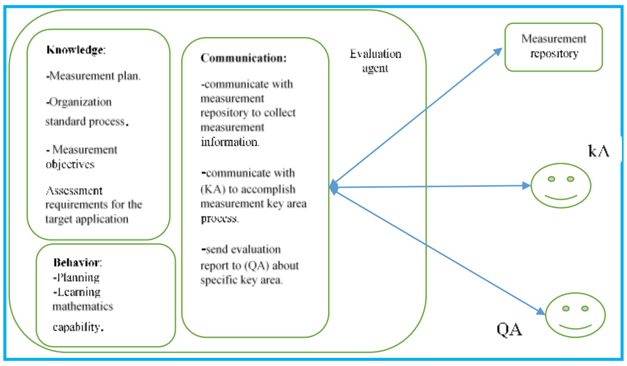

Knowledge:
-Measurement plan.
-Organization standard process.
- Measurement objectives
Assessment requirements for the target application
Behavior:
-Planning
-Learning mathematics capability.
Communication:
-communicate with measurement repository to collect measurement information.
-communicate with (KA) to accomplish measurement key area process.
-send evaluation report to (QA) about specific key area.
Evaluation agent
Measurement repository
kA
QA

Fig.3 Internal structure for evaluation agent (EA)

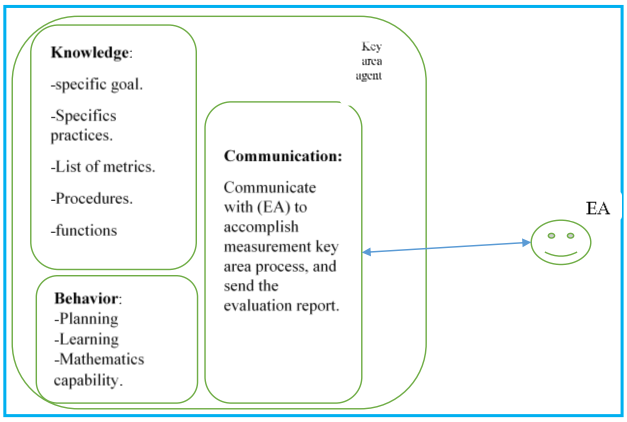


Fig.4 Internal architecture for key area agent (KA)

3.2.3 Quality agent (QA)

Quality agent works in the third layer. It follows the MAS key process area to identify issues that must be addressed to achieve a performance level.

The numbers of quality agent (QA) depend on the situation of the enterprise and the ability to produce many suggestions to improve the performance levels. It has intelligent features to enable it producing the suggested plans that the enterprise has to follow to improve the performance level.

Quality Agent use proactive and learning capabilities for mapping. It analyzes the results of evaluation to identify the current performance level depending on the score between (zero – 20+). The score starts from (0-4) for level one ,(5 - 9) for level two , (10 - 14) for level three , (15 – 19) for level four , and (20 +) represents level five, the highest level of performance.

1.level1: Initial (chaotic ‹ad hoc ‹individual heroics) -the starting point for use of a new or undocumented repeat process.

2.Level2: Repeatable -the process is at least documented sufficiently such that repeating the same steps may be attempted.

3.Level3: Defined -the process is defined/confirmed as a standard business processes.

4.Level4: Managed -the process is quantitatively managed in accordance with agreed-upon metrics.

5.level5: Optimizing-process management includes deliberate process optimization/improvement.

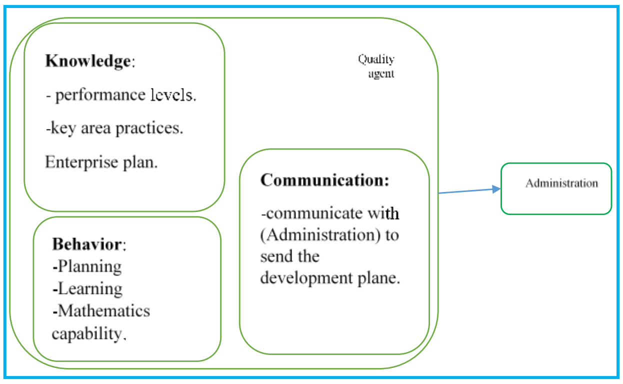


Fig5. Internal architecture for Quality agent (QA)

4 Modeling the Multiple Agent System (MAS)

The Multiple Agent System (MAS) is model using the Computation Structure Model (CSM). Formal representation of the model consists of two directed graphs, one is called the data flow graph, the other is called the control flow graph. Data Flow Graph shows the input and the output of each operation in the MAS whereas the Control Flow Graph (CFG) determines the sequence of executing these operations.

We focus on using CFG to show the MAS functions. CFG consists of a sequence of operations and a set of control nodes. Control nodes consist of the following nodes:

1) Start node: It indicates the beginning of the MAS operation.

2) End node: it terminates the MAS operation (The End node does not exist if the MAS is continuously running).

3) Condition node: It evaluates a logic expression that describes the status of some variables and decides one of the alternative paths.

4) Fork node: It creates multiple threading to accommodate concurrent paths (threads).

5) Join nodes: It is used to synchronize activities among multiple paths (threads).

Operation nodes are basically agents of different types performing their functions.

