A Dual Simulation Environment for Simulating MAS in Telecommunication Networks

NADA MESKAOUI
Lebanese University
Hadath – Lebanon
Lebanon
nadames@inco.com.lb

DOMINIQUE GAITI
University of Troyes
12 Rue Marie Curies
Troyes, Cedex
dominique.Gaiti@utt.fr

KARIM Y. KABALAN
American University of Beirut
Bliss Street
Beirut - Lebanon
kabalan@aub.edu.lb

Abstract— this paper presents a dual simulation environment for simulating different type of telecommunication networks integrating intelligent agents. Agents are considered as advanced tools for resolving complex issues in networking based on intelligent and dynamic features. To test the efficiency of these proposals, new simulation environments, integrating both agents and network components, are required. In this paper we propose an extension with intelligent capabilities to a networking platform. This dual simulation environment has been tested for implementing agents in a DiffServ network to improve its performance. Simulation results show the efficiency of integrating agents within telecommunication networks and also prove that such a dual simulation environment is needed to test new techniques based on agents and multi-agent systems in networking.

Key-Words: Artificial intelligence, telecommunication, networks, single-agent systems, multi-agent systems, Diffserv network, platforms.

1 Introduction

New approaches, in any domain, have to be validated by simulations before its real implementation. For this reason, different types of platforms were conceived in various domains to help researchers testing the efficiency of their proposals.

Some Existing platforms are dedicated to simulate telecommunication networks and others for multi-agent systems. Networking platforms do not have intelligent capabilities. Also, platforms conceived for multi-agent systems' simulation stress on the definition of the intelligent capabilities of the agent without taking into consideration the telecommunication networks' elements. Network elements could be modeled and built within these platforms but this means that every networking component should be redefined, modeled and represented by agents [1]. Standard components and mechanisms that constitute a telecommunication network are not provided in this type of platforms.

The objective of our work is to propose an intelligent framework to model and simulate intelligent features in networking in order to prove the efficiency of intelligent approaches – based on agents and multi-agents systems – in the improvement of telecommunication networks' performance.

This paper presents a platform having the ability to simulate both networks and multi-agent systems. Having this platform, one can specify any type of agent's behavior to interact with any node structure. The proposed platform is detailed in section 4 while sections 2 and 3 present, respectively, the existing platforms for networks and Multi-Agent systems simulations. Section 5 shows some simulation results and a conclusion is provided in section 6.

2 Platforms for network simulation

In telecommunication networks simulations, two platforms' categories are identified: open platforms for standard network simulations able to implement new techniques and network functionalities and platforms specialized in the simulation of a specific type of networks able to implement new techniques but in the frame of their objectives.

2.1 Network Simulator "NS-2"

The Network Simulator "NS-2" [2] is an open, extensible, event-driven simulation engine implemented in C++ and using the MIT's Object Tool Command Language [3] (OTcl - an object-oriented version of Tcl [4]), as a command and configuration interface.

A network topology is realized using three primitive building blocks: nodes, links, and agents. Agents are the objects that actively drive the simulation and could be thought of as the processes and/or transport entities that run on nodes (end hosts or routers). Traffic sources and sinks, dynamic routing modules and the various protocol modules are all examples of agents.

Ns implements modules for some standard protocols and techniques [5]. Different routing protocols are available such as unicast, multicast, and hierarchical routing.

The explicit congestion protocol XCP has been added to Ns-2.28. XCP is a feedback-based congestion control system that uses direct, explicit, router feedback to avoid congestion in the network. It is designed for both scalability and generality.

A Differentiated Services "DiffServ" module has been integrated into Ns-2.1b8. DiffServ, is an IP QoS architecture based on packet marking that allows packets to be prioritized according to user requirements. During the time of congestion, more low priority packets are discarded than high priority packets.

Ns provides a module for simulating mobile networking. This module covers the internals of a mobile node, routing mechanisms and network components that are used to construct the network stack for a mobile node.

Extensions that enable the simulation of satellite networks are also available in ns. In particular, these extensions enable Ns to model either traditional geostationary ``bent-pipe'' satellites with multiple users per uplink/downlink and asymmetric links, or geostationary satellites with processing payloads (either regenerative payloads or full packet switching), or polar orbiting LEO constellations such as Iridium and Teledesic. Other modules are also available, like MPLS, radio propagation, direct diffusion and others.

