An Agent-based Paradigm for Allocating Multi-Provider Service Demands

M. Calisti and B. Faltings
Laboratoire d'Intelligence Artificielle
Swiss Federal Institute of Technology
CH-1015 Lausanne, Switzerland
{calisti, faltings}@lia.di.epfl.ch

Abstract

The increasing number of competitors and the growing traffic demand are the main factors pushing for a more dynamic and flexible service demand allocation mechanism. Human interactions are becoming more and more inadequate to solve this task, since many different issues have to be considered for optimising all possible choices and strategies. Software tools are becoming fundamental for supporting human decisions and/or for reducing the need of human intervention. Nowadays, agent technology promises a good support for pro-active *and* autonomous *network control, that would enable the automation of many network provider tasks. In order to prove the feasibility of such automation through the use of agents, a multi-agent simulator for the allocation of service demands has been developed. This paper describes the simulator and aims to give a useful feedback for agent developers.*

1 Introduction

For many distributed applications requiring intelligent, autonomous and communicating software entities the multi-agent technology seems to be the most promising answer. Among others and despite the scepticism, the networking community is also investigating the deployment of autonomous agents for different purposes: network control, network management, etc. In particular, in a multi-provider environment the distribution and the heterogeneity of actors, resources and technologies suggests a management solution based on static and/or mobile distributed software entities. Such autonomous entities would have the ability to directly invoke effective changes to switch and router controllers, without the intervention of human operators. Several good papers on this subject can be found in [10]. For evolving the interaction between distinct network providers the NPI-mas, *Network Provider Interoperability-multi agent system*, has been proposed and developed [3]. The basic idea is to provide an efficient and flexible mechanism for self-interested agents representing different network operators for the allocation of service demands spanning distinct networks. In order to prove the feasibility of this paradigm, a multi-agent simulator has been developed.

This paper focuses on the implementation of NPI-mas aiming to describe the software components which the simulator consist of for:

- Showing the feasibility of distributed algorithms that support the allocation of multi-domain service demands. The simulations validate the algorithms and give quantitative evaluations of the performance that could be obtained in specific network scenarios.
- Sharing with the readers the lesson learned by programming distributed, autonomous and communicating software entities.
- Presenting a multi-agent paradigm that could be integrated in a real network in order to both support human operators and automate many inter-domain management steps.

First, we present the architecture of the NPI-mas simulator including a description of its main structural components. Evaluation results are then presented and a comparison of our approach to previous work on the multi-provider service demand allocation is given. Final remarks and comments on future work conclude the paper.

2 An Agent-Based Simulator

NPI-mas has been conceived and developed as the virtual place that allows the simulation of multi-domain service demands allocations. A service demand is defined as $d_k ::= (x_k, y_k, qos_{req,k})$, where x_k is the source node, y_k the destination node, and $qos_{req,k}$ the required Quality of Service. A demand may be anything from a video-conference to a virtual link in a Virtual Private Network. In

our framework the QoS requirements correspond to *bandwidth* and the *end-to-end delay*.

Three main types of agents populate NPI-mas. The *End User Agent* (EUA) that acting on behalf of a final end user expresses needs and formulates service demands, the *Service Provider Agent* (SPA) that processes service demands and contacts a *Network Provider Agent* (NPA) that owns or directly controls specific network resources. The NPA contacts peer-operators, whenever interactions are needed, e.g., when the service demand spans several networks. This paper mainly focuses on the *NPA-to-NPA* interactions.

2.1 The NPI-mas Architecture

The NPI-mas simulator is entirely implemented in Java. JATLite [1] has been adopted as the main framework for the agent development. The three main components of the simulator are:

- The *Router* that receives messages from the registered agents and routes them to the correct recipient/s. All agent communications rely upon TCP/IP sockets.
- The *Agent Management Interface*, that is a graphical interface that allows the selection of a multi-domain scenario, the creation of agents, and the visualisation of simulations' outputs.
- The *agents' tribe*: several EUAs and SPAs can be created. A NPA is associated with every network provider in the simulated scenario.

