An analytical comparison of the client-server, remote evaluation and mobile agents paradigms

A. Puliafito¹, S. Riccobene², M. Scarpa³

¹ Università di Messina, Dipartimento di Matematica
Contrada Papardo, S. Speroni, 98166 Messina, Italy. e-mail: apulia@ingegneria.unime.it

² Università di Catania, Dipartimento di Matematica
V.le A. Doria 6, 95125 Catania, Italy. e-mail:sriccobene@cs.unict.it

³ Università di Catania, Istituto di Informatica e Telecomunicazioni,
V.le A. Doria 6, 95125 Catania, Italy. e-mail:mscarpa@iit.unict.it

Abstract

In this paper we deal with the study of the actual convenience of using the agent programming paradigm for accessing distributed service. We try to point out the benefits of such a communication paradigm, by providing an analytical study of its basic features in comparison with client-server approach and remote evaluation. The results that we have obtained show how agents must not always be considered the only solution to any communication issue, since in several cases their use might even reveal a drawback. In this paper we present several models of non-Markovian Petri nets, which have been solved through the WebSPN tool, and we provide a close comparison between the agents technique, the client-server and the remote evaluation. We also focus our attention on providing some practical remarks, which can help the developer during the design, in order to select the communication paradigm which best suits to the features of the application that has to be developed.

Keywords: Client-Server, Remote Evaluation, Agent technology, Petri nets.

1 Introduction

Communication among distributed processes is an essential requisite in nowadays computing systems. A communication paradigm represents the set of rules to be followed in exchanging data and synchronizing the execution of processes. The nature of currently available computing systems is pushing a lot towards a distributed approach which assumes that computing resources and data are no longer located on the same machine, but migration of code and data is executed in order to speed up the whole execution process. Code to data and data to code are alternative strategies to be adopted [1].

The classical client-server approach assumes that the client functionalities are somehow disjoint from the execution power of the server. This means that locally executing processes retrieve data from remote databases, which constitute shared resources inside the
distributed system. Actually, to allow the shared data to be accessed from different applications, huge (and not filtered) amount of data is retrieved on the client, who executes the processing of the data in order to extract the required informations. Recent technological improvements are allowing new alternatives to this communication paradigm [2].

Generally speaking, the classical data to code approach is now reversed to accomplish a code to data strategy, which aims to promote the use of remote computing resources widely available and easily accessible [3, 4]. Language such as Java strongly supports this kind of communication paradigm as it allows to write the code once and to execute it many times on heterogeneous hardware and software environments [6]. The great success of this language has made the development of distributed applications in heterogeneous and mobile environments very popular and a very effective approach to the more classical client-server solutions [7]. Software code which can eventually migrate on its own among different servers, till the target assigned from the user has been pursued, is usually identified as an agent, to stress the strong autonomy that this communication paradigm allows in controlling code execution [8, 9, 10]. The word agent is appearing always more often and the suspect that a new buzzy word has been created is sometimes legitimate. An increasing literature has been produced which proposes code mobility and mobile agent technology as the only available solution to implement distributed applications or to access the services offered by a distributed system [11]. Interesting applications exist and challenging new ones are being developed, although the so called killer application has not been identified yet. Such application will be the one which can univocally demonstrates the advantages of mobile agent among alternatives paradigms.

In between the client-server and the agent communication paradigms a third possibility can be identified which is usually referred to as remote evaluation [17]. This paradigm assumes that code migrates on the server where it is executed, but then only the data are transferred back to the client: code migration between servers is not considered.

