A Video Transmission Framework Using Components and Multi-Agent Systems

Vincent Pretre, Christophe Lang, Jean-Christophe Lapayre, Nicolas Marilleau

A video transmission framework using components and multi-agent systems

Pretre V.*, Lang C.*, Lapayre J-C*., Marilleau N.*

*Laboratoire d'Informatique
de l'Université de Franche-Comté
CNRS FRE 2661
16 route de Gray
25030 Besançon CEDEX
{pretre, lang, lapayre, marilleau}@lifc.univ-fcomte.fr

Abstract

This paper presents an application of video transmission over Internet, which goal is to be used in a cooperative platform.
This application uses proxies during the transmission to adapt the video (changing the size, the framerate and/or the encoding format). Adaptation is a necessity in this kind of application, due to the diversity of receivers (computers, mobile phones, PDA ...).
During the design process, we chose to use component oriented programming and multi-agent systems. We present here how these two paradigms help us to have a flexible and evolutive application, and, for each transmission's step, what is the most appropriated solution.
At the end, we also present tests that have been made to evaluate the power needed by the proxies in order to discuss about benefits that can be brought by our architecture.

Keywords:

video transmission, component oriented programming, multi-agent system, video adaptation.

1. INTRODUCTION

Video transmission is an important part of cooperative applications, it makes possible the transport of information that could not be possible with standard text transport.
It can be really useful in e-health applications, that a doctor can diagnose his patient distantly, discuss with a colleague about a patient or view a medical act live.

The major goal of this paper is to present our video transmission application, based upon multi-agent systems and component oriented programming, two paradigms which are actually not much used in this type of application.

The main problem with such application is the diversity of supports that will receive video. For example, same video file can be received by a computer and a mobile phone. We could use the same format for the two receivers, but it would not be an efficient solution. If we optimize the video for the computer, it will certainly be hard to read for the phone (too large for the screen, too hard to decode and too big for its bandwidth), and if we optimize it for the phone, it will not be smart to read it on the computer (video too small).
The diversity in the receivers is not the only one, there is also numerous formats for the video encoding, and all receivers will not support same formats.
In order to resolve these problems, one solution is to adapt the video for each receiver.

We first speak about Video adaptation and the role of agents and components in this area. The next part presents tests we have done in order to evaluate the power needed by the adapters, and which parts of the adaptation are the most greedy.
In another part, we present the architecture of our application, where we use agents and/or components, and what they bring more than classic programs.
Finally, we conclude and present our future works.

2. VIDEO ADAPTATION

We have localized three parameters involved in video adaptation. The first one is the framerate (the number of images composing a second of the video). When we reduce it, the video gets smaller and is easier to decode. It is chosen according to the power and the bandwidth of the receiver.
The second one is the size images composing the video. Its reduction make video smaller and easier to decode. As the framerate, it is chosen according to the bandwidth and the power of the receiver, but there is one parameter added, the size of the receiver's screen.
The last parameter we can change is the encoding format of the video. There exists two major types of encoding formats, non-destructive and destructive. The first one produces video easy to decode (because the images are pure,

the decoder has not to recreate them), but heavy (because, as said before, the images are pure). The second type produces light videos, but hardest to decode (images being part destructed, decoder has to recreate them). The encoding format is the hardest parameter to choose. It is a compromise between needed bandwidth and power. It also has to be chosen according to the known formats of the receiver.

Currently, there exists two major ways for the video adaptation. The first one is to adapt the video at the source. The advantage of the method is that it does not need computer over the network to realize the adaptation. But there are two major disadvantages with this method. The first one is that the sender will have more work to do, and may be quickly overloaded, the second one is that if we have to send the same video to several receivers, but with different parameters, the sender will have to multiply the adaptations. The second solution is to use computers (which will be called 'adapters') over the network that will do the adaptation. This solution is more expensive and harder to set up, because it needs to create discussion protocols between sender, receiver and adapters. But great advantages of this solution are its modularity and the fact that the sender does not have to do adaptation. Modularity is given by the fact that the sender and the receiver will be minimal, and when we want to add new functionalities to the application, we do not have to change them, but only adapters.

