Performance Engineering in Agent-based Systems Concepts, Modelling and Examples

Cornelius Wille, Reiner Dumke, Stanimir Stojanov[+]

Otto-von-Guericke-Universität Magdeburg,
Fakultät für Informatik, Postfach 4120,
39016 Magdeburg,
Tel.: 0391-67-18664,
Fax: 0391-67-12810
(wille,dumke)@ivs.cs.uni-magdeburg.de

[+]University of Plovdiv, Dept. of Mathematics and Informatics,
4000 Plovdiv, 236 Bulgaria blv.,
Bulgaria, csstani@pu.acad.bg

Abstract

This paper presents a kind of application of the software performance engineering to the area of agent-based systems.

In a first part we will describe the general aspects and contents of multi agent systems (MAS) architectures. Then, a short presentation of the main software performance principles should motivate the measurement approaches for the MAS. Software agents can be intelligent as well as flexible. This involves the possibility of using traditional methods of software performance evaluation and controlling based on some classical approaches like "a posteriori" or "fix it later" technologies.

*Our paper presents new approaches in performance measurement of agent-based systems relating to software aglets and MAS based on the new **MALINA** concept.*

Keywords: Software performance engineering, multi agent systems, software aglets, agent performance, MALINA concept

1 Introduction

Software agents combine a lot of facilities and features such as mobility, intelligence, reactivity, adaptability, autonomy etc. In order to consider the measurement points on an agent or agent-based system it is necessary to analyse the potential architecture of the software agent. In the most complex case we can establish the following components of an agent as shown in figure 1.

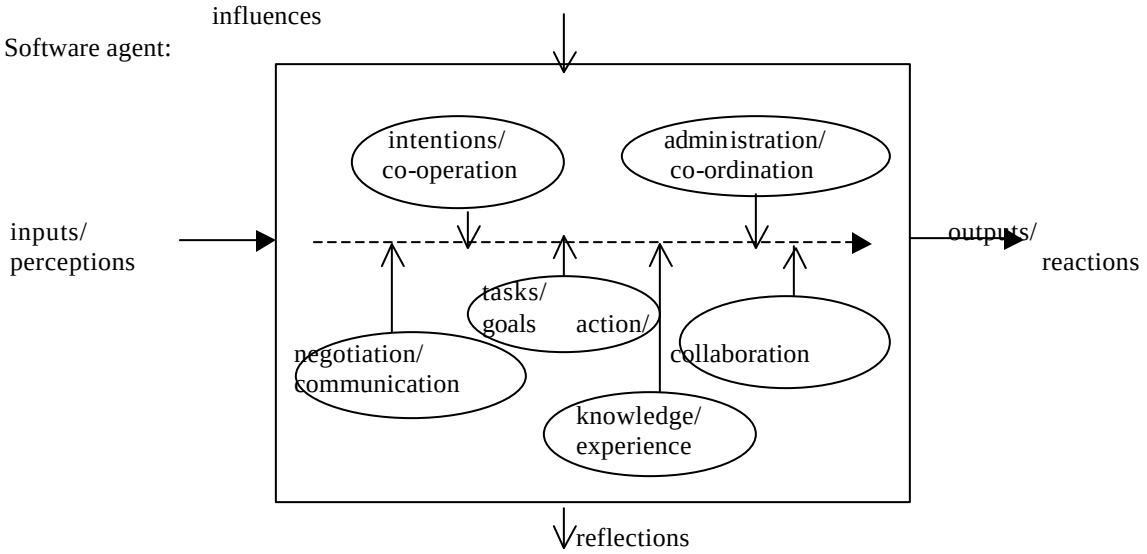

Fig. 1: Components of a general software agent

In the same manner we will characterise the general architecture of an agent-based system (see also ([Dumke 00c] and [Ferber 99]). This model should consider both kinds of the object (as passive object which will be the operation basis, and the active objects as agents themselves). On the other hand, we also consider the fact of usability of a MAS as shown in the figure 2.

Software agent system:

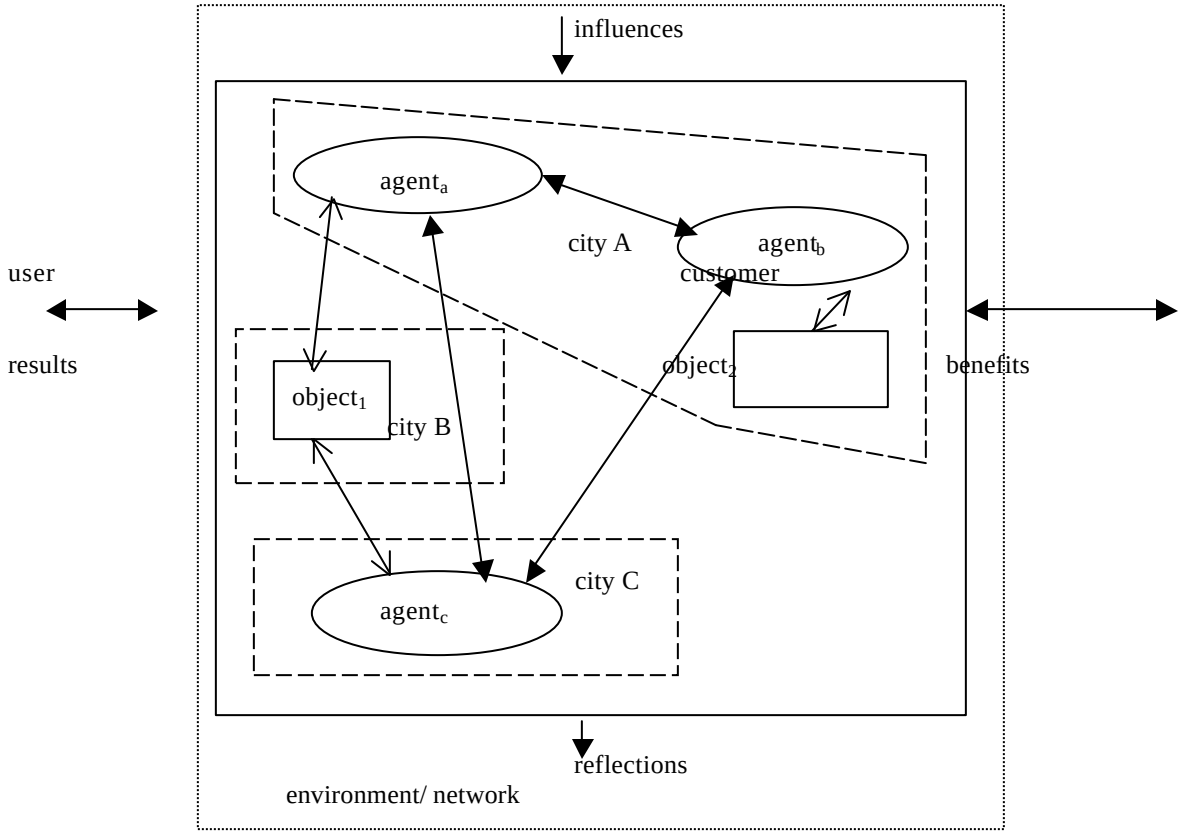

Fig. 2: General architecture of a MAS

Furthermore, agent-based systems intent also some essential *software engineering aspects* which must be also considered in the case of system evaluation and measurement. Such aspects are for instance (see [Dumke 00b], [Hayzelden 99], [Schmietendorf 00a])

- The different stages of an agent and a MAS during the software specification, design and implementing process,
- The different kinds and contents of agent or MAS documentation,
- The different foundations of the MAS implementation such as Aglets, Telescript etc. and the different techniques of implementation, for instance mobile objects,
- The different factors relating to the current basis systems, platforms or net topologies and architectures,
- Last but not least, the different role of quality aspects such as reliability, performance and security.

