Mobile Agents Self-optimization with MAWeS

Emilio Pasquale Mancini[1], Massimiliano Rak[2],
Salvatore Venticinque[2], and Umberto Villano[1]

[1] Università del Sannio, Benevento, Italy
{epmancini, villano}@unisannio.it
[2] Seconda Università di Napoli, Via Roma 29, IT-81031 Aversa (CE), Italy
{massimiliano.rak, salvatore.venticinque}@unina2.it

Abstract. In mobile agents systems, classical techniques for system optimization are not applicable due to continuous changes of the execution contexts. MAWeS (MetaPL/HeSSE Autonomic Web Services) is a framework whose aim is to support the development of self-optimizing autonomic systems for Web service architectures. In this paper we apply the autonomic approach to the reconfiguration of agent-based applications. The enrichment of the Aglet Workbench with a Web Services interface is described, along with the extensions to the MAWeS framework needed to support the mobile agents programming paradigm. Then a mobile agents application solving the N-Body problem is presented as a case study.

1 Introduction

The mobile agents programming paradigm is an emerging approach for distributed programming. Agents-based platforms are considered good solutions in many fields, such as GRID [1,2,3] or SOA (Service Oriented Architecture) [4,5]. In mobile agents systems, classical techniques for system optimization (such as *ad-hoc* tuning, performance engineered software development, ...) are hard to apply. This is essentially due to continuous changes of the execution contexts, as an agent is able to suspend its own execution, to transfer itself to another agent-enabled host and to resume its execution at that destination. So, even if the mobile agents approach may help to develop performance-oriented applications, in practice the only solution to guarantee critical requirements seems to be the use of an architecture able to auto-configure and to auto-tune until the given requirements are met. Moreover, when an agent moves itself, it impacts the state of the new system. A prediction of the modified state can help to make good choices for agents reconfiguration.

Autonomic computing [6,7,8,9], whose name derives from the autonomic nervous system, aims to bring automated self-management capabilities into computing systems. In previous papers, we have introduced MAWeS (MetaPL/HeSSE Autonomic Web Services) [10,11], a framework whose aim is to support the development of self-optimizing autonomic systems for Web service architectures. It adopts a simulation-based methodology, which allows to predict system performance in different status and load conditions. The predicted results are used for

B. Kågström et al. (Eds.): PARA 2006, LNCS 4699, pp. 1158–1167, 2007.

a feedforward control of the system, which self-tunes *before* the new conditions and the subsequent performance losses are actually observed.

MAWeS is based on two existing technologies: the MetaPL language [12] and the HeSSE simulation environment [13]. The first is used to describe the software system and the interactions inside it; the latter, to describe the system behavior and to predict performance using simulation. Using MAWeS, it is possible to add self-adaptive features both at *service level*, i.e., building up services that optimize themselves in function of their overall usage [11], and at *application level*, i.e., focusing on the application behavior [10]. In the latter case, a standard client application interface, `MAWeSclient`, provides the general services that can be used and extended to develop new applications.

In this paper we propose to apply the proposed approach to the reconfiguration of agent-based applications integrating the Web Service and mobile agent programming paradigm. We aim at defining the potentiality of this approach and at proposing an architecture able to build self-optimizing agents. In order to achieve this result, we had to extend both mobile agent platforms and the framework architecture. Moreover, we had to build extensions to the description language and the simulation engine.

The reminder of the paper is organized as follows. The next section presents previous work and the scientific background. Section 3 describes our proposal, presenting the extension made to both the mobile agents platform and the MAWeS framework. Section 4 deals with an example, showing a mobile agents-based NBody application and its integration within MAWeS. The paper will end with our conclusions, which summarize the results obtained, and with a discussion of our future work.

2 Background

Our proposal for mobile agents self-optimization relies on previous research and on a large amount of existing software tools. In order to present the work done, in this section we introduce the main concepts about the mobile agent programming paradigm adopted, describing the chosen platform. Moreover, we describe the state of the art of the optimization framework adopted, MAWeS.

A mobile agent is a software agent with an added feature: the capability to migrate across the network, together with its own code and execution state. This paradigm allows both a pull and a push execution model [14]. In fact, the user can choose to download an agent, or to move it to another host. Mobility can provide many advantages for the development of distributed applications. System reconfiguration by agent migration can help to optimize the execution time by reducing network traffic and interactions with remote systems. Furthermore, stateful migration allows to redistribute dynamically the agents for load balancing purposes. Several different criteria can guide agent distribution, such as moving the execution near to the data, exploiting new idle nodes, or allocating agents on the nodes in such a way that communications are optimized.