Figure, shows the proposed MAS system. It consists of two major steps: Measurement and improvement plan generation. In the first stage, a number of Info agents are running to measure system performance and collect necessary information. Collected information is organized in different reports as guided by administration needs. In the second stage, four Key Area (KA) agents are functioning in parallel to perform four different operations described above. Each KA feeds its results to a matching evaluation agent. Each evaluation agent will conduct its process as pre-defined and sends results to the quality agent. Evaluation agents are synchronized before the quality agent starts its operation. The quality agent unitizes evaluation agent’s analysis to define an improvement plan.

This completes one MAS cycle. The proposed improvement plan is implemented and after a period of time (three months in the system under consideration) before the second cycle starts.

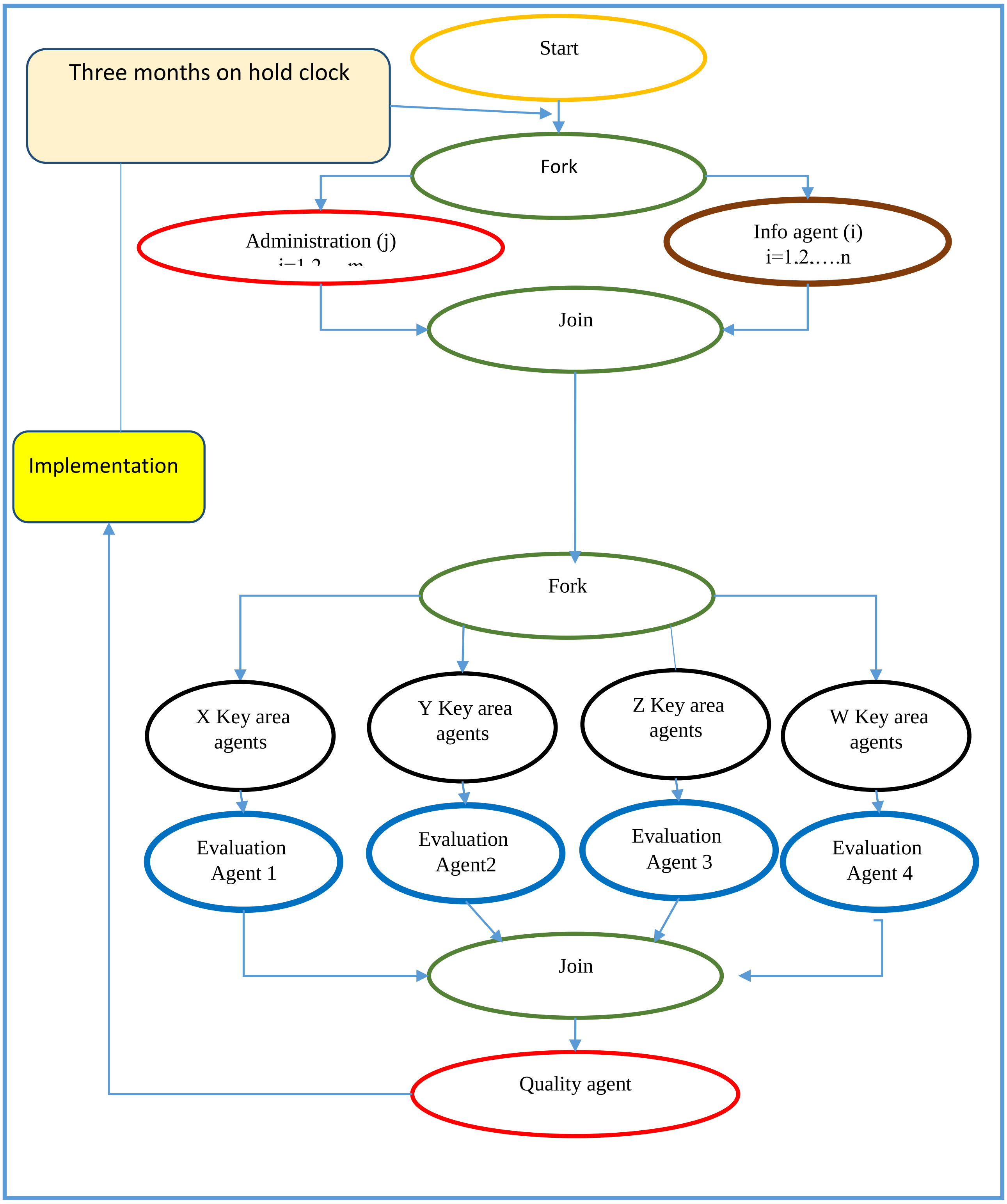


Fig.6 Frame work architecture

4.1 Measurement repository structure:

Info agents are working in parallel collect different measurements. They store collected data in the measurement repository. Information collected in the measurement repository is concurrently access by different agents and administration. This might create possible access conflicts in accessing the measurement repository. Hence it is necessary to develop the right structure for the measurement repository to allow concurrent access without conflicts. In this work, we propose using the

4.2. Producer/Consumer model:

Producer/consumer processes are quite common in operating systems [17] A producer process produces information that is consumed by a consumer process. In order to allow these processes to run concurrently, we must create a pool of buffers (memory) that can be filled and emptied by the producer and the consumer, respectively. The producer and the consumer must be synchronized, so that the consumer does not try to consume items which have not yet been produced yet. In this situation, the consumer must wait until an item is produced. Figure …. Shows the communication protocol between one producer and one consumer.