2.2 J-Sim

J-Sim [6] is an open, component-based, compositional simulation environment, proposed in the frame of research cooperation between the Ohio State University and the University of Illinois at Urbana-Champaign, and built entirely in Java upon the notion of the autonomous component architecture.

The behaviors of the J-Sim components are defined in terms of contracts and can be individually designed, implemented, tested, and incrementally deployed in a software system. A system can be composed of individual components in much the same way a hardware module is composed of IC chips. Moreover, components can be plugged into a software system, even during execution. This makes J-Sim a truly platform-neutral, extensible, and reusable environment.

J-Sim also provides a script interface to allow integration with different script languages such as Perl, Tcl or Python. A fully integrated J-Sim with a Java implementation of the Tcl interpreter (with the Tcl/Java extension) already exists and is called Jacl. So, similar to NS-2, J-Sim is a dual-language simulation environment in which classes are written in Java (for NS-2, in C++) and "glued" together using Tcl/Java. However, all the public classes/methods/fields in Java can be accessed naturally in the Tcl environment without the need to export them to the Tcl environment as required in NS-2.

This component-based architecture makes it possible to compose different simulation scenarios – for different networks' architectures – from a set of basic components and classes proposed by J-Sim and/or defined by the user.

The generic node structures defined in J-Sim along with the different implemented techniques are detailed in the following sections. This is because our proposal is based on this platform.

2.2.1 A generic node structure

J-Sim proposes an implementation of an abstract network model "INET" [7] on the top of the component-based architecture. This model defines the generic structure of a node and several generic network components with their associated contracts.

The internal node structure, either an end host or a router, is defined in INET in two layers instead of four layers as in the TCP/IP model or seven layers as in the OSI model. These two layers are the Upper Protocol Layer "UPL" and the Core Service Layer "CSL" presented in Figure 1. The UPL contains transport, signaling, and application protocol modules. The CSL provides a set of well-defined services, which are common in most network architectures, to modules in the UPL. In some sense, the network layer, the link layer and the physical layer are encapsulated into the core service layer.

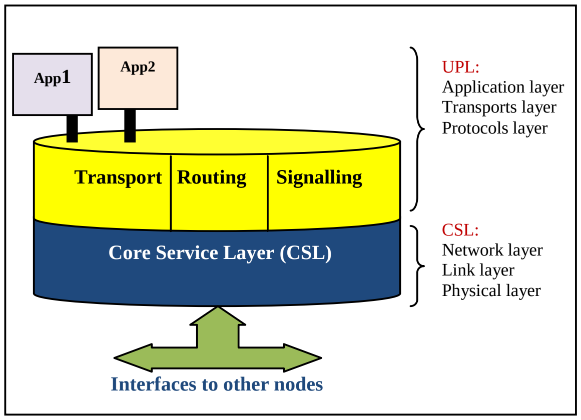

Figure 1. The J-SIM architecture

Users may then compose a network scenario in a plug-and-play fashion, by connecting components in their desired manner. Users may also subclass an appropriate component and redefine new attributes and methods to incorporate their own protocols and algorithms.

2.2.2 Implemented techniques

Different techniques are implemented within J-Sim like IntServ, DiffServ, MPLS and Active Networks.

The IntServ technique is specified within J-Sim in 18 Java classes [6]. This implementation defines quality of services requirements in terms of end-to-end delay, inter-destination jitter, inter-packet jitter, and minimum bandwidth packet loss rate.

A DiffServ package [8] is built on the top of the INET network model. This package implements the various classes that may serve as the building blocks for DiffServ, and includes markers/taggers, service profile, policers/shapers and queue management algorithms.

The implementation of MPLS [9] within J-Sim defines two basic components: forwarding table component and MPLS component. The forwarding table keeps information about the configured labels within the MPLS network by associating an IP prefix or an incoming label with an outgoing interface, and the MPLS component routes packets according to the configuration of the forwarding table.

The Active Network [10] was implemented in a package that holds classes specific to the active network functionalities. In addition, minor modifications to some existing Java classes were also introduced.

Other platforms

NS-2 and J-Sim are platforms conceived for network simulations, not related to any type of networks. In the same category, there is OPNET [11], which is a commercial network simulator. Other types of platforms for simulating a specific network environment, such as INSANE [12] and ANTS [13] that respectively simulates ATM [14] and active networks [15] were also proposed.

3 Platforms for MAS simulation

Agents and MAS oriented programming is an advanced software-modeling paradigm proposed by researches in the distributed artificial intelligence domain. It addresses the need for software systems to exhibit rational human-like behavior in different domains.