Due to the existence of so many communication paradigms with so strongly different approaches the following questions arise:

- **Which is the communication paradigm should I use?**
- **Is it possible to identify application scenarios where one approach is more convenient than the others?**

The spirit of this paper is well synthetized in these two questions. And derives from the observations of the authors that an analytical evaluation of the three communication paradigms is (probably) not available in the literature, although many papers exist which discuss about the power of the mobile agent approach against more traditional communication paradigms [5]. Our goal consists in trying to understand if mobile agents are really to be used in any circumstance or if a meditated use should be done because some cases exist where client-server or remote evaluation are more convenient. In order to provide an analytical evaluation of the performance offered by such paradigms, we have developed some Petri net models which capture graphically the main aspects of these communication paradigms [16, 14]. We have also used the modeling tool named WebSPN for their numerical evaluation [12, 13]. WebSPN has been extremely useful both for its friendly graphical interface and also for the possibility it offers to deal with non-exponentially distributed events. The results we have obtained are quite interesting
and provide some insight on the real performance offered by the client-server, the remote evaluation and the mobile agent communication paradigms. We presume that our work could be useful also to practitioners and software developers to justify some intuitive choices on one side and to help them in better design their systems on the other.

The rest of this paper is organized as follows: in section 2 we introduce the three communication paradigms highlighting their distinguishing features. In section 3 we describe the Petri net models we have developed and the basic assumptions behind them. In section 4 we present a case study, justify the chosen parameters and provide an analytical comparison of the results. Finally, in section 5 we present the conclusions.

2 Communication paradigms

In this section we will examine the three communication paradigms mentioned before. However, we need to outline that they are often grouped as Client - Server interaction, even if their structural and behavioural differences are considerable. Below we will use the term Client - Server (CS) only to identify the first paradigm presented.

2.1 The Client - Server (CS) paradigm

In the typical Client - Server protocol, data processing mainly takes place in the host client. In fact, the work of the server is limited to the mere execution of some basic procedures for the data retrieval and storage. Before being sent to the client, data only undergo a soft initial filtering. The host server behaves as a simple remote storage system. Together with all of the other servers and the interconnection network, makes the whole system to form a big “repository” of information available to the different clients.

The server usually makes available some procedures for handling the stored data which are designed for responding to criteria of general effectiveness. The actual data processing is therefore left to the host client, where the user can execute ad hoc procedures for the kind of processing desired.

This type of CS scheme is used when we want to create a very simple system from the management point of view, or structures with a high level of security. In fact, the procedures for accepting the requests and selecting the data embedded into the servers, and there is no way of changing them at runtime. An advantage of this architecture is the possibility of controlling the type and the ways of access to the data stored in the server. In other words, this type of system is very “closed” from the point of view of the access to the data. The server places at the user’s disposal only what is expected during the design of the database. Consequently, the level of security is very high.

At the same time, the system is also very “closed” from the point of view of performances. In fact, if the user has specific requests concerning the modes of data processing, and if the server (or better its manager) does not provide for that specific type of operations, the only possibility commits in retrieving much more data than needed, and then to perform the operations of processing and selection in the client. In these cases, the server provides a huge amount of documents, in order to assure a wide basis of selection.

Of course, all that causes an overload of both the server and the communication system. In fact, the amount of data exchanged may be considerable. Consequently, the host client must have its own processing capability. Of course, this cannot be low.
2.2 The Remote Evaluation (REV) paradigm

Unlike the typical Client - Server, the paradigm of Remote Evaluation (REV) implies that server receives not only the processing requests from the client, but also the whole code needed for performing operations of selection on the data stored. The response of the server is therefore limited to sending the information that can be actually used and required by the client, with no additional overhead. Besides, since the user can use a customized code in the server, the data sent in output are “ready for the use”, and they only need negligible additional processing. From this point of view we can also think of an environment with host clients equipped with minimum processing potentialities.

The initial cost is therefore higher in comparison with the CS paradigm, and is localized in the opening stage of sessions. In fact, the code for the data processing can be of considerable size and we can easily assume that its size is higher than a simple retrieval request. Of course, this cost is counterbalanced by the reply stage (transmission of search results from the server to the client), because the amount of data passing through the network is more limited.

During the stage of design, a system with Remote Evaluation must be created by considering more detailed aspects in comparison with the CS. In fact, the processing architecture of the different servers must be similar (or very well known), so that the code sent by the client can be easily executed on all of the hosts. From this point of view, we can think of a common platform for code execution. This also implies the need for creating protection elements that could assure a high level of security.

