An Intelligent Interface for the Dynamic Negotiation of QoS in ARCADE

Zeina JRAD
LIPN Laboratory
University of Paris XIII
Avenue Jean-Baptist Clement
Villetaneuse 93430, France
zj@lipn.univ-paris13.fr

Francine KRIEF
LIPN Laboratory
University of Paris XIII
Avenue Jean-Baptist Clement
Villetaneuse 93430, France
krief@lipn.univ-paris13.fr

Abstract

Since several years, the network area is in continual progress and one of the important functions that should be introduced in Internet Next Generation consists of the policy-based servers. These servers of control will realize an equivalence between the demands of the clients and the network resources. The objective of the project Arcade is to define an intelligent interface capable of accomplishing the best choice possible on the allocation of the policies that concern the QoS, the mobility and the security. In this paper, we develop a part of Arcade called NIA (Negotiation Intelligent Agent) that plays the role of intermediary between the user and the rest of the system and is designated to help him access to the Internet New Generation. We use agent technology to dynamically negotiate the SLS on the user's behalf, follow the user's behavior to be able to anticipate the negotiations and manage the re-negotiations.

1. Introduction

The purpose of the RNRT[1] project ARCADE [9] (Architecture of Adaptive Control of the IP Environments) is to set a general model for the control of the IP networks. This control is based on the determination of a profile for each user in order to reserve to him adapted resources. These resources can be dynamically allocated and should be controlled according to specific policies. The control should be applied on the security, the mobility and the quality of service.

Most applications cannot dynamically express their QoS requirements to obtain the adapted level of service. For each application, the customer and the provider have to agree on rules of assignment of service levels. They sign a contract called SLA [1] (Service Level Agreement) which is then translated into high-level policies. These policies are not directly executable by the network devices. They must be translated into intermediate and then into low level policies which are understandable by network devices.

COPS-SLS (Common Open Policy Service - Service Level Specification) [2] is a protocol that allows dynamic negotiation of the technical parameters of SLA. This negotiation process is complex because the user has to provide the technical parameters he needs. In this context, we propose to help the user by introducing an intelligent user interface. This interface contributes to the process of dynamic negotiation of the technical parameters of the SLA. This is done by detecting the needed values of these parameters according to the work of the user.

The paper is organized as follows. Section 2 presents the concept of a dynamic negotiation, Section 3 describes the architecture of Arcade. Section 4 presents the reasons behind the introduction of the agent technology in Arcade. Section 5 enumerates the different components of the intelligent user interface, section 6 resumes what was done in the implementation and we conclude in section 7 with our future works.

2. The dynamic negotiation of SLS

[1] French Research Project in Telecommunication

0-7803-8577-2/04/$20.00 ©2004 IEEE

In order to have guaranties of QoS, the user has to sign a Service Level Agreement (SLA) with the chosen service provider. There is no SLA standardization but there are some significant European projects working on the subject, for example TEQUILA [3], EURESCOM P906 QUASIMODO [4] and SEQUIN [5]. The technical parameters of the SLA can be dynamically negotiated using a protocol proposed by the IETF called COPS-SLS [6]. The SLS (Service Level Specifications) is constituted of pairs of parameters and values that define the service provided to a data flow. It contains especially the applicability of the SLS, the flow identifier of a SLS, the traffic conformity and performance guarantees. Concerning the performance guarantees, they can be expressed in a quantitative or qualitative way (e.g. low delay and medium loss). In Tequila [7] (used by COPS-SLS), the identified parameters are delay, packet loss, jitter and throughput.

In the COPS-SLS model, the SLS-PDP represents the provider and the SLS-PEP represents the client. The client here is just a logical entity who requests, possibly on behalf of other entities, network resources. It may be an end-host, a gateway of a local network, or another ISP. We have not only policies to manage network devices in a domain, but also policies to manage each type of client in the domain. COPS-SLS comprises two phases which are the configuration and the negotiation [8]. The configuration phase follows the COPS provisioning mode. The domain negotiation policies are provisioned down to the customer. These policies represent the services offered by a provider, this can be the negotiable parameters, the value constraints of these parameters or the dynamic level of the negotiation. The configuration phase is also useful for the provisioning of pre-defined service templates that can be requested by the customer in case there is a correspondence with his own needs. After successfully installed the configuration supplied by the PDP, the negotiation phase can begin. This phase follows the outsourcing mode. The PEP sends a request for its desired level of service. The PDP can accept or reject the request or propose another level of service to the client. The PEP installs policies according to the decision and sends a report. If both decision and report are positive, the contract is established and the user gets the quality that corresponds to the negotiated level of service. At any time, the network can send an unsolicited decision to change the parameters of the negotiation phase if, for example, he is not able to maintain the requested level of service. The negotiation phase will be guided by the updated configuration.