Traditional software systems make it difficult to model rational behaviors and often programs written in these systems experience limitations, especially when attempting to operate in real-time environments.

MAS platforms are characterized by their architectures able to build systems that work on the full range of tasks expected from an intelligent agent, from highly routine to extremely difficult, open-ended problems. They have the ability to represent and use appropriate forms of knowledge and a full range of problem solving methods. They interact with the outside world and learn about all the aspects of the tasks executed by the agents and their performance on them. These types of platforms are goal-oriented, based on states and operators. They continually try to select and apply operators to states until the achievement of a specific goal.

Different platforms were proposed for multi-agent systems simulation like JAFMAS, DECAF, JACK, DEMAS and others, each having its proper implementation and fields of application.

3.1 JAFMAS

The Java-based Agent Framework for Multi-agent Systems "JAFMAS" [16] is a multi-agent system platform written in Java for developing cooperation knowledge and protocols between different agents. It provides sixteen main Java classes and the user, who desires to develop his application using JAFMAS, has to extend only four of these classes.

This platform provides the essential communication, interaction and coordination mechanisms to application developers and offers the possibility to define different agents' behaviors and enables these agents to work together and coherently achieve their goals and those of the multi-agent system as a whole.

3.2 DECAF

The Distributed Environment Centered Agent Framework "DECAF" [17], [18] toolkit proposes a platform suitable for research activities in the multi-agent systems domain. It allows the design, development, and operation of intelligent agents to achieve solutions in complex software systems. It provides the necessary architectural services, for building intelligent cooperating agents: communication, planning, scheduling, execution monitoring, coordination, learning and self-diagnosis. Control and programming of DECAF agents is assisted by a graphical user interface called the plan-editor.

DECAF provides a platform for development of different agents' behaviors and interactions. DECAF is written in Java and makes extensive use of the Java threads capabilities to get improved performance as each agent runs concurrently within its own thread.

3.3 JACK

JACK Intelligent Agents [19] is an agent oriented development environment fully integrated within the Java programming language to provide agent-oriented extensions to Java. This extension defines new base classes, interfaces and methods, provides additions to the Java syntax to support new agent-oriented classes, definitions and statements and also provides semantic extension (runtime differences) to support the execution model required by an agent-oriented software system.

An agent is presented in JACK as an entity having a set of believes about the world, a set of events to respond to, a set of goals and a set of plans that describes how to handle the different goals and events.

3.4 DMAS

The Distributed Multi-Agent System Builder "DMAS Builder" [20] is a 100% Java development environment/tool that supports the development of multi-agent systems that are totally independent from the used operating system. It has two main components: the development environment and the package of classes supporting agent-oriented programming.

DMAS provides a set of classes to help building a MAS application. The "Agent" class is the highest class in the agent hierarchy and contains features and methods required for any type of agents. An agent has an identifier, a

"Mailbox" to receive and send messages, a list of tasks to achieve, and a specific behavior. The agent could also build and integrate a knowledge base thanks to a list of classes provided for this purpose.

DMAS offers a graphic specification and validation of all the provided components, a complete and automatic source code generation (in java) of all system components, and an exploration mechanism allowing system development and execution independent from DMAS builder. It also supports very good code extensibility and offers an abstraction of communication mechanisms and protocols between agents and various components.

4 The proposed platforms

To build a platform with both networking and MAS capabilities, we believe that it would be easier to extend one of the existing platforms in the networking domain with intelligent functionalities instead of extending a MAS platform with networking features. This is because our objective is to prove the efficiency of agents and MAS in resolving complex issues in networking, so the same agents' infrastructure, with just different agents' behaviors, will be integrated in different types of networks. As most networking platforms implement different types of networks, it is easier to use them instead of implementing the different networks' types in a MAS platform.

4.1 Choice of a Network simulator

Different simulation environments were proposed to simulate telecommunication networks as described in the previous sections. All these platforms, each within its objectives, have succeeded in providing users with approximation of the realistic functioning of their networks. To choose a platform for our proposal, we had to evaluate different aspects like the architecture of the software, the simulation framework and technique, the performance scalability, and the implementation of the standard Internet protocols. As a result the J-Sim/INET simulation environment was chosen.