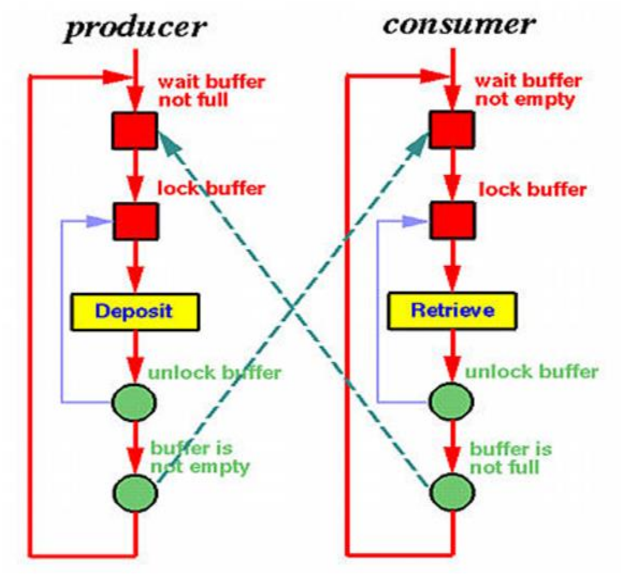


Fig.7 Producer/Consumer model

In our case, every info agent and administrator are producer and every Evaluation agent and key area agent are consumer. In between, we have a Repository. This Repository consists of three parts: a storage area to keep the raw data collected by the info agents and administrator, an organizer that formats the raw data and presents it as instructed by the evaluation and key area agent, the organizer stores its output in another storage area to be ready for the evaluation agent and key area agent to use. Figure gives the block diagram for this communication protocol.

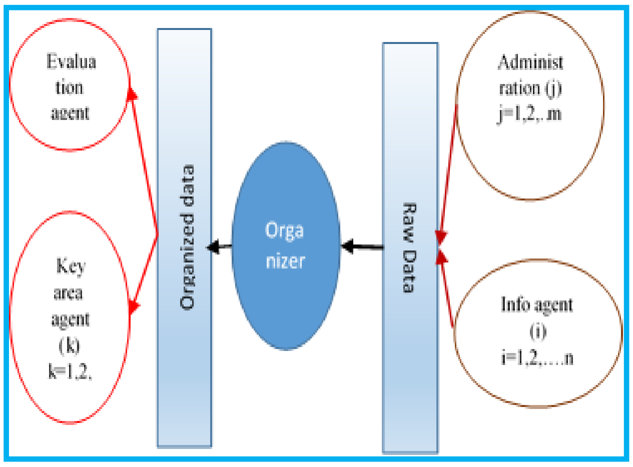


Fig.8 Measurement repository structure

5 Bank Operations Using Multi-Agent System

As discussed in the previous sections, Multi-Agent Systems have a great potential in many domains. In this part, we are discussing how to utilize the technique in improving bank operations. Bank operations are part of one's everyday life. That is why all banks strive to provide the best service to achieve customer satisfaction. Banks provide a lot of their services now online. However, a lot of services must be done in person at one of the bank branches.

In this case study, we investigate, using Java Agent Development Environment JADE [18-19], operations of a bank that has more than one branch ranging from headquarter, medium size to small size branches. The goal is to provide recommendation on how to best utilize services provided in each branch to reach customer satisfaction, which is basically represented as the total time a customer spends inside a branch to get his/her request done.

The following sections include a system overview and operation details. In addition, we present some experiments performed using this system to evaluate its efficiency. At the end, we come up with conclusions and recommendations on using the proposed system.

5.1 System Overview

Our system is a multi-agent recommender. Each bank branch is represented as an agent. In addition, we have a core agent, which we will refer afterwards by “the recommender agent”. Agents collaborate with the recommender agent to provide the best service utilization advices. Figure-1 shows the system architecture.

A bank, represented as an agent, has the following properties that describe the services provided:

Branch ID: a unique identifier for that bank branch in our environment.

Branch Size: describes the size of the branch [headquarter, medium, small].

Services: a list of services that this specific branch offers.

Small: [deposit, withdraw, balance statement, bank checks, open account, transfer, ATM].

Medium: add [wire transfer] to the above list. Headquarter: add [loans] to the above two lists.

Customer Service Desks: a list of desks available at this branch. Each desk type is identified by: a) how many of this type, b) cost per desk and c) the average processing time.

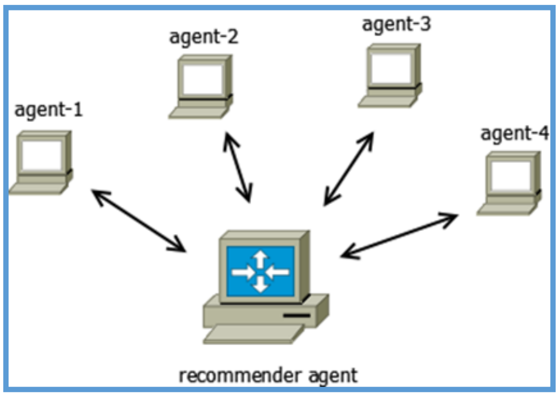


Fig. 9 Multi agent system

5.2. System Operation

Step 1: agents register themselves to the recommender

Each agent send its bank properties described above as a manifest file to the recommender agent. The recommender agent keeps track of each and every agent in the system. As these agents represent the bank branches, the recommender at this step is aware of all branches capabilities and services offered.