2.3 The Mobile Agent (MA) paradigm

A Mobile Agent is an executable code that can move from a host to another, either according to decisions taken independently or according to outside parameters or after some interactions with other agents. The context of the program is transferred, as well as the code, during the migration. This way, there is a kind of suspension of the execution of the program, waiting for the subsequent resume state on a remote machine.

The system of Remote Evaluation is a more limited approach than the MA. In fact, a code migration is present in the REV, but there is always a direct interaction between the client and the server. This means that the code sent by the client returns the data directly to the source. Besides (when this operation is done), the process is completed, so the context of execution of the program is limited to the single host.

Conversely, the mobile agent system can be used for performing the research operations more effectively. In fact, the agent has the procedures for operating on the database according to the ways desired by the user, and can also make independent decisions, such as migration to other sites or returning the results obtained to the user, if they are considered sufficient. In this sense, the interaction between the user and the agent is limited to the stages of transmission and return of data. What takes place within this time limit depends only on the way the agent was designed. So we can say that the user loses the control of the agent, once it has been sent. This feature implies two direct consequences: 1) the size of the code of the agent is higher than the one needed for a simple REV (the structure of an MA is more complex) 2) when the agent migrates, it has to carry the partial results of the processing performed in the hosts visited before. We can therefore
expect that the amount of data transferred in each migration tends to increase. The agent can decide to limit the data considered interesting for the user dynamically, even by discarding the data selected in the previous hosts. Agents with a maximum quota of user data, which can be moved in each migration, can therefore be designed.

3 Models of Petri nets

In this section we describe some Petri net models we have developed in order to provide an analysis of the performance offered by CS, REV, and MA. We assume that a data search has to be executed, involving a set of $N$ DB-servers. All the servers have to be contacted, and some of them more than once. The numerical solution of the PN models is provided in the following section, where all the parameters are fixed and justified.

3.1 Petri net model of the CS paradigm

The system modelled is a typical interaction between a client and a set of servers. Let us fix their number to $N$ distinct units. Let us suppose that the client requires the processing by all of the $N$ servers (that is, the session is not ended before contacting all of the servers).

![Diagram](a)

![Diagram](b)

![Diagram](c)

Figure 1: CS, REV and MA models for computing the distribution of the response time

We can think of a single server as a procedure which receives in input a limited amount of data (the actual request), operates on a much higher amount (the whole database of the server), and returns a certain portion (not too small) to the client. In other words, the amount of returned data can be compared to the whole database, while the size of the request is much smaller.

The logical sequence of operations can be synthetized as follows: the client sends a data processing request to a server, which processes it and sends back the data directly to the client. A local processing, much more complex than the first one, starts in order to decide whether to request the same server for further operations, or to proceed with some requests to the next server. For example, this is the typical process that takes place when a search is done through the Internet by using a search engine. In fact, the
query format is very simple and general, so the reply consists of a long list of related documents. The user’s task is the filtering of the information obtained.

The processing stage will end when the client has processed all of the $N$ requests sent to the $N$ servers. However, according to the scheme used, the processing actually performed by each server (that is, for each high-level request) can be more than one. An explicit communication is expected by the client for each of them. Consequently, the total processing time cannot be fixed a priori, since it changes according to both the single operation and the number of operations performed for each high-level request.

The system described so far has been modelled by using a Petri net, described as follows (see figure 1a). At the beginning, the place $\text{Queries}$ contains $N$ tokens, which represent the $N$ high-level requests that the client has to send to the servers. Also place $\text{Ready}$ contains a token, indicating that an high-level request is ready to be processed. $\text{Start}$ is an immediate transition enabled by the simultaneous presence of tokens in $\text{Queries}$ and $\text{Ready}$, representing the request generation process. Once $\text{Start}$ fires, a token is placed in $\text{Req buffer}$ which indicates that the client has a request queued ready for being sent through the network.