2 Software Performance Engineering for Agent-based Systems

2.1 Performance Engineering during the Software-Development

The presentation of the performance engineering methodologies should be reduced to the general principles which are defined by [Smith 94] (see also [Dumke 00a] and [Schmietendorf 00]):

P01: ***Fixing-point Principle***: *"For responsiveness, fixing should establish connections at the*

earliest feasible point in time, such that retaining the connection is cost-effective."

P02: ***Locality-design Principle***: *"Create actions, functions, and results that are close to physical computer product."*

P03: ***Processing Versus Frequency Trade-off Principle***: *"Minimize the processing times frequency product."*

P04: ***Shared-resource Principle***: *"Share resources when possible. When exclusive access is required, minimize the sum of the holding time and the scheduling time."*

P05: ***Parallel Processing Principle***: *"Execute processing in parallel (only) when the processing speedup offsets communication overhead and resource contention delays."*

P06: ***Centering Principle***: *"Identify the dominant workload functions and minimize their processing."*

P07: ***Instrumenting Principle***: *"Instrument systems as you build them to enable measurement and analysis of workload scenarios, resource requirements, and performance goal achievement."*

P08: ***Structuring Principle of Physical Concurrency***: *"Introduce concurrency only to model physically concurrent objects or processes."*

P09: ***Tuning Principle***: *"Reduce the mean service time of cyclic functions."*

P10: ***Data Structure Rule***: *"Augment structures with extra information or change the structure so that it can be accessed more easily."*

P11: ***Store Precomputed Results Rule***: *"Compute the results of expensive functions once, store the results, and satisfy subsequent requests with a table look-up."*

P12: ***Caching Rule***: *"Store data that are accessed most often to make the cheapest to access."*

P13: ***Lazy Evaluation Rule***: *"Postpone evaluation until an item is needed."*

P14: ***Packing Rule***: *"Use dense storage representations to decrease storage cost by increasing the time required to store and retrieve data."*

P15: ***Interpreter Rule***: *"Represent common sequences of operations compactly and interpret as required."*

Obviously, most of these principles are addressed to the software product and give only some hints to the software process. Regarding our general MAS architecture we can point out these principles in the presentation in figure 3.

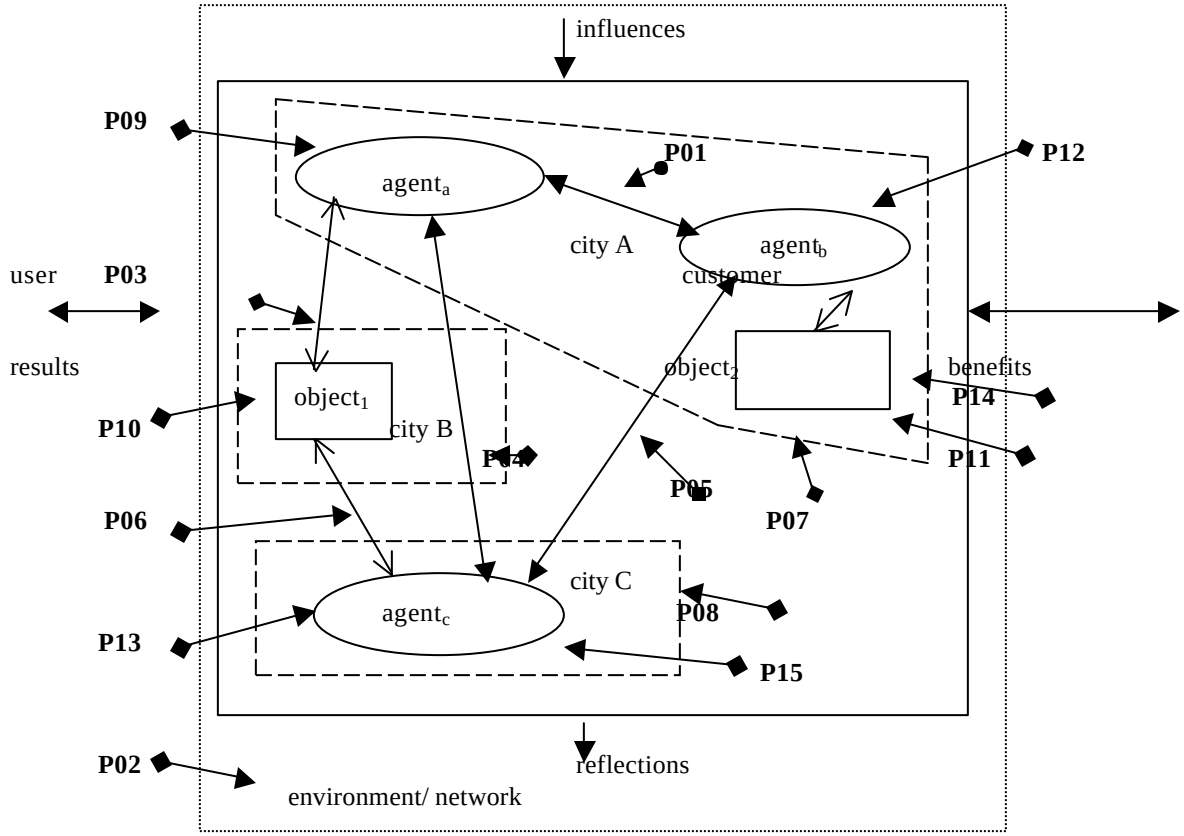

Fig. 3: Principles of performance aspects in an agent-based software system

The possibilities of tool support in the software development and in the object-oriented paradigm lead to the idea of the process evaluation based on software performance engineering defined as Performance Engineering Maturity Model (**PEMM**) [Schmietendorf 99]. This method of evaluation also implies an improvement of the general software development process caused by the earlier consideration of the performance aspects. The techniques to achieve these goals are performance-based prototyping, costs estimation etc. (see [Schmietendorf 01]).

2.2 Fundamental Kinds of Performance for Agent-based Systems

Now, we will define a whole set of performance metrics related to the product, process and resources technologies and components of the software agents and the MAS. In considering the above remarks, we should involve all the general aspects of software agents kinds and their facilities, interactions and intentions. Note, that the metrics-based analysis of the agent behaviour is one of the new and extended areas in software measurement of agent-based systems (see [Dumke 00]). The performance metrics set is described in the following Table 1.

