View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Institucional de la Universidad de Almería (Spain)

Tracking Traitors in Web Services via Blind Signatures

J.A. Álvarez-Bermejo[1] and J.A. López-Ramos[2]

[1] Dpt.Arquitectura de Computadores
Universidad de Almería
jaberme@ual.es
[2] Dpt. Álgebra y Análisis Matemático
Universidad de Almería
jlopez@ual.es

Abstract. This paper presents a method and its implementation, built on the blind signatures protocol, to trace users sharing their licenses illegally when accessing services provided through Internet (Web services, Streaming, etc). The method devised is able to identify the legitimate user from those users who are illegally accessing services with a shared key. This method is robust when detecting licenses built with no authorization. An enhancement of the protocol to identify the last usage of a certain license is also provided, allowing to detect a traitor when an unauthorized copy of a license is used.

1 Introduction

The distribution of software, protected by licenses, has always been an issue for the intellectual creators and their business' models. Implementing countermeasures for software piracy is a must for the software industry even before the boom of the Internet era, the *RojaDirecta* saga should make clear why the software industry -and the content industry- is looking for new enforcement tools [1]. When the Internet was not so popular, sharing licenses was hard and the methods employed to protect software were merely based in built-in passwords. Reverse engineering, decompilation and other not so advanced techniques made it possible; a further step was then adopted, hash functions and cryptography. As the code to check licenses was embedded in the program, the illegal usage and duplication of licenses were an easy task. Support to avoid the discovery of built-in hash functions was also provided from the Operating Systems in order to harness possible abuses. As the Internet began to be used as a medium to distribute licensed software (p2p networks are the proof), the licensing protection mechanism began to be weaker, those who cracked software posted their tricks or distributed software packages to let any user unprotect software.

As the Internet acquired more relevance, protecting licenses and software became harder. Software is illegally cracked and then redistributed, in a high percent of the distributions *malware* is included in the distributed packages,

therefore creating ways to protect software is necessary not only for the sake of the creators' economy but for the integrity of the users, that might be compromised. As a solution, many authentication services were moved to Internet servers, using licensed software means that it might be necessary establishing a first connection to a server in order to authenticate the software and let it run. Examples of such software range from antivirus software to games. Implementing a license protection is a hot spot in software factories like the antivirus industry. The antiviruses need a license to retrieve updated virus signatures databases in order to keep the system protected. The strength of an antivirus, among others, relies on its databases. Antivirus companies invest so much effort in building such databases. A duplicated license harness this useful effort.

This paper proposes a method and a feasible architecture to access services through the Internet, using licenses that cannot be replicated, duplicated, shared, or even cracked. Concurrent usage of licenses would trigger the alarm system. When licenses are not used concurrently, the method is able to detect the abuse and report it to the legitimate user of the original license. The licenses do not add information on users' identity as opposed to other contributions like the one developed in [2] and protects identity of legitimate users as in [3]. This work not only detects false licenses but also abuses (like duplications, stolen keys, etc) of original licenses.

1.1 Agent Based Computing

Internet is not as robust as your company's wifi network. Traditional methods to establish connections to Internet servers do not offer the dynamism that a robust licensing mechanism may need so we based our method on a layered multiagent architecture [4].Distributed systems based on agents are very attractive because of their inherent scalability and autonomy. Viewing software agents (usually in the form of objects) as brokers and the interactions among agents as the exchange of services, a multiagent system closely resembles a community of human beings doing business with each other, and are widely envisioned to be able to perform many commercial activities on behalf of human beings. Useful properties of agents (see [5]) that distinguish them from traditional processes or threads:

- reactive: responds fast to changes in the environment.
- autonomous: controls its own actions.
- proactive: does not simply act in response to the environment.
- temporally continuous: continually running process.
- communicative/sociallyable: communicates with other agents
- learning/adaptive: changes its behaviour based on its previous experiences.
- mobile : able to migrate from one machine to another.
- flexible: the answer to an event is not predefined.

A multiagent system is a collection of agents that work in conjunction with one another. They may cooperate to achieve a common goal or compete to achieve

individual goals. In this system the use of agents enhances communication and allows for speedier transfer of data. Agent oriented computing differs from centralized object based computing in several ways and as such requires different analysis and design methodologies [6]. Devices such as smartphones have seen a major upsurge in popularity. Using a multiagent layered architecture to stream media to those devices is efficient, as shown in [7].