For our application, we focus on the second solution, but we decided to extend it with agents and components. With these two methods, the senders, receivers and adapters can be distributed, and can use others computers power on their local network.

3. BRINGING COMPONENTS AND AGENTS TO VIDEO ADAPTATION

3.1 Multi-agent systems

Distributed tools, particularly Multi-Agent Systems (MAS) are an interesting area that can help us for our problem. A MAS is composed by autonomous entities, called agents which interact together and with their environment in order to accomplish tasks [6]. A MAS is also composed by an environment where agents can evolve.

Each agent has its environment perception and does actions in accordance with its behavior and its view. In [8], Jacques Ferber suggests a "weak" and a "powerful" definitions of agent. MAS are used in many domains as geography, robotic, sociology.... We can organize agents into two main categories. An agent can be cognitive in other words, an agent can keep information by using a memory and does action in accordance with its view and its memory. Contrary to cognitive agent, reactive agent doesn't have any memory. Therefore, it replies to its view by doing an action on his world. Cognitive agents are divided into sub-types which one is interesting. Indeed BDI agent and particularly VON-BDI agent [1] (Value Obligation, Norm, Belief, Desire, Intention) are humanistic agents. Each of them associates beliefs with desires to compute intentions. VON-BDI agent derives of BDI agent. Few parameters have been appended to BDI agent in order to introduce Value, Obligation and Norm concepts. These new properties allow us to represent habit (value and norm) and prohibition.

In many areas, MAS are more and more employed to solve some -usually hard to solve- problems. They give a new distributed view to those problems. Thus, they often bring new solutions that can not be handled by more monolithic strategies. Moreover, each agent is an improved object that can take some decisions in accordance with the context and its past. That is why, we need to explore this area to reveal the contribution to our problem.

Currently, there is not many applications using multi-agent systems for video transmission : [3] presents the use of multi-agent system to decode a mpeg2 video, and [4] uses agents in adaptative routers.

3.2 Components

Component is an useful concept in software engineering because it helps computer scientists to design and develop softwares. It is a basic element we associate with other components in order to create an application.

Frédéric Peschanski in [10] defines software components as *reuseable elements which compound applications*. In fact, they can be seen as entities which realize particular tasks according to their functionalities and their behaviours. They are *independent* and *reusable* because they can be deployed in different systems without adding tools.

A component contains several interfaces. They are defined by operations we call in order to configure entities or to realize tasks. Each of them has internal functions which implement component behaviour. They are called by user through interfaces. So, a component can be imagined as a box we access, thanks to its interfaces, in order to do tasks.

Several component models are presented in the literature. Each of them have few specificities. These properties imply an area where a component of a model can be used. For example, *JavaBeans* [7] allow developers to create component, *a Bean*, which can be used and managed with a graphical user interface.*Enterprise Java Beans* [2] or *Corba Components* [9] are useful to create distributed applications based on component.

Distributed components like EJB are managed by application servers like *JONAS* or *JBOSS*. Their goal is to manage component life cycle (create or destroy an instance), persistance (associate a component with a database), transaction and so on. In fact, an application server allows components to be accessed by network customers and execute component code.

A distributed component based approach is an interested way we need to explore. Indeed this type of element allows us to create services used by agents in order to achieve their tasks.

Like agents, components are not currently very used in video transmission. However, two papers talk about their utilisation with video. The first one is the Friend platform ([11]), which is based on component oriented programming and provides video conferences services.
The second one is about the Chili software([5]), a teleradiology platform built on components, which can do video transmission.

Components and agents are similar on two points: they can be called on a distant computer and they help to have a flexible application, but the agent is autonomous, unlike the component.
This mainly means two things: the agent does not only react to the application that calls him, he can also react to his environment (for example the load of the processor) and he can continue his work even if the program which called him has been stopped.
A component can continue to exist after the stop of the calling application, but it will not do anything more, just waiting for an application to ask him work.
A component can continue to exist after the stop of the calling application, but it will not do anything more, just waiting for an application to ask him work.

4. PRELIMINARY TESTS

As we wrote above, the tests we made have two goals: evaluate the power needed by adapters, and locate which parts of the adaptation are the most asking for power.

