Indicators for Self-Diagnosis: Communication-based Performance Measures

Michael Rovatsos[1], Michael Schillo[2], Klaus Fischer[2], and Gerhard Weiß[1]

[1] Department of Informatics, Technical University of Munich,
Boltzmannstraße 3, 85748 Garching, Germany
{rovatsos, weissg}@informatik.tu-muenchen.de
[2] German Research Center for Artificial Intelligence (DFKI),
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
{schillo, kuf}@dfki.de

Abstract. Multiagent systems (MAS) have found their way into industrial applications in recent years and appear to be one of the most promising technologies that originated in AI research in recent years. However, evaluation standards as they are common e.g. in the scheduling or database systems communities are largely amiss. In this paper, we propose *communication-based performance measurement* (CBPM) as a new method that is particularly suitable for open, communication-intensive MAS, and argue that it can be used as to design *indicators for self-diagnosis* by the MAS itself. The ability of such self-diagnosis is a prerequisite for MAS with self-repairing and self-optimising properties required by the *autonomic computing* view. CBPM is based on the idea that important aspects of the external behaviour of a MAS can be measured in terms of the communication processes within them. We present different levels of communication-based performance measurement: frequency analysis of performatives and analysis of complex message patterns. Several examples of analyses of inter-agent communication based on FIPA-ACL and the contract-net protocol in implemented, complex, market-oriented MAS demonstrate the usefulness of our approach. We conclude that these performance measures provide useful information about MAS and pave the way for devising autonomic self-improvement methods for these systems.

1 Introduction

In the last few years, multiagent systems (MAS) have been increasingly successful in industrial applications [9], particularly as a paradigm used for systems that have to operate in complex, dynamic, distributed domains. Recently, *autonomic computing* [10] has been proclaimed as a new approach to industrial systems that are capable of *self-management*, i.e. that have *self-configuring*, *self-healing*, *self-optimising* and *self-protecting* capabilities. It is believed that this can be achieved through a peer-to-peer collaboration of *managers* that control the use of computational resources (databases, networks, storage facilities, etc.), measure system properties and decide what steps to take to improve performance, usability or security in the system.

From a MAS perspective, these managers can be seen as agents that observe the system and take appropriate action. However, this can be particularly difficult in *open*, *communication-intensive* systems (cf. Internet agents, ubiquitious computing) where the internal design of system components is highly encapsulated and hence not always

accessible for manager agents. The increasing popularity of technologies such as Web Services and open platforms for agent-based service deployment such as Agentcities [1] calls for methods to measure and influence the behaviour of system resources at the level of *communication* rather than direct control.

Starting from this observation, this paper develops methods for *communication-based performance measurement* (CBPM) for MAS that are based on an analysis of the communication processes that unfold during operation of these systems. We claim that these measures have the potential to become valuable *performance indicators* when such systems are analysed by manager agents, and that this *self-diagnosis* is capable of providing useful guidance in trying to meet external requirements such as *self-repair* and *self-optimisation*. To this end, we propose several such performance measures and illustrate their usefulness with practical examples.

The remainder of the paper is structured as follows: Section 2 describes briefly our intuitions regarding communication-based performance measurement and the general ideas behind it. Then, in Section 3, we introduce the generic model of MAS that we use to define measures. Section 4 provides detailed definitions for the proposed performance measures. This is followed by illustrative examples in Sections 5 and 6. Finally, we sum up with a discussion in Section 7.

2 Why Communication-based Performance Measurement?

In the process of designing and developing software systems, developers have several attributes of the final product in mind, such as e.g. availability, security, modifiability. Ideally, software engineering methods should give off-the-shelf advice on how to proceed in the process of engineering the system in order to achieve these requirements. This is a core aspect of *performance engineering* [5], where it is common practice to distinguish between

- *internal attributes* of the product (i.e. those which can be measured purely in terms of the product itself) and
- *external attributes* of the product (i.e. those which can only be measured with respect to how the product relates to its environment).