3. The Architecture of Arcade

The architecture conceived in this project concerns the policy-based servers and the dynamic negotiation of the SLS (using an extension of the protocol COPS) between the policy-based server and the nodes of the IP network. The control on the choice of the policies uses a billing function that allows the invoicing of the services supplied by the network.

Figure 1 represents the architecture of the project. The main components of the architecture are the Arcade middleware and the Arcade managed components.

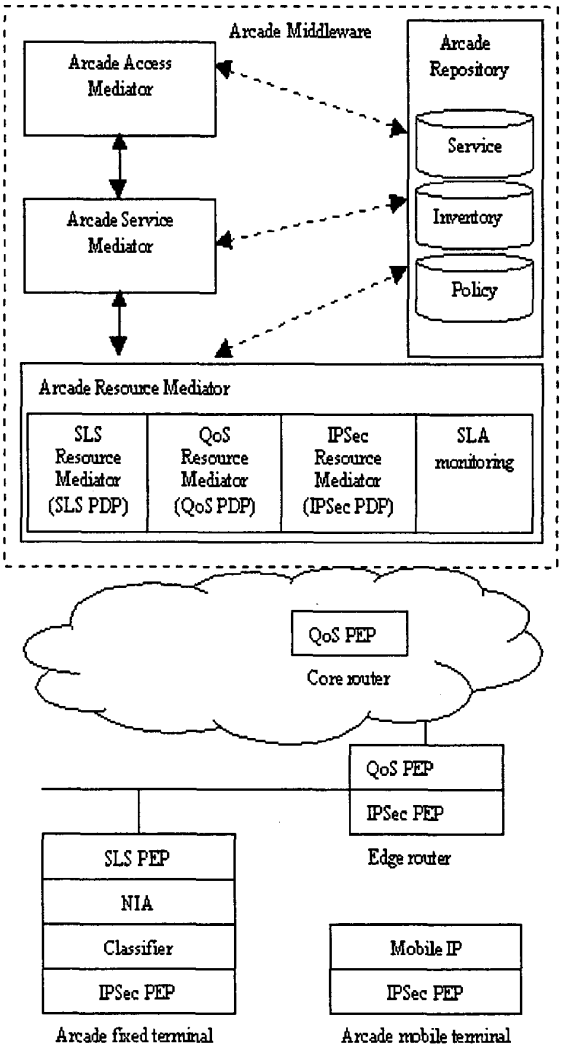

Figure 1. The Arcade architecture

3.1. The Arcade middleware

It provides the provisioning service of SLA for the final users of Arcade. It is composed of the Arcade Access Mediator, the Arcade Service Mediator, the Arcade Resource Mediators and the Arcade repository.

3.1.1. Arcade Access Mediator. It establishes a direct contact with the end-user (for SLA determination, subscription). It presents to the service provider (SP) a simplified view of the services offered by the network operator (NO) and translates the requests of service to the Service Mediator in an appropriate form.

3.1.2. Arcade Service Mediator. It supervises the management of the physical access to the services via the underlying network, using the Resource Mediator(s). The Service Mediator do not establish a direct contact with the end-user (for SLA determination, subscription). It establishes management rules of the QoS and the security in order to satisfy the requests made by the Network Operator through a graphical interface generated by the Access Mediator. Then, these rules are communicated to the Policy Manager who is responsible of the configuration of the PEP. The repository contains data that represents the information base of the network.

