PROCEEDINGS OF THE WORKSHOP ON APPLICATIONS OF SOFTWARE AGENTS
ISBN 978-86-7031-188-6, pp. 18 - 24, 2011

Self-organized Multi-agent System for Service Management in the Next Generation Networks

Mario Kusek and Gordan Jezic

University of Zagreb, Faculty of Electrical Engineering and Computing,
Department of Telecommunications, Unska 3, HR-10000 Zagreb, Croatia
{gordan.jezic, mario.kusek}@fer.hr

1 Extended Abstract

Next Generation Networks (NGN) aim to offer a wide variety of advanced telecommunications and multimedia services. Introduction of these services will be enabled mainly by two factors: increased bandwidth in the access network and convergence of different legacy networks towards a universal all-IP core network. The offer of large number of services in the NGN environment will arise the need for service provisioning and management procedures.

Service management on emerging telecommunication systems that are distributed over a wide area is a hard task because it is not easy or even possible to perform final testing on a remote target system, as well as on system in operation [1-3]. Experiences show that it is possible for new software running on a target system to give a result different from the one obtained on test system [4]. The reasons are mostly the structural and/or functional differences between both systems. Therefore, only implementation and testing on the actual target system can give the answer whether the new software solves the problem (i.e., error, new operational circumstances, enhancement, and maintainability improvement) or not. Service management and software configuration operations in distributed systems become very demanding tasks as the number of computers and/or geographical distances between them grow. The situation gets worse with an increase in the complexity of the network and the number of nodes.

This paper describes a method for service provisioning in an environment with a large number of distributed network servers and different versions of services placed across them. We have developed the agent based system called Remote Maintenance Shell (RMS) capable for remote control and management of services. To be specific, we present solutions for getting the service to the right place, starting it, and providing maintenance (upgrading with new versions). We have defined possible service distribution strategies for execution of operations and in this paper we consider applying of self-organized agents as a possible solution for this problem.

1.1 RMS System

RMS represents a protected environment for service control, manage and maintenance based on mobile agents [5]. It includes the following remote operations that support service maintenance: delivering service to a remote system, remote installation/un-installation, program starting/stopping, tracing and trace data collection, maintaining several versions of service, selective or parallel execution of two versions, and version replacement [6].

The main advantages of RMS over similar tools for remote installations are the following:

- It provides the possibility to test and trace the software on the actual target system where it must be deployed, which is the only way to make sure that the software will work properly when put online;
- It provides selective and parallel software execution modes, which are particularly suitable for introducing new software or upgrading the existing one without stopping the system.

MA-RMS (Multi-Agent RMS) is the upgraded version of RMS, which is based on a multi-agent system paradigm. In RMS tasks have to be assigned one-by-one by the human administrator, and they are performed by individual agents. MA-RMS makes it possible for the human administrator to assign only the desired end-state of software environment on one or several remote locations, which is then automatically processed by the MA-RMS system. Input by the human administrator is analyzed and decomposed into a set of operations, which are then distributed to multiple mobile agents to perform them. Those agents migrate to their respective locations, and continue to communicate and coordinate in order to accomplish the given goals. This approach makes it significantly easier to perform service provisioning in large distributed systems, since it allows automatic replication of the desired configuration across a large number of servers instead of many individual installations.

The position of the MA-RMS in a network node of the distributed system is shown in Fig. 1. The main role of the MA-RMS system is to provide the requirements for service management in the distributed system and handle several versions of services (verX) on multiple network nodes simultaneously, without suspending or influencing normal services operation. MA-RMS system uses an agent platform as a base for agent creation and management, and adopts all security mechanisms, as well as other basic features from the agent platform.

In the environment with a large number of network nodes and different kinds and versions of the heterogeneous services placed on the network nodes, MA-RMS system is capable to remotely control and manage the service. Mobile agents are the carriers of all operations. They offer several important advantages that make them especially suitable for implementation in the distributed systems, as agents can be employed to perform tasks as a team.

The limitations of this approach can be found in the fact that agent platform must be installed and always started on all network nodes in the system. This will additionally increase the load on the node, which can lead to some stability issues (particularly on the nodes which are already running at or very near their projected capacity).

Moreover, Java Virtual Machine must be run on all of the nodes supporting MA-RMS.

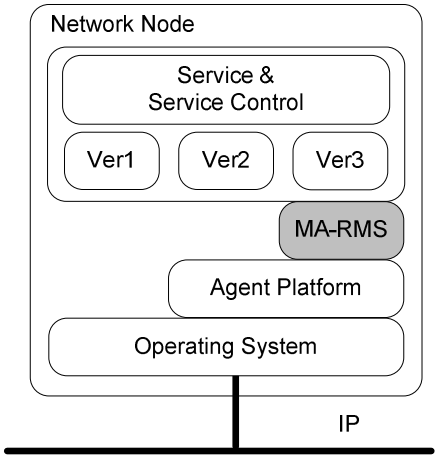

Fig. 1. MA-RMS in a network node

MA-RMS is based on three principles [7], as follows:

- Design for remote maintenance: An application should be designed according to specific rules in order to fit the requirements for remote maintenance;
- Low resource implementation: Only MA-RMS parts needed for a specific maintenance job are activated, all other remain inactive in order to save the system resources for regular operation;
- Agent-supported maintenance session: Software agents support all remote operations requested by a maintenance session and guided by user.