In general, internal attributes are more domain-independent and easy to measure, which does not hold for external attributes. Therefore, it is highly desirable to predict the values of external attributes (e.g. usability and comprehensibility) on the basis of measurements regarding internal attributes (such as resource load, errors during stress testing, etc). From the standpoint of autonomic computing, manager agents take over the role of the performance engineer during operation of the system: they must measure *internal* attributes of the system (such as exchange of information between components, data storage strategies etc.) to be able to predict *external* attributes, such as optimal responsiveness, data security, etc.

When dealing with open systems of communicating agents, managers are additionally confronted with systems in which the precise operation of computational subprocesses (agents) cannot be predicted *a priori*: achieving effective measurement and control is very difficult in the absence of full knowledge of the internal design of other agents. It is therefore not surprising that the issue of *performance measurement* of MAS

has been largely avoided[3] despite the fact that successful engineering methods have been proposed for MAS in recent years by researchers in the field of *agent-oriented software engineering* [7, 13].

Effectively, all we are left with as a basis for performance measurement in open MAS is the *communication* observed in the system, and this is the central idea behind *communication-based performance measurement* (CBPM): to use *communication data* between agents in the system as data material that is suitable for performance measurement, and then defining appropriate measures for this kind of data. Thereby, we define any data as communication that is either (a) a textual message passed between two or more agents or (b) some (physical) action that an agent performs publicly, i.e. an action that can be observed by at least one agent other than itself.

Although open systems "force" us to take such an approach by the encapsulation of agent internal computation, two fundamental properties of MAS, namely that

- MAS are (usually) based on deliberative, knowledge-based agents, and that
- agents in MAS communicate using high-level languages such as KQML [8] or FIPA-ACL [6]

also suggest that the approach offers several *advantages*:

1. Even if reliable mental models of the agents are not available in open systems, high-level ACLs allow us to derive the states of social commitments, the intentions of agents etc. to a certain degree. This is due to the semantics we *impute* on the ACL performatives – even if they are violated by agents, this will eventually become evident in observable agent behaviour (e.g. if a promise is not kept). Thus, high-level ACLs allow for a tracing of much more abstract types of interactions and dependencies than, for example, low-level network communication.
2. This kind of measurement allows us to abstract from non-communicative properties of the system so that we can immensely reduce the global complexity of the system. Knowing that agents are able to process knowledge-level representations (of tasks, environment states, etc.) allows us to ignore the details of their internal reasoning (in fact, in open systems, we have no other choice); at the same time, we can expect at least those aspects that are relevant to the interaction between components to become visible in the contents of messages, and hence open to an analysis by the manager agent (who is a knowledge-based agent in turn).

Motivated by these advantages, we need to identify and precisely define concrete measures. As a prerequisite for these definitions, we first introduce the underlying model of MAS.

3 A Generic MAS Model

Our MAS consists of a set of agents $\mathcal{A} = \{a, b, c, \ldots\}$ (that need not be fixed over time), which (among other things) are able to communicate with each other by using

[3] Rare and only application-specific exceptions include work on resource management for grid computing [3] or on mobile agents, where performance measures concentrate on computation overhead for message passing and agent migration [4].

a speech-act based agent communication language (like KQML [8] or FIPA-ACL [6]) the semantics of which is accessible to every agent[4].

Furthermore, we assume that the MAS has to perform tasks taken from a fixed set of possible tasks $\mathcal{T}$ and that they "arrive" at the system at arbitrary points in time (they may enter the system via some central "manager" agent or through different agents). To allow for a more detailed evaluation, we assume that a real-valued measure

$$c : \mathcal{T} \rightarrow \mathbb{R}$$

for the cost of the tasks is defined by the observer of the system that is measuring its performance.