3.1.3. Arcade Resource Mediator. It is responsible for the network performance asked by service providers. It is composed of four components called Policy Decision Points (PDP) in the architecture defined by the Policy Framework WG of the IETF:

- *QoS PDP*: This is used to configure the QoS in the edge routers, terminals and the other routers. The QoS manager transmits to the concerned equipment the management rules of the QoS built up by the management system of Arcade. The rules are based on the parameters of service available in the database Service and then saved in the Policy Repository (the base of rules of QoS).
- *SLS PDP*: It allows the negotiation of the required level of service to the different terminals using a classifier and connected to the network (SLS PEP). Based on a set of rules of acceptance/rejection of the connection, the SLS policy Manager allows to accept, to reject or to propose an alternative SLS to a given client.
- *IPSec PDP*: The security manager is used to configure secure links between two entities of the network (terminals or edge routers). As in the QoS PDP, it transmits to the equipment the rules elaborated by the management system of Arcade.
- *SLA monitoring*: This is responsible of the configuration of the measurement tools. These tools assure the Service Level Monitoring. It supervises the collection of the statistics generated by the measurement tools and the production of the ad-hoc reports and it is controlled by the Arcade service mediator.

3.1.4. Arcade repository. It maintains an information base of the managed network elements along with their state and their capabilities. The constitutive elements of Arcade repository are the Policy repository, the Service repository and the Inventory repository.

- *Policy repository*: It allows the stock of management rules (profiles, activation planning, deployed rules, etc.). The database should implement the model IETF PCIM [10].
- *Service Repository*: It stocks the profiles and the definitions of the offered services. This service repository will save also the description of the SLS along with the information used by the Billing module.
- *Inventory repository*: This is a database whose model propose a technical view of the managed network. This database is accessible from the management system of Arcade and should implement an information model based on CIM.

3.2. The managed components in Arcade

The different components managed in Arcade are the network elements, the fixed terminals and the mobile terminals.

3.2.1. Arcade network elements. They are composed of Edge and Core Routers. These elements are called the Policy Enforcement Points (PEP) in the Architecture defined by the Policy Framework WG of the IETF [11]. The resources of the PEP are managed according to the SLA.

3.2.2. Arcade fixed terminals. They are composed of an IPSec PEP, a QoS PEP, a classifier and a NIA.

- *IPSec PEP:* It enforces the security policy rules.
- *SLS PEP:* It enforces the QoS policy rules negotiated with the SLS PDP.
- *Classifier*: It allows the recognition and the classification of the packets for each application of the terminal [12]. Once classified, the packets are marked with the DSCP (Differentiated Service Code Point) value. This mark depends on what had been

negotiated between the SLS PDP and the SLS PEP.

- *Negotiation interface Agent (NIA)*: It permits to provide to the SLS PEP the elements of the SLS to negotiate and then to configure the classifier in function of the value of DSCP negotiated. It determines the values of the parameters of SLS according to the work of the user and his needs in terms of quality of service. It sends these parameters to the SLS PEP which negotiate with the SLS PDP a certain level of service. If the negotiation is successful, the NIA gets a DSCP value that it communicates to the classifier along with other information necessary for marking the IP packets as the type of the active application. The work of the NIA is a difficult task that consists on following step by step the behavior of the user and even to communicate with him sometimes in order to be able to estimate the needed quality and to associate this with a well determined values of SLS.

3.2.3. Arcade mobile terminals. They have an IPSec PEP.

4. The agent technology in Arcade

Policy-based management offers much more than an automation of the router configuration using a database, which can be done with other architectures. A server of rules interprets and combines the static information recorded in the bases with other circumstantial data to make its decision and to send its directives to the networks nodes. If the rules of policies change, the network behavior changes. So policy-based management is dynamic. This characteristic is very important. The operator must be able to adapt to the environment (evolution of the customers' needs, unexpected behavior of the network, deployment of new services in an environment with a high competition). Furthermore the operator works in a distributed environment. Different entities are involved in the decision-making (i.e. Policy Manager and PDP). So agents are well adapted to policy-based management because of their main characteristics namely the adaptability, the autonomy, the ability to communicate with the others in order to solve some common problems in a decentralized manner. They also have learning aptitudes [13].

An agent is a software or physical entity to which a certain mission is allotted and which is able to achieve this mission in an autonomous way and in co-operation with other agents [14]. Autonomy is an interesting propriety. It avoids excessive information exchanges between distributed entities. It allows the system to react faster, for example in case of SLA violation.

Due to the main characteristics of the agent

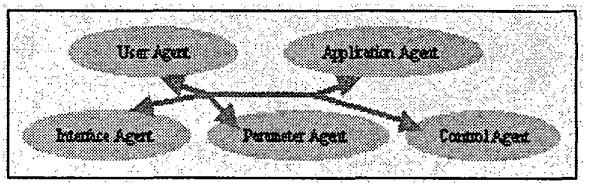

Figure 2 . The proposed MAS

technology described above, the introduction of intelligent agents in the policy-based networking environment can help the network to react faster if a problem occurs, e.g. in the case of a degradation of service. It can also simplify the management and allow a decentralized control of the QoS.