Product Performance Metrics	
Software Agents	***Agent-based System***
Agent design level: ➢ *Software agent size:* a large agent size can cause a low performance and mobility (as **implicit performance**) ➢ *Software agent component structure:* the structure does affect the coupling effects (as **structure performance**) ➢ *Software agent complexity:* a high computational complexity leads to a weak performance (as **immanent performance**) ➢ *Software agent functionality:* a high functionality can injure the performance (as **action performance**)	**System design level:** ➢ *Agent system size:* a small agent system size can cause an overhead (as **potential performance**) ➢ *Agent system component structure:* the system structure relates to the distributed performance (as **architecture performance**) ➢ *Agent system complexity:* this aspect influences the system applicability (as **entropy performance**) ➢ *Agent system functionality:* the system functionality affects their efficiency (as **model performance**)
Agent description level: ➢ *Software agent development description level:* the description level determines the maintainability of an agent (as **change performance**) ➢ *Software agent application description level:* this evaluation considers the usability of an software agent (as **usability performance**) ➢ *Software agent publication description level:* a high publication level supports the spreading of the agent use (as **distribution performance**)	**System description level:** ➢ *Agent system development description level:* the system description affects of system maintenance (as **maintenance performance**) ➢ *Agent system application description level:* a good application description is a precondition for an efficient use of the whole system (as **using performance**) ➢ *Agent system publication description level:* a good system publication supports the spreading (as **marketing performance**)

Product Performance Metrics	
Software Agents	***Agent-based System***
Agent working level: ➢ *Software agent communication level:* a high communication intensity can affect a flexible application (as **communication performance**) ➢ *Software agent interaction level:* this aspects expresses the activity of an agent (as **interaction performance**) ➢ *Software agent learning level:* this level is based on the type of an agent and his roles in the system (as **learning performance**) ➢ *Software agent adaptation level:* the facility of adaptation of the agent implementation (as **adaptation performance**) ➢ *Software agent negotiation level:* this level determines the success of an agent activity relating to common tasks (as **negotiation performance**) ➢ *Software agent collaboration level:* a high collaboration of an agent classify his roles in the given tasks (as **collaboration performance**) ➢ *Software agent co-ordination level:* a high level determines the role of the agent in an administration hierarchy (as **co-ordination performance**) ➢ *Software agent co-operation level:* this level determines the effectiveness of common tasks realisation (as **co-operation performance**) ➢ *Software agent self-reproduction level:* this level determines the stability of an software agent itself (as **reproduction performance**) ➢ *Software agent performance level:* a high agent performance is related to all kinds of agent activities (as **operation performance**) ➢ *Software agent mobility level:* this aspect consider the efficiency relating to the agent movement (as **mobility performance**) ➢ *Software agent specialisation level:* a high specialisation can lead to a high performance (as **suitability performance**)	**System working level:** ➢ *Agent system communication level:* this level characterises the intensity of the conversations (as **advising performance**) *Agent system interaction level:* many interactions are based on a high co-operation (as **team performance**) ➢ *Agent system knowledge level:* this aspect determines the knowledge-based foundation of the agent-based system (as **knowledge performance**) ➢ *Agent system living level:* this aspects is based on the adaptability of the agents and characterises the system maintenance effort (as **life performance**) ➢ *Agent system conflict management level:* a high conflict management causes this aspect (as **conflict solution performance**) ➢ *Agent system community level:* a high community level is caused on collaboration (as **community performance**) ➢ *Agent system management level:* a efficient management determines the agent organisation level (as **management performance**) ➢ *Agent system application level:* this aspect is based on an effective task-oriented agent co-operation (as **application performance**) ➢ *Agent system stability level:* a high stability level includes the agent self-reproduction and other system error handling facilities (as **stability performance**) ➢ *Agent system performance level:* this level includes the agent performance and the performance of the environment (as **processing performance**) ➢ *Agent system flexibility level.* the mobility behaviour of all agents is considered here (as **flexibility performance**) ➢ *Agent system organisation level:* this level leads to an efficient distribution of the agent roles and their administration (as **organisation performance**)

Tab. 1: Performance metrics for the product evaluation of MAS

In the next table we define performance metrics for the process evaluation of software agents and agent-based systems. We will define three aspects per agent characteristics. This leads us to the performance metrics set shown in the Table 2.

Process Performance Metrics	
Software Agents	***Agent-based System***
Agent development life cycle: ➢ *Software agent phases level:* a high phase level also expresses a good efficiency (as **development performance**) ➢ *Software agent milestones level:* this level expresses the correct timing of the agent development (as **cycle time performance**) ➢ *Agent requirements workflow level:* this level is caused by the timely realisation of the requirements for the agent implementation (as **ensuring performance**)	**System development life cycle**: ➢ *Agent system phases level:* this level is caused for an efficient system realisation (as **forming performance**) ➢ *Agent system milestones level:* this level is related to the aspects in the planning time of their realisation (as **realisation performance**) ➢ *System requirements workflow level:* a high workflow level evaluates the appropriateness of the realised system requirements (as **requirements performance**)
Agent development method level: ➢ *Software agent methodology level:* this level means that the development method should be efficient to the kind of agent implementation (as **methodology performance**) ➢ *Software agent paradigm level:* a high paradigm level is caused by an efficiency of the implementation (as **paradigm performance**) ➢ *Software agent CASE level:* this level expresses the tool-support during the agent development (as **tool performance**)	**System development method level:** ➢ *Agent system methodology level:* a high methodology level expresses the use of appropriate development techniques (as **conception performance**) ➢ *Agent system paradigm level:* this level determines the appropriateness of the chosen techniques (as **technology performance**) ➢ *Agent system CASE level:* this level includes the set of different tools in order to support the system development (as **CASE performance**)
Agent development management level: ➢ *Agent project management level:* a high management level is involved in the system project management (as **administration performance**) ➢ *Agent configuration management level:* this level expressed the efficiency of version control for the agent (as **version performance**) ➢ *Agent quality management level:* this level expresses the used quality assurance techniques related to the agent development (as **quality assurance performance**)	**System development management level:** ➢ *System project management level:* this level describes the timing and the appropriate use of resources (as **process performance**) ➢ *System configuration management level:* this level is caused by a version control for all parts of the agent-based system (as **configuration performance**) ➢ *System quality management level:* this level includes the different quality assurance techniques (as **system maturity performance**)

Tab. 2: Performance metrics for the process evaluation of MAS

In the same manner, we will define a set of performance metrics usable for the different kinds and aspects of the resources in the agent-based system development (see Table 3).

Resources Performance Metrics	
Software Agents	***Agent-based System***
Agent developer level: ➢ *Agent developer skill level:* a high skill level causes an efficient developer specialisation for agent implementation (as **skill performance**) ➢ *Agent developer communication level:* the communication is an indicator for an efficient resolving of any questions (as **team level performance**) ➢ *Agent developer productivity level:* a high productivity includes the functionality and the quality of the software agent (as **developer performance**)	**System developer level:** ➢ *System developer skill level:* this level includes different kinds of knowledge (as **experience performance**) ➢ *System developer communication level:* a high communication level is based on the successful participation design techniques (as **team performance**) ➢ *System developer productivity level:* this level is related to the development of the different system components (as **staff performance**)
Agent software resources level: ➢ *Agent software paradigm level:* this level keeps the efficiency of the chosen paradigm (as **component performance**) ➢ *Agent software performance level:* this level is a precondition for the agent performance itself and is related to the used system software (as **agent basis performance**) ➢ *Agent software replacement level:* a high replacement level keeps a good level of agent maintenance and migration (as **replacement performance**)	**System software resources level:** ➢ *System software paradigm level:* this level is divided for the different system components (as **COTS performance**) ➢ *System software performance level:* a high performance level of the used COTS determines the system performance (as **MAS basis performance**) ➢ *System software replacement level:* this level is divided in the evaluation of the different components of the agent-based system (as **migration performance**)
Agent hardware resources level: ➢ *Agent hardware reliability level:* this level includes the different platforms which will be used by a mobile agent (as **reliability performance**) ➢ *Agent hardware performance level:* this level considers also the potential types of platforms (as **hardware performance**) ➢ *Agent hardware availability level:* a high availability level is a precondition for the mobility of an agent (as **availability performance)**	**System hardware resources level:** ➢ *System hardware reliability level:* this level includes all platforms of the implemented environment of the agent-based system (as **system reliability performance**) ➢ *System hardware performance level:* this level is a basis for the efficiency of the agent-based system (as **platform performance**) ➢ *System hardware availability level:* this level expresses the stability of the system use (as **guarantee performance**)

Tab. 3: Performance metrics for the resources evaluation of MAS

This formal listing of performance metrics in the tables above implies the investigation of relations and causalities between these metrics. But, we will not consider these aspects here. Our new approach for agent-based systems evaluation is addressed to the behaviour of the software agents in a MAS. Further investigations can be found in [Dikaiakos 01] [Koblick 99] and [Monasce 98].