J-Sim/INET is a dual-language environment that seems to be a simple and well-defined component-based software architecture. Java is used to create components and a script language is used as a glue or control language that integrates components in a desired fashion at run time and provides a high-level dynamic control. These facilitate fast prototyping of customized simulation scenarios, and runtime configuration and diagnosis. Otherwise, J-Sim provides synchronization methods to further extend programming flexibility, and have the ability to work with both discrete event simulation and real-time process-based simulation.

J-Sim implements different techniques like DiffServ, IntServ, MPLS and Active Networks and supports major Internet protocols like IP, TCP, UDP, OSPFv2, BGP-4, and the pseudo-network interface drivers NIC, sockets, global Internet topology construction and IP address assignment. This offers the user the flexibility of implementing an application for any type of standard networks without the need to define standard protocols and techniques.

Concerning NS-2, its architecture is not so-structured and the mixture of compiled and interpreted classes makes it difficult to understand and validate NS-2 codes. Otherwise, NS-2 provides substantial support for simulation of TCP, routing, and multicast protocols, but due to the special node structure in NS-2, it is non-trivial, and sometimes difficult, to include other protocols/algorithms or accommodate other network architectures in it [6]. In addition, NS-2 has just some implementations of the standard techniques.

Finally, the other platforms for Networks' simulations are dedicated for a specific network. It is so obvious that these types of specialized environments cannot be used in our proposal as an open platform able to support simulations for any network type and protocol is required.

4.2. Choice of a MAS simulator

Different simulation environments were defined for the simulation of multi-agent systems as described above. An evaluation of these platforms was proposed in [20]. This evaluation considers the following criteria: the tool's methodology and flexibility, the learning and communication facilities, the simplicity of the transition between development and implementation, the database and MAS management support, the automatic code generation and graphical support, the code extensibility, the simplicity of the implementation and deployment, and finally documentation.

Results showed that JACK offers great possibilities at the implementation level even if it requires a lot of effort, but it doesn't have any specification at the methodology level and suffers from a great gap in terms of graphic utilities. DECAF and JAFMAS are environments supporting different levels of development process and emphasize planning and interactions between agents but do not deal with implementation and deployment. Finally, DMAS is considered in [20] as the "idealistic" complete MAS development environment that puts emphasis on development facility and simplicity, provides useful implemented options to programmers and offers several agent types, each having different characteristics and behaviors.

For our proposed solution, we need to extend the J-Sim simulation environment with intelligent functionalities required for telecommunication networks applications. As none of the predefined platforms is dedicated for MAS simulation within telecommunication networks, we decided to build an extension to J-Sim inspired from all the existent MAS platforms. This gives us the flexibility to choose from each MAS environment the features that fit the most the telecommunication networks' requirements and also to add some required elements and features.

4.3 J-Sim Extension

The implementation of the proposed model is based on an extension to the J-Sim simulation environment. J-Sim provides many standard functionalities and mechanisms for the building of a network node structure. These mechanisms and services are provided as components and packages.

To add intelligent capabilities to the J-Sim node structure, we defined an agent package. The description of this package, its implementation and its interface with other existing packages and components within J-Sim are next presented along with the way to define agents' behaviors related to specific applications.

4.3.1 The agent package – Architecture

The Agent package is developed in JDK1.4. As shown in Fig. 2, the agent package is structured in different components that provide the essential interaction/coordination and data management mechanisms for application developers.

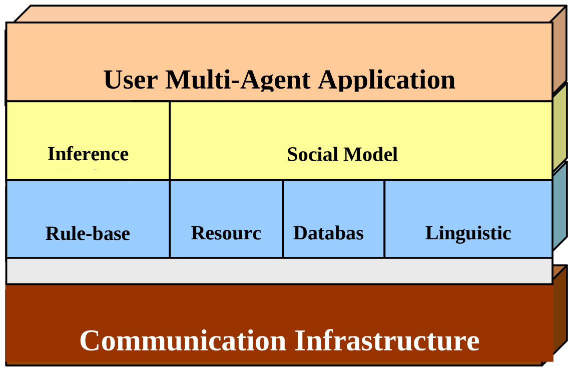

Figure 2. *The Agent Package Architecture*

The user, according to the required agents' type and behavior, defines the multi-agent system application. This application benefits from the services provided by the social model, which determines the way the agent communicates, cooperates, and interacts with other agents in the system in order to bring about a coherent solution. During its communication, the agent exchanges messages using a common agent-independent language defined by the Linguistic layer. In these messages, the agent exchanges resources and knowledge and then stores acquired data in the data model. To reason on this data, the agent implements a rule-base that contains the needed rules to decide about its future actions. These rules are invoked by the inference engine.