In Arcade, we propose the introduction of an intelligent interface in the terminal in order to facilitate the dynamic negotiation of SLS between a user and a service provider. The dynamic negotiation of SLS using COPS-SLS is a process which is complex. The user has to identify his needs in term of QoS and this task is difficult because QoS is an ambiguous term. Effectively, we can distinguish different QoS levels:

- User QoS: It is the QoS perceived by the user. Generally, it is a subjective concept which depends on various parameters such as service, application, bearable level of degradation, task (leisure/professional use), urgency degree.
- Network QoS: It is the QoS associated with the network devices.
- Server QoS: It is the local QoS associated with the server.

The end-to-end QoS is the QoS perceived by the user. It depends on both network and server QoS. From a user's point of view, the most important QoS is the user QoS which focuses on user-perceivable effects, rather than their causes within the network which concerns the service provider or the network operator.

In our interface, the automation of the SLS negotiation process is designated to simplify the relationship between the user and the service provider. The intelligent interface makes the (re)negotiation process transparent for the user and it accomplishes the following tasks:

- negotiation or re-negotiation of the level of service taken into account the user's and the applications needs;
- management of the degradation of service requested by the service provider.

- control of the QoS parameters.

5. The Intelligent Interface Model

The intelligent interface is a multi-agent system that contains six agents, each of which has a certain role to accomplish. Agents are well adapted to policy-based management because of their main characteristics namely the adaptability, the autonomy, the ability to communicate with the others in order to solve some common problems in a decentralized manner. They also have learning aptitudes.

Figure 2 describes the proposed MAS (Multi-Agent System). The communication, the coordination and the negotiation among the agents lead the system to a more efficient output. They can exchange information or demand services to each other. Each agent has a base of knowledge in which it can save information and from which it can retrieve data that helps it taking its decisions. An agent has access only to its own base of knowledge but it can contact another agent to ask it for local data that it possesses. The agents of our system are described as follows:

5.1. User Agent

The agent identifies the user to see if he is new or old, i.e. known or unknown to the system. It then analyzes his work along with the applications he uses in order to see if he belongs to one of the user's profiles. These profiles correspond to users who were previously working on this terminal, knowing that users who have similar requirements in terms of quality of service have the same profile. This agent includes three modules:

5.1.1. User's identification (UA1). After getting the user's login and password, the module identifies the user according to the information in his base of knowledge. If the user is known to the system, it may anticipate for him by negotiating directly the QoS based on his old profile. On the other side, if he is a new one, the module collects the maximum of data concerning his work and the applications he uses and attributes to him a new profile.

5.1.2. Data modification and memorization (UA2). New user's behavior is memorized and old user profiles is modified according to the new requirements of the user. Thus, data in the base of knowledge of the agent needs to be updated continuously.

5.1.3. Behavior analysis (UA3). The behavior of the user is what leads the module to attribute to him a certain profile. That's why the module needs to analyze the followed steps and the used applications.

5.2. Application Agent

Applications can be distributed into many categories according the type of the supported information (data, voice, image...) or the need for communication (delay, jitter...). The QoS requirements for an video application are not the same as those for an audio application [15]. The profile of the application will then be determined according to these categories and to the requirements of the user. In other words, not only applications of different types, but also similar applications used by different users, require different quality of service. The determination of the profile can be accomplished in two steps:

5.2.1. Classification module (AA1). It determines the type of the active application then associates it with one of the class of application defined in the base of knowledge of the agent. Applications are classified according to their requirements in terms of QoS.

5.2.2. Profile determination (AA2). It collects the information necessary to determine the profile of the active application. It may contact the user by the intermediary of the Interface Agent in order to ask him for his demands or the importance of his work. It may also contact the User Agent looking for saved information or profile that helps predicting the required quality of service. Thus, the final profile will be based on the requirements of both of the user and the application.

5.3. Interface Agent

It plays the role of intermediary between the user and the system. It defines the graphical interfaces needed for the communication with the user. It may send him messages or questions and tries to help him choose the answers that best match with his profile or needs.

5.3.1. Message translation (IA1). It determines which of the messages should be transmitted to the user according to the request of the other agents. It is also responsible for the conversion of the requests into a comprehensible language. Messages should be in the easiest format in order not to complicate the contribution of the user to the task accomplished by the agents.