2.3 Special Aspects of the Performance Controlling for Software Agents

Now, we concentrate our investigation to the performance evaluation and improvement addressing product aspects. In Figure 1, we have presented the different kinds of components of an agent relating to his autonomous actions. This aspect includes an assumed intelligence of an agent as well as the possibility to control his performance kinds and levels.

Hence, the classical a posteriori techniques of performance controlling and tuning are appropriate in this case. These facilities are shown in Figure 4.

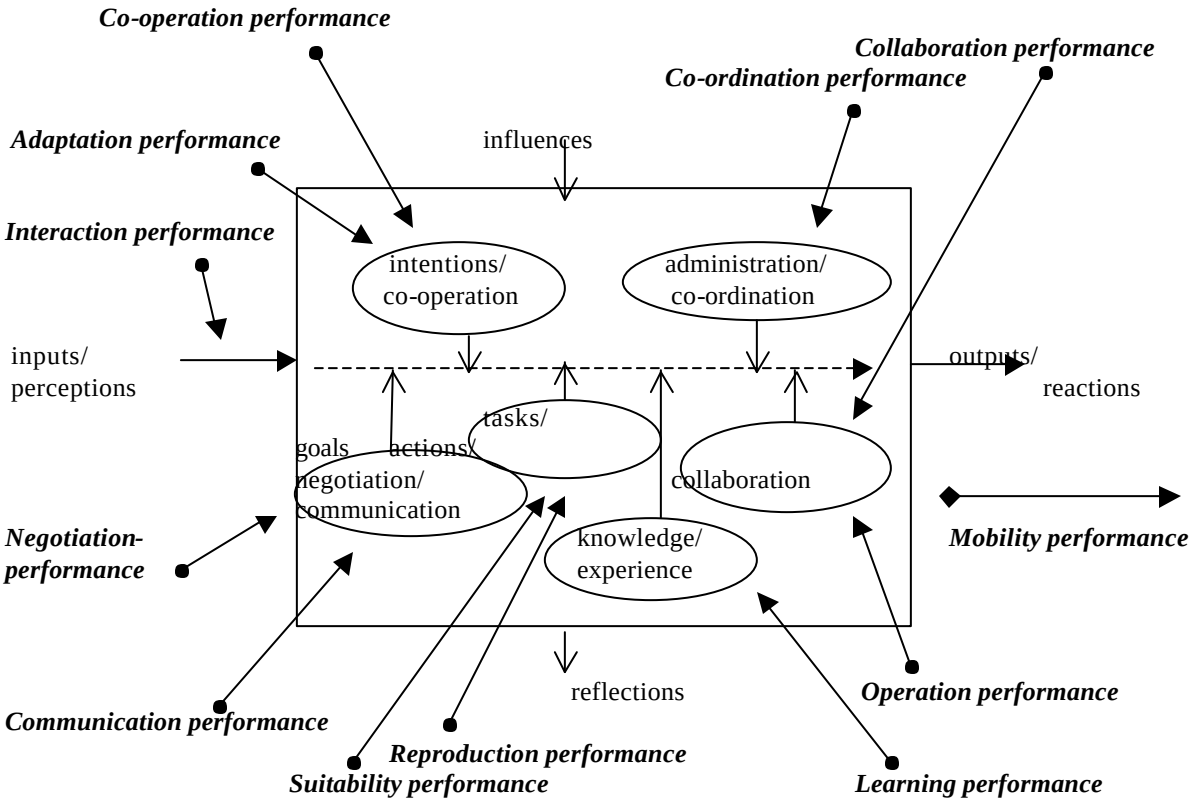

Fig. 4: Performance metrics of agent's run time

In the following, we will define these performance aspects and metrics in an explicit manner in order to execute their characteristics in some cases of their application. The descriptions are given in an alphabetical order.

- ***Adaptation performance:*** The adaptation performance quantifies the time effort for the migration of the software agent in order to keep further tasks in the agent-based system.
- ***Collaboration performance:*** The collaboration performance will be executed through the average time effort for task realisation of an agent relating to the other agents which participate at the task.
- ***Communication performance:*** The communication performance will be expressed by the average time of message sending and receiving in order to work on a common task.
- ***Co-operation performance:*** The co-operation performance includes the summary of the collaboration performance, co-ordination performance, and negotiation performance.
- ***Co-ordination performance:*** The co-ordination performance is caused by the average time effort of the co-ordination between the agents in order to work on a common task.
- ***Interaction performance:*** The interaction performance characterises the time to reply during the interaction of the software agent.
- ***Learning performance:*** The learning performance executes the effort for extension and adaptation of the knowledge of a software agent.
- ***Mobility performance:*** The mobility performance determines the average dwell time of an agent in the visited cities.
- ***Negotiation performance:*** The negotiation performance determines the average time effort for the preparation and co-ordination of a task solution between software agents.
- ***Operation performance:*** The operation performance executes the average time effort for the realisation of the agent tasks/operations.
- ***Reproduction performance:*** The reproduction performance is addressed to the effort of the

regeneration and/or reproduction of the fundamental operations of a software agent.

- ***Suitability performance:*** The suitability performance evaluates the loss of efficiency caused by the redundancy in the functionality of a software agent addressed to the problem solution.

Based on theses definitions, we can execute in a first approximation the appropriate kinds of performance of an agent-based software system. The following table shows a simplified execution of the different types of the MAS operation performance. This approximation excludes the special characteristics and influences of the agent environment and the MAS resources and intends the average execution.

MAS operation performance	An approximate execution
Advising performance:	*average communication performance*
Application performance:	*average co-operation performance*
Community performance:	*average collaboration performance*
Conflict solution performance:	*average negotiation performance*
Flexibility performance:	*average mobility performance*
Knowledge performance:	*average learning performance*
Living performance:	*average adaptation performance*
Management performance:	*average co-ordination performance*
Organisation performance:	*average suitability performance*
Processing performance:	*average operation performance*
Stability performance:	*average reproduction performance*
Team performance:	*average interaction performance*

Tab. 4: Determination of the different kinds of the MAS operation performance

On the other hand, the performance metrics of the run time of an agent are structured considering the whole tasks of the MAS. For instance, we can establish the following groupes of performance characteristics:

- ***Realisation of o task:***
  Co-operation = {communication, negotiation, interaction, collaboration, operation},
- ***Pre activities for task realisation:***
  Preparation = { suitability, mobility },
- ***Post activities after task realisation:***
  Improvement = { learning },
- ***Background of the agent activities:***
  Living facilities = {adaptability, reproduction}.

In the following we will discuss some examples of performance measurement based on concrete software agent systems.

4 First Applications for Aglets-based Systems

4.1 Performance Measurement of Software Aglets

Software aglets are a special kind of a MAS implemented in the Java programming language [Lange 98]. The notion *aglet* is build from the special kind of small Java applications as *applet* and the word *agent*.