Step 2: agents send their daily operation data

After the registration step, each agent should send the daily operation log. This log represents the clients who have already received a service at this specific branch that day. Our system operates on log files that have the following format:

File type: Comma Separated Value (CSV)
Fields: CUSTOMER_ID,
SERVICE_REQUESTED, ARRIVAL_TIME,
SERVICE_START_TIME,
SERVICE_FINISH_TIME

Times are represented as timestamp of the following format: YYYY-MM-DD HH:MM:SS

Step 3: the recommender analyzes the log data

As the recommender receives log data for a branch, it automatically starts analyzing it. The main goal of this step is to get the average waiting time during the day. This value represents the average time a client waited in line for his service to start. We put the value on a scale to measure customer satisfaction. We assume that a customer will likely be unhappy if he waits for more than 5 minutes to start talking to a customer service representative at a desk .2 Problem Formulation

Please, leave two blank lines between successive sections as here.

Mathematical Equations must be numbered as follows: (1), (2), …, (99) and not (1.1), (1.2),…, (2.1), (2.2),… depending on your various Sections.

4.2.1 Subsection

When including a subsection you must use, for its heading, small letters, 12pt, left justified, bold, Times New Roman as here.

5.2.2 Sub-subsection

When including a sub-subsection you must use, for its heading, small letters, 11pt, left justified, bold, Times New Roman as here.

up to 70% of 5 mins	up to 5 minutes	more than 5 minutes
satisfied client		angry client

Step 4: the recommender sends local recommendations

Using the analysis data, the recommender reply to an agent with a recommendation to change some aspect of its manifest file. Two possible local advices the system may provide:

- To add more desks to reduce waiting time.
- To remove desks to reduce total cost.

The recommender replies with one advice at a time. The agent then adjusts the manifest file (properties) and re-computes the log results (SERVICE_START_TIME and SERVICE_FINISH_TIME) for the same log data. Then, the agent sends the log again to the recommender to perform analysis and give advice. This conversation keep running back and forth until the analysis reaches an average waiting time with the minimum cost possible. At this point, the local recommendation conversation stops and the system marks this bank branch as utilized. The recommender does this with each single branch agent till all branch agents are utilized locally.

Step 5: the recommender output global recommendations.

As the recommender keeps track of local recommendation done for each branch, we were able to further extend the system to provide global recommendations. Two possible global advices the system may provide:

- To transfer a desk from one branch to another. This happens in case a branch will remove a desk to reduce cost and another branch will add a desk to reduce waiting time.
- To swap two desks in two different branches. This happens when a branch needs to remove a desk of type 1 and puts another desk of type 2 and the other branch needs the opposite.

5.3. Log Data Generation

To put the above proposed algorithm into action, we had to generate random data to perform our experiments. Here is how we did so. For simplicity, we assumed that customers arrive at a branch in a uniformly distributed manner. So, we used the uniform distribution random number generator to get arrival times. Services requested are also randomly selected from the list of services available at this specific branch. An agent computes the complete log file (to be sent to the recommender) based on the manifest file (the properties). Each time the recommender advises to perform some changes in the manifest file, this doesn't change the original log data. It affects the processed data that has a start and finish time for each client.

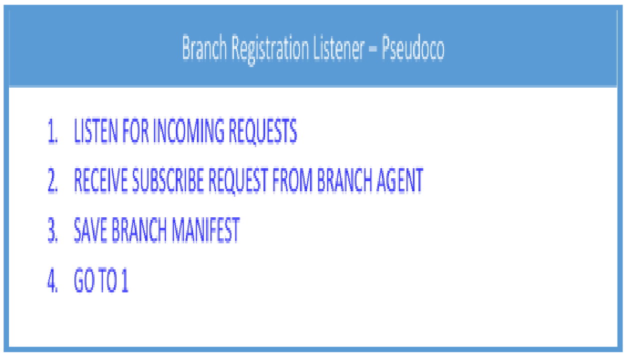
How the agent computes the service start and finish times? First, the agent keeps track of all available desks according to the manifest file. It marks all of them as available at the beginning. For each new arriving client, it checks if there is an available desk or not. If so, it marks this desk as busy from the arrival time till the arrival time plus the average processing time of that specific desk. If no available desks, it checks the first desk to finish and extend its busy duration till its previous client finish time plus the average processing time of that specific desk. This ensures that every arriving client receives service at the earliest convenience in the order of arrival. Although this might seem very tight, it improves the results of our recommender for more accurate advices.

5.4. Recommender Algorithm

The recommender has two main threads that listen to other agents (branches) requests.

- Branch Registration Listener

In this thread, the recommender agent listens to agents that send a message with SUBSCRIBE performative. The content of this message contains the manifest file of this specific branch. As discussed previously, the manifest file describes the services provided in this branch; number of desks, cost and average waiting time per each. The recommender then keeps track of this manifest for further analysis.