The AMI (Figure 1) enables first the selection of a randomly generated network scenario. Next, all agents populating the pre-selected scenario are created. Each agent automatically registers with the *router*, its name and its address. In addition, every NPA recovers the data describing the network topology from dedicated *management information databases*. Finally, a text window displays the simulation outputs such as the computed end-to-end paths from the source to the destination network, the selected path along which the inter-domain routing is started, the different results of intermediate steps of the demand allocation process and final negotiation outcomes.

Every NPI-mas agent is associated with a graphical interface that displays all messages that are sent and received by the agent. From the interface associated with an EUA, it is possible to enter a specific service demand. An auxiliary window is used to enter a service demand in terms of source and destination nodes, required amount of bandwidth, available budget, required end-to-end delay and temporal constraints. Next, the EUA sends a *call for proposal* to a selected SPA.

[1] JATLite is a set of packages that facilitates the agent framework development using Java. On-line documentation can be found at: http://piano.stanford.edu/

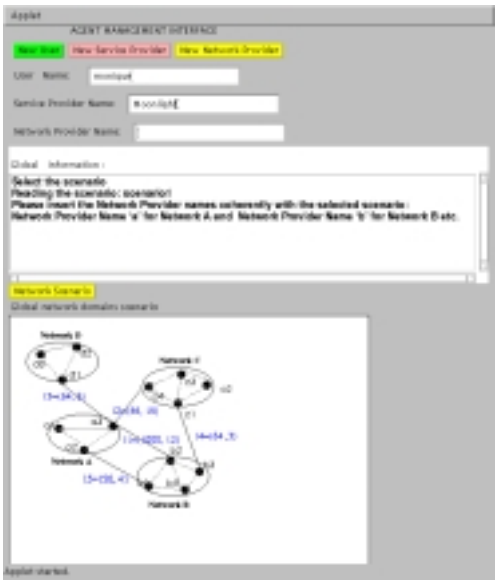

Figure 1. *The Agent Management Interface.*

An SPA represents both the *client* of the network services offered by the NPAs and the *provider* of a variety of telecommunications services to customers (EUAs). The SPA acts as a (currently very simple) *matchmaker*, finding suitable agents (NPAs), and accessing them on behalf of the requesting agent (EUA). In the future, this entity will be enhanced by introducing more sophisticated *brokering* and *recruiting* capabilities.

Since the main goal of our simulator is to make use of the algorithms and the techniques designed and developed for the inter-domain QoS-based routing, we focused on the development of Network Provider Agents. Because of multiple tasks an NPA has to carry on several sub-components have been implemented. NPA *input* can be either messages coming from other agents or human user commands. An NPA *central controller* is responsible for the coordination of several parallel activities, for processing inputs, for interfacing the agent world with human operators, for getting data characterising the network state, etc. In our simulated scenario, the data is stored in a database that is dynamically updated during the simulation. In a real network, the data would be retrieved directly from the network management platform through the use of ad hoc *wrappers*. The NPA *communicator module* parses agents' messages, maintains communication channels, controls and manages all ongoing conversations. The NPA *CSP expert* is a specialised sub-structure that is responsible for CSP modelling and for applying typical CSP consistency and searching techniques. The NPA *negotiator module* generates strategies for the controller. Considering the current state of the network, the various constraints, the utility function, and the interaction

protocol, this module produces the strategy to be followed at every step of the negotiation process. Possible *outputs* of the NPA activity are either messages to other agents, or internal actions, such as changes in the data configuration, or presentation of options to human operators.

2.2 NPA-to-NPA Interactions

- Every NPA has an aggregated view of the multi-domain network topology that is used for computing the abstract paths to other networks. An *abstract path* is an ordered list of distinct providers' networks between the source and the destination network. We call *initiator* the NPA that first receives a request from a SPA. The choice of a specific abstract path P is based on the following heuristics: (1) Eliminate all the paths that do not guarantee enough bandwidth. (2) Among the paths left select the cheapest. (3) If still more than one path exists, chose the path which has, after having accepted the incoming demand d_k, the largest bandwidth left [2].