Software aglets can be implemented with help of the Aglets Software Development Kit (ASDK) from IBM which defines the MAS environment as ***Tahiti server***. The basic operation of software aglets are:

- ***Creation:*** The definition of a mobile agent in a special context.
- ***Cloning:*** The implementation of a same type of a software agent at a special run time.
- ***Dispatching:*** The transformation of a aglets from one city to another.
- ***Retraction:*** The transformation of the software agent back to his origin.
- ***Activation*** respectively ***Deactivation:*** The possibility of stopping/ starting a software aglets.
- ***Disposal :***The destruction of a software agent in the MAS.

The following Figure 5 shows these basic operations of software aglets in the predefined environment.

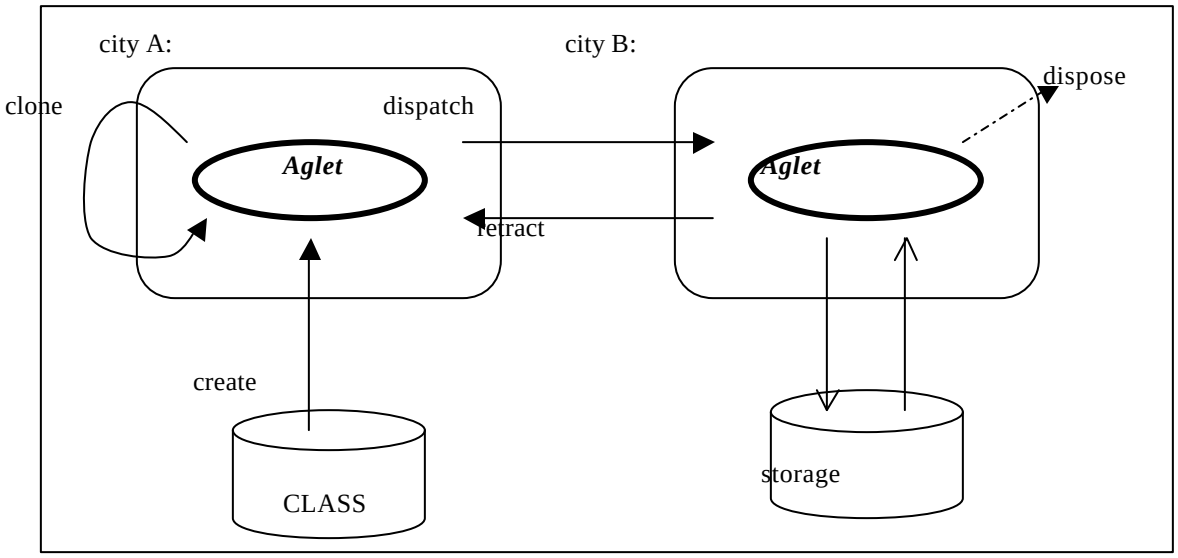

Abb. 5: The aglet life cycle model

In order to demonstrate some aspects of performance analyses, we will present a simple ***measurement agent*** as aglet, that measures the size (as lines of code (LOC)) of objects (as Java classes and/or further aglets) in a special place in the MAS respectively in a city.

Figure 6 shows the general definition of our measurement aglet working in the aglets environment on a special place.

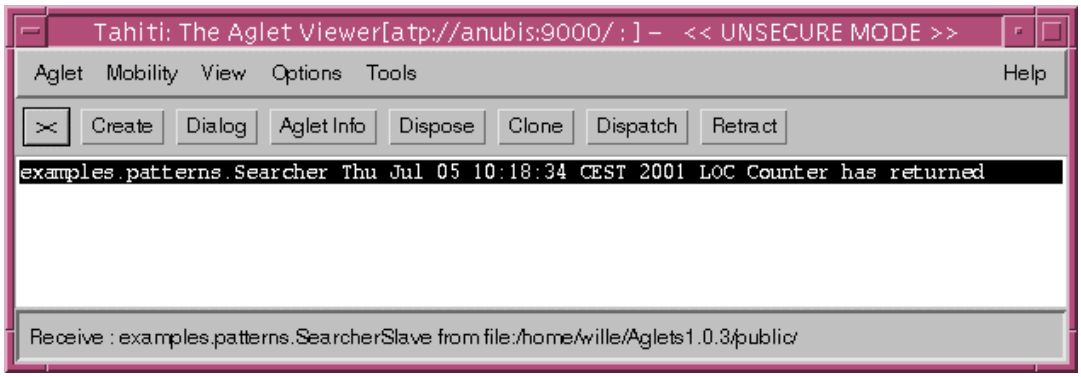

Fig. 6: Creation of a measurement aglet

Figure 7 demonstrates the application of our measurement agent in a local area network.

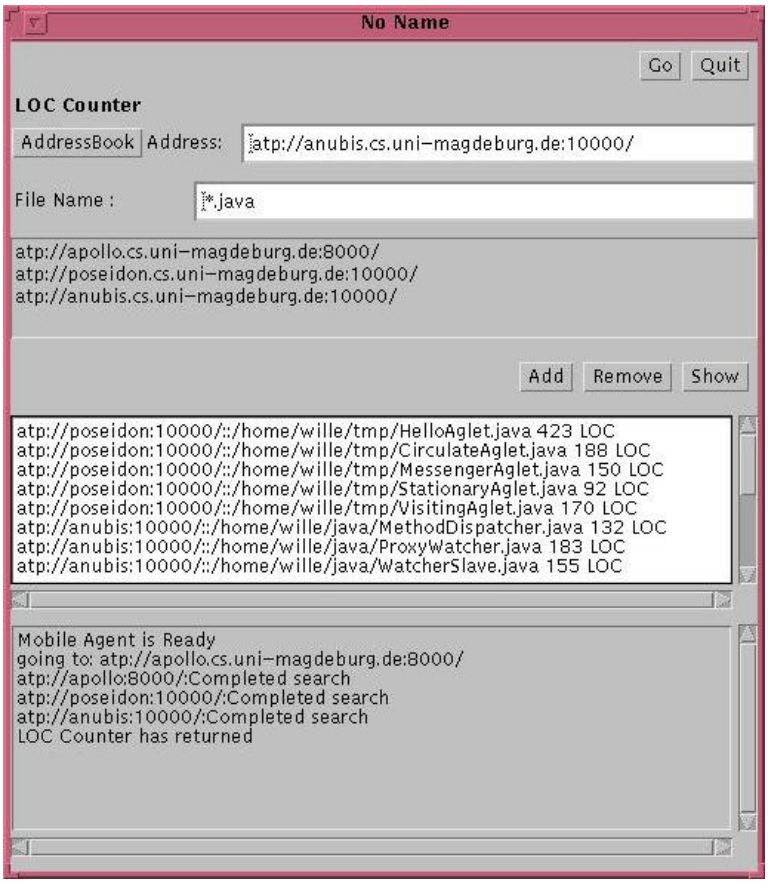

Fig. 7: Application of the measurement aglet for LOC counting

The next figure shows the determination of the ***operation performance*** of a software agent based on the aglets functionality above.