1.2 Multiagent layered architecture proposal

A multiagent layered architecture, as shown in figure 1, is the implementation chosen for the method proposed in this paper to protect media/content streaming, software or whatever that is covered by a license restriction.

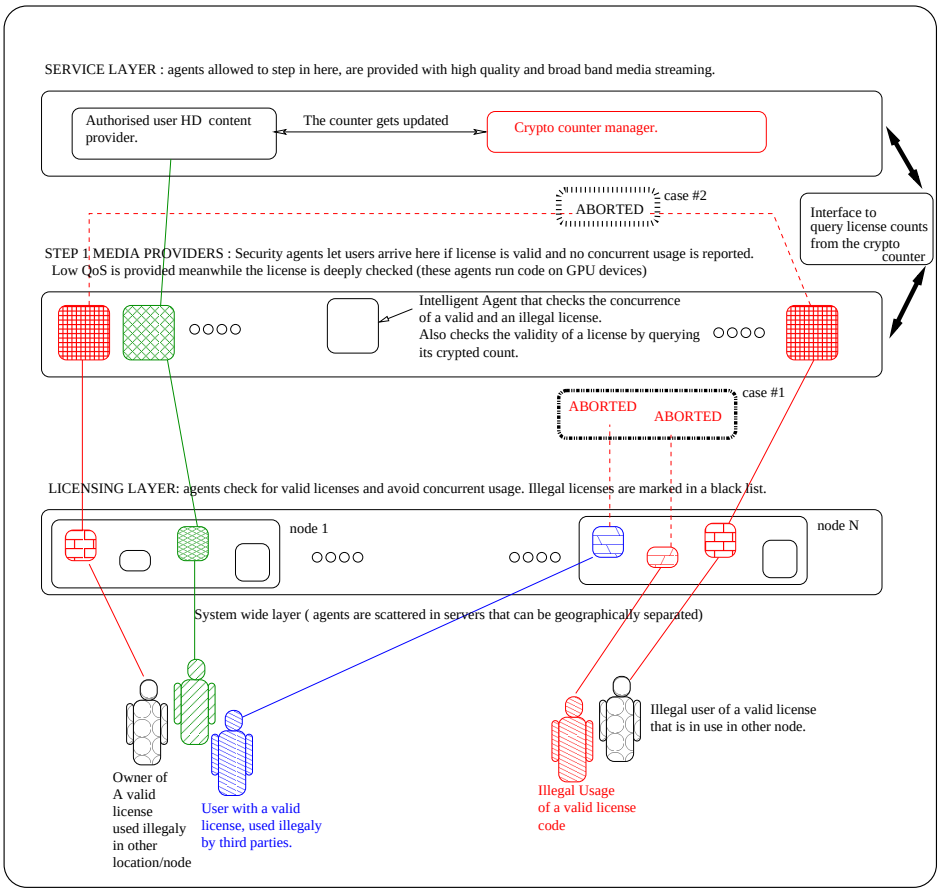

Fig. 1. Multiagent layered architecture proposal

As sketched in figure 1, our proposal establishes several layers of cooperating and intelligent agents to solve the problem of whether a license is valid or not. An agent can migrate to the upper layer taking with itself all the necessary information about the owner of the license. When an user connects to the server, an agent is selected to manage the licensing procedure and the connection at the first stage, the *Licensing Layer*, here the agents verify whether there is another license running at the same server using the protocol devised in section 2. In figure 1 it can be seen that if there exist several copies (see figure 1, case 1) then the connection is aborted and the owner of the license is reported (the license is then blacklisted). It may happen that the connection with an illegal copy of a license (while the valid one is active) is taking place in a different server, in this case the agent would proceed to the next step, the *Media Providers* layer where the user is provided with a low quality streaming, meanwhile the agent is verify globally if the license corresponds to a legitimated user. If another copy is running, then as in case 1, the licenses are blacklisted and the users are reported (see figure 1, case 2). Otherwise, the agent proceeds to migrate to the next layer, the *Service Layer*, where a High Definition media is streamed to the end user. The last scenario may contemplate a non-concurrent-in-time usage of licenses, in such situation the **cryptocounter**, see section 2.2, is used to provide valid access to the content. Following sections are devoted to deeply explain the protocol devised in this section.

2 Method

The method used to detect, in this paper, illegal usage of licenses is based on a existing protocol for digital cash over finite fields. The original protocol can be found in [8, Chapter 11].

We are assuming that the services provided by the Web Server make use of a session key, namely K_S.