The general view that we have of such task-based MAS is that when tasks come in, a negotiation process is initiated, which leads to either agreement or conflict. In the case of agreement, agents agree on a coordinated joint plan which they then execute. Otherwise, the task fails and the next task is processed. If agents reach consensus on how to perform the task, they execute the joint plan, and enter additional negotiation loops in case of further conflicts. Also, in some cases, plans may fail during execution.

In the following definitions, we will assume that a FIPA-ACL-subset of speech acts is used by the agents and we will use a simplified syntax for messages of the format

$$\texttt{performative}(\texttt{sender}, \texttt{receiver}, \texttt{content})$$

where `sender` and `receiver` are symbolic names for agents and `content` is either some other message or some first-order logic formula. As the set of performatives we define for the scope of this paper

$$\begin{aligned}\texttt{performative} \in \{\ & \texttt{inform}, \texttt{inform\_done}, \texttt{inform\_ref},\\ & \texttt{agree}, \texttt{accept\_proposal}, \texttt{request},\\ & \texttt{cfp}, \texttt{reject\_proposal}, \texttt{propose},\\ & \texttt{failure}, \texttt{not\_understood}, \texttt{refuse}\}\end{aligned}$$

which is a subset of the performatives defined by FIPA [6]. In real-world applications, this set is usually extended or restricted according to the requirements of the interaction protocols used.

Finally, and maybe most importantly, in all measures we define we will use *message-time* as the underlying time-scale, i.e. instead of measuring (real) time, we measure all quantities with respect to the *total number of messages* ($TNOM$) that have occurred during operation. In the following, let

$$M = \{m_1, m_2, \ldots m_n\}$$

denote the set of all messages ($TNOM = |M|$). If needed, we can partition M into sets

$$M_T = \{m_{T_1}, \ldots m_{T_{n_i}}\}$$

[4] Otherwise agents have to make huge efforts to "understand" communication, a topic which opens a whole new dimension of complexity and is not regarded here. However, obeying *semantics* by no means implies benevolence regarding *pragmatics*.

for task $T \in \mathcal{T}_{curr}$ where $\mathcal{T}_{curr} \subseteq \mathcal{T}$ is the set of past processed tasks (including the task currently processed) such that

$$M = \biguplus_{T \in \mathcal{T}_{curr}} M_T$$

The advantage that measuring messages offers compared to measuring time directly is that we can (a) quantify communicative phenomena in relation to the total amount of communication so as to assess their importance in the context of all ongoing communication, (b) neglect time spent on intra-agent reasoning, and hence be able to concentrate on crucial (social) properties of the system. Although this is an abstraction, the loss of accuracy can be neglected in many deployed systems, as the time spend for communication between agents (on different machines) dominates other processes.

4 Communication-based Performance Measures

4.1 Basic Measures

Basic measures rely on counting messages and atomic message types (i.e. certain performatives) and are therefore the simplest CBPM measures that serve as a starting point for any analysis of social system properties. The first measure we introduce is *messages per task and cost* ($MPTC$) and is computed by using the formula

$$MPTC = \sum_{T \in \mathcal{T}_{curr}} \frac{M_T}{c(T)}$$

Here the quantity $MPTC$ reflects how much communication is "spent" on a task per cost. In order to obtain comparable quantities of $MPTC$, the numbers of messages M_T are normalised with the cost of the tasks, following the intuition that it is justified for more expensive tasks to require more communication than cheaper ones. Normally, we expect MAS to be operating most efficiently, for which $MPTC$ is minimal (except, of course, if $MPTC$ takes on *extremely* small values which would mean that tasks are completed or fail without almost any communication – in which case communication does not make a difference or is not working properly).

This measure can be further refined by distinguishing assigned tasks between "failed" and "successfully completed" tasks

$$\mathcal{T}_{curr} = \mathcal{T}_{succ} \uplus \mathcal{T}_{failed}$$

and computing a "fail-fast" variant of $MPTC$

$$\text{ff\_}MPTC = \alpha \cdot \sum_{T \in \mathcal{T}_{succ}} \frac{M_T}{c(T)} + \beta \cdot \sum_{T \in \mathcal{T}_{failed}} \frac{M_T}{c(T)}$$

By using $0 \leq \alpha \ll \beta \leq 1$ we can weigh the amount of communication spent on failed tasks stronger than that spent on tasks successfully completed, thus implicitly expecting effectively communicating agents (that minimise $\text{ff\_}MPTC$) to realise at an early stage that a task cannot be completed.