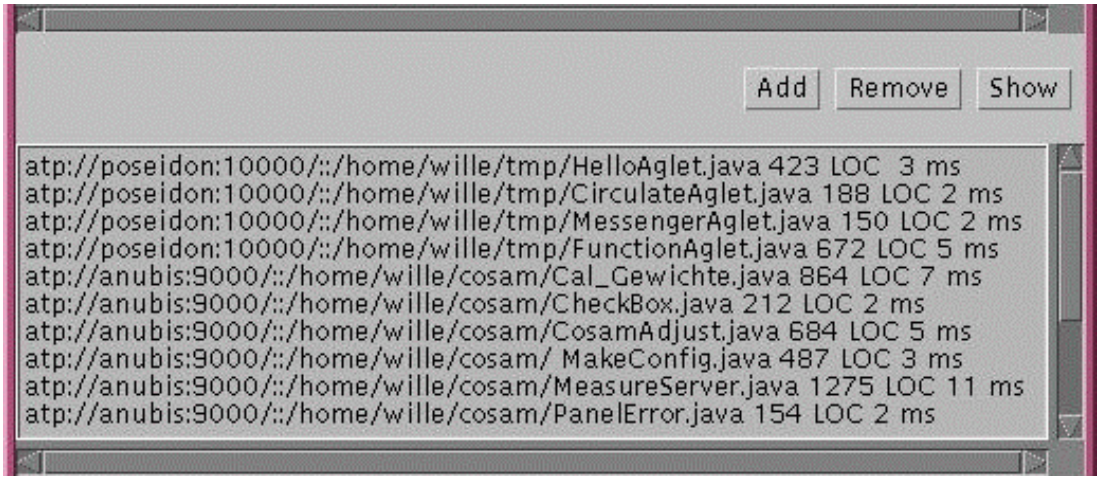

Fig. 8: Protocol of the operation performance of a software agent

Furthermore, we can extend this performance analysis by defining *performance constraints* such as limits of the operation time of the agent. This leads us to an example of an evaluation of the ***collaboration performance***. The following Figure 9 presents such an example of time constraints for our LOC counter aglet above

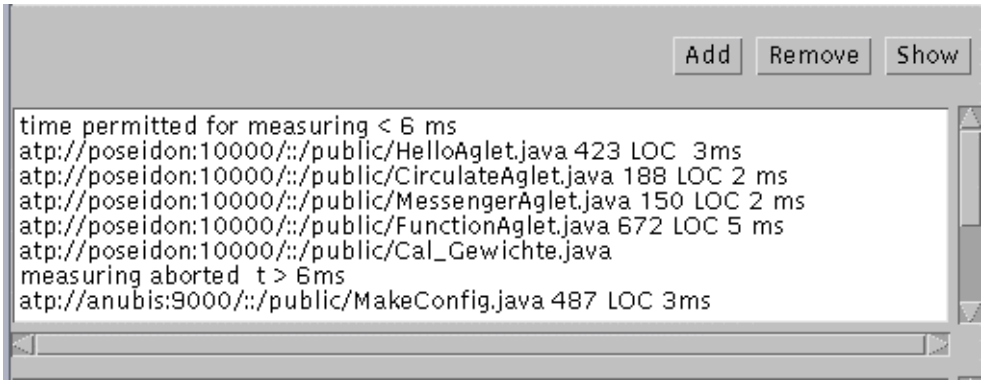

Fig. 9: In tention of the collaboration performance of an aglet

In the next example we will consider the mobility of software agents. Figure 10 shows the dispatching of an aglet from one place to another in a local area network. This aglet produces the city information on every place he has switched.

Fig. 10: Protocol of the mobile aglet producing city information

This aglet can be use in order to demonstrate an example for analysis of the ***mobility performance***. Figure 11 presents an example of a city hopper aglet and prints the dwell time in the special network place.

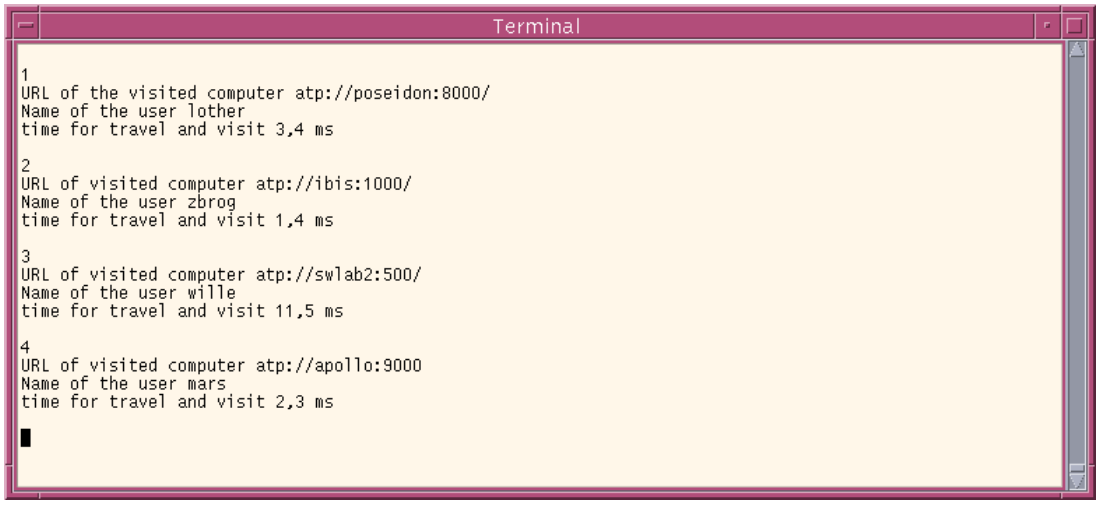

Fig. 11: Protocol of the execution of the mobility performance of an aglet

The analysis of the mobility performance can be extended by the use of constraints for the different work places (as cities) and the use of intelligent algorithms to optimise the aglet operation in special networks.

4.2 Performance Measurement in MAS based on the MALINA Concept

MALINA (Multi-Agent Local Integrated Network Associations) is an environment for the development of multi-agent applications and it is a more precise extension of the concepts given in [Stojanov 96], [Stojanov 98], [Stojanov 00]. This is a more common approach to intelligent agents and multi-agent systems, an approach that does not focus just on the agents' ability to "think" or to make the agent very clever, but also embraces all constitutive parts of one multi-agent system in a vast technology for the development of distributed applications in a very short time.

The technology supports a bottom-up approach for developing multi-agents applications. In MALINA the questions about how to overcome the bounds of intelligence and how to make agents capable to obtain all advantages of the multi-agent society, how they could be able to use other agents' knowledge and to act more "successfully" while trying to solve complex task on behalf of the user are solved with applying three main steps – operational agents' configuration (using an abstract agent architecture), determining of the static infrastructure of the application in terms of "agent cities" and specification of the agent associations. MALINA is specified in following three levels:

- *Abstract level* – at this level a hypothetical infrastructure which provides the theoretical framework of the technology is defined.
- *Conceptual level* – conceptions are decomposition and detailisation of the hypothetical infrastructure in order to prepare the development of supporting programming tools and an appropriate developing environment The following four conceptions which are a basis for the developing environment of the technology are founded.
  - ✓ *Abstract Agent Architecture* – specifies the architecture of an abstract generic agent by help of which the agents of different applications can be configurated
  - ✓ *AgentCities* – specify a static infrastructure for multi-agent applications
  - ✓ *AgentAssociations* – provides various possibilities for building agents associations
  - ✓ *MobileServices* – give different ways for automatic generation of mobile agents.
- *Development level* – this level includes the development tool for the technology.

In the MALINA technology the abstract agent is a genetic structure with the following main parts: agent machine including the agent interface, agent local control, functionality (specialization) of the agent and agent mentality.

Agent interfaces consist of perceptive and influence mechanisms. Using its sensors the agent receives information from other agents and from the multi-agent environment. In our technology that is the city in which the agent lives. Each agent sends information to the other agents through its effectors, and using the

effectors it is capable to change the environment or to operate into the city employing its functionality and specialization or its social role. It is not enough just to send and receive information through sensors and effectors. Mechanisms for processing and understanding these information are needed. In the abstract agent architecture we have separated this process over two phases:

- Agent training – during the configuration of the operational agent the designer has to "teach" the agent what it should do according to received performative (influencing the ***adaptation performance*** and the ***learning performance***).
- Run time processing – this is a practical mechanism for reasoning that enables the agent to use all knowledge obtained during the training (influencing the ***co-operation performance*** and the ***suitability performance***).