4.3.2 The Agent package - java classes

We propose an implementation of the agent package in 18 main java classes that provide the essential elements and entities for the developers of intelligent applications within a telecommunication network. Five of these classes, were originally taken from the JAFMAS platform with some modifications to adapt them to our proposed model. The classes of this package and the relationship between them are shown in Figure 3.

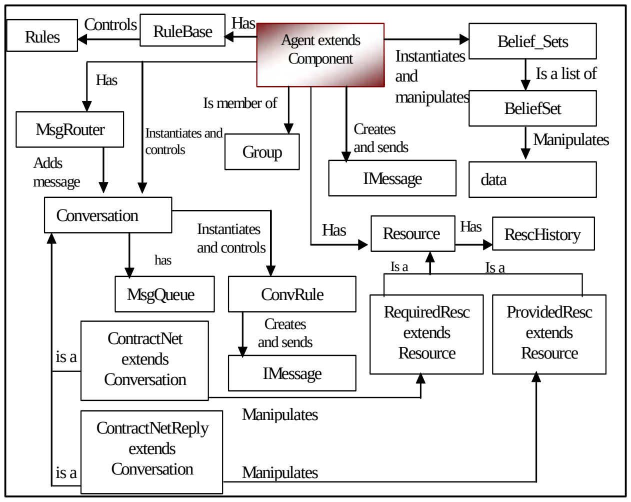

Figure 3. *Classes of the package and their*

The relationship between the different models and java classes is described in Figure 4.

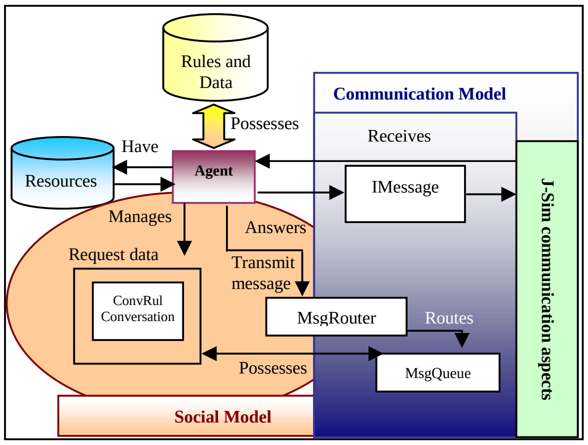

Figure 4. *Relationship between the different entities of the agent package*

The agent processes data and rules, has a list of resources and manages conversations with other peers. When it receives a message, it takes different actions according to the message intent: if the message is related to a conversation, the agent sends it to the message router for processing. The message router directs it to the related conversation for treatment. The message will be so stored in the message queue of the conversation. The received message may hold information that modifies the conversation state and may induce rules execution or require being stored in the agent's database. If the agent receives a message providing data, it updates its database. In case the agent receives information concerning a resource, it directs it to the concerned resource manager for processing.

4.3.3 Interfacing the agent package and other components

The agent package was integrated to J-Sim and implemented on the top of the core service layer (CSL) as shown in Figure 5. This gives the agent the possibility to benefit from the different services provided by the CSL especially those related to message transfer and information retrieval.

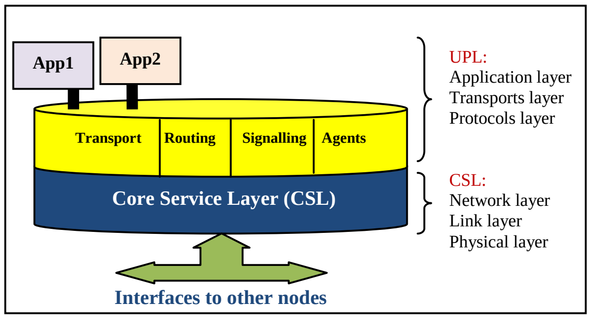

Figure 5. *The J-SIM architecture*

The core service layer provides a set of well-defined services to the modules implemented in the upper protocol layer. These services are:

- ***Data services*** give the upper protocol layer "UPL" the possibility to send and receive packets to and from the network, via the data services ports.