- Next, the *initiator* contacts all the NPAs along P requesting them to locally determine the set S of possible internal routes for allocating d_k.

- If all providers are locally consistent, i.e., S non empty, the *arc consistency* phase is started. During this phase, the NPAs exchange information about *inter-domain* constraints. All not compatible network access points are discarded, so that every NPA reduces the set S. This phase involves a propagation of messages among neighbours, in order to revise the set S for every access point that is discarded. How to ensure that this propagation terminates is described later in Section 3.1.

- If the arc consistency is successful, i.e, all NPAs have a non empty set S consistent with inter-domain constraints, the negotiation for selecting a specific end-to-end route is started. The initiator broadcasts a *call-for-proposal* to all NPAs along P. Every contacted agent is supposed to follow the *collect-offers* interaction protocol, Figure 2. An offer consists of a specific local route at a certain price. The *initiator* evaluates all received offers and elaborates possible global offers for the EUA, which involves *pricing* and *profit maximisation*q. The negotiation is successful whether the EUA accepts the offer. The initiator confirms to the NPAs the results of the transition. If the negotiation fails and the end-user does not modify its requirements the demand request is rejected and the initiator notifies all other NPAs.

3 Behind the Evidence

NPI-mas agent communication relies on the use of TCP-IP sockets for the exchange of messages. The JATLite router is responsible for forwarding all messages to the appropriate recipient. A first version of NPI-mas makes use of KQML [6]. KQML facilities are in fact offered within JATLite for the use of such a language between z agents [2]. The *content language* used by NPI-mas agents is an ad hoc set of basic components such as *propositions*, *objects* and *actions*. For instance, *arc-consistency-successful* is a proposition that can be either *true* or *false* whether the inter-domain constraints are satisfied or not. For the future development of NPI-mas, we envisage the usage of the *Constraint Choice Language*, CCL [8]. This language has been expressly designed and developed for agents deploying Constraint Satisfaction Problem techniques, which our solving algorithms are based upon. The common *ontology* that all our agents are referring to is not explicitly and formally formulated. However, the context of a conversation is defined through the use of those common terms and expressions.

Ongoing agent conversations expect certain message sequences, which are fixed by the *interaction protocols*. In NPI-mas for many agent interactions we have been inspired by the standard *fipa-request-protocol*, although the performative we have used belongs to KQML and not to FIPA ACL. This required some semantic mapping between similar communicative acts of the two languages. One of the most delicate interactions is the one described by the *collect-offers* protocol, Figure 2. This protocol has been designed as a one-to-many relationship, that the *initiator* starts. A call for proposal, *cfp*, is sent from the *initiator* to all *peer NPAs* involved in the multi-provider demand allocation process. Every NPA can either *agree* or *refuse* to make a proposal. Between the agreement and the *propose* there is a timeout t_{out} for a peer NPA has to formulate an offer. When the *initiator* has received the offers from all peer operators, a unique *global offer* for the EUA can be formulated.

At this point, a second conversation following the *global-offering* protocol is nested with the previous one. If the EUA accepts the global offer, then the initiator confirms the peer NPAs that the demand will be allocated. If the EUA refuses the offer the message sent to the peer NPAs is a *failure*. The nested conversation is time constrained in order to ensure the termination of the *collect-offers* conversations.

There are two main categories of data structures deployed in NPI-mas: the *durable* ones and the *perishable*

[2] We have also developed a communication language package that enables the use of the FIPA ACL language [7] within JATLite. See http://liawww.epfl.ch/ calisti/ACL-LITE