The transmission phase of the request through the network is modelled by the transition $\text{Send req}$. If $\text{Comp1}$ contains a token, it means that the server has received the request and starts the processing stage, which lasts for a variable length of time modelled by transition $\text{Search}$. Once the server has completed the operation a token enters place $\text{Answ buffer}$ to indicate that it is waiting to send the reply data. The actual transmission time is modelled through the transition $\text{Send resp}$.

A token in place $\text{Comp2}$ indicates that the client has received the reply, and is processing the data locally (transition $\text{Filter}$). Once this phase is over (a token is in place $\text{End session}$), the client has to decide whether to send another request to the same server or to proceed with the next server. This is modelled with the two transitions $\text{Redo}$ and $\text{Next}$, which are probabilistic immediate. If $\text{Redo}$ fires the server will be required for further data. Conversely, if $\text{Next}$ fires triggers the current session is ended and the client proceeds with the next server (a token enters place $\text{Ready}$).

It must be noticed that this model has an absorbing state. We can therefore evaluate the mean time to absorption (MTTA) and its distribution. The condition that identifies this state is $\left( \#\text{Ready} = 1 \right) \land \left( \#\text{Queries} = 0 \right)$, where the symbol $\#P$ denotes the number of tokens in the place $P$.

### 3.2 Petri net model of the REV paradigm

Figure 1b shows the Petri net model for the REV protocol which derives from the behavioral description provided in section 2.2. We assume that $N$ servers have to be accessed to satisfy the query. Thus, place $\text{Queries}$ is initialized with $N$ tokens. As long as the servers are contacted and reply, the number of tokens in this place decreases. Transition $\text{Send code}$ models the time needed for sending the code and the request through the network. $\text{Send code}$ is enabled if a token is also in $\text{Ready}$, indicating that the current session is over, and a new session can be started to the next server.

Place $\text{Comp1}$ represents the server with a pending request. The processing time of the code provided by the client to the server is modelled through transition $\text{Search}$. When it fires, a token enters place $\text{Comp2}$ which indicates that the server has retrieved a certain
amount of data and is operating on them, in order to select what is actually useful to
the user. This filtering operation is modelled by the transition $\text{Filter}$.\(^1\) A token in place
$\text{End session}$ says that the server has achieved partial results, and is deciding whether to
perform new local operations ($\text{Redo}$ fires) or to end the current session, by returning the
results of the processing to the client ($\text{Next}$ fires). Transition $\text{Send resp}$ models the time
for sending the reply data.

### 3.3 Petri net model of the MA paradigm

The Mobile Agents paradigm differs from the Remote Evaluation for the following fea-
tures:

- a process can migrate during its execution. This means that the amount of band-
width occupied by the migration is higher than the code migration, since the state
of the process (which is being executed) has to be transferred.

- The migration does not imply an interaction with the client. In the protocols
previously described, the client controls the management and the selection of the
servers. The client receives the data in each session and performs any new query.
In a system based on Mobile Agents, the client only starts the session for the
first time. The rest is delegated to the agent. This leads to two elements: a) the
communication between the servers and the client may be slow, but this will
not greatly affect the overall performances of the whole system; b) the connection
between the client and the server might even be discontinuous, or might not be
assured during the whole operation. This can lead to a higher flexibility of use.

In figure 1c is shown the model of a mobile agent system. A token in place $\text{Start}$
denotes the client in a ready state, waiting to start the session. Through the transition
$\text{Send agnt}$, the initial transfer of the agent to the first server is modelled. The type of this
transition is therefore the same as the one of the REV, since at the beginning only the
code of the agent is transferred, without the corresponding state. Place $\text{Comp}1$ represents
the server with the agent in an execution state, for an amount of time modelled by the
timed transition $\text{Search}$. A token in $\text{Comp}2$ indicates that the server has ended the first
stage of general selection, and is going to refine the data obtained; the execution time
of the refining code on raw data is represented by the transition $\text{Filter}$. The server has
completed a processing cycle once a token enters place $\text{End session}$, and has to decide
whether to continue or to make the agent migrate.