Set up phase: The Server chooses large primes p and q such that $p = 2q + 1$ and g the square of a primitive element in $\mathbb{Z}_p$. The server calculates $g_1 \equiv_p g^{k_1}$ and $g_2 \equiv_p g^{k_2}$ for k_1 and k_2 secret, and random numbers and makes public

$$\{p,\ q,\ g,\ g_1,\ g_2\}$$

Three hash functions are also selected: H_1, that takes an integer and returns, also, an integer and H_2 and H_3 that take 5 and 4 integers respectively and return each, a single integer mod q.
The Server chooses its secret identity x and computes and makes public

$$\{h_0 \equiv_p g^x,\ h_1 \equiv_p g_1^x,\ h_2 \equiv_p g_2^x\}$$

2.1 Protocol

1. When an user logs into a content Server by the first time, the agent selected to guide him through the connection, chooses u randomly and sends $I \equiv_p g_1^u$ to the content Server.

2. The Server stores I along with some information to identify the user and sends, to the agent, a private ticket that will be used to get the corresponding session key K_S every time the user wishes to access the service.
3. The Server chooses w and computes and sends to the agent the triple

$$\{z' \equiv_p (Ig_2)^x,\ h \equiv_p g^w,\ k \equiv_p (Ig_2)^w\}$$

4. The agent then chooses seven random numbers $(s, x_1, x_2, n_1, n_2, n_3, n_4)$ and computes:
   - $A \equiv_p (Ig_2)^s$.
   - $B \equiv_p g_1^{x_1} g_2^{x_2}$.
   - $z \equiv_p z'^s$.
   - $a \equiv_p h^{n_1} g^{n_2}$.
   - $b \equiv_p k^{sn_1} A^{n_2}$.
   - $a' \equiv_p h^{n_3} g^{n_4}$.
   - $c \equiv_q n_1^{-1} H_2(A, B, Z, a, b)$ and $d \equiv_q n_3^{-1} H_2(A, B, z, a, H_1(T))$, where $H_1(T)$ denotes the hash value of the user's ticket T.
5. The agent sends to the Server, the pair $\{c,\ d\}$
6. The Server computes and sends back to the agent the pair

$$\{c_1 \equiv_q cx + w,\ d_1 \equiv_q dx + w\}$$

7. The agent computes $r \equiv_q n_1 c_1 + n_2$ and $r' \equiv_q n_3 d_1 + n_4$.
   The license corresponding to the ticket T will be then

$$L = \{A,\ B,\ z,\ a,\ b,\ r,\ a',\ r'\}$$

Theorem 1. *The following equalities hold for a* $L = \{A,\ B,\ z,\ a,\ b,\ r,\ a',\ r'\}$ *corresponding to the ticket* T*:*

$$g^r \equiv_p a h_0^{H_2(A,B,z,a,b)},\ A^r \equiv_p b z^{H_2(A,B,z,a,b)},\ g^{r'} \equiv_p a' h_0^{H_2(A,B,z,a,H_1(T))}$$

Proof. Firstly we recall that $r' \equiv_q n_3 d_1 + n_4$ and thus $r' \equiv_q n_3 dx + n_3 w + n_4$. Then $d \equiv_q n_3^{-1} H_2(A, B, z, a, H_1(T))$

$$\begin{aligned} a' h_0^{H_2(A,B,z,a,H_1(T))} &\equiv_p h^{n_3} g^{n_4} h_0^{H_2(A,B,z,a,H_1(T))} \\ &\equiv_p (g^w)^{n_3} g^{n_4} (g^x)^{H_2(A,B,z,a,H_1(T))} \\ &\equiv_p g^{wn_3} g^{n_4 + xH_2(A,B,z,a,H_1(T))} \\ &\equiv_p g^{wn_3 + n_4 + xH_2(A,B,z,a,H_1(T))} \\ &\equiv_p g^{wn_3 + n_4 + xn_3 d} \\ &\equiv_p g^{r'} \end{aligned}$$

Analogously we get $g^r \equiv_p a h_0^{H_2(A,B,z,a,b)}$

Let us show now that $A^r \equiv_p b z^{H_2(A,B,z,a,b)}$.