For these tests, we have to handle different operations:

- decoding a video;
- decoding, then encoding a video in the same format (with the previous operation, we can evaluate the load of the encoding operation);
- changing the framerate of a video;

All these operations are done on mpeg2 encoded video, it is currently the only format supported by our application. We will not do tests of changing the video size, this operation is not yet supported by our application.

These tests have been executed on five computers. The first one, named Roc, has an Athlon K6(2) 300MHz processor, and run with the GNU/Linux Debian Sarge OS.
The second computer, called Diabolo, has an Athlon 1400+ processor, and uses the GNU/Linux Debian Sid OS.
The third one is called Winnie, its processor is an Intel Pentium III rated at 1200Mhz, and runs with GNU/Linux Debian Sarge.
The fourth one, is called Boromir and has a processor. Its operating system is GNU/Linux Fedora core 3.
The last one, Paki1, has a 2600MHz Intel Pentium 4 processor, and uses GNU/Linux Debian Sarge OS.

4.1 Presentation of the tests

For these tests, we only use kernels of our agents and components. These kernels are written in C++, whereas the agents and components are written in Java (using JNI to communicate with their kernels).
The fact that only kernels are tested makes that the load noticed is not the final application one, but this will be useful later when the application will be finished, to compare and evaluate the power needed by agent and component layer.

During our tests, we did not work on streams but on files, that means that there are disk accesses that will not be present in the final application. For each computer, we evaluated the time to read the complete file (using the 'time cat file > /dev/null/' command). There are no writing during our test, the output is redirect to /dev/null. These commands gave these results:

- Roc, 0.613 sec.
- Diabolo, 0.171 sec.
- Winnie, 0.143 sec.
- Boromir,
- Paki,

Computers we use can execute several programs at the same time. We will use this property to evaluate how many videos a computer can adapt.
A test is conclusive when the time needed to treat a video is lesser than the length of the video. If this time is greater, we will have latency in the treatment, and so for the receiver.

The computers are isolated from the network during the tests, so there can not be perturbed during tests.

To evaluate the power needed by the adaptation, we use two Unix commands: 'cat /proc/loadavg', gives us the load of the processor before the execution, and 'time', which gives us the time taken by the program to run and the percentage of processor used.

In the aim to have representative tests, we run each test one hundred time. Results shown in this paper are the average of the results obtained.
To simplify the reading of the results, we use these abbreviations:

- clBt: CPU load before test;
- clDt: CPU load during the test;
- dT: duration test;
- mdt: minimum duration test;
- Mdt: maximum duration test;
- nvt: number of video treated at the same time;
- vfAa: video's framerate after adaptation;
- eR: efficiency ratio ($adT/videoduration$), the test is conclusive when this value is lesser than one.

4.2 Test results

The first test we done is the decoding of a 34.64 seconds long video. The same video will be used for all the tests. Results of these tests are available in table 1.

The goal of the second test is to decode a video and encode it again in the same format. To do this, we extract each frame of the input video and put it in the output video.
We have ran this test several time on each computer to evaluate how many videos can be treated by each computer. We incremented the number of video adapted until the efficiency

Table 1: Decoding tests

machine	clBt	clDt	dT	mdt	Mdt	eR
Roc	1.02	98.97	7.67	7.39	8.72	0.21
Diabolo	0.23	97.82	1.37	1.35	1.49	0.03
Winnie	8.45	98.42	1.24	1.2	1.58	0.03
Boromir	6.05	98.95	1.01	0.99	1.07	0.02
Paki	7.91	98.94	0.79	0.78	0.84	0.01

Table 2: Decoding/encoding on Roc

nvt	clBt	clDt	mclDt	MclDt	dT	mdt	Mdt	eR
1	1	99.99	98	99	40.1	39.55	42.1	1.21

Table 3: Decoding/encoding on Diabolo

nvt	clBt	clDt	dT	mdt	Mdt	eR
1	1.79	96.45	6.11	5.91	7.26	0.17
2	2.53	87.89	12.33	11.89	13.51	0.35
3	3.65	83.34	18.53	17.93	19.65	0.53
4	4.98	89.19	24.7	23.75	25.76	0.68
5	7.30	72.97	31.09	29.94	32.03	0.89
6	8.29	76.89	39.21	36.1	40.87	1.13