The stage of decision concerning any migration of the code is modelled through the
two immediate transitions $\text{Redo}$ and $\text{Next}$. If $\text{Redo}$ fires the processing continues locally
(a token enters $\text{Comp}1$); if $\text{Next}$ fires the code migrates to another server (token in place
$\text{Decision}$). With regards the migration a distinction between two possibilities can occur:
either the agent migrates to a new server (firing of $\text{Migration}$), or go back to the client.
In the latter case (firing of $\text{Send resp}$), we do not need to move the code, but only the
reply data.

\(^1\) We separate the retrieval from the filtering process in order to allow a more correct comparison with
the PN model of the CS. In fact in the last one, the two processes are performed in two different places
The place Next srv is at first filled with N-1 tokens, which represent the N-1 servers remaining to be visited (the first one is represented by the initial token in Start). As long as there are some tokens in this place, the transition Send resp is disabled, while the Migration is enabled. In other words, if Next srv is not empty only the migration to another server can take place. A token in place End indicates that the session is over.

4 Performance evaluation of the paradigms

The Petri net models introduced in the previous sections have been evaluated in order to perform a comparative analysis of the three paradigms of distributed processing under considerations. We have selected the completion time of an application for data retrieval on the Web as an index of quality to base the comparison on.

4.1 Selection of the numerical values

In order to properly compare the CS, REV, and MA, we have fixed the following parameters, which are common to all of the models:

- size of a request ($D_{req}$): we have supposed that the request made by the client is small and constant for any possible situation. This corresponds quite well with what happens in the most part of client-server interactions.

- Time for searching the data in the server ($1/\lambda_{sre}$). The exact value for this time and its distribution greatly depend on the type of application examined and on the characteristics of the server used. However, we have fixed the distribution of the searching time as exponential, without losing the generality.

- Processing time of data ($1/\lambda_{elab}$). This processing can take place both in the server and in the client, depending on the protocol used. If this stage takes place in the server, it can be formally the same as the one of research. However, we have preferred to render both operations explicit, in order to compare the protocols better. Also this stage, like the previous one, greatly depends on the application and on the hardware system used, so we have fixed an exponential distribution.

- Size of the replies to queries ($D_{min}, D_{max}$). The reply to a query can realistically be considered included between a minimum and a maximum value, with a uniform distribution. We can suppose that the minimum value of a reply has the same size as the query, and that the largest reply is one or two order of magnitude higher [5].

- Size of the code ($D_{cod}$). We have assumed that the size of the code is the same in the REV and MA cases. In the second case the state of the migrating process is included in the additional data that the agent carries with itself. Besides, we have supposed that each migrating agent transfers only a limited portion of information with itself. Our idea is that, while the agent performs its operations, it updates its basis of knowledge, and discards the old information or what is less pertaining with the current operation. This value has been indicated with the parameter $n$. 

Throughput of the communication network. Two transmission rates are present in the system considered: high \((T_{\text{high}})\) and low \((T_{\text{low}})\). In many real situations the client is connected to the network through a low-speed link, while the different hosts are connected by a high-speed links.

The values assigned to the timed transitions of the PN models are calculated according to the formulas shown in table 1, where we have used the following relations:

\[
D_{1\text{min}} = D_{\text{min}} + D_{\text{cod}} \quad D_{1\text{max}} = n \cdot D_{\text{max}} + D_{\text{cod}}
\]

The numerical values selected for this case study are summarized in table 2.

<table>
<thead>
<tr>
<th>Table 1: CS, REV, MA model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CS model</strong></td>
</tr>
<tr>
<td>Transition</td>
</tr>
<tr>
<td>Send req</td>
</tr>
<tr>
<td>Search</td>
</tr>
<tr>
<td>Send resp</td>
</tr>
<tr>
<td>Filter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>REV model</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition</td>
</tr>
<tr>
<td>Send code</td>
</tr>
<tr>
<td>Search</td>
</tr>
<tr>
<td>Filter</td>
</tr>
<tr>
<td>Send resp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>MA model</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition</td>
</tr>
<tr>
<td>Send agnt</td>
</tr>
<tr>
<td>Search</td>
</tr>
<tr>
<td>Filter</td>
</tr>
<tr>
<td>Send resp</td>
</tr>
<tr>
<td>Migration</td>
</tr>
</tbody>
</table>

Table 2: Numerical values used to compute the parameters of the Petri nets.