$$
\begin{aligned}
bz^{H_2(A,B,z,a,b)} &\equiv_p k^{sn_1} A^{n_2} z'^{sH_2(A,B,z,a,b)} \\
&\equiv_p (Ig_2)^{wsn_1} (Ig_2)^{sn_2} (Ig_2)^{xsH_2(A,B,z,a,b)} \\
&\equiv_p (Ig_2)^{wsn_1+sn_2+xsH_2(A,B,z,a,b)} \\
&\equiv_p (Ig_2)^{wsn_1+sn_2+xn_1sc} \\
&\equiv_p ((Ig_2)^s)^{wn_1+n_2+xn_1c} \\
&\equiv_p A^{wn_1+n_2+xn_1c} \\
&\equiv_p A^{n_1(w+cx)+n_2} \\
&\equiv_p A^r
\end{aligned}
$$

since $H_2(A, B, z, a, b) \equiv_q n_1 c$ and $c_1 \equiv_q w + cx$.

8. The user accesses the content Server and demands the license.
9. The agent in charge of driving the validation procedure, sends to the Server the pair $(h(T), L)$ corresponding to the currently logged user.
10. The Server checks that the equalities of Theorem 1. If these holds, then the Server stores the license and computes the hash $H_3(A, B, H_1(T), t)$ where t denotes a time-stamp.
11. The agent now computes

$$s_1 = H_3(A, B, H_1(T), t)us + x_1 \quad \text{and} \quad s_2 = H_3(A, B, H_1(T), t)s + x_2$$

where u is the private information generated by the agent in step 1 and s, x_1 and x_2 are the random integers generated also by the agent in step 4.

12. The Server checks that $g_1^{s_1} g_2^{s_2} \equiv_p A^{H_3(A,B,H_1(T),t)} B$ and if so, then the client is sent the session key K_S encrypted using T and the pair $(h(T), L)$ is stored along with the 4-tuple $(s_1, s_2, H_3(A, B, H_1(T), t))$ until the user leaves/logs out the system.

Theorem 2. *With the above notation, the following equality holds*

$$g_1^{s_1} g_2^{s_2} \equiv_p A^{H_3(A,B,H_1(T),t)} B$$

Proof. Since $I \cong_p g_1^u$, $A \equiv_p (Ig_2)^s$ and $B \equiv_p g_1^{x_1} g_2^{x_2}$ we get that

$$
\begin{aligned}
g_1^{s_1} g_2^{s_2} &\equiv_p g_1^{H_3(A,B,H_1(T),t)us+x_1} g_2^{H_3(A,B,H_1(T),t)s+x_2} \\
&\equiv_p (g_1^u)^{H_3(A,B,H_1(T),t)s} g_1^{x_1} g_2^{H_3(A,B,H_1(T),t)s} g_2^{x_2} \\
&\equiv_p (g_1^u g_2)^{H_3(A,B,H_1(T),t)s} g_1^{x_1} g_2^{x_2} \\
&\equiv_p ((Ig_2)^s)^{H_3(A,B,H_1(T),t)} g_1^{x_1} g_2^{x_2} \\
&\equiv_p A^{H_3(A,B,H_1(T),t)} B
\end{aligned}
$$

Proposition 1. *The above protocol avoids the usage of a certain license by two different users simultaneously, it protects and avoids the usage of a stolen license and the reuse of a recorded message previously submitted for a requested service.*

Proof. If a legitimated user shares all his private information (which comprises the ticket, the license and all the private numbers generated through all the steps of the protocol) with someone else and this fake license is used to authenticate $(H_1(T), L)$ along with another triple $(s_1', s_2', H_3(A, B, H_1(T), t')$ while the legitimated user (or a third person holding the information) is still logged

in the system, then the Server, as it is shown in [8, 11.1.9], the private information u that identifies the legal user by means of $I \equiv_p g_1^u$ is derived from $u \equiv_q (s_1 - s_1')(s_2 - s_2')^{-1}$.

Note that if someone steals a license $(H_3(T), L)$, in order to get the service, the user needs to know, also, the private information u to produce s_1 and s_2. Trying to forge these numbers implies satisfying the Theorem 2 which is a hard problem since it involves the discrete logarithm.

The reuse of a recorded message $(H_1(T), L)$ is equivalent to the previous comment.

Proposition 2. *The above protocol provides anonymity to users.*

Proof. It easily observed that through this protocol there is no possibility for anybody, including the Server, to compute u, which is the only information that identifies the user.