Due to previous experiences and for the facilities it offers, we chose to adopt the Aglet Workbench, developed by IBM Japan research group [15]. Af far as interoperability is concerned, the Aglet Workbench is compliant with the MASIF specification [16]. It relies on a transport protocol that is an extension of http.

The MAWeS Framework has been developed to support the predictive autonomicity in Web Service-based architectures. It is based on two existing technologies: the MetaPL language [12] and the HeSSE simulation environment [13]. The first is used to describe the software system and the interactions inside it; the second, to describe the system behavior and to predict performance using simulation.

MetaPL is an XML-based meta-language for parallel program description, which, like other prototype languages, can also be used when applications are not (completely) available [12]. It is language independent, and can be adopted to support different programming paradigms. It is structured in layers, with a core that can be extended through *Language Extensions*, implemented as XML DTDS. These extensions introduce new constructs into the language. Starting from a MetaPL program description, a set of extensible filters makes it possible to produce different program *views*.

HeSSE is a simulation tool that allows to simulate the performance behavior of a wide range of distributed systems for a given application, under different computing and network load conditions. It makes it possible to describe distributed heterogeneous systems by interconnecting simple components, which reproduce the performance behavior of a section of the complete system (for instance a CPU, a network ...). These components have to reproduce both the functional and temporal behavior of the subsystem they represent.

The MAWeS framework uses MetaPL descriptions and HeSSE configuration files to run simulations. Through the execution of multiple simulations, differentiated by one or more parameter values, it chooses the parameter set that optimizes the software execution. MAWeS is structured in three layers. The first one is the front-end, which contains the software modules used by final users to access the MAWeS services. The second one is the core, which includes the components that manage MetaPL files and make optimization decisions. The last one contains the Web Services used to obtain simulations and predictions through MetaPL and HeSSE.

MAWeS operates adopting two strategies:

Service Call optimizations (simulation, evaluation of the choices and application tuning) take place at application startup (increasing the application startup latency).

Reactive optimizations (simulation, evaluation of the choices and application tuning) take place asynchronously with the application and do not affect the application performance, except when the framework decides to change the application behavior.

The MetaPL/HeSSE WS interface defines a set of services that make it possible to automate the application of the methodology. When the MAWeS framework is firstly used, it is necessary to describe the components in MetaPL. This

can be done before starting the development to obtain a prototype view to analyze (and to optimize), or, in parallel with it, to verify the design choices.

Inside the meta-description, it must be suitably identified the set of parameters that can be modified by the optimization engine. MAWeS will automatically perform a set of simulations varying the values of these parameters to find the optimal value set. The user can specify the tunable parameters by means of the *autonomic* MetaPL extensions, which define new MetaPL elements for the `Mapping` section [10].

3 MAWeS and Agents Integration

As pointed out in the introduction, the mobile agents paradigm is a powerful programming technique, which can help to reconfigure an application in a very easy and clean way, distributing data in a distributed system and possibly achieving better performance through load balancing. On the other hand, evaluation of overhead and actual performance of the application may be a hard task: agents run in an hosted environment (the mobile agent platform) that hides completely the hardware behavior. So, it is difficult to find out the best configuration for an application. The most viable solution seems to be the development of self-adapting applications.

The MAWeS Framework was initially developed in order to help application self-configuration, embedding into them a client that evaluates many different hardware/software configurations (i.e., distributions of software onto a distributed system and application parameters) in a completely different context, namely Web Services and hence service-oriented applications. In order to adapt the MAWeS framework to applications based on the mobile agents paradigm, it is necessary:

- to describe the evolution of a mobile agent application, in order to make it possible the prediction of its performance behavior on the target environment. This leads to the requirement for a Mobile Agent Extension for the MetaPL description language;
- to simulate the execution of the mobile agent based application. This involves the development of a new HeSSE library modeling the agents and their runtime support;
- the framework should be able to control and to change the mobile agents-based application. So it has to communicate with agents and/or their platform and to be able to move/clone/destroy/... the agents and to communicate with them through messages. Moreover, an agent should be able to invoke the framework in order to notify new changes into the environment.

The last point opens a new universe of problems, as it involves the integration between mobile agents application and the web services programming paradigm, which is the only form of interaction supported by MAWeS. We focused on two different ways to integrate a WS interface to mobile agent systems:

Mobile Agents as WS clients. A mobile agent is able to access to an external web services, i.e., it directly performs SOAP requests. This approach lets an agent to invoke MAWeS services.