### 4.2 Solution of the models and analysis of the results

The models described in the previous paragraph have been analytically evaluated by using the WebSPN tool. This tool allows to associate timed transitions with firing times having non-exponential distributions (GEN transitions), with no need to fix limits on how they are enabled. Besides, WebSPN can manage different memory policies [15]...
associated with GEN transitions. As we will see below, this characteristic has been used for obtaining some specific results.

### 4.2.1 A preliminary evaluation of the CS, REV and MA paradigms

Notwithstanding the programming paradigm used for the creation of the distributed application, we expect a stage when we decide whether we can visit another server or we need to continue processing on the same one because we have not had a satisfactory reply, yet. In the proposed models this behaviour is described by a pair of immediate transitions in conflict. Each of them is associated with a probability that the event represented takes place. If \( p_r \) is the probability of using the same server for the subsequent processing, \( 1 - p_r \) represents the probability that we decide to move to a different server for searching for other data. In the Petri net models this means associating a firing probability \( p_r \) to the transitions \textit{Redo} (see Figure 1a,b and c) and a firing probability \( 1 - p_r \) to the transitions \textit{Next}.

Of course, the performances of the system greatly depend on this parameter, and, once the feature of the network and the processing capabilities of the servers have been fixed, an increase in the service times is expected with the increase of \( p_r \). It must also be noticed that \( p_r \) is a typical parameter of the specific application examined and of the type of Data Base used for storing the information.

In order to evaluate the behaviour of the system varying this parameter, we have done some experiments in which we have fixed the throughput of all the communication network to 50 kbps. The firing rates concerning the stage of searching and processing of data have been fixed to \( \lambda_{src} = \lambda_{elab} = \frac{4}{sec} \). Such rates are valid for all of the experiments made, since they do not depend on the programming paradigm used, but on the computational ability of the computers used.

![Figure 2: Mean Time to Completion vs \( p_r \)](image)

Figure 2 shows a graph in which we have made a diagram of the mean time to completion according to the variation of \( p_r \) for each paradigm examined. The results of these experiments point out two aspects:

1. the typical CS is not always the worst of the three paradigms, since it provides the best performances in terms of mean time to completion for values of \( p_r \leq 0.7 \);
2. We cannot say a priori that Mobile Agents are the best approach for creating a distributed application, since they (at least in the specific scenario considered) can hardly assure a better behaviour than the other ones.

Due to the previous remarks, we need to consider the approach to be followed according to the conditions of use and to the application. We need to notice that, even if the CS paradigm can be a valid alternative, it is affected by the variation of $p_r$. Conversely, both REV and MA have a slight variation in the mean time to completion.

![Figure 3: Mean Time to Completion vs throughput of the subnet when $p_r = 90\%$.](image)

In figure 3 we show a graph in which we have made a diagram of the mean time to completion for the application according to the variation of the throughput of the communication network in the three possible scenarios (CS, REV, MA), fixing $p_r = 0.9$. We have made this choice because we think this is a realistic value for the type of application examined. The throughput has been changed from a minimum of 10 kbps (slow network) to a maximum of 1 Mbps. The values of the parameters of transitions for this set of experiments are shown in table 3. In this table we have used the notation $(a, b)$ for the parameters of a uniform distribution between $a$ and $b$, while the deterministic distribution is identified with the value $\gamma$ of its threshold.

Of course, the transitions that model the data transmission through the communication subnet will be influenced by the bandwidth of the subnet. We also need to outline that the MA model has been analyzed assuming that the speed of all the transmission network is the same. As we have described in section 3.3, this is not always true, and the implications of this scenario will be discussed below.