Branch Registration Listener – Pseudoco

```
1. LISTEN FOR INCOMING REQUESTS
2. RECEIVE SUBSCRIBE REQUEST FROM BRANCH AGENT
3. SAVE BRANCH MANIFEST
4. GO TO 1
```

- Branch Log Listener

In this thread, the recommender agent listens to agents that send a message with CALL FOR PROPOSAL performative. The content of this message contains the complete log of a day to be analyzed. Given the fact that we have the manifest of this branch agent in our records, we can provide accurate recommendation to the branch agent

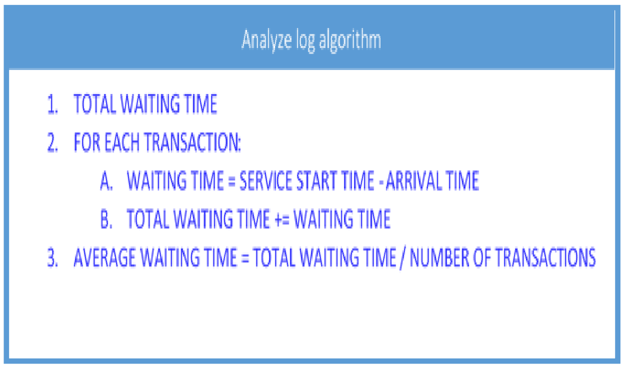
In the above pseudocode, there are two important steps we analyze below.

When adding a desk, we choose between three types of desks (expensive but has a very small processing time, moderate with acceptable waiting time or cheap but has a large processing time). Removing a desk is arbitrary, that is we choose a desk at random to remove.

Local recommendation is finished when the recommender reaches a steady state for that branch. At this point, the recommender send the final manifest file to the branch agent.

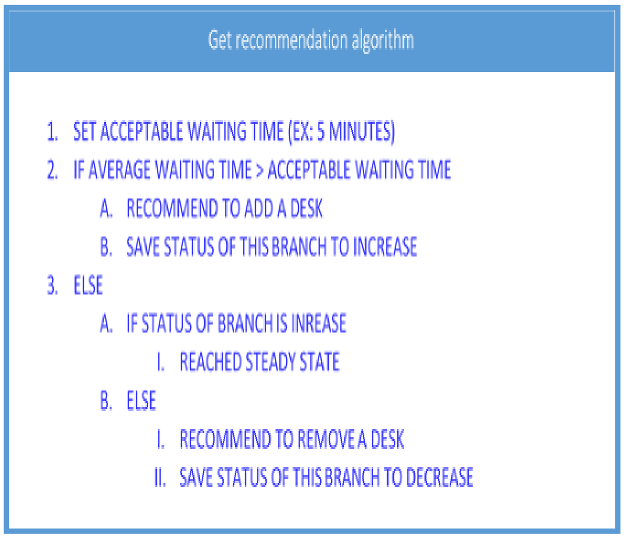
6. Testing Experiments and System Verification

To prove that our algorithm is true, we performed a set of experiments with different generated data to test our use cases.



```
Analyze log algorithm

1. TOTAL WAITING TIME
2. FOR EACH TRANSACTION:
    A. WAITING TIME = SERVICE START TIME - ARRIVAL TIME
    B. TOTAL WAITING TIME += WAITING TIME
3. AVERAGE WAITING TIME = TOTAL WAITING TIME / NUMBER OF TRANSACTIONS
```



```
Get recommendation algorithm

1. SET ACCEPTABLE WAITING TIME (EX: 5 MINUTES)
2. IF AVERAGE WAITING TIME > ACCEPTABLE WAITING TIME
    A. RECOMMEND TO ADD A DESK
    B. SAVE STATUS OF THIS BRANCH TO INCREASE
3. ELSE
    A. IF STATUS OF BRANCH IS INREASE
        I. REACHED STEADY STATE
    B. ELSE
        I. RECOMMEND TO REMOVE A DESK
        II. SAVE STATUS OF THIS BRANCH TO DECREASE
```

6.1. Local Recommendation

6.1.1. Experiment(1)

Objective:

To reduce the waiting time in a branch.

Operation Result

In this experiment, we have set up a branch that has a small number of desks and generated heavy traffic for it. From the resulting processed log file, we can see that the last client would leave the branch at 9:00 PM (while he entered before the branch closes at 5:00 PM). This was because the number of desks couldn't satisfy the arrival rate of the clients.