5.3.2. Display module (IA2). It is responsible for the determination of the graphical interfaces or windows and the display of the messages to the screen.

5.4. Parameters Agent

It determines a general profile for the negotiation of the QoS, which will be based on the application and the user's profile which represents the demands of the user and the requirements of the application at the same time. It also determines the values attributed to all of the negotiation parameters depending on the constraints of the system and the service providers.

5.4.1. SLS parameters (PA1). It searches for the pertinent parameters necessary for describing a service according to the needs of the user and the requirements of the application. In this work, we are using the parameters defined in Tequila-SLS [7].

5.4.2. Predefined values (PA2). It searches for the allowed predefined values fixed in the contract of negotiation between the two parts, by contacting the corresponding SLS PDP and sending a request of configuration of the user terminal.

5.4.3. Values determination (PA3). After the determination of the convenient parameters and the accepted values, PA3 should be able to learn how to associate the best values of those parameters with the general profile of negotiation determined by the system.

5.5. Control Agent

It plays the role of a link between the terminal and the service provider mainly for requests

of Internet connection and services negotiation. This task is represented in the following modules:

5.5.1. Communication module (CA1). When an agent needs to communicate with the service provider, it sends a request to the Control Agent. The CA1 module will establish the link between the two sides.

5.5.2. Evaluation module (CA2). It is needed to assure that both of the two parts respect the negotiated services. The satisfaction of the user is deduced from the analysis of his behavior. Actions as the ending of an active application or a sudden demand of disconnection can be feedback into the module to indicate the dissatisfaction of the user.

5.5.3. Negotiation module (CA3). It sends the deduced values of the SLS parameters to the SLS PEP and asks him to begin the process of negotiation with the SLS PDP of the service provider. Both of the user and provider has the right to ask for a change in the services. This corresponds to a request of re-negotiation between the two parts.

6. Interaction Model and Verification of the system

In this paper, we represent the interactions of our proposed system using an AUML (Agent-based Unified Modeling Language) model [16]. There are messages sent to and from agents in order to look for certain information or answer other's request. In figure 3, when the user activates an application requiring an Internet connection, the User Agent creates a new instance of the Application Agent and asks for the classification of that instance (AA1). This corresponds to query (q1). Meanwhile, UA proceeds to the identification of the user (UA1). AA replies (r1) to the first query and asks for the identity of the user (q2). Two possibilities are represented in the model:

1. The user is unknown, which means that he works for the first time on this terminal. In this case the system does not have any information concerning his old work. After getting the reply (r2) from UA, AA begins to search for the profile of the application (AA2). It sends a query to the Interface Agent asking for a communication with the user to get more information about his needs and requirements (q3). IA translates the request into a comprehensible message (IA1), sends it to the user, waits for his answer (IA2), and then translates the choices of the user into a reply message that he sends to AA (r3). AA sends the profile of the application in an inform message to the Parameter agents (i1). PA1 starts determining the SLS parameters just after getting (i1), then PA contacts CA asking for the configuration of the terminal (q4) in order to determine the SLS predefined values (PA2). PA fills into the SLS parameters (PA3) and sends them to the Control Agent (i2) in order to start the process of negotiation with the SLS PDP (CA3).