Looking more closely at the properties of messages exchanged rather than only counting them, we can determine the average usage of certain message *types* if we

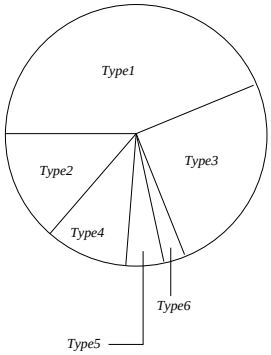

Fig. 1. A message type partition chart.

assume that the set of relevant performatives has been partitioned into such types according to certain criteria. For the scope of this paper we suggest the following partition for the list of performatives given above:

- $Type1 = \{\texttt{request}\}$ for messages that indicate when agents require non-local information from others,
- $Type2 = \{\texttt{inform}, \texttt{inform\_done}, \texttt{inform\_ref}\}$ for messages that can only be a reply to some question and hence indicate propagation of information among agents, or are used for synchronisation,
- $Type3 = \{\texttt{cfp}, \texttt{propose}\}$ to denote messages that indicate offers to accept a task or calls for such offers,
- $Type4 = \{\texttt{reject\_proposal}, \texttt{refuse}\}$ for messages that indicate when $Type3$ messages fail,
- $Type5 = \{\texttt{accept\_proposal}, \texttt{agree}\}$ for messages that indicate when $Type3$ messages succeed, and finally
- $Type6 = \{\texttt{not\_understood}, \texttt{failure}\}$ for messages indicating either messages that are not understood or requests that cannot be handled.

In analogy to a frequency analysis of each performative (which we will come back to later), we define the *mean message type usage (*$MMTU$*)* to be

$$MMTU(x) = \frac{1}{|\mathcal{T}_{curr}|} \sum_{T \in \mathcal{T}_{curr}} \frac{|\{m \in M_i | type(m) = x\}|}{|M_T|}$$

for $x \in \{Type1, Type2, Type3, Type4, Type5, Type6\}$ to compute the average percentage of a certain message type per task. Using these values, a visualisation called *message type partition chart* can be derived that provides us with a message type profile for a specific MAS as shown in the example in Figure 1. In this example, we are dealing with a MAS in which the largest portion of all messages is spent on gathering non-local information, i.e. agents are very busy attempting to obtain information from their peers. The chart also reveals that many of these questions go unanswered ($MMTU(Type2) < MMTU(Type1)$) so obviously "information exchange" is not efficient in this system (either questions are posed that cannot be answered, or agents refuse to answer too often).

Apart from asking other agents, the agents in this system are also very busy offering and asking for services ($Type3$), and these attempts are much more often unsuccessful than successful ($MMTU(Type4) > MMTU(Type5)$). Quite often, there is no reaction at all, which is reflected by

$$MMTU(Type4) + MMTU(Type5) < MMTU(Type3)$$

Obviously, such measurements are valuable starting points for improving system performance. Moreover, $MMTU$ can be easily further refined, which makes it a very flexible and powerful measure. As an example, we might compute its value also depending on task cost as in

$$MMTU(x, c_1, c_2) = \frac{1}{|\mathcal{T}_{curr}^{c1,c2}|} \sum_{T \in \mathcal{T}_{curr}^{c_1,c_2}} \frac{|\{m \in M_i | type(m) = x\}|}{|M_T|}$$

where

$$\mathcal{T}_{curr}^{c_1,c_2} = \{T \in \mathcal{T}_{curr} | c_1 \leq c(T) \leq c_2\}$$

is the set of all tasks that whose cost lies in the interval between c_1 and c_2. This would allow us to make more precise statements with respect to the distribution of message types, and the same would be the case if we parametrise $MMTU$ with e.g. certain subsets of agents, of agent types, spatial regions in a network, etc.

