JAAF-S: A FRAMEWORK TO IMPLEMENT AUTONOMIC AGENTS ABLE TO DEAL WITH WEB SERVICES

Baldoino F. dos S. Neto, Andrew D. da Costa, Carlos J. P. de Lucena
PUC-Rio, Computer Science Department, LES, Rio de Janeiro, Brazil

Viviane T. da Silva
Universidade Federal Fluminense, Computer Science Department, Niterói, Brazil

Manoel T. de A. Netto
PUC-Rio, Computer Science Department, LES, Rio de Janeiro, Brazil

Keywords: Agent, Self-adaptation, Semantic web service and Framework.

Abstract: Due to the widespread interest and deployment of web services and service-oriented architectures in industry, it is necessary to develop systems able to, at run-time, discover, reason and select services. Considering that agents present properties like reasoning, autonomy, pro-activity and self-adaptation, the multi-agent system is a paradigm that fits these concerns. Agents can be used to autonomously and pro-actively discover services, decide about the most appropriate service and adapt themselves if they face a problem while using the selected service. In this paper we focus on a framework (Java self-Adaptive Agent Framework for Service – JAAF-S) to implement self-adaptive agents able to adapt themselves while searching and using web services. The framework also provides support to three main agent-related properties: autonomy, pro-activity and reasoning. JAAF-S extends the JADE framework that already gives support to autonomy and pro-active agents, provides reasoning methods based on rules, cases and genetic algorithms as well as mechanisms to discover and select web services.

1 INTRODUCTION

As mentioned in (Huns, Singh et. al., 2005), service-oriented computing (SOC) has taken hold in business in, for instance, the use of shipping services in e-commerce transactions; the aggregation of hotel, car rental, and airline services; or the book-rating services for libraries, consumers and bookstores. Given the widespread interest and deployment of web services and service-oriented architectures in industry, SOC represents a fundamental shift in the way web applications are developed.

Therefore, it is necessary to provide techniques to discover, invoke, compose and monitor web services. In this context, Semantic Web Service (SWS) (McIlraith, Son,and Zeng 2001) has been pointed as a way to address these issues. SWS provides a way to capture the data and metadata associated with a service together with specifications of its properties and capabilities, the interface of its execution, the prerequisites and consequences of its use.

Although SWS can solve some of the mentioned issues, the complexity of current systems has directed the software engineering community to look for systems able to adjust or adapt their behavior in response to requirement changes. Considering that adaptive agents present properties like: reasoning, learning, autonomy and pro-activity, multi-agent system is a paradigm that fits on these concerns.

In order to support the creation of agents with such capabilities, the Java Autonomic Agent Framework for Services (JAAF-S) was proposed. It provides mechanisms to develop self-adaptive agents by the implementation of different self-adaptation processes based on services. Nonetheless, it provides reasoning methods based on rule-based reasoning (Costa, Lucena et. al., 2008), case-based reasoning (Amodt and Plaza, 1994) and genetic algorithm (Mitchell, 1998), and also service discovery and selection implementation that can be used by the agents. JAAF-S can be instantiated to implement, for instance, biological, ubiquitous computing and autonomic computing systems.

This paper is organized as follow. Section 2 presents some related work. In Section 3 the Java Autonomic Agent Framework for Service is detailed, while Section 4 states a case study by describing the use of the framework to help implementation of agents capable of discovering and selecting Web services and also able to adapt themselves to changes in the requirement. Finally, Section 5 concludes and presents some future work.

2 RELATED WORKS

Rainbow (Garlan, Cheng, Huang, Schmerl and Steenkiste, 2004) uses an abstract architectural model at runtime to monitor the properties of an executing system, to evaluate the model for constraint violation, and — if a problem occurs — to perform global and module adaptations on the running system. However, Rainbow presents two limitations: (i) it uses mechanisms based on a fixed self-adaptation process, based on (Dobson, S., Denazis et. al., 2006), to monitor and adapt the system behavior at runtime, and (ii) it uses only constraints (rules) to verify problems (or violations) beyond utility functions (Petrucci and Loques, 2007) to determine the most appropriate adaptation within a set of applicable ones. Designed to treat these limitations, the JAAF-S enables the elaboration of different self-adaptation processes and provides not only rule-based reasoning and utility functions, but also case-based reasoning (CBR) mechanisms. According to CBR is an efficient way to implement some of the properties of autonomic systems. Beside this, the JAAF-S provides support to discovery services based on OWL-S (Martin et. al., 2009). Such technology presents important properties to perform automatic discovery, selection and composition of services.

The Agent Building and Learning Environment (ABLE) (Bigus, Schlosnagle, Pilgrim et. al., 2002) provides an autonomic management in the form of a multi-agent architecture; that is, each autonomic manager is implemented as an agent or a set of agents, thus allowing different autonomic tasks to be separated and encapsulated into different agents. Although ABLE provides different reasoning mechanisms, it does not have the intention of providing support to the elaboration of different self-adaptation processes.