As long as the network provides more bandwidth (and thus a lower transmission speed), there are less differences among the paradigms, since the computation time becomes more important than the transmission one. However, the results shown are limited to an evaluation of the mean times to completion. Actually, more precise information about the performances provided come from the knowledge of the distributions of completion times, rather than from the knowledge of mean times. The models proposed allow obtaining such quantities by requiring the probability of being in the absorbing state in transient.
Table 3: Firing time parameters used in the models

<table>
<thead>
<tr>
<th>Throughput (kbps)</th>
<th>Send req (sec)</th>
<th>Send resp (sec)</th>
<th>Send code (sec)</th>
<th>Send resp (sec)</th>
<th>Send agent (sec)</th>
<th>Send resp (sec)</th>
<th>Migration (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.8 (0.8, 24.0)</td>
<td>32 (0.8, 24.0)</td>
<td>32 (0.8, 24.0)</td>
<td>32 (0.8, 24.0)</td>
<td>32 (0.8, 24.0)</td>
<td>32 (0.8, 24.0)</td>
<td>32 (0.8, 24.0)</td>
</tr>
<tr>
<td>20</td>
<td>0.4 (0.4, 12.0)</td>
<td>16 (0.4, 12.0)</td>
<td>16 (0.4, 12.0)</td>
<td>16 (0.4, 12.0)</td>
<td>16 (0.4, 12.0)</td>
<td>16 (0.4, 12.0)</td>
<td>16 (0.4, 12.0)</td>
</tr>
<tr>
<td>50</td>
<td>0.16 (0.16, 4.8)</td>
<td>6.4 (0.16, 4.8)</td>
<td>6.4 (0.16, 4.8)</td>
<td>6.4 (0.16, 4.8)</td>
<td>6.4 (0.16, 4.8)</td>
<td>6.4 (0.16, 4.8)</td>
<td>6.4 (0.16, 4.8)</td>
</tr>
<tr>
<td>100</td>
<td>0.08 (0.08, 2.4)</td>
<td>3.2 (0.08, 2.4)</td>
<td>3.2 (0.08, 2.4)</td>
<td>3.2 (0.08, 2.4)</td>
<td>3.2 (0.08, 2.4)</td>
<td>3.2 (0.08, 2.4)</td>
<td>3.2 (0.08, 2.4)</td>
</tr>
<tr>
<td>200</td>
<td>0.016 (0.016, 0.48)</td>
<td>0.64 (0.016, 0.48)</td>
<td>0.64 (0.016, 0.48)</td>
<td>0.64 (0.016, 0.48)</td>
<td>0.64 (0.016, 0.48)</td>
<td>0.64 (0.016, 0.48)</td>
<td>0.64 (0.016, 0.48)</td>
</tr>
<tr>
<td>500</td>
<td>0.008 (0.008, 0.24)</td>
<td>0.32 (0.008, 0.24)</td>
<td>0.32 (0.008, 0.24)</td>
<td>0.32 (0.008, 0.24)</td>
<td>0.32 (0.008, 0.24)</td>
<td>0.32 (0.008, 0.24)</td>
<td>0.32 (0.008, 0.24)</td>
</tr>
</tbody>
</table>