Proposition 3. *Only the corresponding authority (the server) is able to provide valid licenses.*

Proof. Computing numbers, that verifies the Theorem 1, involves discrete logarithms, for instance even a legal user knowing $A,\ B,\ z,\ a$ and T will need to solve a discrete logarithm in order to produce r' such that $g^{r'} \equiv_p a' h_0^{H_2(A,B,z,a,H_1(T))}$

However, since an user might share his key with somebody else and due to anonymity, this unauthorized usage could be detected only in the case in which the legal and the unauthorized users make use of it simultaneously as noted in Proposition 1.

2.2 Detecting copies of licenses

The purpose of this section is to settle an extension to the method in order to detect when a copy of a license, when no other copy of it is active at the moment, is used and determine who is responsible of the corresponding sharing (violation).

We now introduce a new parameter in the license L that is nothing but a cryptocounter where the only one able to update values of the counter is the owner of a certain private key. A version also based on the discrete logarithm is easily obtained as follows:

Let $f \in \mathbb{Z}_p$ the initial value of the counter and let k_1 be random. Then the first value of the counter is $(a_1, b_1) = (g^{k_1} \ mod\ p, (g^c)^{k_1} \ mod\ p)$. To increase the counter we make $(a_2, b_2) = (a_1 g^{k_2} \ mod\ p, b_1 (g^c)^{k_2} g \ mod\ p)$, for c some secret value owned by the Server.
Now given $(a_n, b_n) = (a_{n-1}^{k_n} \ mod\ p, b_{n-1}(g^c)^{k_n} g \ mod\ p)$, then $a_n^{p-1-a} b_n = g^n$ and so only the Server, who is the only knowing a can get g^n and thus, via a simple search, n.

The application is now clear. Thus a license is defined as

$$L = \{A,\ B,\ z,\ a,\ b,\ r,\ a',\ r',\ c'\}$$

for c' a cryptocounter as above. The Server has to store a copy of the license and the 4-tuple appearing in Step 12. Then each time a license is used, the cryptocounter is increased by the Server. When the user leaves the system, the Server sends the user L' with the corresponding value c' increased.

Thus, the protocol is the same until step 9. In step 10, the Server checks the same equalities as above and that the corresponding c' equals the stored value. In case the value that c' gives rise is less than the value that provides the stored one, this means that a copy of a license is being used to access the service. Then the protocol follows the same Steps 10, 11 and 12 and operates as in Proposition 1 to detect the legal user that shared all his private information.

Thus, the only way to avoid fraud detection when using this protocol is that both legal and unauthorized users share the information that the Server sends back after leaving the service each time. This attack could be carried out when the group of people which the legal user shares his license with is formed by few and trusted and people, since in other case it is quite probable that one of the members of the group shares the information to others and the situation could become one of those that the protocol detects as fraudulent.

3 Implementation

The implementation of such a system need to be scalable. Common agents' platforms tend to be inefficient to handle thousands of agents concurrently operating on a same server with different calculations (checking licenses' validity). The implementation we propose here is one built on high performance concurrent and migratable objects provided by the charm++ [9] runtime. Implementations for high performance and costly applications are proven to be efficient in terms of concurrent usage of resources, scalability and load adaptivity issues when using message-driven computing. To build our model, high-level abstractions are useful to target the model of the implementation without having to deal with details of the architecture underneath [10]. A farm cpan based model (see Fig.2) was used to have fully concurrent charm objects per processor. Load balancing is executed asynchronously as exposed in [11]. Figure 2 shows an implementation proposal for the licensing server. To instrument the agents layer, the platform proposed in [12] was used as a model for our implementation so both, agent communications and overloaded computing nodes were efficiently reduced.

As figure 2 shows agents are labeled as *slave objetcts*. Whenever a new server is added to the infrastructure, then a new *node object* is created to host and monitor the computation. Slaves can be sent to the new node when the node is set to available. Slave objects communicate using messages (remote invocation