The *local control* of the abstract agent is designed in two levels – a meta-level for communication and co-ordination and a planning level including aspects of the ***communication performance, co-ordination performance,*** and of the ***collaboration performance***. In MALINA technology the agents can communicate with each other through KQML (see [Finin 94], [Labrou 97]). The agent *meta-level* is the part from the abstract agent that deals with the processing of the received requests (in form of KQML performatives) and with the construction of the answers (also KQML performatives). It is the mediator between the rough KQML performative and it's processing from the agent corresponding to the internal rules given by the designer. A internal rule looks like following:

performative
(performative_name
:parameter$_1$ <word>
: parameter$_2$ <word>
…
: parameter$_N$ <word>)

$$\frac{\text{action}_1, \text{action}_2, \ldots, \text{action}_N : \text{condition}_1, \text{condition}_2, \ldots, \text{condition}_N}{\text{ResultFromActions} \rightarrow \text{ComposeAnswer}(\text{parameter}_1, \ldots, \text{parameter}_M)}$$

Such rules are interpreted from the meta-level applying rules from the common sense reasoning and default logic. If a performative with name *performative_name* has been received (with the corresponding parameters), then:

1) Check consistency with the conditions given in the list: $condition_1, \ldots, condition_N$.
   - ✓ If there is no contradiction with agent's mental states (intentions and commitments) ⇒ go to step 2.
   - ✓ Else ⇒ ask for the next received performative.
2) Perform the partial plan from list: $action_1, action_2, \ldots, action_N$;
3) According to the result from plan execution ⇒ compose an answer to the corresponding agent.

Depending on agent's social role in the city or multi-agent society it inhabits different performatives are included into the agent's meta-level.

Agents are computational systems that inhabit and interact with dynamic, and not entirely predictable environments (relating to the ***interaction performance***). They decide for themselves, on the basis of their individual beliefs, goals, etc., how to respond to the environment and other agents (relating to the ***negotiation performance***). In the MALINA technology we have separated the different mental states that give to the agent all the knowledge it has about the world to make conclusions and the processing mechanism of these mental states. This mechanisms referring to the agent's mentality are part of the abstract agent's local control. In the case of a minimal agent's capability we need beliefs, intentions and commitments as agent's mental states. An agent's beliefs include beliefs concerning the world, beliefs concerning the other agents in the MAS, and beliefs concerning the agent itself. The beliefs of the agent may be incomplete and insufficient and if classical representations are used the agent will not be able to solve a lot of problems if it always needs all the information to be present in its beliefs. To cope with this problem we use principals of default logic in the presentation and interpretation of agent beliefs. So the local control of agents needs a mechanism to process complex default rules, and to make conclusions, to ask questions using common sense reasoning and default logic principals. An agent may update its beliefs by observing the world and by receiving messages from other agents using its sensors and effectors.

The *Agent Cities* concept specifies one open infrastructure for modelling multi-agent systems in distributed environment. An application could be considered as a set of different "cities". Each city includes logically related agents. A city can be constructed in three steps: specification and

identification of the city, definition of the agent locations and distribution of the agents over the city. Each operational agent after its configuration is put into a given city and as an inhabitant of this city the agent participates in the social life of the city. But to perform any social activities agents need mechanisms to communicate with one another and with the institutions of the city. That is why, when the cities are created, they have to be put into operation by means of defining some communication mechanisms. The city consists of three main parts (see Fig. 12).

The *Kernel* manages the communication among the agents. It supports two address spaces (logical and physical) and the transformation between them.

In the MALINA technology we propose a *measurement and evaluation module* which implements some ideas of the approach proposed in this paper. By means of appropriate metrics we try to gather quantity data for the evaluation of the agent behaviour (the local aspect) and of the features of the multi-agents application as a whole (the global aspect). By the local aspect we intend to improve the operation of the separated agent and the loading of their local resources. We distinguish different kinds of resources – devices, system software, information sources etc. The main goal of the global aspect is the optimization of the static infrastructure of multi-agents application and the effective use of the global resources. The two aspects tie close together in relation to some metrics.

In the technological framework two kinds of measurements are possible - off-line and on-line measurements. The *off-line* measurements will be accomplished during the "agents' teaching" phase which is a part of the testing. The measurement's data are visible for the designers of the multi-agents application. In this way they can improve with help of development tools the technology of the static infrastructure of the system. For the support of the off-line measurements we are going to define a new set of specialized KQML-performatives. The *on-line* measurements and evaluations take place during the running of the application (run-time measurements). The gathered data are used mainly for supplying of the city's mobility module. This kind of the measurements are transparent for the application designers and users. They cause dynamically changes of the system which can be only booked.

A good software technology must give an account of its ability to supporting applications with high performance and effectiveness (especially their ***operation performance***) . This point of view is very important for the distributed information systems. A possible way for increasing the performance is the use of mobile services. The MALINA technology provides tools for generating and controlling mobile services in form of mobile agents. Here an approach is proposed based on the notion of *conditional mobility*.

While the static agents are implemented as a separated process over a computer the mobile agents can be moved across the entire net. They can refresh its operation at the new location. From a conceptual point of view these agents can be interpreted as traveling, dynamic, wandering, roaming or migrating. The sense of the mobility is the performance increasing which can be achieved by better use of the available resources. For example, a service (agent) will be moved near to a host, which offers free resources.

In MALINA we understand the mobility as ability to movement of the agents across the "cities". According to the mobile agents conception the mobility is always conditional, e.g. it can be provoked from different reasons (motives). There are systems that support unconditional mobility. In our approach the mobility is interpreted as a dynamic changeable property. This property springs up from the satisfaction of proper conditions. In the mobility conception we assume that the services provided from a multi-agents application are a priori static. By the application configuration the designers and the administrators set the default location of the agents. If the run-time establishs a situation which satisfies the mobility conditions then the mobility status of selected agents can change, i.e. they become mobile. After the mobility necessity get over the run-time restores the static infrastructure. The static infrastructure can be modified only off-line by help of the development tools of the technology. The conditional mobility in MALINA is supported with help of a system for measuring and evaluate the agent behaviour especially their ***mobility performance***).