The work in (Poggi, Tomaiuolo and Turci, 2007) proposes a framework with the aim of supporting agent-based service oriented architecture. The peculiar characteristic and strength of this work is the integration of the agent technology with web services, workflow, rule engine and semantic web. Similar to this, the JAAF-S works with web services, rule engine and semantic web. But in addition our framework provides case-based reasoning and different mechanisms to discover and select web services.

3 JAAF-S

JAAF-S is a framework implemented by the use of software agents and extends JADE (Bellifemine, Caire, Trucco, Rimassa, Mungenast, 2007), a FIPA-compliant framework to implement multi-agent systems (MAS) developed in Java, in order to represent four concepts: (i) agents that perform self-adaptation, (ii) plans executed by agents representing self-adaptation processes (or control-loops), (iii) activities that are the steps of such processes, and (iv) techniques that can be used to discover and select services.

More details about the JAAF-S implementation can be seen in the class diagram depicted in Figure 1, which illustrates the main JAAF-S classes. The self-adaptive agents are represented by the *AdaptationAgent* class and the self-adaptation process by the *PlanAdaptation* class. The *PlanAdaptation* class extends the JADE *FSMBehaviour class* that provides support to the implementation of finite automata composed of activities (or behaviors) represented by the *Behaviour* class. Therefore, in order to implement an autonomic agent it is necessary to perform three tasks: (i) create an agent by extending the *AdaptationAgent* class, (ii) create new activities or instantiate some default provided by the framework, and (iii) create a self-adaptation process by extending the *PlanAdaptation* class.

In order to work with services it is important to instantiate *MatchingSelection* and *SelectionStrategy* classes to give support to the discovery and selection of services.

JAAF-S already provides a self-adaptation process, mentioned on (Dobson, Denazis et. al., 2006), represented by the *ControlLoop* class that is composed of four activities: Collect, Analyze, Decision and Effector, quickly introduced below. Due to space limitations, we will detail in the

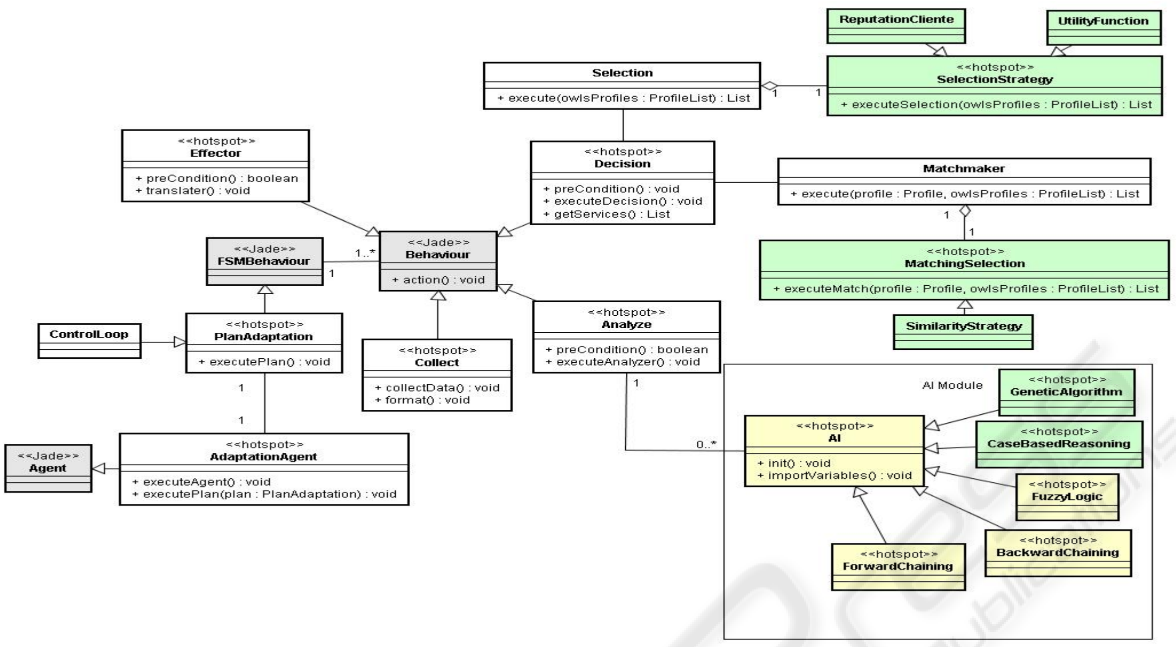
ReputationCliente
UtilityFunction
Selection
+ execute(owlsProfiles : ProfileList) : List
<<hotspot>> SelectionStrategy
+ executeSelection(owlsProfiles : ProfileList) : List
<<hotspot>> Effector
+ preCondition() : boolean
+ translater() : void
<<hotspot>> Decision
+ preCondition() : void
+ executeDecision() : void
+ getServices() : List
Matchmaker
+ execute(profile : Profile, owlsProfiles : ProfileList) : List
<<Jade>> FSMBehaviour
<<Jade>> Behaviour
+ action() : void
<<hotspot>> MatchingSelection
+ executeMatch(profile : Profile, owlsProfiles : ProfileList) : List
SimilarityStrategy
ControlLoop
<<hotspot>> PlanAdaptation
+ executePlan() : void
<<hotspot>> Collect
+ collectData() : void
+ format() : void
<<hotspot>> Analyze
+ preCondition() : boolean
+ executeAnalyzer() : void
AI Module
<<hotspot>> GeneticAlgorithm
<<hotspot>> CaseBasedReasoning
<<hotspot>> AI
+ init() : void
+ importVariables() : void
<<hotspot>> FuzzyLogic
<<hotspot>> BackwardChaining
<<hotspot>> ForwardChaining
<<Jade>> Agent
<<hotspot>> AdaptationAgent
+ executeAgent() : void
+ executePlan(plan : PlanAdaptation) : void
1
1..*
0..*