From this experiment, we expect the recommender to give another manifest file that has a larger number of desks that together reduces the waiting time to the green scale. Here is a typical conversation between the recommender and the branch agent:

```
recommender:recommender agent -> started ..

Branch b1 is ready.

Branch b1 has processed all clients ..

recommender:recommender -> received manifest ..

recommender:recommender -> BankBranch{branchID=b1, branchSize=headquarter, services=[deposit, withdraw, balance, checks, open, transfer, atm, wire, loan], availableDesks=[Desk{deskID=d1, cost=200.0, averageProcessingTime=3}, Desk{deskID=d1, cost=200.0, averageProcessingTime=3}, Desk{deskID=d2, cost=100.0, averageProcessingTime=8}, Desk{deskID=d2, cost=100.0, averageProcessingTime=8}, Desk{deskID=d3, cost=50.0, averageProcessingTime=12}, Desk{deskID=d3, cost=50.0, averageProcessingTime=12}]}

recommender:recommender -> received log ..

b1:branch -> successfully submitted manifest

b1:branch -> recommender says: received manifest successfully

recommender - > average waiting time for b1 is 128.078 minutes

recommender - > sending increase recommendation for b1 with new manifest ->

BankBranch{branchID=b1, branchSize=headquarter, services=[deposit, withdraw, balance, checks, open, transfer, atm, wire, loan], availableDesks=[Desk{deskID=d1, cost=200.0, averageProcessingTime=3}, Desk{deskID=d1, cost=200.0, averageProcessingTime=3}, Desk{deskID=d2,
cost=100.0,          averageProcessingTime=8},
Desk{deskID=d2,                        cost=100.0,
averageProcessingTime=8},  Desk{deskID=d3,
cost=50.0,          averageProcessingTime=12},
Desk{deskID=d3,                          cost=50.0,
averageProcessingTime=12},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3}]}

b1:branch -> sent updated log based on
recommender proposal ..

recommender:recommender -> received log ..

recommender - > average waiting time for b1 is
42.31400000000001 minutes

recommender  -  >  sending  increase
recommendation for b1 with new manifest ->

BankBranch{branchID=b1,
branchSize=headquarter,      services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0,          averageProcessingTime=3},
Desk{deskID=d1,                        cost=200.0,
averageProcessingTime=3},  Desk{deskID=d2,
cost=100.0,          averageProcessingTime=8},
Desk{deskID=d2,                        cost=100.0,
averageProcessingTime=8},  Desk{deskID=d3,
cost=50.0,          averageProcessingTime=12},
Desk{deskID=d3,                          cost=50.0,
averageProcessingTime=12},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3}]}

b1:branch -> sent updated log based on
recommender proposal ..

recommender:recommender -> received log ..

recommender - > average waiting time for b1 is
0.11874999999999999 minutes

recommender - > reached steady state for b1
with the following manifest

BankBranch{branchID=b1,
branchSize=headquarter,      services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0,          averageProcessingTime=3},
Desk{deskID=d1,                        cost=200.0,
averageProcessingTime=3},  Desk{deskID=d2,
cost=100.0,          averageProcessingTime=8},
Desk{deskID=d2,                        cost=100.0,
averageProcessingTime=8},  Desk{deskID=d3,
cost=50.0,          averageProcessingTime=12},
Desk{deskID=d3,                          cost=50.0,
averageProcessingTime=12},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3}]}

b1:branch -> manifest is now optimized as
follows:

BankBranch{branchID=b1,
branchSize=headquarter,      services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0,          averageProcessingTime=3},
Desk{deskID=d1,                        cost=200.0,
averageProcessingTime=3},  Desk{deskID=d2,
cost=100.0,          averageProcessingTime=8},
Desk{deskID=d2,                        cost=100.0,
averageProcessingTime=8},  Desk{deskID=d3,
cost=50.0,          averageProcessingTime=12},
Desk{deskID=d3,                          cost=50.0,
averageProcessingTime=12},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3},
Desk{deskID=rec1,                      cost=200.0,
averageProcessingTime=3}]}
```

As you can see in the above conversation, the recommender suggested adding two desks to reduce the average waiting time below the acceptable limit.

6.1.2. Experiment 2

Objective:
To reduce the cost of desks in a branch.

Operation Result
In this experiment, we have set up a branch that has the same number of desks but generated little traffic

for it. From the resulting processed log file, we can see that the branch will just close in time. This was because the number of desks were very satisfactory to clients' needs.
From this experiment, we expect the recommender to give another manifest file that has a less number of desks that reduces the cost of operating this branch, yet has an average waiting time in the green scale. Here is a typical conversation between the recommender and the branch agent:

```
commender:recommender agent -> started ..

Branch b1 is ready.

Branch b1 has processed all clients ..

recommender:recommender -> received
manifest ..

recommender:recommender ->
BankBranch{branchID=b1,
branchSize=headquarter, services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0, averageProcessingTime=3},
Desk{deskID=d1, cost=200.0,
averageProcessingTime=3}, Desk{deskID=d2,
cost=100.0, averageProcessingTime=8},
Desk{deskID=d2, cost=100.0,
averageProcessingTime=8}, Desk{deskID=d3,
cost=50.0, averageProcessingTime=12},
Desk{deskID=d3, cost=50.0,
averageProcessingTime=12}]}

recommender:recommender -> received log ..

b1:branch -> successfully submitted manifest

b1:branch -> recommender says: received
manifest successfully

recommender:recommender - > average waiting
time for b1 is 0.0 minutes

recommender:recommender - > sending
decrease recommendation for b1

New manifest -> BankBranch{branchID=b1,
branchSize=headquarter, services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0, averageProcessingTime=3},
Desk{deskID=d2, cost=100.0,
averageProcessingTime=8}, Desk{deskID=d2,
cost=100.0, averageProcessingTime=8},
Desk{deskID=d3, cost=50.0,
averageProcessingTime=12}, Desk{deskID=d3,
cost=50.0, averageProcessingTime=12}]}

b1:branch -> sent updated log based on
recommender proposal ..

recommender:recommender -> received log ..

recommender - > average waiting time for b1 is
26.334 minutes

recommender - > sending increase
recommendation for b1 with new manifest ->

BankBranch{branchID=b1,
branchSize=headquarter, services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0, averageProcessingTime=3},
Desk{deskID=d2, cost=100.0,
averageProcessingTime=8}, Desk{deskID=d2,
cost=100.0, averageProcessingTime=8},
Desk{deskID=d3, cost=50.0,
averageProcessingTime=12}, Desk{deskID=d3,
cost=50.0, averageProcessingTime=12},
Desk{deskID=rec2, cost=100.0,
averageProcessingTime=5}]}

b1:branch -> sent updated log based on
recommender proposal ..

recommender:recommender -> received log ..

recommender - > average waiting time for b1 is
0.038999999999999986 minutes

recommender - > reached steady state for b1
with the following manifest

BankBranch{branchID=b1,
branchSize=headquarter, services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0, averageProcessingTime=3},
Desk{deskID=d2, cost=100.0,
averageProcessingTime=8}, Desk{deskID=d2,
cost=100.0, averageProcessingTime=8},
Desk{deskID=d3, cost=50.0,
averageProcessingTime=12}, Desk{deskID=d3,
cost=50.0, averageProcessingTime=12},
Desk{deskID=rec2, cost=100.0,
averageProcessingTime=5}]}

b1:branch -> manifest is now optimized as
follows:

BankBranch{branchID=b1,
branchSize=headquarter, services=[deposit,
withdraw, balance, checks, open, transfer, atm,
wire, loan], availableDesks=[Desk{deskID=d1,
cost=200.0, averageProcessingTime=3},
Desk{deskID=d2, cost=100.0,
averageProcessingTime=8}, Desk{deskID=d2,
cost=100.0, averageProcessingTime=8},
Desk{deskID=d3, cost=50.0,
averageProcessingTime=12}, Desk{deskID=d3,
cost=50.0, averageProcessingTime=12},
Desk{deskID=rec2, cost=100.0,
averageProcessingTime=5}]}
```