Figure 4: Completion time distributions when $p_r = 0.6$, 0.7, 0.8 and 0.9

In figure 4 we show the distributions of the completion times for the three paradigms varying $p_r$ from 0.6 to 0.9. The graph $a$ shows that a probability of 0.95 of completing the processing is reached in a time $t^{0.95}_{CS} \simeq 117$ s in the case of CS, and in a time $t^{0.95}_{REV} \simeq 110$ s in the REV case. This difference increases if higher completion probabilities are required. For example, if we wanted to be sure that the application has concluded the processing (i.e. the probability is close to 1.0), we would have $t^{1.0}_{CS} \simeq 170$ s and $t^{1.0}_{REV} \simeq 120$ s. In figure $b$ we show a similar graph in case $p_r = 0.7$. In this case we absolutely need some information about the probability distributions, in order to make a distinction among cases that seem to be similar (CS and REV have the same mean time to completion). From the graph of the probability distribution we conclude that the behaviour of the CS is worse than the one of the REV, since in this case we have $t^{0.95}_{CS} \simeq 160$ s and $t^{0.95}_{REV} \simeq 117$ s. If we search for the maximum time to absorption, we can see that $t^{1.0}_{CS} \simeq 215$ s, $t^{1.0}_{REV} \simeq 130$ s, $t^{1.0}_{MA} \simeq 250$ s. Thus, not only we can find a considerable difference between CS and REV,
but we can also notice that the difference in the performances between CS and MA is smaller than what we would have expected from the data concerning mean times. Graphs of figures c and d shows the behaviour for \( p_r = 0.8 \) and 0.9. We can notice that the CS paradigms performs worse then REV and MA if a completion time with high probability is required.

### 4.2.2 REV and MA versus network speed

The results obtained in the previous section make us think that the CS paradigm has the worst behaviour among the three. Below we will therefore focus our attention on the REV and MA paradigms. This evaluation will be done after fixing \( p_r = 0.9 \); thus we will no longer mention this parameter. If this takes a different value, we will need to reconsider the exclusion of the CS paradigm.

This time the proposed models have been solved by changing the speed of the communication network that connects the client and the servers. As to the MA paradigm, we have considered these two cases:

1. the interconnection network of the servers is the same as the one that connects the servers to the client. The transmission speed of data is the same in any stage of the communication (in this case, we will speak of MA1 model);

2. the interconnection network among the servers is a high-speed network, while the one that connects the client with the servers changes (in this case, we will speak of MA2 model).

The MA2 model is justified if the user is connected to the system through a Mobile computer. In the MA2 model the throughput of the high-speed network has been fixed to 1 Mbps. The parameters for non-exponential transitions therefore remain the same as the ones in table 2, except than for transition Migration, which describes the migration of the agent among the servers. The firing time of this transition is uniformly distributed, with parameters \((0.008, 0.24)\).

![Graph showing mean time to completion of REV and MA paradigms.](image)

Figure 5: Mean time to completion of REV and MA paradigms.

The mean times to answer obtained from the solution of the models according to the variation of the throughput of the network are shown in the figure 5. As we could expect,
the increased speed of the network causes some improvements in the answer times of the system. Anyway, we need to point out that the increase does not result to be graded with reference to the network. In fact, the curves whose speed is higher than 200kbs are very close. In such conditions, the processing time in the server is predominant than the communication time between the server and the client. Besides, we confirm that the mobile agent paradigm MA1 has the worst behaviour in conditions of homogeneous network. This is due to the fact that the communication is overloaded by the transfer of whole pieces of code when an agent migrates. Conversely, a lower amount of data needs to be exchanged in the REV. If the delay due to the migration of the agent is compensated by using a fast transmission network among the servers (MA2 model), the mobile agent paradigm shows its values. In fact, the graph points out how the mean times to answer are always the best that can be obtained for the application examined. Besides, such times remain quite constant when the speed of the communication network with the client changes.

5 Conclusions

The Mobile Agents (MA) paradigm is playing an increasing role in modern communication systems. Different applications exist which take advantage from code migration regarding the field of information retrieval, systems management, user mobility. We believe that this new communication paradigm is very promising, but recognize the lack of an evaluation of its performance, mainly against more traditional communication paradigms such as client-server and remote evaluation. In this paper we have provided an analitycal evaluation of these paradigms based on the use of Petri nets. The models we have developed show that mobile agents are not always the more convenient solution as the performance they offer strictly depend on external factors such as the network speed and the application itself. It appears that the MA paradigm should be successfully adopted when the communication subsystem is fast enough or in all the cases when the user may be temporarily disconnected from the system due to mobility or network unavailability. Numerical values are provided which give an insight on the opportunity to use one communication paradigm or another identifying practical situations of interest for researchers and practitioners in the area of distributed and mobile systems.

References