Table 4: Decoding/encoding on Winnie

nvt	clBt	clDt	dT	mdt	Mdt	eR
1	2.42	98.78	6.78	6.74	7.13	0.19
2	2.71	85.95	13.51	13.43	13.91	0.39
3	3.81	90.31	20.26	20.1	20.7	0.58
4	4.83	88.72	26.99	26.82	27.39	0.77
5	5.84	78.96	33.67	33.43	34.02	0.97
6	6.59	74.29	40.40	40.07	40.99	1.16

Table 5: Decoding/encoding on Boromir

nvt	clBt	clDt	dT	mdt	Mdt	eR
1	1.01	98.84	4.06	4.02	4.44	0.11
2	1.65	96.94	8.12	8.05	8.57	0.23
3	2.73	94.55	12.16	12	12.64	0.33
4	3.83	85.98	16.17	15.73	16.59	0.46
5	4.84	72.18	20.23	19.94	20.62	0.58
6	5.57	60.31	24.26	23.82	24.72	0.7
7	6.87	62.95	28.29	27.7	28.76	0.81
8	7.22	62.17	32.33	31.81	32.75	0.93
9	8.90	65.27	36.34	35.42	36.99	1.04

Table 6: Decoding/encoding on Paki1

nvt	clBt	clDt	dT	mdt	Mdt	eR
1	3.18	99.06	3.42	3.42	3.44	0.09
2	1.76	90.64	6.86	6.8	6.92	0.19
3	2.67	90.83	10.29	10.23	10.37	0.29
4	3.75	85.59	13.73	13.51	13.9	0.39
5	4.71	86.49	17.15	16.35	17.91	0.49
6	5.47	72.87	20.57	20.36	20.91	0.59
7	6.82	91.82	23.99	22.62	24.82	0.69
8	7.79	90.53	27.41	27.11	24.74	0.79
9	8.78	89.28	30.83	30.58	31.1	0.89
10	8.94	93.38	34.35	34.11	34.53	0.99
11	10.34	84.43	37.70	35.44	40.09	1.08

Table 7: Evaluation of encoding duration

machine	duration evaluation	eR
Roc	32.43	0.93
Diabolo	4.74	0.13
Winnie	5.54	0.15
Boromir	3.05	0.08
Paki	2.63	0.07

ratio was greater than one. Results of these tests are available in tables 2, **??** and 3.

With these two tests, we can evaluate the power needed by the encoding, we obtain results shown in table 7.

The third test is about the changing of the video framerate. To do this, we extract each frame of the input video, but we do not put all in the output video.

The mpeg2 format has a limited number of possible framerate, so we can not use all the framerates we want. To do this, we fake a change of framerate, output video is declared with a framerate of 25 images by second (the framerate of the input video), but we build the video as if it had another framerate.

For example, if we want to have an output video with a framerate of 5 images by second, the 25th first images of the video are declared as if they were composing the first second of the video, but compose in fact the 5th first seconds of the video.

This gives video which seems to be accelerated when it is played by a classical player. The one used is our application plays the video slowly, so the video's framerate looks changed.

Results of these tests are available in tables 8, 9, 10, 11 and 12

4.3 Conclusions of the tests

As we thought before the tests, decoding and encoding video ask lot of CPU power, and a simple computer may not be powerful enough to adapt many videos.

Agents and components will be very useful for these operations, making possible to use power of the other computer available on the network. As we have seen with Roc, even computer which are not very powerful can be used for some

Table 8: Changing framerate on Roc

clBt	clDt	dT	mdt	Mdt	vfAa	eR
1.02	97.45	34.92	34.3	37.7	20	1.008
1.02	97.97	33.04	32.62	33.8	15	0.95
1.28	96.26	31.29	30.02	33.85	10	0.9
1.26	97.98	25.06	24.74	25.9	5	0.72

Table 9: Changing framerate on Diabolo

clBt	clDt	dT	mdt	Mdt	vfAa	eR
2.12	44.05	12.50	12.27	13.58	20	0.36
2.32	43.91	11.89	11.62	12.83	15	0.34
1.58	45.92	10.59	10.37	11.56	10	0.30
4.97	46.91	8.49	8.06	9.42	5	0.24