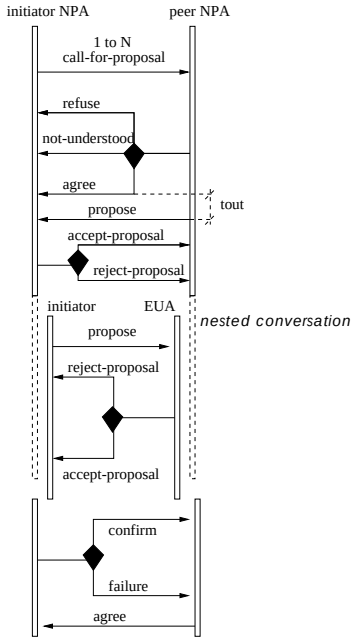

Figure 2. *Nesting interactions: the collect-offers protocol and the nested global-offering protocol.*

ones. The former category includes all these data structures that are created at the beginning of the simulation and that will last during all its duration. The *bootstrap* structure is common to all agents, since it collects an aggregated view of the overall network scenario, i.e., the providers' networks list, the inter-domain links characteristics, the boundary access points, etc. Every agent has also access to a private permanent data structure that contains information about its own resources. A distributed management information structure guarantees in fact a flexible and scalable approach to the service demand allocation. Furthermore, privacy and security are better controlled by having non-shared information structures.

The *perishable* data structures are all these objects that are instantiated during the simulation and that are used by agents for a specific demand allocation. These structures are created, and eventually updated at run-time, and finally deallocated whenever the specific demand allocation process terminates.

3.1 The Lesson Learned

Somebody could argue that agents are just objects by another name, however when implementing software agents that are supposed to be *autonomous*, *reactive*, *pro-active* and *social* the intrinsic difference becomes more evident, at least to an agent developer. Even though objects encapsulate some state, can communicate via message passing, has methods corresponding to operations that may be performed on this state, the main difference is that an object usually has a single thread of control and does not have a flexible behaviour. Basicly, while an object controls its state, an agent controls its behaviour.

Despite 'standard' solutions for supporting agent interactions exist [9], [6], the way to implement them is not always so standard. Specific technical issues that a developer has to face are summarised in the following.

Naming and addressing issues. No matter what kind of platform is used, an agent needs to be uniquely identified in the system he is living in. The 'identity card' of an NPI-mas agent consists of three major fields: the name, the profession and the address. The agent *name* is automatically checked by NPI-mas whenever a new agent is created. The name must be unique. The *profession* identifies if it is an EUA, a SPA or a NPA. This allows the creation of agents with specific skills, that correspond to specific roles played in the multi-provider demand allocation process. The *address* includes the agent name, the host-name (i.e., the machine where the agent is running), and a time-stamp corresponding to the moment at which the agent is created. Addresses are handled transparently by the JATLite router.

Handling multiple conversations. Every agent can be involved in more than one conversation at the time. A conversation identifier, *conv-id*, is used for that purpose for every non isolated communicative act. A vector of *conversation* objects has to be dynamically maintained by every agent.

Handling multiple data structures. Analogously to what happens with multiple conversations, every agent manages vectors of data structures, since different objects need to be instantiated for different and parallel allocation demands. The dynamic update of such vectors is the most delicate part since messages from different agents can concern the same data structure. To face this concurrency problem we adopted a FIFO policy, serialising the access to the data structures.

Detection of a global state. During the 'arc consistency' phase there is a propagation of messages among agents in order to revise the set of possible local routes. This phase ends when all agents are consistent (or eventually as soon as one of them is inconsistent) and when no messages are pending in the system, i.e., when the *global state* is stable [4]. In order to detect that this state is reached a control mechanism, namely a Java thread instantiated by the NPA *initiator*, is used. The initiator receives notifications about the state of every single agent involved in the arc consistency and about the messages that are spread around in the system. This allows to maintain and update a global state, that is periodically checked by the control thread.