Mobile Agent Platform as a WS server. The mobile agents platform has a Web Services interface. A WS client can access the mobile platform and send messages to the agents and/or change their state (clone, migrate, ...)

In order to simplify the implementation, here we focus only on the MAWeS Service Call Optimization strategies. This means that we aim at optimizing the starting distribution of the mobile agents to the distributed environment. Once the application is started, its behavior does not change. As a consequence, we implemented only the second proposed technique for mobile agent and WS integration, modifying the existing Aglets platform in order to add a Web Services interface.

Moreover, we assume that the available distributed environment does not change frequently, and so the data about the available platforms and their performance may be collected off-line, thus being available when the application starts. This implies that the MAWeS framework does not need to discover dynamically the changes in the execution environment (such as the availability of a new agent platform), as this information is provided to the framework off-line.

3.1 Mobile Agents and Web Services

To support the interaction between a mobile agent system and MAWeS, we enriched the Aglet Workbench [15] with a Web Service interface. We developed a Web Service that implements a SOAP bridge, enabling any requestor to invoke the agent platform facilities and to interact with the agents hosted locally or with remote ones. Furthermore, many basic platform facilities such as creation, cloning, dispatching have been exported as services. The available methods provide functionalities for agents creation, disposal, cloning, migration and discovery, along with point-to-point and multicast messaging. These methods invoke the Aglets API to exploit the functionalities of the agent context that has been created at startup. In order to add a Web Service interface to the agent platform, we turned it into a web application, to let it be executed in a container of the chosen application server, Jakarta Tomcat.

It should be noted that communication among agents and among agent servers exploits the Agent Transfer Protocol. On the other hand, communication between web service clients and the platform relies on SOAP messages, which are handled directly by Tomcat. SOAP messages may be service requests, such as agent creation or migration, or messages to be forwarded to the agents. The enhanced web agent platform will take care of distinguishing between the two, forwarding the incoming SOAP messages to the agents or performing the invoked service.

3.2 The MAWeS Agent Extended Architecture

Mobile agent-based applications launch depends on the platform chosen for their execution. Usually the graphical interface offers a screen in which the user

chooses the class to be executed. The agents code is already on the server or accessible from a remote codebase. Thanks to the newly developed Web Services interface, we are able to create and to run an agent through a web services client application, which invokes the target environment. This client application is an extension of the standard MAWeS client.

The agent-based MAWeS client contains the description of the target mobile agents, MetaPL documents, and the link to their code (i.e., the link needed by the platform to create and to start up the agent). When the user starts up the client, it queries the MAWeS engine, which returns the list of servers to be adopted, the number of agents to start on each server and, possibly, application-dependent parameters.

The MAWeS core was extended in order to maintain information about the available mobile agents platforms. At the state of the art, the core retrieves the list of the available hosts and their performance parameters using an UDDI register. This means that parameters are statically stored. Future extensions will provide a protocol to check that the platforms are active and to measure their performance. The resulting architecture is shown in Figure 1.

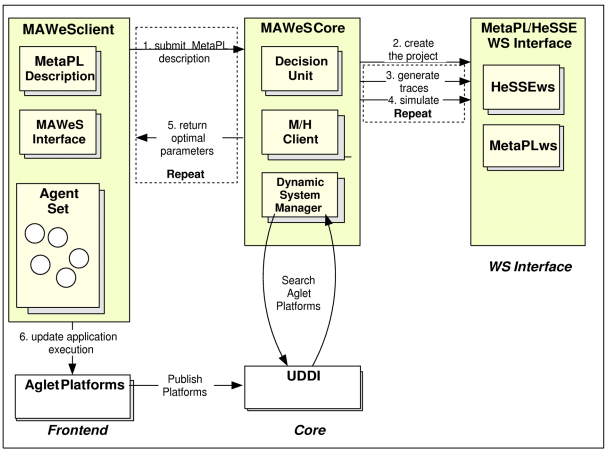

Fig. 1. The MAWeS Extended Framework for Mobile Agents

3.3 MetaPL Extension

The newly developed MetaPL extension defines a set of new MetaPL elements:

`Agent` this element replaces the standard MetaPL `Task` element and contains all the agent code. `name` and `id` attributes identify the agent.

`Create` this element represents an operation of agent creation. It describes the case in which an agent invokes the platform in order to create a new agent locally. The attribute `agent` contains the name of the type of agent created.

`Clone` this element represents an operation of cloning by an agent; it describes the case in which an agent invokes the platform in order to duplicate itself.