Table 10: Changing framerate on Winnie

clBt	clDt	dT	mdt	Mdt	vfAa	eR
1.94	96.71	6.26	6.21	6.6	20	0.18
6.28	96.69	5.96	5.92	6.3	15	0.17
6.05	94.82	5.76	5.48	6.66	10	0.16
3.96	96.94	4.58	4.53	4.95	5	0.12

Table 11: Changing framerate on Boromir

clBt	clDt	dT	mdt	Mdt	vfAa	eR
1.01	96.14	3.69	3.66	4.05	20	0.1
0.99	96.22	3.54	3.51	3.95	15	0.1
0.52	96.39	3.25	3.23	3.61	10	0.09
3.60	96.59	2.75	2.73	3.27	5	0.07

Table 12: Changing framerate on Paki

clBt	clDt	dT	mdt	Mdt	vfAa	eR
1.71	96.49	3.07	3.07	3.08	20	0.08
7.29	96.55	2.93	2.93	2.95	15	0.08
6.87	96.7	2.69	2.69	2.7	10	0.07
3.77	96.86	2.25	2.25	2.26	5	0.06

Table 13: Framerate change and weight gain

new framerate	new weight	lost frames	weight gain
20	4	20%	17%
15	3.8	40%	21%
10	3.3	60%	32%
5	2.6	80%	46%

work (for example changing the framerate).
Next, we will see how agents can help use to use all the available power on the network.

Another conclusion can be done on the third test (framerate changing). Before this test, we thought that changing the framerate would have an important impact on the video's size.
But, as we can see in table 13, even after a big framerate change (25 to 5), the video's weight has not much decreased (50%, whereas we remove 80% of the frames).
But we can also see that the framerate change has an impact on the CPU power needed, that seems logic, because as we have seen before, encoding is greedy and changing the framerate ask less encoding.

In the next part, we propose a new architecture to handle video adaptation problem.

5. ARCHITECTURE OF OUR APPLICATION

The presentation of our application will be done in three parts. In the first one, we present different actions done during a video transmission.Basically, it will be called : Video Transmission Protocol
The second part presents, for each part, how can we use agents and components.
The third part presents how we use networks of agent to balance the load on the computers used in our application.
The last part speak about choices we have done for our application.

5.1 Video transmission protocol

The first action is the **negotiation**, which is done between the sender, the receiver and the adapters.
During this phase, the format of the video (encoding format, size and framerate) sent and received, and the adaptation work are chosen.

The second action is the **source catching**, it only concerns the sender. The source can have multiple formats, like stream, webcam or file.

The third action is the **encoding**, and is done by the sender. In the case of a file reading, this action does not exists. But if the sender catches images from a webcam, it will have to encode it to do a video.

The fourth action is the **video sending**, in which the sender sends the video to the adapter.

The fifth action is the **video reception**, it is done by the adapter.

The sixth action is the **adaptation of the video**, which is done by the adapter. As we have seen during tests, adaptation is composed of three phases, decoding, treatment and re-encoding.

After adaptation, the adapter sends the video to the receiver. This is in fact the same than the fourth and fifth action.

The seventh action is the **video decoding** which is done by the receiver.

The last action is the **video displaying**, which is done by the receiver.

Presented as it, these actions seem to be executed sequentially, but in fact they do not. Actions are during all along the transmission, and are executed in the same time.

5.2 Using agent and component

Agents and components presented here are called on the local network. We consider this network to be fast and sure, unlike Internet for example.
This makes that we are able to transfer big amount of data between agents or components and the program which calls them without any problem (lost data or long time to transfer).

For the negociation phase, we use agents. We chose them because of their environment perception. This make them able to detect the actual load of the computer, and so evaluate which encoding will be the most efficient.
As explained in the next section, these agents are also used for the load balancing of the application.
One negociation agent is run on each actors of the transmission (sender, adapter and receiver) and a network is created between them to negociate all along the transmission).
We do not use components in this phase because they do not have the environment perception. They could have it, but it is not the role of a component. Furthermore, component are considered to be reacting objects: a program sends it datas, the component treat them and return the result to its caller. This is why components are not adapted to this phase.