- ***Identity services*** allow the upper protocol layer "UPL" modules to query or configure the identities of the node.
- ***Routing table services*** allow UPL modules to query or configure the routing table of the node.
- ***Interface/Neighbor services***, allow UPL modules to query the interface/neighbor information related to the different interface(s) of the node.
- ***Multi-cast services*** offer the possibility to join and leave a group;
- ***Packet filter configuration services***, allow UPL modules to configure the packet filters in the core service layer;

Other interfaces could be defined between the agent package and applications or protocols implemented on the top of the core service layer. Specification of these types of interfaces is related to the intelligent application defined by the user as extension to the agent package. In order to establish an interface between the agent package and a specific component, the user has to define a port in both components and specify a contract to formalize the exchange of information.

4. How to implement a specific application

In order to implement a specific application, the user has to proceed in two steps: Specification and implementation.

4.1 Step1 – Specification

Specification means to identify the different types of agents, members of the multi-agent system, the groups, the production rules and resources.

According to the multi-agent system specificities, one or different agent types may be defined. In case all the agents have the same aims and provide common functionalities, only one agent type is required. For each defined type, the user has to specify the agent behavior, which defines the way the agent treats incoming information and interacts with other agents.

Within the multi-agent system, different comities and groups having specific properties could be defined. The identification of these different groups should be also performed within this phase. This is in addition to the specification of the resources required and provided by the agent to store data along with the different reasoning rules used by the agent to decide future actions.

4.2 Step2 – Implementation

The first task in the implementation is to define the extension to the "agent.java" class. This class should contain parameters and methods that reflect the agent type and determine its behavior.

The agent's knowledge/belief is represented in a tuple-based relational model. We have so to determine the name and type of each element of a specific relation. This requires an extension of the "BeliefSet.java" class for each required relation type. An extension of the data class is also required to define the different types of elements that constitute a relation.

In order to define the different resources required and provided by the agents, extension to both "ProvidedResc.java" and RequiredResc.java" classes should be implemented for each resource. Also, the production rules needed by the agent are specified in the extension to both "Rules.java" and "RuleBase.java" classes.

Finally, a list of java classes that extend both "Conversation.java" and "convRule.java" classes should be implemented to specify the interaction of the agent with other peers within the multi-agent system. As conversation is defined as an agent's plan to achieve a goal, based on interactions with other agents, we have to identify every possible conversation that each agent can engage in, and represent those conversations by developing a finite state machine model for each of them. These finite state models identify the different conversation classes in the application. The finite state machine model that represents conversations is rule-based descriptions of what an agent does in certain situations. These rules define each transition of the finite state machine. We have so to identify in the extension of "ConvRule.java" the various execution conditions and actions for each rule of the conversation.

5. Simulation results

We tested our dual-simulation environment by integrating agents in a DiffServ network to resolve the problem of service quality degradation of some specific DiffServ traffic types in case congestion occurs.

We consider two sources S1 and S2 that respectively send to two destinations D1 and D2 traffic with two different levels of quality of services, where the traffic sent by S1 demands a higher level of quality of services than the traffic sent by S2. The throughput of both sources S1 and S2 is configured as to force congested situation in one of the DiffServ network links.

Figure 6 shows great loss of S2 packets if congestion occurs and no agents are implementation within the DiffServ nodes. After the integration of agents within the DiffServ nodes better results were perceived. These agents have a behavior that tries to increase the S2 traffic throughput (minimize drops) and balance the traffic load within the network when congestions occur. These results are shown in Figure 7.

Figure 7 shows for both S1 and S2 traffics, the difference between the throughputs perceived by the sources and the destinations. In this simulation all the S1 and the S2 traffic are received by the destination, without drops of S2 packets.

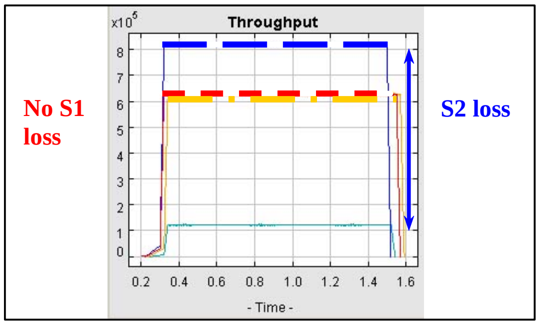

Figure 6. *DiffServ network without*

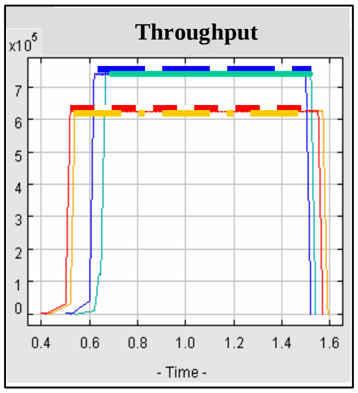

Figure 7. *The throughput of the S1 and S2 traffic after the aaent's intearation*

6. Conclusion

This paper proposes an implementation of a new type of platforms for simulating agents and multi-agent systems within telecommunication networks. The implementation is based on an extension of the J-Sim networks' simulation platform with intelligent capabilities. The extension has been inspired from platforms dedicated to MASs simulations.