Integration of JATLite with 'external' Java code. The modularity of the JATLite's architecture enables developers to build agent-based systems that deploy only specific

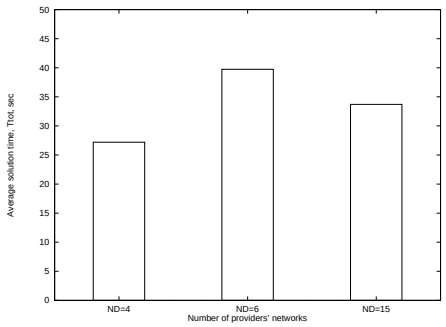

Figure 3. *Three different simulated scenarios have been considered. T_{tot} has been computed over a set of 300 randomly generated demands.*

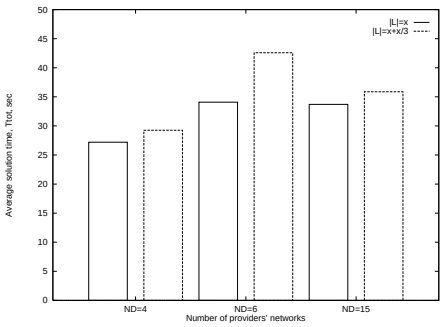

Figure 4. *T_{tot} increases when increasing the number of inter-domain links, since the search space increases.*

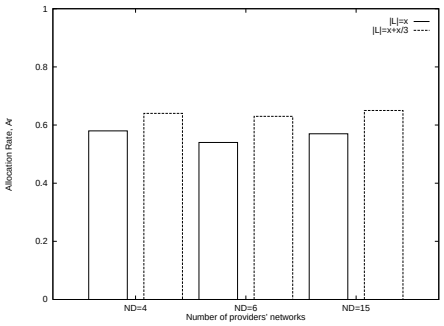

Figure 5. *The graphic shows the increment of demands successfully allocated, when increasing the inter-domain capacity.*

packages of such a tool. The on-line documentation of both JATLite and Java has been sufficient for the integration of all the NPI-mas components.

3.2 Simulation Results

The performance metrics that have been observed for evaluating the NPI paradigm are the *average demand allocation time*, T_{tot}, and the *allocation rate*, $A_r := nbsuccess/nbdemands$, with $nbsuccess$ the number of demands successfully allocated, and $nbdemands$ the total number of service demands.

Simulations with a variable number N_D of networks (Figure 3), and therefore of NPAs [3], have been run in order to check the scalability of our algorithms and in parallel the usability of communicating agents for the multi-provider service allocation purpose. The average values that has been obtained for T_{tot}, i.e., $[25, 40]$ seconds, can be considered a positive result when compared to the current delays required by human operators to negotiate solutions and topology changes. Given a fixed number N_D of providers' networks, we then tested the influence of a different number $|L|$ of inter-domain links. Increasing $|L|$ has a double effect: (1) The complexity of the solving process increases by augmenting the number of possible access points combinations between neighbour networks: T_{tot} increases, see Figure 4. (2) The probability that at least one end-to-end path satisfying the QoS requirements of demands to be allocated exists augments: A_r grows, see Figure 5. Similar results have been obtained when varying the complexity of the internal topology of every network. A greater number of network nodes and intra-domain links can augment the search space and can introduce an additional computational overhead. However, as for an increment of $|L|$, if the topological changes correspond to an increment of the network resources, the number of demands successfully allocated increases.

[3] In our simulations $N_D \in [4, 15]$, which correspond to a realistic multi-provider scenario. However, we are testing the NPI paradigm for greater values of N_D.

3.3 Contribution

Many multi-agent systems are already available and the most common classifications distinguish between architectures for *reactive agents*, *deliberative agents* and *interacting agents* (see [12] for good references). Hybrid architectures have also been developed for integrating reaction and deliberation. However, some platforms are too specific, some others are not freely available, and more in general no pre-existing agent platform was offering a support for any kind of Constraint Satisfaction (CS) technique. Furthermore, when we started the development of NPI-mas no FIPA compliant agent platform was available.

NPI-mas has been conceived as an agent platform which (1) can flexibly make use of ad hoc Constraint Satisfaction algorithms; (2) supports agent communications either via

KQML or FIPA ACL; (3) allows the simulation of different underlying network scenarios, (4) supports the simulation of automatic multi-provider service demand allocations. NPI-mas agents could be integrated within a real network by providing ad hoc *wrappers* in order to interface the *agent-based* and the *non-agent* worlds, i.e., the network management and control level. These wrappers would take into account the low level characteristics of specific underlying network technologies.