`Dispose` this element represents an operation of disposal by an agent (or an application). It describes an invocation to the platform in order to destroy an agent. The attribute `agentID` contains the identifier of the agent to be disposed.

`Activate/Deactivate` these elements represent an operation of activation/deactivation by an agent, an invocation to the platform in order to pause or awake an agent. The attribute `agentID` contains the identifier of the agent affected by the instruction.

`handleMessage` this element represents the activation of the message handling mechanism inside the agent. Message passing adopts the standard message passing MetaPL extension.

`Platform` this element, containedin the mapping section, represents an available platform. It supports the `platformID` attribute, which identifies the platform.

`AgentInstance` this element, contained in the mapping section, represents an available agent. It supports the `agentID` attribute, which uniquely identifies the agent and a `StartPlatformID`, which describes the platform on which it is created at startup.

All the elements support an optional `PlatformID` attribute, which reports the identifier of the target mobile agent platform. Figure 2 shows an example of MetaPL description. It contains the description of an agent A which creates an agent B and migrates on a new platform. Agent B just migrates on a new platform.

3.4 HeSSE Library

HeSSE is a complex simulation environment, easily extensible due to the adoption of a component-based approach. It is out of the scope of this paper to give

```
<MetaPL>
   <Code>
    <Agent name="A">
        <Create name="B" /> <Migrate aglet="test" platformID="2" />
    </Agent>
    <Agent name="B"> <Migrate aglet="test" platformID="2" /> </Agent>
  </Code>
  <Mapping>
    <Platform platformID="1"/> <Platform platformID="2"/>
    <AgentInstance name="A" platformID="1"/>
    <AgentInstance name="B" platformID="1"/>
  </Mapping>
</MetaPL>
```

Fig. 2. An Example of Mobile Agent MetaPL Description

a detailed description of the simulation models adopted by the simulator and of the details of the models developed to simulate the agents. This section will give an overview of the newly developed components and of their behavior and usage. In order to simulate the Aglets mobile agents platform, we developed a library that extends the features of an existing Message Passing library. The newly developed library offers the following components:

Aglet_Daemon. It is the Aglets daemon, and corresponds to the mobile agent platform. This component offers to the simulation environment the service to control the agents.

Aglet_Data. This component is used only to maintain common information among multiple platforms.

Aglet. It reproduces the behavior of a single agent. It is fed with a trace file, whose format is the same of the "original" HeSSE Message Passing trace files, extended with constructs to take into account the typical agent operations (create, clone, dispose, migrate, activate, deactivate).

4 An Example of Tool Execution

As case study, we propose a mobile agent application able to solve the problem of gravitational interaction between n different bodies (NBody problem) [14]. The sequential algorithm that solves the well-known N-body problem computes, during a fixed time interval, the positions of N bodies moving under their mutual attraction. The program repeats, in steps of a fixed time interval, the construction of an octal tree (octree) whose nodes represents groups of nearby bodies, the computation of the forces acting on each particle through a visit of the octree, the update of position and speed of the N particles. Figure 3 describes the behavior of the application. Note that the application involves a set of different agents. The main agent, `master`, waits for a message from the available workers, in order to coordinate them. Each worker, once created, signals the master and starts computations. The MAWeS Client contains the MetaPL descriptions, and at application startup, it sends them to the MAWeS Core. In this case we does not take into account any application specific parameter, so the tool has only to get the list of available platforms, and decide how many agents to start and where. The list of available platforms is stored in an UDDI register together with their performance indexes useful for simulation. The MAWeS core, so, generates a new set of mapping sections, as shown in Figure 3 (mapping section) composed of the list of available platforms, the Number of Agents to be created and the list of Agent Instances. The tool generates a different mapping section for each different configuration to be tested. By default, it generates a number of mapping sections that is two times the number of available platforms, with an increasing number of agents of the worker type. The resulting document is given to the MAWeS component, which performs performance evaluations (MH-Client), simulating the configuration and returning the predicted response time. The MAWeS core returns to the client the mapping section, together with a

```
<Code>
 <Agent name="master" >
    <Loop iteration="Nstep" > <Block>
        <Loop iteration="NAgentstep" > <Block>
            <Receive kind="ready" />
        </Block></Loop>
        <Multicast kind="go" /> <Multicast kind="subtree"/>
        <Multicast kind="subforces"/> <Multicast kind="posANDvel"/>
    </Block></Loop>
 </Agent>
 <Agent name="Worker">
   <CodeBlock region="Initialize" />
    <Loop iteration="Nstep" > <Block>
       <Send to="master" kind="ready" /> <Receive kind="go" />
       <Codeblock region="BuildTree" />
       <Multicast kind="subtree" /> <Receive kind="subtree" />
       <Codeblock region="Compute" />
       <Multicast kind="subforces"/> <Receive kind="multicast" />
       <Codeblock region="Update" />
       <Multicast kind="posANDvel"/> <Receive kind="posANDvel" />
    </Block></Loop>
  </Agent>
</Code>
 <Mapping>
    <Platform platformID="1"/> <Platform platformID="2"/>
    <NumberOfAgents value="3" variable="NAgent"/>
    <AgentInstance name="master" platformID="1"/>
    <AgentInstance name="worker" platformID="1"/>
    <AgentInstance name="worker" platformID="2"/>
 </Mapping>
```