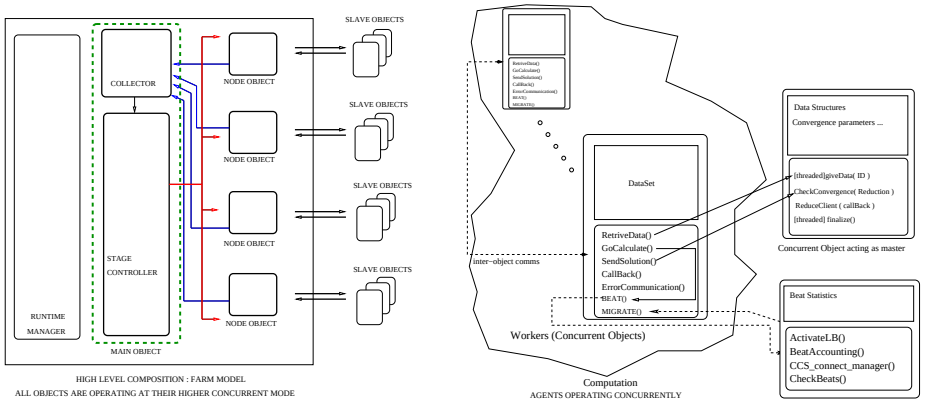

Fig. 2. Multiagent layered implementation proposal

of their methdos, accesible via proxy objects). In order to avoid that the monitoring of the computation affects performance, agents are able to send *signals* to a special object that do not belong to the computation. This object gathers signaling information from agents and is able to detect a decrease in performance [11] so load balancing is invoked asynchronously and agents are moved to other computing nodes.

4 Conclusions

A new protocol to protect services provided through the Internet has been shown and proved to work efficiently when illegal usage of licenses is taking place, even when the licenses are not in use at the same time. Also, a layered multiagent architecture was sketched in order to show a feasible implementation. The agents' platform proposed is built by composing high performance and concurrent objects that communicate using asynchronous messages. Such issue enables the proposed platform to interleave computation and communication phases, so when an agent is communicating, then another is efficiently scheduled to do its CPU related computations. We have, also, used and enhanced our proposal by following the implementation sketched in [12] to relieve the effect of unmaskarable latencies and to efficiently move agents from one processor to another.

These latency hiding issues and the migratable characteristic of the cited objects are prone to build high level and efficient compositions whose details are not tied to machine specific architectures because the charm++ runtime deals with them for us. The proposed platform, that we have developed, to implement the security protocol exposed in this paper, seizes these properties.

References

1. Picker, R.C.: The yin and yang of copyright and technology. Commun. ACM **55**(1) (2012) 30–32
2. Lou, X., Hwang, K.: Collusive piracy prevention in p2p content delivery networks. Computers, IEEE Transactions on **58**(7) (july 2009) 970 –983
3. Ding, Y., Fan, L.: Traitor tracing and revocation mechanisms with privacy-preserving. In Wang, Y., ming Cheung, Y., Guo, P., Wei, Y., eds.: CIS, IEEE (2011) 842–846
4. Kosuga, M., Yamazaki, T., Ogino, N., Matsuda, J.: Adaptive qos management using layered multi-agent system for distributed multimedia applications. In: ICPP. (1999) 388–394
5. Franklin, S., Graesser, A.C.: Is it an agent, or just a program?: A taxonomy for autonomous agents. In Müller, J.P., Wooldridge, M., Jennings, N.R., eds.: ATAL. Volume 1193 of Lecture Notes in Computer Science., Springer (1996) 21–35
6. Silva, D., Braga, R., Reis, L., Oliveira, E.: A generic model for a robotic agent system using gaia methodology: Two distinct implementations. In: Robotics Automation and Mechatronics (RAM), 2010 IEEE Conference on. (june 2010) 280 –285
7. Leetch, G., Mangina, E.: A multi-agent system to stream multimedia to handheld devices. In: Computational Intelligence and Multimedia Applications, 2005. Sixth International Conference on, IEEE (2005) 2–10
8. Trappe, W., Washington, L.: Introduction to cryptography: with coding theory. Pearson Prentice Hall (2006)
9. Kale, L., Arya, A., Bhatele, A., Gupta, A., Jain, N., Jetley, P., Lifflander, J., Miller, P., Sun, Y., Venkataraman, R., Wesolowski, L., Zheng, G.: Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge. Technical Report 11-49, Parallel Programming Laboratory (November 2011)
10. Capel Tunon, M., Lopez, M.: A parallel programming methodology based on high level parallel compositions (cpans). In: Electronics, Communications and Computers, 2004. CONIELECOMP 2004. 14th International Conference on. (feb. 2004) 242 – 247
11. Alvarez-Bermejo, J.A., Roca-Piera, J.: A proposed asynchronous object load balancing method for parallel 3d image reconstruction applications. In: Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I. ICA3PP'10, Berlin, Heidelberg, Springer-Verlag (2010) 454–462
12. Jang, M.W., Agha, G.: Adaptive agent allocation for massively multi-agent applications. In Ishida, T., Gasser, L., Nakashima, H., eds.: Massively Multi-Agent Systems I. Volume 3446 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2005) 575–575