The service demand allocation is a complex and articulated process that is even more delicate for networks that aim to provide QoS guarantees [5], especially in a multi-provider context where every provider tries to maximise his own utility and knowledge about the network topology is restricted by the network providers for strategic reasons.

Although multi-domain QoS routing has been tackled from many directions (ATM and SDH network [11], billing [1], multi-domain management in the MISA [4] project, agent interactions for routing multimedia traffic over distributed networks, see [7] and the FACTS project [5]) no previous work has addressed the possibility of dynamic negotiations about more than one path at a time.

In NPI-mas the use of Distributed Constraint Satisfaction techniques and the deployment of autonomous agents making use of a compact aggregation of network resources availability characteristics, offer the capability of: (1) accelerating the allocation of multi-provider service demands, by automating many steps currently performed by humans. (2) Supplying standard solutions that abstract from technical details, by using a common and standard agent communication language and a standard ontology. (3) Supplying consistent solutions without the need of explicating internal and confidential data, such as network topologies, negotiation strategies, pricing mechanisms, etc. (4) Supporting human decisions, or, in a more future scenario, of replacing human operators. (5) Integrating economic principles within *self-interested* agents, in order to optimise the revenue.

4 Final Remarks

Implementing the NPI-mas system has been essential to validate theoretical concepts previously defined, as well as to destroy many of the prior certainties and modify some concepts about both agents and service demand allocation. The results obtained through the simulation prove the potential of our paradigm.

Beyond more realistic simulations and more exhaustive data analysis, there are several directions that the authors are considering for the future development of NPI-mas. First of all, more sophisticated negotiation techniques and more accurate pricing mechanisms need to be integrated. Furthermore, more complete and realistic representations of networking data and multi-provider service demands would bring additional value to the simulations. The possibility of creating *coalitions* among NPAs, in order to take advantage of a higher degree of coordination, is an alternative way of determining a global service offer for end-users. More work on the *service provider* level, i.e., more sophisticated brokering capabilities would be added to the SPAs agents. Additional work on the interaction between the human end-user and the EUA together with more sophisticated brokering capabilities for the SPAs are currently under investigation.

[4] http://www.misa.ch

[5] http://www.labs.bt.com/profsoc/facts

References

[1] C. Bleakley, W. Donnelly, A. Lindgren, and H. Vuorela. TMN specifications to support inter-domain exchange of accounting, billing and charging information. *Lecture Notes in Computer Science*, 1238, 1997.

[2] J.-Y. L. Boudec and T. Przygienda. A Route Pre-Computation Algorithm for Integrated Services Network. Technical Report TR-95/113, LRC-DI, EPFL, Lausanne, Switzerland, 1995.

[3] M. Calisti, C. Frei, and B. Faltings. A distributed approach for QoS-based multi-domain routing. *AiDIN'99, AAAI-Workshop on Artificial Intelligence for Distributed Information Networking*, 1999.

[4] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *TOCS*, 3(1):63–75, Feb. 1985.

[5] S. Chen and K. Nahrstedt. An Overview of Quality of Service Routing for Next-Generation High-Speed Networks: Problems and Solutions. *IEEE Network*, pages 64–79, November/December 1998.

[6] T. Finin et al. Specification of the KQML Agent-Communication Language – plus example agent policies and architectures, 1993.

[7] Fipa. Fipa spec. 97 v2.0. *Foundation for Intelligent Physical Agents*, 1997.

[8] Fipa. Fipa spec. 99, v0.2. *Foundation for Intelligent Physical Agents*, 1999.

[9] Foundation for Intelligent Physical Agents. FIPA Specifications, Oct. 1997. http://www.fipa.org/spec/.

[10] A. L. G. Hayzelden and J. Bigham, editors. *Software Agents for Future Communication Systems: Agent Based Digital Communication*. Springer-Verlag, Berlin Germany, 1999.

[11] D. Karali, F. Karayannis, K. Berdekas, and J. Reilly. Qos based multi-domain routing in public broadband networks. *Lecture Notes in Computer Science*, 1430, 1998.

[12] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.