Fig. 3. NBODY with mobile Agent MetaPL Description

link to the available platforms. Then the MAWeS client starts the agent on the chosen platforms.

5 Conclusions and Future Work

In this paper we have proposed an extension to the MAWeS framework for building self-optimizing mobile agent applications. We have shown its new architecture, and the main extension to the description language MetaPL and to the HeSSE simulator. Moreover, the Aglet mobile agents platform has been extended to turn it into a web application. We have shown how the framework can work on a simple example. As this paper presents only the tool architecture and its main features, a next step for our research will be a detailed performance analysis of the approach, pointing out the conditions in which the tool can be useful.

References

1. Zhang, Z.R., Luo, S.W.: Constructing grid system with mobile multiagent. In: Int. Conf. on Machine Learning and Cybernetics, vol. 4, pp. 2101–2105 (November 2003)
2. Tveit, A.: jfipa - an architecture for agent-based grid computing. In: AISB'02 Convention, Symposium on AI and Grid Computing (2001)
3. Aversa, R., Martino, B.D., Mazzocca, N., Rak, M., Venticinque, S.: Mobile agents approach the grid. In: Martino, B.D.M., et al. (eds.) Engineering the Grid: status and perspective, American Scientific Publishers (2006)
4. Greenwood, D., Calisti, M.: Engineering web service - agent integration. In: IEEE Conference of Systems, Man and Cybernetics, The Hague, IEEE Computer Society Press, Los Alamitos (2004)
5. Lyell, M., Rosen, L., Casagni-Simkins, M., Norris, D.: On software agents and web services: Usage and design concepts and issues. In: 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia (July 2003)
6. Birman, K.P., van Renesse, R., Vogels, W.: Adding high availability and autonomic behavior to web services. In: Proc. of 26th Int. Conf. on Soft. Eng., Edinburgh, UK, pp. 17–26. IEEE Computer Society Press, Los Alamitos (2004)
7. IBM Corp.: An architectural blueprint for autonomic computing. IBM Corp., USA (October 2004)
8. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
9. Zhang, Y., Liu, A., Qu, W.: Software architecture design of an autonomic system. In: Proc of 5th Australasian Workshop on Software and System Architectures, Melbourne, Australia, pp. 5–11 (April 2004)
10. Mancini, E., Rak, M., Torella, R., Villano, U.: Predictive autonomicity of web services in the MAWeS framework. J. of Computer Science 2(6), 513–520 (2006)
11. Mancini, E., Rak, M., Villano, U.: Autonomic web service development with MAWeS. In: Proc. of 20th Int. Conf. AINA06, Austria, pp. 504–508 (April 2006)
12. Mazzocca, N., Rak, M., Villano, U.: MetaPL a notation system for parallel program description and performance analysis. In: Malyshkin, V. (ed.) PaCT 2001. LNCS, vol. 2127, pp. 80–93. Springer, Heidelberg (2001)
13. Mancini, E., Mazzocca, N., Rak, M., Torella, R., Villano, U.: Performance-driven development of a web services application using MetaPL/HeSSE. In: Proc. of 13th Euromicro Conf. on Parallel, Distributed and Network-based Processing, Lugano, Switzerland (February 2005)
14. Grama, A., Kumar, V., Sameh, A.: Scalable parallel formulations of the barnes-hut method for n-body simulations. Parallel Computing 24, 797–822 (1998)
15. Lange, D., Oshima, M.: Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley, Reading (1998)
16. Milojicic, D., et al.: Masif: The omg mobile agent system interoperability facility. In: Rothermel, K., Hohl, F. (eds.) MA 1998. LNCS, vol. 1477, pp. 50–67. Springer, Heidelberg (1998)