1.2 RMS Architecture

RMS is a distributed system comprising two main components: RMS Console and RMS Maintenance Environment (RMS ME). RMS Console is the client part of the RMS, which offers a GUI through which the administrator performs management actions on the remote systems. RMS Maintenance Environment is the server part of RMS, which must be preinstalled on the remote systems in order for them to be managed by RMS. RMS is implemented in Java, so it can be installed on any operating system with Java Virtual Machine installed. It should be noted that, although RMS can be used for client-side software management, it is designed primarily for the use on servers. All operations in RMS are executed by mobile agents. A mobile software agent is a program that represents its user in the network. The most important characteristic of mobile agents is their ability to migrate autonomously between the network nodes. RMS uses JADE as the underlying agent system [8]. Once the administrator defines the operations to be performed at the remote system(s), they are assigned to

one or several mobile agents, which then migrate to the remote system(s) and perform the operations at the actual target system.

Mobile agents are used because they offered several important advantages that made them especially suitable for implementation of a centralized system for remote software management, such as RMS. Namely, the use of mobile agents produces the following benefits in RMS.

- Completely decentralised operation execution – only operations assignment is centralized in the management station, while the actual execution of the operations takes place at the target remote system(s).
- Increased parallelism – agents can do their jobs in parallel, since they are completely autonomous after they are sent into the network and they don't have to be controlled by the management station.
- Increased asynchrony – once the administrator defines the desired operations and the agents are sent into the network, it is no longer necessary to maintain a permanent connection with the remote systems.
- Reduced sensibility to network latency – since the agents migrate to the remote system, none of the interactions during software maintenance are made over the network. Instead, they are carried out locally at the remote system.
- Fexible configuration of remote testing procedures – since the testing is performed by a mobile agent, it is possible to dynamically reconfigure that agent and thus adapt the testing procedure without changing the RMS Maintenance Environment itself.

1.3 RMS Features

RMS features can be roughly divided in two main categories: basic and advanced. What separates them is the amount and type of additional work that has to be done for the software to be managed with RMS using these features.

Basic features are those related to deployment, maintenance and execution control (software migration, installation, starting, stopping and basic version handling). They include the mechanisms for bringing the software to the targeted remote systems, configuring it properly on each of them, starting and stopping it. Basic version handling offers the possibility to stop the old version and replace it with the newer one. It is possible to maintain several versions of the same software and switch between them, but only one can be active at any given time (in the normal execution mode).

Advanced features are tracing, testing and advanced version handling. As mentioned before, tracing and testing are performed on the actual target system. Besides normal execution mode, in which only one version is active, RMS also provides parallel and selective execution modes.

1.4 Agent Distribution Strategy

When deploying services at a large number of remote locations using software agents, two parameters significantly affect the deployment time. The first one is the

number of agents in the team. Increasing number of agents produce bigger load in the network. The second parameter affecting deployment time defines which service should be assigned to which agent. The two parameters are defined by service distribution strategies [9]. We have currently defined the following service distribution strategies:

R1: a single agent executes all services on all nodes;

2. R2: an agent executes a single service on only one node;
3. R3: an agent executes all services on only one node;
4. R4: an agent executes a specific service on all nodes;
5. R5: an agent executes a specific service only once on all nodes;
6. R6: services are assigned to the agents in order to exploit maximal parallelism in service execution. Mutually independent services are assigned to different agents, in order to execute them simultaneously on nodes with parallel execution supported;
7. R7: a hybrid solution combining R4 and R3. An agent is responsible for a specific service on all nodes; all other agents execute all other services, each on a different node;
8. R8: a hybrid solution combining R5 and R3 (specialization of R7 in the way R5 is specialization of R4).

For example the R3 service distribution strategy distributes all the services, which have to be executed, on one node to one agent. This strategy has proven to be the best strategy when executing services in networks where the network bandwidth is large compared to the size of the components each service is made of. In order to perform fast analysis on different network topologies we developed a MAN Simulator [10, 11].

Current service distribution strategies used by the MA-RMS are only optimal for some network topologies with certain network parameters, such as the R3 strategy mentioned before. The current service distribution selector first analyzes the network topology and network conditions. It then selects the service distribution strategy, which will most likely yield best solution according to the analysis. The problem with real networks is that their conditions change and are never ideal [9].

1.5 Genetic algorithm for optimizing service distributions

The genetic algorithm is used by researchers to solve a variety of search and optimization problems. All of these problems have one thing common, exact algorithms cannot find an optimal solution in a reasonable time. Genetic algorithms do not always yield the optimal solution due to randomness in implemented operators. However, it guarantees that it will find a suboptimal solution in a reasonable time. This is satisfactory for us since an exact algorithm will take too long to find a solution. In [chapter] we have designed genetic algorithm for optimizing service distribution to software agents. We have compared results with different strategies and different network link bandwidth.