Agents will also be used for the source catching. They will be usefull in this part for giving modularity to our application, for each type of source (different webcam model for example) we have a different agent, with the same access interface. This makes our application more modular, to which we can easily add new sources without having to reconfigure it.
The components can not be used for this phase, for the same reason given above. It is not a reaction phase.

For the encoding part, both agents and components are usefull, for two reasons. First, they can be executed on a distant computer, this makes possible to balance the load over the network. The second point is their modularity. Like for the different sources, we can create different agents and/or components for each video encoding format.
We can also use multiple agents or components for the same encoding, distributing this action other the network, using the power available on the network to encode video more quickly.
We will also use agents and/or components for the video decoding, these two actions are quite similar, and we get the same advantages using agents and components in this phase.

The sending phase is realized by agents. The reason is still the modularity, we use a different agent for each transmission protocol we use.
Only one agent is used for each transmission, there is no interest to use multiple agents for one transmission.
Components can not be used for this, the reason is, once again, the non-reacting type of this action.
The phase of reception is close to the sending, so we also use agents for it.

There is two actions for which we have not yet assigned component or agents. The first one is the video display. This is a basic operation, which does not need modularity nor much processor power. It will de done directly by our application.
The second action is the adaptation. As we have seen with the tests, there is not really adaptation phase, it is decoding, treatment then encoding. The treatment are simple operations which do not need much power. As the display, it will be done directly by the application core.

The core of our application, which role is to manage the communication between all the parts, is an agent too.
The main reason is that it simplify the communications. The agent framework we use (MadKit - www.madkit.org) handle the communications between the different agents.
The core will also realize the treatment in the adaptation, as told above.

5.3 Using agents to manage load

We have briefly spoken about using agents to watch for the load of the computer, we will now detail this idea. As presented above, we have actually two networks of agents, the first one is dedicated to the transmission (capturing the source, encoding and decoding the video, sending and receipting the video) and the second is used for the negotiation.
The agents presents on this network will be shared with a third network, the load watching network. This network is in fact a set of subnetworks, each of them placed on the local network of each actor of the transmission (sender, receiver and adapters). These subnetworks do not communicate with each other, information is shared through the negotiation network.

The load watching subnetworks are composed of agents which role is to get the load information of each computer used in the transmission (it can be a computer where a component or an agent is running for example).
When one of these computers get overloaded, his agent will send a message on the watch loading network, in the aim to find another computer to run one of its component or agent. After that, there is two solutions possible: another computer on the network is found, and the component or agent which takes too much power is moved to a computer

which can accept it. The operation is transparent for all other actors of the network. In the second case, if no host can be found, a message is sent on the negotiation network to find another adapter to do the work.

5.4 Overview plan

The figure 1 represents an agent and a component as they are shown in the next plans.
In order to simplify the reading of the plans, we do not show any detailed plan of the complete application.

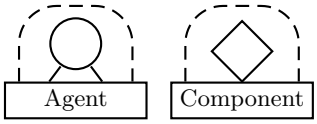

Figure 1: Representation of agents and components

The figure 2 present a example of sender, distributed on the network. It only uses agents, but could use components for the encoding phase.
We do not show figure of a receiver, it is quite the same thing that a sender. The main difference is that there is no source capture and sending agent, replaced by a receiver agent. The treatment chain is inversed of course.
The box on the left represents a computer on the local network, on which the agent in care of the encoding is executing. The edges on the right represent the communication with the other actors of the transmission: the negociation agent communicate with the adapters and the receiver's negociation agents, the sending agent with the first adapter's reception agent.