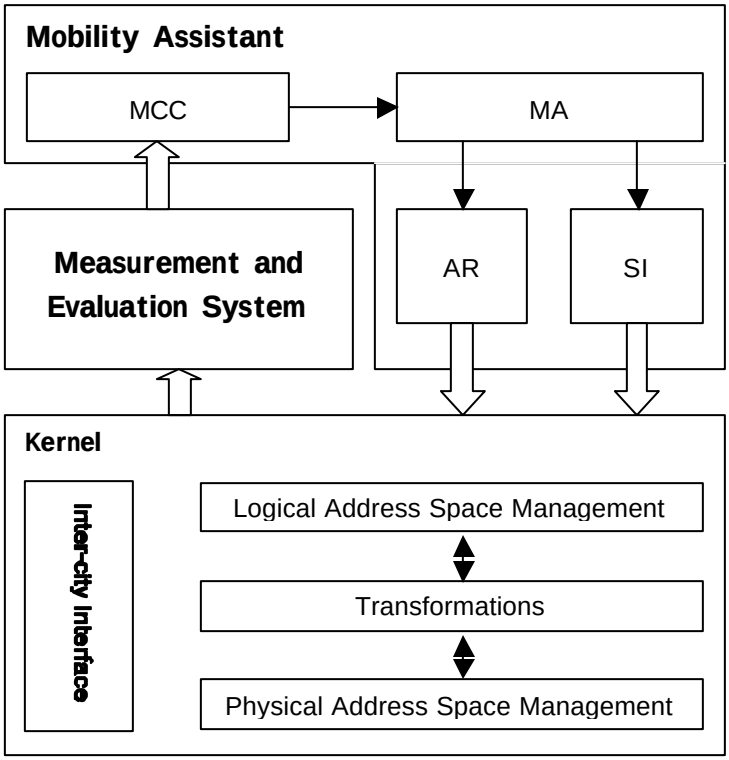

Fig. 12. Architecture of a city

The mobility manager includes the following components:

- *MCC (Mobility Conditions Check)* – this component checks periodicly if any mobility condition is arised. The mobility conditions can be specified during the system's configuration. For checking-up conditions the data provided by the measuring module are used. A set of marked agents that have to become mobile will be returned as result.
- *MA (Mobility Assignment)* – MA assigns to the selected agents the property mobility. The property is limited in the time. After the fixed time point the mobility is not valid. If it is needed the time interval can be extended.
- *AR (Agents Relocation)* – in this module are implemented the following operations:
  - *Copy* – generates a new copy of an agent which is mobile
  - *Move* – an agent will be moved to a new "city" without generating a copy
  - *Clone* – generates a new mobile copy with modified functionality.
- *SI (Static Infrastructure)* – restores the static infrastructure of the application.

There are different conditions to arise of mobility. If a agent wants to get information from some sources located over different platforms it can send a request to all platforms through RPC-like (Remote Procedure Call) methods. However, if the data size is too large then we have to look for solutions to optimize the traffic. The agents can process the removed data more effective when the services, which are necessary, are offered on the removed site. The relocation of the agents can *increase the performance*. Disadvantage of this type of mobility is the fact that the removed site has to give CPU cycles to support the mobile process.

In the current version of this conception there are two scenarios which specify the mobility conditions considering the agent (system) working performance:

- *Mobile assistant* – a set agent has to realize too many requests. In order to accelerate the temp of work the system generates a mobile copy of the agent, which will be located in another "city". The new "city" offers a free resource in this time. The set of requests will be separated between two agents. After the work is done the mobility condition is not valid yet. The run-time restores the static infrastructure.
- *Accelerate the volume of messages* – MCC establish that between two agents which are in

different "cities" run intensive exchange of messages. The system moves one of them in the "city" of the other. So in this way we accelerate the exchange between them. When we finish the condition of mobility isn't satisfied.

5. Conclusion

The software measurement related to the agent-based systems – also the *general* performance aspects - is rarely considered today. Hence, we define at first the general measurement aspects and the intentions of software agents and MAS. Based on a general definition of a software metrics set addressed to the aspects of the product, process and resources in MAS development and application, we have investigated the special characteristics of the agent work performance. Two examples are given in order to demonstrate the general measurement approach: the performance measurement of software aglets and the discussion of performance intentions in a new agent-based system concept of MAS development.
Further investigations are directed to the integration of special performance evaluation methods in more complex aglets and for other MAS technologies.

6. References

[Dikaiakos 01] Dikaiakos, M.D.; Samaras, G.: *Performance Evaluation of Mobile Agents: Issues and Approaches.* In: Dumke et al.: Performance Engineering, Lecture Notes in Computer Science 2047, pp. 148-166

[Dumke 00] Dumke, R.: *Anwendungserfahrungen eines objektorientierten Measurement Framework.* In: Dumke/Lehner (Hrsg.): Software-Metriken, DUV, 2000, S. 71-93

[Dumke 00a] Dumke, R.; Koeppe, R.: *Konzeption einer Web-basierten SPE-Entwicklungsinfrastruktur.* Tagungsband des 1. Workshop Performance Engineering in der Software-Entwicklung (PE2000), Mai 2000, Darmstadt, S. 1-16

[Dumke 00b] Dumke, R.; Koeppe, R.; Wille, C.: *Software Agent Measurement and Self-Measuring Agent-Based Systems.* Preprint Nr. 11, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2000

[Dumke 01] Dumke, R.; Rautenstrauch, C.; Schmietendorf, A.; Scholz, A.: *Performance Engineering – State of the Art and Current Trends.* Lecture Notes in Computer Science 2047, Springer Publ., 2001

[Dumke 00c] Dumke, R.; Schmietendorf, A.; Stojanov, S.: *Komponentenorientierte Entwicklung verteilter Multiagenten-Applikationen.* Tagungsband des 2. Workshop komponentenorientierte betriebliche Anwendungssysteme (WKBA 2), Februar 2000, Wien, S. 69-79

[Ferber 99] Ferber, J.: *Multi-Agent Systems – An Introduction to Distributed Artificial Intelligence.* Addison Wesley Publ., 1999

[Finin 94] Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.: *Specification of the KQML Agent-Communication Language.* The DARPA Knowledge Sharing Initiative External Interfaces Working Group, University of Toronto, 1994

[Hayzelden 99] Hayzelden, A.L.G.; Bigham, J.: *Software Agents for Future Communication Systems.* Springer-Verlag, 1999

[Koblick 99] Kobilck, R.: *Concordia.* Comm. Of the ACM, 42(1999)3, pp. 96-99

[Lange 98] Lange, D.B.; Oshima, M.: *Programming and Developing Java Mobile Agents with Aglets.* Addison-Wesley Verlag, 1998

[Labrou 97] Labrou, Y.; Finin, T.: *A Proposal for a new KQML Specification*. University of Maryland Baltimore County, 1997.

[Monasce 98] Monasce, D.A.; Almeida, A.F.: *Capacity Planning for Web Performance.* Prentice Hall 1998

[Schmietendorf 00] Schmietendorf, A.: *Modellbezogene Notationen, Methoden und Tools für ein Software Performance Engineering.* Preprint Nr. 15, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2000

[Schmietendorf 00a] Schmietendorf, A.; Dimitrov, E.; Dumke, R.: *An Overview about UML-based Performance Engineering.* Preprint Nr. 14, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2000

[Schmietendorf 01] Schmietendorf, A.; Dumke, R.; Foltin, E.: *Risk-driven effort-estimation of task within the software performance engineering.* Proceedings of the ESCOM 2001, April 2001, London, pp. 49-57

[Schmietendorf 99] Schmietendorf, A.; Scholz, A.: *The Performance Engineering Maturity Model*. Metrics News 4(1999)2, S. 41-51

[Smith 94] Smith, C.: *Performance Engineering*. In: Marciniak: Encyclopedia of Software Engineering, Vol. 2, John-Wiley Verlag, 1994, S. 794-810

[Stojanov 96] Stojanov, S.: *A Multi-Agent System for the Solution of Statistical Problems*. Workshop "Concurrency, Specification and Programming", 25-27.09.1996, Berlin, 181-189.

[Stojanov 98] Stojanov, S.; Kumurdjieva, M.: *MASTT-Technology and Its Application in Electronic Commerce Systems*. Workshop "Concurrency, Specification & Programming", September 28-30 1998, Berlin

[Stojanov 00] Stojanov, S.; Kumurdjieva,M.; Dimitrov, E.; Schmietendorf, A.: *Technologicak Framework for Development of Agent-based Applications*, Workshop "Concurrency, Specification & Programming", October 9-11 2000, Berlin, 299-311