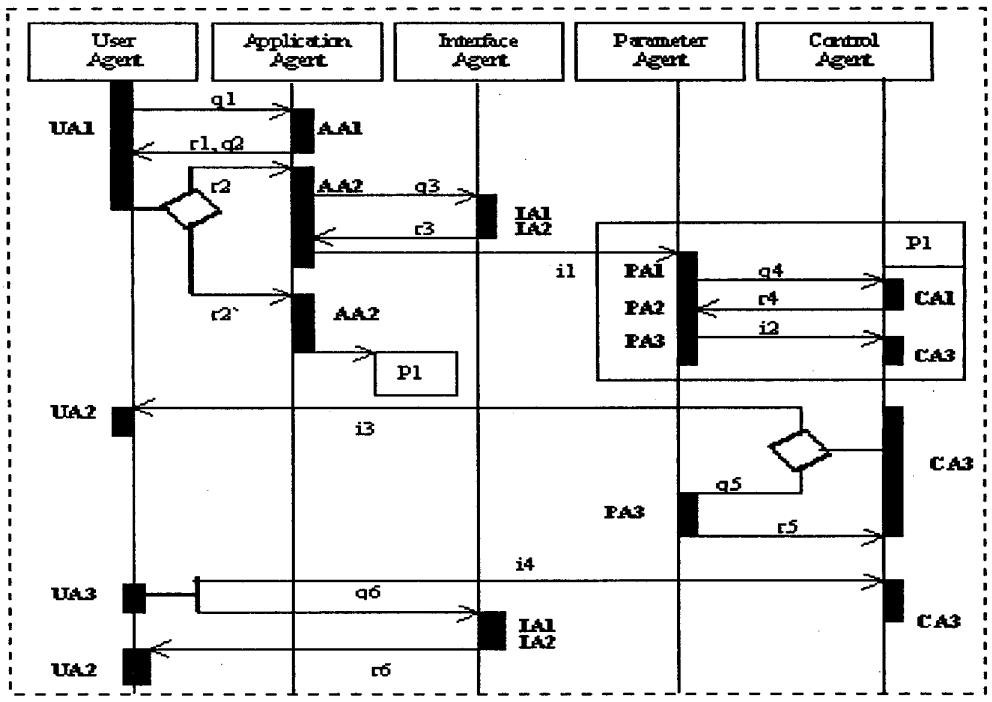

Figure 3 . AUML Representation of the System

2. The user is old, has a well-known profile and has asked for an automatic procedure once he was connected before. Thus, the system knows the type of applications he used to activate and therefore his requirements in terms of quality of service. In this case, the Application Agent doesn't need to communicate with the user since he knows previously his choices. P1 indicates the nested sequence of actions that is repeated many times according to the interactions of the system.

In both of the two cases, the Control Agent can systematically send messages to the User agent to transmit to him information (i3) relative to the current user. UA will then save all possible data concerning the user, his profile and the services negotiated for him (UA2). This is in case of a successful negotiation between the SLS PEP and the SLS PDP. Otherwise, CA can transmit to PA the demand (q5) of the provider to re-negotiate certain parameters or because the evaluation of the actual QoS shows the need for a re-negotiation.

PA looks for the values of SLS that can best describes the new services and sends them to CA (r5). On the other hand, UA may detect the dissatisfaction of the user while analyzing his behavior (UA3). It sends then an inform message to the Control Agent (i4) and contacts the user through the Interface Agent in order to better understand his behavior (q6, r6).

7. The implementation of the NIA

The implementation of the NIA was performed using a multi-agent platform called oRis [17]. The oRis environment was conceived and developed in the laboratory of LIL-ENIB (France). It contains a programming agent oriented language and a simulation motor. This is a multi-agent environment where each agent has an autonomous behavior that allows it to communicate and to interact with the other agents.

7.1. The characteristics of the platform

oRis is a language that takes the procedural tools of C and the object paradigm of Java to which was added some dynamic properties and agent oriented functionalities. This is what adds to the language a generic aspect, unlike many other agent oriented languages that are specific to a given application.

In oRis, the agent is an object that has a particular method, the *main()*, that represents its behavior. This method is executed at each step of the simulation without being called. It is called automatically at each creation of an agent unlike other methods where we need constructors and destructors.

Figure 4 shows how a declaration of an agent is done. Each method that has a main() is considered as an agent. That how we differentiate between an agent and an object. In this method, we should detail the behavior of the agent. We notice that in oRis, the constructor is represented by the method *new()* and the

destructor by the method *delete()*, and that the principal code is in the 'execute' code block.

```
Class Agent                    // Declaration of the class 'Agent'
{
int age;                       // 'Agent' contains an integer 'age'
void main(void)                // 'Agent' has a method 'main()' → this is an agent
};
void Agent::main(void)         // Definition of the method 'main()'of the class 'Agent'
{
printf("The age of ", this," is ", ++age);
}
execute                        // Code to be executed (initialisation)
{
printf("----Begin----");
for (int I=0;I<3;I++)          // To instantiate five 'Agent'
new Agent;
printf("----End----");
}
```

```
----Begin----
----End----
The age of Agent.2 is 1
The age of Agent.3 is 1
The age of Agent.1 is 1
The age of Agent.1 is 1
```

Figure 4 . An agent oriented program

In oRis, writing an agent oriented application means the instantiation of agents. The 'execute' code block is simply used to initialize the application by creating instances; then, the instances themselves are what make the development of the application and not the principal program. Thus, the end of the 'execute' code block does not represent the end of the application and this is the main difference compared to the principal function of the languages C/C++.