The proposed platform was tested for integrating agents and MAS in telecommunication Networks. Different agents' behaviors has been defined and compared, for example an agent behavior was specified to prevent and resolve congested situations in networking. The simulation results show great improvement in the network behavior after the integration of the intelligence represented by the agents' implementation. Other simulations are presented in our previous works [22] and [23], they all give better results after agents integration.

Finally, we can conclude that the proposed dual simulation environment enabled to measure the efficiency of adding different levels of intelligence - represented by intelligent agents - in different types of networks. We believe that agents and MASs will help the future networks' generation reaching a high level of service treatment and network management and control. This requires the usage of the dual simulation environment described in this paper to test and compare different agents' behaviors.

References

[1] Merghem L., “Une Approche Comportementale pour la Modélisation et la Simulaion des Réseaux de Télécommunications”, P.H.D. report, Paris 6 University (France), 2003.

[2] NS-2, available at: www.isi.edu/nsnam/ns/, updated 2008.

[3] OTCl http://otcl-tclcl.sourceforge.net/otcl/, 2007

[4] TCL http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html

[5] http://www.isi.edu/nsnam/ns/doc/node1.html

[6] J-Sim team, Ohio State University, available at: www.J-Sim.org, updated 2005.

[7] http://www.j-sim.org/drcl.inet/inet_tutorial.html, 2003

[8] http://www.j-sim.org/drcl.diffserv/diffserv.html , 2003

[9] http://www.info.ucl.ac.be/~bqu/jsim/mpls_desc.html , 2003

[10] http://www.j-sim.org/contribute.html, 2006

[11] http://www.opnet.com , 2008

[12] Mah B. A., “INSANE Users Manual”, the Tenet Group, Computer science division of the University of California at Berkeley, available at: http://citeseer.ist.psu.edu/499053.html

[13] Wetherall D., Guttag J. and Tennenhouse D., “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”. *IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[14] Muller N., “ATM Introduction to Asynchronous Transfer Mode”, available at: www.ghostship.com/infosyssec/secatm1.htm, 1995.

[15] Tennenhouse D., Smith J., Sincoskie D., Wetherall D. and Minden G., “A Survey of Active Network Research”, *IEEE Communications Magazine*, Vol. 35, No. 1, 1997, pp 80-86.

[16] JAFMAS team, available at: http://www.ececs.uc.edu/~abaker/JAFMAS/

[17] McGeary F., “DECAF Programming: An Introduction”. Foster McGeary Computer and Information Sciences University of Delaware mcgeary, available at: www.eecis.udel.edu/~decaf/papers/DECAFIntro.pdf, April 9, 2001.

[18] Graham J. R., Decker K. S. and Mersic M., “DECAF – A Flexible Multi-Agent System Architecture”. Kluwer Academic Publisher, September 1, 2000.

[19] Barbuceanu M and Fox M., “COOL: A Language for Describing Coordination in Multi Agent Systems”. *In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[20] Garneau T. and Delisle S., “A New General Flexible and Java-Based Software Development Tool for Multi-Agent Systems”. *In Proceedings of the 2003 International Symposium on Information Systems and Engineering (ISE 2003),* Montreal (Quebec, Canada), July 2003.

[21] *J-Sim team, Ohio State University, “The Abstract Network Model (INET)”, Part I, available at: : www.J-Sim.org, December 15, 2003.*

[22] N. Meskaoui, D. Gaiti and, K. Y. Kabalan, “Implementation of a multi-agent system within a Diffserv network to improve its performance.” In *Proc. of the 2003 International Symposium on Information Systems and Engineering, ISE2003,* Montreal, Canada, July 2003.

[23] N. Meskaoui, “A framework to model and simulate multi-agent systems within telecommunication networks: new environment, tools and behaviors,” P.H.D. thesis, Paris 6 University, Paris, France, 2004.