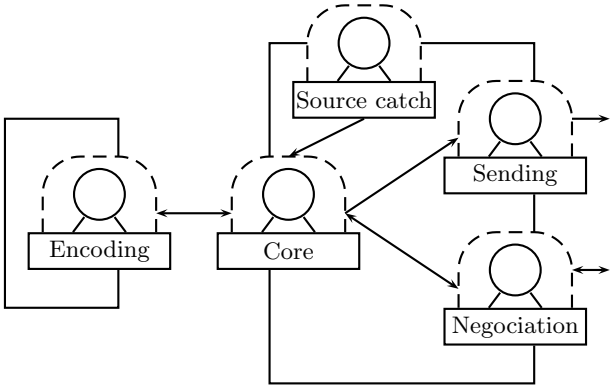

Figure 2: Example of sender

The figure 3 shows an adapter, which uses component for the decoding phase, and multi agents for the encoding.
On this figure and the precedent, load managment subnetworks are not shown in order to clear the plan.

6. CONCLUSION AND FUTURE WORK

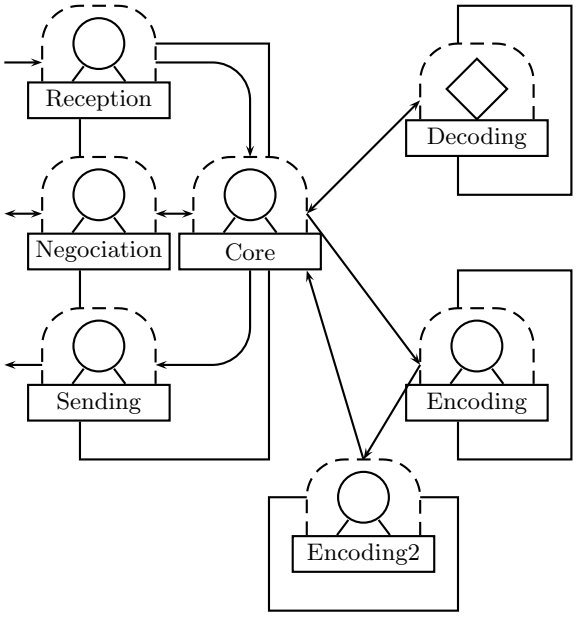

Figure 3: Example of adapter

As we have seen, the major contribution of the components and agents in video transmission is that we can use entire network to work on adaptation, whereas classical applications only use one computer as adapter.
They also make our application able to use computers which are not directly connected to the Internet (for example protected by a firewall), if there is at least one computer which has a direct access. This allows our application to use power which could not be used by other applications.
Our system of load balancing will make our application capable of using times of inactivity of every computer of the network to do the adaptation, and move the work once the computer gets used again.

The next work will be to set up the agent and component framework up to our kernels. Once this done, we will be able to run new tests, which will make us able to evaluate the load added by components and agents.

7. REFERENCES

[1] Gordon Beavers and Henry Hexmoor. In search of simple and responsible agents. In *The GSFC Workshop On Radical Agents*, 2002.

[2] Linda G. De Michiel. Ejb 2.1 specification. Technical report, SUN Microsystems Inc, 2003.

[3] Massimo De Santo, Mario Molinara, and Gennaro Percannella. On the applicability of the multi-agent system paradigm for parsing videos. In *IEEE HICSS-35, Hawaii*, 2002.

[4] Giuseppe Di Fatta, Salvatore Gaglio, Giuseppe Lo Re, and Marco Ortolani. Adaptive routing in active networks. In *OpenArch*, 2000.

[5] Uwe Engelmann, Andre Schroter, Markus Schwab, Urs Eisenmann, and Hans-Peter Meinzer. Openness and flexibility : from teleradio to pacs. In *CARS'99*,

pages 534–538. Elsevier, 1999.

[6] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[7] Graham Hamilton. Javabeans(tm) specification 1.01. Technical report, SUN Microsystems Inc, 1997.

[8] Ferber Jacques. Les systmes multi-agents : un aperu gnral. *Technique et science informatiques*, 16:979–1012, 1997.

[9] OMG. Corba component version 3. Technical report, Object Management Group, 2002.

[10] Frdric Peschanski, Thomas Meurisse, and Jean-Pierre Briot. Les composants logiciels : Evolution technologique ou nouveau paradigme ? In *Objets, Composants, Modles (OCM'2000)*, pages 53–65, Nantes, France, 2000.

[11] Jack P.C. Verhoosel, Harold J. Batteram, and Rudynell S. Millian. The friend platform: conquering complexity using distributed software components. In *Software Symposium 2000*, 2000.