As shown in Figure 4, the 'execute' code block contains only instantiations and the execution of the application takes place at the exit of this block and runs continuously. We note also in the execution of the application showed above that the agents are designed randomly. Thus, the instance 'Agent.2' is displayed before 'Agent.1'.

7.2. The communication in oRis

In oRis, the communication among the agents can be done in many ways:

7.2.1. Synchronous communication. This is what we use implicitly when we develop an application object oriented. In the synchronous communication, the flow of execution that invokes a method continues itself the execution of the code of that method and then returns back to the treatment of its own code when the invoked method is terminated.

This corresponds to a direct call of the methods. The invocation of a method by an object does not create a new flow of execution at that object but simply directs the activity of the calling object towards a treatment that manipulates the data of the designed object.

7.2.2. The reflex links. Among the available communication methods, oRis provides a solution to survey the data of an instance in order to be informed when they are modified. A reflex link is an object assigned to an attribute and whose methods are activated automatically each time the concerned attribute is modified. It is possible to associate a certain number of reflex links with the same attribute of the same instance.

7.2.3. Point to point communication. It concerns a communication based on an explicit sending of messages from point to point in an asynchronous treatment. This type of communication is based on the class 'Message which is automatically defined in oRis. It represents the class of base of all the messages that the entities can exchange. The unique information that this object contains by default is the sender of the message. In order to add other information in the message to be expedited, it is necessary to create a derived class from the class 'Message' and then add the fields needed to the communication.

To accomplish the communication point-to-point, the sender starts by instantiating an object of the type 'Message' or any other derived type. It provides the different fields of the message and terminates by invoking the method *sendTo()* of the message by passing as an argument a reference to the destination (it's the message that knows how to reach the receiver). This latter invocation places a message in the queue (First In First Out) of the internal messages of the receiver.

In order that the receiver be able to treat the received message, it should consult its proper queue of messages: this is the role of the methods *getNbMessages(), getNextMessage()* among others.

7.2.4. Diffusion. The communication point-to-point imposes on the sender to precise the destination of the message. However, if the instance does no know which entity is capable of executing the needed service, the diffusion of messages seems to be the best solution.

The messages of this method are similar to those in the previous one except in the way of sending and receiving them. Instead of using the method *sendTo()* of the message in order to send it to a particular

destination, the sender invokes the method *broadcast()* of the same message.

7.3. The agents of the NIA

The intelligent interface developed in the project Arcade is called the NIA (Negotiation Intelligent Agent) and is designated to negotiate dynamically the QoS parameters.

The implementation of this MAS in oRis corresponds to creating three files:

-MyAgents.pkg which contains the declaration of five classes where each one corresponds to an agent along with the needed methods and attributes.

-MyAgents.imp which is the implementation of the methods of each agent along with their constructors and the messages to be exchanged among them.

-MyAgents.ors for the instantiation of the agents.

The communication among the agents of the NIA is based on either the point-to-point or the diffusion methods. There are many types of messages depending on the requests made by the agents and the demanded information. The massages contain data and/or names of functions to be executed depending on whether the message is designated to inform another agent or to demand a service. The method *main()* belongs to each agent and is activated at the instantiation of an agent until the end of the instance. This is how the agent receives the messages send by the other agents and identifies their contents.

Concerning the user's profile determination, for an unknown user, the NIA communicates with him searching for his needs and preferences. In the corresponding class, we have the method *isUsrKnown()* that represents the work of the first module of this agent in determining whether the user is know by the system or is connecting for the first time.

In the user agent as well as in other agents that has a base of knowledge, we have the method *toSave()* that memorizes the user behaviors and profiles and updates them continuously according to the new requirements.

The method *usrProfile()* is the one that attributes a profiles for each user according to the applications he's using and his needs in terms of quality. For an old user, this profile can be a direct and automatic negotiation of the class of service without going through the communication phase. This procedure is periodic in order to give the user the possibility to change his mind.

Concerning the Interface Agent that plays the role of intermediary between the user and the system, it may use graphical interfaces implemented in Java in order to communicate with the user. Figure 5 shows the one that it uses to ask the user about his preferences in terms of quality in case where the NIA is unable to get itself this information.