4.2 Measuring complex message patterns

The measures introduced above already provide useful information about the amount of communication and its distribution over classes of performatives, but they do not allow for a *syntactic* analysis of entire dialogues and protocols. To achieve such an analysis, we introduce *message patterns* where we interpret sequences of messages as lists of strings and use upper-case variables $A, B, \ldots$ to denote messages or message fields that may be referred to by later messages in the same sequence. Furthermore, we use $*$ as a wildcard symbol that stands for an arbitrary message sequence. For example, a pattern

$$p = [\texttt{accept}(A, B, \texttt{do}(A, X)), *, \texttt{do}(A, X)]$$

describes a set of messages that starts with an acceptance of agent A towards agent B to perform action X and ends with A actually performing X [5]. Likewise,

$$q = [\texttt{accept}(A, B, \texttt{do}(A, X)), (\neg\texttt{do}(A, X))^n]$$

stands for the set of sequences in which A does not fulfil its commitment for at least n steps after committing itself to do X.

Clearly, such patterns can be efficiently matched against messages in a message log. Hence, for any such pattern p we can measure the average length of its occurrence

$$mean\_length(p) = \frac{1}{\mathcal{T}_{curr}} \sum_{matches(m,p) \wedge m \in M} length(m)$$

[5] Remember that this is only a communication if the execution of X is observable for both parties – the performative `do` (which is not part of the language definition, in the sense of FIPA-ACL) is used to signify such observable action execution.

where $matches(m, p)$ is a boolean function that returns `true` iff pattern p matches message sequence m and $length(m)$ is the number of messages in sequence m. Alternatively, we can define the *task-relative* average length of p as

$$mean\_length(p) = \frac{1}{\mathcal{T}_{curr}} \sum_{matches(m,p) \wedge m \in M} \frac{length(m)}{|M_T|}$$

and, of course, also its frequency

$$frequency(p) = \frac{|\{m \in M | matches(m, p)\}|}{|M|}$$

Average lengths and frequencies can be used to define a number of other useful performance measures. For instance, consider a MAS in which agents exchange proposals concerning a multiagent plan to execute a task and where their peers can either `agree` to a proposed plan or reject a proposed plan by `reject_proposal`. Assume further that for a plan to be executed, all agents have to accept it. We can then define

$$p_a = \bigcap_{Q \in \mathcal{A}} [\texttt{propose}(P, Q, X), *, \texttt{accept}(Q, P, X)],$$
$$p_c = \bigcup_{Q \in \mathcal{A}} [\texttt{propose}(P, Q, X), *, \texttt{reject}(Q, P, X))]$$

as the set of sequences in which *all* agents (*at least one* agent) Q eventually accept/reject P's proposal X. Accordingly, we can define

$$MTTA = mean\_length(p_a),$$
$$MTTC = mean\_length(p_c)$$

where $MTTA$ stands for *mean time to agreement* and $MTTC$ stands for *mean time to conflict*. Likewise, patterns can be defined for task allocation, resource allocation, conflict resolution, negotiation processes, etc. In particular, if a specific set of *interaction protocols* is used in the MAS, properties of enacted instances of these protocols can be quantified, e.g. the number of bids in an auction, the *mean time to accepted bid* etc.
Quite evidently, these kinds of measures can be superior when it comes to predicting the behaviour of external attributes: if we consider *responsiveness* as a required external attribute in a parallel execution environment for agents, surely $MTTA$ is more expressive than $TNOM$, as probably many messages are sent in parallel while we are interested in the length of the sequence until an agreement is reached.