From the previous experiments [9] we know that the R3 distribution strategy always performs the best in scenarios where the time needed to migrate all service components is significantly smaller then time needed to deploy these services. However,

when the link speed was reduced the genetic algorithm was able to generate a better distribution then the R3 strategy.

In scenarios with the bottleneck in the network the genetic algorithm produced the best results. The reason for this is because the time needed to migrate service components was comparable to the time needed to execute them. In such cases, the genetic algorithm was able to produce distributions in which the first agent had more services assigned to it then the second agents.

The same applies in situation where there are dependencies between different services. Most distributions generate a solution in which agents have to wait for other agents due to long migration times. The genetic algorithm is capable of optimizing the order agents deploy each service increase of parallelism in service execution and avoiding the waiting times.

In this experiment we have proved that the genetic algorithm can give better results then the other strategies but we have not calculate the time for executing the genetic algorithm into account. The second disadvantage of genetic algorithm and defined strategies is that in planning process the planning agent needs to have all relevant data from the network and when the condition in the network changes dynamically the agents does not adapt. That is the reason why we consider using self-organizing agents.

1.6 Self-organizing agents

There is no unique definition of self-organization but there is list of properties that usually include [12]:

- System appears to have spontaneous order.
- The overall state of such system is an emergent property.
- Interconnected components are organized in productive way based on local information.
- Complex system can self-organize.
- The process of self-organization is near the “edge of chaos”.

By some scientists the evolution is combination of natural selection (e.g. in evolutionary algorithms) and self-organization. The self-organized agents use local data to make its decisions. The main idea is to have agents that are committed to the goal and each agent itself chooses specific actions. The user gives list of operations to all agents in the team. Each agent then decides (e.g. random) to which operation will execute. After that agents goes to the node where action must be executed and checks if the action is executed or not. If action is not executed then it executes. If the action is already executed then the agent chooses another operation from the set of operations that should be executed on the same local network. If that operation is also executed then it tries to execute another operation on the same local network. If all operations on that local network are executed then the agent chooses operation on another network. The job is finished when all agents find that all operations are finished.

This is not optimal solution because all agents have to go to each node from the initial list. We will have to investigate some mechanisms to reduce number of nodes that each agent visits. One solution is to have blackboard on each node and when one

agent enters the node it checks the blackboard and updates its list of executed operations and updates the blackboard. Then it chooses the next operation for execution.

The second solution is threshold value. If the chosen operation is already executed then the agent updates its threshold value and if the threshold value is over boundary then it aborts execution of operations in that local network. There could be also the threshold for local networks as well. If agent aborts its execution it informs user agent upon its findings and if all agents are aborted and all jobs are not executed then the user agent can choose different strategy for remaining jobs.

References

1. Pigosky, T.M, Practical Software Maintenance, Wiley, New York (1996)
2. IEEE Std. 1219: Standard for Software Maintenance, Los Alamitos, CA, IEEE Computer Society Press (1993)
3. Canfora, G., Cimitile, A, Software Maintenace, Chapter 2 of Handbook of Software Engineering and Knowledge Engineering, Volume 1, World Scientific Publishing Company, 1st edition (2002)
4. Lovrek, I., M. Kos, B. Mikac, "Collaboration between academia and industry: Telecommunications and informatics at the University of Zagreb", *Computer communications*. **26**, 5; pp. 451-459 (2003)
5. Jezic, G., M. Kusek, I. Ljubi, "Mobile Agent Based Distributed Web Management", *Proc. 4th Int. Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies*, Vol. 2, pp. 679-682, Brighton (2000)
6. Jezic, G., M. Kusek, I. Ljubi, K. Jurasovic, Mobile Agent-Based System for Distributed-Software Maintenance, L.C. Jain, N.T. Nguyen (Eds.): Knowl. Proc. & Dec. Mak. in Agent-Based Sys., SCI 170, pp. 43–69.
7. Java Agent DEvelopment Framework – JADE, http://jade.tilab.com/
8. Lovrek, I., Caric, A., Huljenic, D.: Remote Maintenance Shell: Software Operations using Mobile Agents. In: ICT 2002, International Conference on Telecommunications, Beijing (2002)
9. Kusek, M., K. Jurasovic, I. Lovrek, V. Sinkovic, G. Jezic, Performance Models for Multi-agent Systems and Mobile Agent Network, A. Hākansson, R. Hartung, N.T. Nguyen (Eds.): Chapter 1 of Agent And Multi-Agent Technology For Internet And Enterprise Systems: Studies in Computational Intelligence (2010), Vol. 289, pp. 1-24, Springer
10. Xuan, P., V. Lesser: Multi-Agent Policies: From Centralized Ones to Decentralized Ones. Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-agent Systems Part 3 (2002), pp. 1098–1105
11. Koriem, S.M.: Development, analysis and evaluation of performance models for mobile multi-agent networks. Comput. J. 49(6) (2006) pp. 685–709
12. Kennedy, J., R. C. Eberhart, Y. Shi: *Swarm Intelligence*, Morgan Kaufmann, 2001.