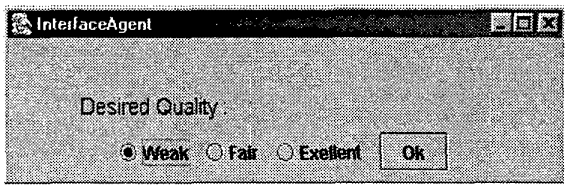

Figure 5 . An Interface for the communication with the user

After getting the necessary information concerning the choice of the user from the Interface Agent or the profile of this latter from the User Agent, the Application Agent uses the method *appProfile()* to determine the profile of the active application. The reasoning of the agent is based on rules that associate the couple of user profile and application type with an application profile. These rules is what constitute the base of knowledge of this agent. And in the same way, the Parameter Agent determines the values of the SLS parameters that corresponds to the application profile using the method *setParam()*.

The last task accomplished by the system consists on communicating with the other modules of the system in order to achieve the main purpose which is the dynamic negotiation of the SLS.

As described above, the first mini-task of the Control Agent consists on sending the SLS parameters to the SLS PEP asking for a DSCP value. This is done by the method *toPEP()* which gets the parameters from the Parameter Agent and transfers them to the SLS PEP by invoking the method *read()* of this latter. This method is implemented as a part of the PEP and is used to calculate the DSCP value using the parameters of SLS. Another method of the PEP called *whatValue()* is invoked by the Control Agent in order to get the DSCP value.

The method toClassif() of the control agent determines what are the information that the classifier needs in order to classify the application on the terminal, then it invokes the method *addDiffMark()* of the classifier to mark the packets of the flow according to the negotiated DSCP value.

8. Conclusion and future work

The intelligent interface proposed in this paper is based on the agent technology and the policy-based networking management in order to make the Internet of today more successful. The agent technology is used

in order to help the user to dynamically negotiate the SLS on the user's behalf, follow the user's behavior to be able to anticipate the negotiations and manage the re-negotiations that mainly use COPS-SLS protocol.

This approach was implemented using a software platform called oRis in the context of the Arcade project. For the portability we are using an oRis/Java interface.

Future works concern the introduction of mechanisms to help the user to choose the best service provider and therefore to add security levels beyond the selection criteria. Another objective will be the introduction of the mobility of the user and the terminal.

9. References

[1] Tjaja Hokmoro J., "SLA Enforced by Policy", PhD thesis, University of Twente, June 2001.

[2] Boyle J., R. Cohen, D. Durham, S. Herzog, R. Raja, A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.

[3] TEQUILA IST Project (IST-1999-11253).

[4] Eurescom P906 QUASIMODO Project.

[5] SEQUIN IST project (IST-1999-20841).

[6] Nguyen T.M.T., N. Boukhatem, Y. El Mghazli, N. Charton, Louis-Louis Hamer, G. Pujolle, "COPS-PR usage for SLS negotiation (COPS-SLS)", draft nguyen-rap-cops-sls-03.txt, Internet Draft, July 2002.

[7] Goderis D., Y. T'joens, C. Jacquenet, G. Memenios, G. Pavlou, R. Egan, D. Griffin, P. Georgatsos, L. Georgiadis, P. Van Heuven, "Service Level Specification Semantics, Parameters and negotiation requirements", internet draft, June 2001.

[8] Chan K., J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith, " COPS Usage for Policy Provisioning", RFC 3084, March 2001.

[9] RNRT ARCADE Project, URL: http://www-rp.lip6.fr/arcade/

[10] "Policy Core Information Model", RFC3060, February 2001, http://www.ietf.org/rfc/rfc3060.txt

[11] R. Yavatkar, D. Pendarakis and R. Guerin, "A framework for policy Based Admission Control", RFC 2753, January 2000

[12] The product of the society QoSMOS is called Traffic Designer 1^(TM). Their page web is www.qosmos.net.

[13] Boukhatem N., B. Campedel, H. Chaouchi, V. Guyot, F. Krief, T.M.T. Nguyen, G. Pujolle, "A New Intelligent Generation for Internet Networks", SmartNet 2002.

[14] Briot J.-P., Y. Demazeau, "principles and architecture of MAS", 2002.

[15] France Telecom. Metrology proposal for Multimedia QoS ITU-Telecommunication Standardization Sector, Document AVD-2152, discussion and proposal. Santa Barbara, 24-28 September 2001.

[16] http://www.auml.org/

[17] http://www.enib.fr/~harrouet/oris.html