5 Example 1: Evaluating Interaction Protocols

To see how applying measures of different complexity can be useful, we now demonstrate the usage of the above classes of measures in the context of a practical and relevant problem. Suppose a specific *protocol manager agent* in the MAS has to decide on the task assignment method used in the system, and that there are one hundred *task manager* agents each of which needs to find a *bidder agent* for a task it has to assign. Suppose for simplicity all bidder agents have identical capabilities, i.e. any bidder can execute any task in principle, yet only a *single* task at a time and the communication deadlines are at the same time. Assume further that the protocol manager agent may choose between the contract-net protocol (CNP) [12] and the contract-net-with-confirmation protocol (CNCP) (for a detailed description, see [11]). Let us further

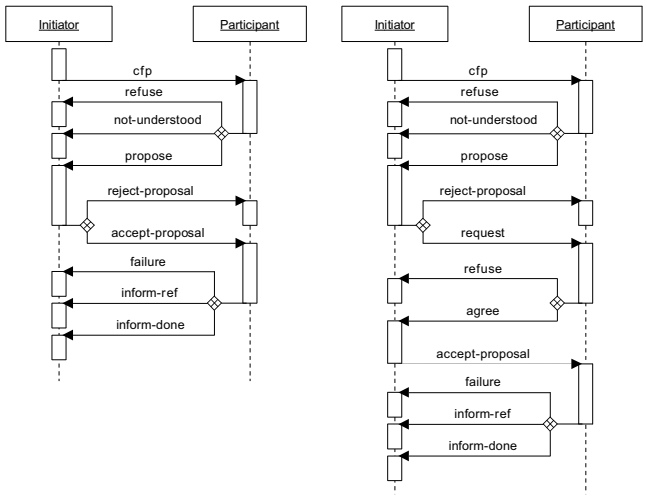

Fig. 2. Sequence diagrams for the contract-net (left) and contract-net-with-confirmation (right) protocols.

suppose that he has tried both mechanisms on comparable problem instances. In a (representative) experiment with our implemented system, the *assigned tasks ratio* (ATR) (i.e. the ratio between tasks successfully assigned and the total number of tasks) was 1.0 using the CNCP while the CNP had an ATR of only 0.65.

How is this possible? The sequence diagrams of the protocols (see Figure 2) do not reveal the crucial difference in the sense that they could be used as the starting point for an informed decision between the two protocols. Let us now apply the measures introduced in the previous sections. First, we evaluate $TNOM$ (20130 for CNP, 30768 for CNCP) and perform an $MMTU$ analysis of the communication that occurred during the experiments (Figure 3). Although it is obvious that the protocols influence the occurrence of certain performatives, in this particular case $TNOM$ and $MMTU$ bring us no closer to an explanation why the ATR of the CNCP is so much higher.

Remembering the message patterns of Section 4.2, we might look at pattern

$$p = [\texttt{propose}(A, B, X, \{\}), *, \texttt{reject\_proposal}(B, A, X)]$$

This means we search agent communication logs for unconditional proposals (side condition is $\{\}$) implying resource allocations (i.e. commitments to be able to perform a task in the future) which are followed by rejection and measure *mean rejected resource allocations* $MRRA = \mathit{frequency}(p)$[6]. And indeed, in our example we find that $MRRA$ equals $zero$ in the CNCP case, and 0.35 in the CNP case. The former case indicates that with the CNCP no resources are allocated before it is clear that they are going to be used, while the latter represents the missing allocations expressed by the low ATR of

[6] This is a simplified version of the pattern used for $MTTC$ in which negotiation only occurs between one manager and one bidder, and it is adapted to the needs of CNP-like negotiation by additionally requiring the empty side condition.

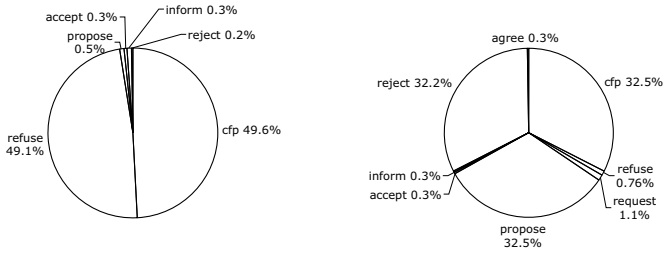

Fig. 3. Message type partition chart for the the same problem solved using CNP (left) and the CNCP (right).

the CNP. This shows at a glance the superiority of the CNCP in the respect that it never produces the sequence "allocating resources" $\rightarrow$ "rejecting this allocation".