As you can see in the above conversation, the recommender suggested first to remove a desk. The agent then processed the same log according to this new settings. However, it increased the average waiting time above the acceptable limit. So, the recommender advises to add a desk, but this time it adds a cheaper desk from the one removed. This satisfies the steady state. In fact in this scenario, it didn't remove a desk completely but suggested a cheaper desk. In other scenarios, the recommender may ask to remove more than one desk.

6.2. Global Recommendation

In this section, we add another layer to the recommender. This global recommendation layer looks at all recommendations given to all branches on our platform. It then suggests either a swap between two desks in two different branches (an expensive desk for a cheaper one and vice versa) or a desk transfer from one desk to another.

The global recommender simply works as follows:

- STEP 1: Keep track of the number of desks added to or removed from each branch.
- STEP 2: Iterate through the list to provide pair-wise transfer recommendation.
- STEP 3: Keep track of the utilized waiting time for each branch.
- STEP 4: Provide swap recommendation for two branches that satisfy the condition that one of them has a waiting time of zero and the other has a waiting time close to the acceptable waiting time set before.

6.2.1. Experiment 3

Objective:

To provide a recommendation that transfers desks from one branch to another.

Operation Result

In this experiment, we have set up a branch that has small number of desks with heavy traffic in the log file. This branch should have a local recommendation to increase the number of desks. In addition, we have set up two branches that has large number of desks but less traffic. Both of these branches got a local recommendation to remove desks to reduce the total cost.

From this experiment, we expect the global recommender to provide us with a recommendation to transfer desks from the second or third branches to the first branch. Here is the output generated by the global recommender for this setting (we have eliminated local recommendation output for simplicity):

```
:recommender -> global recommender started
analysis on 3 branches ..

:global recommender -> transfer from b3 to b1

:global recommender -> transfer from b2 to b1
```

As you can see in the above output, the global recommender ran on three branches and suggested to either transfer a desk from the third branch to the first one or from the second branch to the first one.

6.2.2. Experiment 4

Objective

To provide a recommendation that swaps one desk for another in two branches.

Operation Result

In this experiment, we have set up a branch that has small number of desks with heavy traffic in the log file. This branch should have a local recommendation to increase the number of desks. In addition, we have set up another branches that has large number of desks but less traffic. This branch got a local recommendation to remove desks to reduce total costs.

After the global recommender advises to transfer one desk from one branch to another, it also advises to swap an expensive desk from the branch that has a waiting time of zero with a cheaper desk from the branch that has larger waiting time (note that it is still in the acceptable range of waiting time). This recommendation is done for greater utilization. Here is the output generated by the global recommender for this setting (we have eliminated local recommendation output for simplicity):

These experiments can easily be reproduced using our source code . Note that this output might be slightly different according to each branch settings (manifest file) as well as the log data of clients. Exact output is not guaranteed but the operation of the algorithm is tested on all cases.

5.3. Concluding Remarks

In this case study, we have introduced a multi-agent system that acts as a recommendation system for bank operations. As banks try to do their best to improve customer satisfaction level, this system comes to address the problem of increasing waiting time. It analyzes the log files of customer operations according to the bank settings set before. The system provides generous insights on the waiting time of clients and how to reduce it. In addition, it provides recommendation on how to best utilize resources available across all branches and re-distribute them to meet customer needs.

In conclusion, Bank Operations Multi-Agent System has provided a tested attempt to offer better bank services to the client by reducing the waiting time and the cost of operation. The added value of our branch is that unlike other customer survey systems that are biased toward customer opinion, our system is deterministic and guarantees satisfaction based on actual measurements taken from the history of operation.