This example illustrates that a more complex measure can explain the correlation between an internal attribute $MRRA$ and the external attribute ATR when simpler measures, such as $MPTC$ and $MMTU$ are not.

6 Example 2: Evaluating the Optimal Amount of Communication

Next, picture the situation of a manager agent who maintains the platform on which the above multiagent system is running, with the decision on using the CNCP instead of the CNP already made due to the above considerations. To make the scenario more interesting, assume that the manager agent has no control over the participating agents, which we call *clients* (previously managers), and *service providers* (previously bidders) but it can set the standard protocol to be used. Of course, costs for providing a service may vary, as may the price preferences of clients, i.e. there is no guarantee that tasks get assigned at all. Also, suppose that, for reasons of publicity (and to raise banner advertisement prices), the platform owner is interested in having as many "deals fixed" (tasks assigned) as possible. To reduce bandwidth, one of the parameters the platform may prescribe to participating agents is the maximum number of *contacted agents*, i.e. the maximal number n of calls-for-proposals/proposals (for clients/service providers, respectively). Now assume that the platform manager observes six different series of MAS runs. In the first three runs, 100 service providers and 90, 100, and 110 clients participate with $n = 10$; in another three runs, the same configurations apply, but now $n = 20$ allows for more communication. The platform manager agent, wants to know whether the cost induced by doubling the admissible amount of communication has paid off. To this end, it analyses the patterns

$$p_p = [\texttt{cfp}(P, Q, X), \texttt{propose}(Q, P, Y), \texttt{request}(P, Q, Y)]$$
$$p_r = [\texttt{cfp}(P, Q, X), *, \texttt{refuse}(Q, P, X)]$$
$$p_d = [\texttt{cfp}(P, Q, X), *, \texttt{accept\_proposal}(Q, P, Y)]$$

p_p represents situations in which a proposal is followed by a request, sequences that match p_r represent "failures" of service providers (in which they cannot perform a task although they bid for it because they committed to some other task in the meantime). Sequences that match p_d, finally, mark "deals", i.e. assigned tasks that are properly

r_1	Clients		
n	90	100	110
10	0.21	0.25	0.29
20	0.10	0.15	0.20

r_2	Clients		
n	90	100	110
10	0.51	0.61	0.69
20	0.50	0.67	0.75

Table 1. r_1 and r_2 in different simulations.

carried out. Table 1 shows the values of the following quantities in the six runs we obtained from experiments in the described configurations:

$$r_1 = \mathit{frequency}(p_p) \text{ and } r_2 = \frac{\mathit{frequency}(p_r)}{\mathit{frequency}(p_p)}$$

Two central observations can be made: while an increase of client population from 90 to 110 causes an increase in probability r_1 (i.e. the ratio of proposals that are accepted by the client) by less than 50% in the $n = 10$ case (from 0.21 to 0.29), it *doubles* r_1 in the $n = 20$ case (from 0.10 to 0.20). Of course, it is only natural that if more clients are present, the probability increases with which the price they are willing to pay is met by some service provider. However, it looks as if being allowed to contact more potential partners increases this probability super proportional, when a bigger choice of clients is available.

On the other hand, the second table shows that the probability r_2 of not being able to perform a task although requested to do so by the client increases *stronger* in the case of $n = 20$ (from 0.5 to 0.75) than if $n = 10$ (from 0.5 to 0.69), which makes the benefits of allowing for more communication smaller; at the end of the day, $\mathit{frequency}(p_d) = 0.044$ in the case of $n = 20$ while $\mathit{frequency}(p_d) = 0.084$ in the case of $n = 10$ (with 110 clients). In other words, doubling the amount of communication (and their cost) leads to 98 assigned (and performed tasks), which is only little more than the 93 reached with $n = 10$.

Apparently, increasing the number of calls for proposals and proposals each agent may make is not very advantageous, because it makes service providers send more proposals than they are able to achieve tasks, which limits the benefits incurred by increasing the number of "matches" between clients' and providers' prices.

Therefore, the platform manager should maybe look for alternatives to increasing n if the number of participating clients increases, such as, for example, introducing "matchmaking" agents that find suitable service providers for clients with less communication. This is another example of how using CBPM as an analytical tool may help meet design objectives that cannot be achieved by merely tuning the system in a trial-and-error fashion.

7 Conclusion

In this paper, we suggested *communication-based performance measurement* (CBPM) for multiagent systems, a novel approach to measuring system performance based on measurements of communicative processes among agents. We argued that CBPM measures are particularly suitable as *indicators* for the self-diagnosis of autonomic computing systems, where manager agents conduct these measurements and take appropriate steps to spawn *self-repair* and *self-optimisation* processes.

Our examples of measurements in implemented, large-scale, market-based MAS proved that these measurements provide guidance for such managers, and that they can help improve system performance in *open* systems even without direct access to agents' internal control mechanisms. When employing the full analytical power of CBPM, even simple actions such as reducing message limits or adopting different protocols for the system may significantly influence the global behaviour of the system.

The problem that remains is the online derivation of these performance measures for any given domain. Obviously, this is not an easy enterprise, and much of this process will still rely on the experience of the software engineer. So far, however, we can state that at least for the decisions to be made in our examples, the framework of CBPM supports the process of evaluation. Future work will focus on how agents can be designed that embody elaborate self-diagnosing capabilities and that are able to effectively monitor and improve system behaviour *automatically*. More specifically, we aim at combining the CBPM approach with more elaborate methods for communication analysis at a more *semantic* level, in a similary way as that proposed for the evolutionary development of open MAS through human designers in the EXPAND [2] method.

References

1. AgentCities. http://www.agentcities.org, 2003.
2. W. Brauer, M. Nickles, M. Rovatsos, G. Weiß, and K. F. Lorentzen. Expectation-Oriented Analysis and Design. In *Proceedings of the 2nd Workshop on Agent-Oriented Software Engineering (AOSE-2001) at the Autonomous Agents 2001 Conference*, volume 2222 of *LNAI*, Montreal, Canada, May 29 2001. Springer-Verlag, Berlin.
3. J. Cao, D. J. Kerbyson, and G. R. Nudd. Performance evaluation of an agent-based resource management infrastructure for grid computing. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, Australia*, pages 311–318, 2001.
4. M. Dikaiakos, M. Kyriakou, and G. Samaras. Performance evaluation of mobile-agent middleware: A hierarchical approach. In G. P. Picco, editor, *Proceedings of the 5th IEEE International Conference on Mobile Agents*, volume 2240 of *Lecture Notes in Computer Science*, pages 244–259, Berlin et al., 2001. Springer-Verlag.
5. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, editors. *Performance Engineering*, volume 2047 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin et al., 2001.
6. FIPA. FIPA (Foundation for Intelligent Agents), http://www.fipa.org, 2003.
7. N.R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
8. Y. Labrou and T. Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD, February 1997.
9. V. Parunak. Industrial and practical applications of DAI. In G. Weiss, editor, *Multiagent Systems*, pages 377–421. The MIT Press, Cambridge et al., 1999.
10. IBM Research. Autonomic computing, http://www.research.ibm.com/autonomic/, 2003.
11. M. Schillo, C. Kray, and K. Fischer. The Eager Bidder Problem:A Fundamental Problem of DAI and Selected Solutions. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS '02)*, pages 599–607, 2002.
12. R.G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):61–70, 1981.
13. M. J. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, 1997.