The Bank Operations MAS can easily be extended to address more metrics than what we used (waiting time and desk costs). The recommendation system will provide better advises with the increasing number of branches on the platform. All of these ideas aim at improving customer satisfaction level at banks in a deterministic measurement.

6. Conclusions

A Multiple Agent System (MAS) is a computerized system composed of multiple interacting intelligent agents within an environment. This study presents a MAS framework for improving enterprise performance. The new framework describes an evolutionary improvement path from an ad hoc, immature process to a mature disciplined process. It covers the practices for planning, engineering and managing enterprise performance. The new framework consists of a set of agents with different functions.

The framework consists three categories of agents where number of agents from each category depends on the size and process of enterprise. These categories are:

1) Info Agent: collects all possible performance information and send it to the measurements repository.

2) Evaluation Agent: identifies a cluster of related activities that, when performed collectively, achieve a set of goals necessary for enhancing the performance of the target enterprise.

3) Quality agent: identifies issues that must be addressed to achieve the target performance.

MAS architecture and operation is described using the Computation Structure Model (CSM) [10]. CSM provides information about data interaction among different MAS agents. It also describes in details the algorithm of operating MAS agents including a feedback path required to continuously improve the target enterprise performance.

Finally we use the banking system as a case study to show the merits of our approach. Results prove that target performance can be achieved using the new approach in few cycles of feedback.

References:

[1] Yoav Shoham and Kevin Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations* Cambridge University Press, 2009

[2] E. M. Davidson, S. D. J. McArthur, J. R. McDonald, T. Cumming, and Watt, “Applying multi-agent system technology in practice: Automated management and analysis of SCADA and digital fault recorder data,” IEEE Trans. Power Syst., vol. 21, no. 2, pp. 559–567, May 2006.

[3] S. D. J. McArthur, S. M. Strachan, and G. Jahn,“The design of a multiagent transformer condition monitoring system,” IEEE Trans. Power Syst., vol. 19, no. 4, pp. 1845–1852, Nov. 2004.

[4] McArthur, S. D. J.; Davidson, E. M.; Catterson, V. M.; Dimeas, A. L.; Hatziargyriou, N. D.; Ponci, F.; Funabashi, T “Multi-Agent Systems for Power

Engineering Applications—Part I: Concepts, Approaches, and Technical Challenges IEEE Transactions on Power Systems, Vol. 22, no. 4, pp. 1743-1752, Nov. 2007.

[5] McArthur, S. D. J.; Davidson, E. M.; Catterson, V. M.; Dimeas, A. L.; Hatziargyriou, N. D.; Ponci, F.; Funabashi, T., "Multi-Agent Systems for Power Engineering Applications—Part II: Technologies, Standards, and Tools for Building Multi-agent Systems,"., IEEE Transactions on Power Systems, Vol. 22, no. 4, pp. 1753-1759, Nov. 2007.

[6] D. Koesrindartoto, S. Junjie, and L. Tesfatsion, "An agent-based computational laboratory for testing the economic reliability of wholesale power market designs," in Proc. IEEE Power Eng. Soc. General Meeting, 2005, Jun. 2005, pp. 931–936.

[7] S Chowdhury; S P Chowdhury; P Crossley, *Microgrids and active distribution networks,* Steven age: Institution of Engineering and Technology, 2009.

[8] D P Buse; Q H Wu, "IP network-based multi-agent systems for industrial automation: information management, condition monitoring and control of power systems," London, Springer, 2007.

[9] Genc, Zulkuf; et al. (2013). "Agent-based information infrastructure for disaster management". Intelligent Systems for Crisis Management: 349–355.

[10] Ammar, Reda "Hierarchical Performance Modeling and Analysis of Distributed Software", Chapter 12, Handbook of Parallel Computing: Models, Algorithms, and Applications, edited by S. Rajasekaran and J.H. Reif, Chapman & Hall/CRC Press, December 2007.

[11] Niazi, Muaz; Hussain, Amir (2011). "Agent-based Computing from Multi-agent Systems to Agent-Based Models: A Visual Survey". Scientometrics (Springer) 89 (2): 479–499. doi:10.1007/s11192-011-0468-9.

[12] Naveh, Isaac. "Simulating Organizational Decision-Making Using a Cognitively Realistic Agent Model". Journal of Artificial Societies and Social Simulation.

[13] bKubera, Yoann; Mathieu, Philippe; Picault, Sébastien (2010), "Everything can be Agent!" (PDF), Proceedings of the ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2010) (Toronto, Canada): 1547–1548

[14] Salamon, Tomas (2011). Design of Agent-Based Models. Repin: Bruckner Publishing. p. 22. ISBN 978-80-904661-1-1.

[15] Giacomo Cabri ,Letizia Leonardi ,Franco Zambonelli ,"Mobile-Agent Coordination Models for Internet " , IEEE , 2000.

[16] Panait, Liviu; Luke, Sean (2005). "Cooperative Multi-Agent Learning: The State of the Art" (PDF). Autonomous Agents and Multi-Agent Systems 11 (3): 387–434. doi:10.1007/s10458-005-2631-2.

[17] Anderson, Thomas and Dahlin, Michael, *Operating Systems: Principles and Practice,* Recursive Books; 2 edition (August 21, 2014).

[18] Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood, "Developing Multi-Agent Systems with JADE", Wiley 2007.

[19] http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf, and http://jade.tilab.com