Figure 1: JAAF-S Class Diagram.

following sub-sections only the two most important ones: analyze and decision.

Collect: It is responsible for receiving, filtering and formatting the data provided by the sensor. Therefore, when the sensor informs which data are available, the agent that has received this information should filter and format them in order to be manipulated by others activities of the control loop.

Analyze: The analyze activity is responsible for providing methods to analyze the data collected in the previous activity in order to detect problems (or necessities) and suggest new solutions in OWL-S description. Such descriptions provide data that are useful in defining which services can be used in different situations. Moreover, the framework gives support to three reasoning mechanisms: rules, cases and genetic algorithms. Details about this activity are presented in Section 3.1.

Decision: This activity provides support to automatically discover and select web services that address the target problem (or requirement). Initially, it receives the service descriptions suggested by the previous activity and applies matchmaking algorithms (Srinivasan, Paolucci, Sycara, 2006) between such descriptions and a semantic web service. It aims to meet services that can be used to solve some request or problem. Subsequently, the activity uses selection techniques (reputation, utility function, etc.) to choose the best services.

Effector: This activity receives the selected services from the *Decision* activity, making them understandable (for instance, by translating them) to the application layer. The control-loop can be executed again if any self-adaptation is necessary.

3.1 Analyze Activity

The analyze activity uses the reasoning mechanisms provided by the AI module while analyzing the data it collects. In order to provide a rule-based mechanism, we use the AI module provided by DRPMAS (Costa, Lucena et. al., 2008), which is composed of three algorithms: fuzzy logic, backward chaining and forward chaining. Therefore, in order to develop these algorithms, the ForwardChaining, BackwardChaining and FuzzyLogic classes should be instantiated to implement forward chaining, backward chaining and fuzzy logic reasoning, respectively. The work of the instantiation will be only to define the rules applied.

The second mechanism, case-based reasoning, uses past experiences of similar problems to solve the current problem. As past experiences can be useful for performing self-adaptations, we have incorporated this reasoning mechanism in the framework. We provide an infrastructure (case-base), which enables the storing of past experiences, and algorithm implementations that makes it possible to discover similar cases with the current problem. It is necessary to instantiate the

CaseBasedReasoning class and select the similarity algorithm that will be used in order to implement CBR.

The third mechanism, genetic algorithm, provided by the framework, allows finding exact or approximate solutions to optimization and search problems. Such an approach uses techniques inspired in evolutionary biology, such as inheritance, mutation, selection and crossover (also called recombination). The JAAF-S provides the infrastructure necessary to perform such techniques. In order to apply genetic algorithms it is necessary to implement the GeneticAlgorithm class and elaborate a fitness function (Mitchell, 1998).

3.2 Decision Activity

Service discovery and selection mechanisms have played an important role in service oriented architecture and JAAF-S provides such features.

In order to represent the discovery mechanism, the SimilarityStrategy class implements the service discovery matching algorithm shown in (Srinivasan,, Paolucci, Sycara, 2006). Furthermore, the framework also considers two attributes of the OWL-S Profile: text description and service parameter.

Text description briefly describes the service, summarizes which service is offered, describes what is necessary for the service to work and informs any additional necessary information. The attribute parameter represents a list of properties which informs the quality being provided by the service. This attribute is composed of two sub-attributes. The first sub-attribute is the *serviceParameterName*, which is the name of the current parameter, while that the second sub-attribute, *sParameter*, represents the value of the parameter.

In order to provide selection, two mechanisms are offered by the JAAF-S: reputation and utility function. Reputation of services is represented by the *Reputation* class that is able to identify which services are good or bad based on consumer feedback about the quality of the service. While that utility functions, represented by the *UtilityFunction* class, it performs the selection of services based on the quantitative level of desirability of each service. To do so, it takes a set of OWL-S Profiles as input, verifies the quantitative level of desirability of each attribute in the OWL-S Profiles and outputs the Profiles with the highest level of desirability.

3.3 Hot-spots and Frozen-spots

Since JAAF-S extends JADE, the JADE kernel is also the kernel of JAAF-S and the hot-spots of the JADE are the hot-spots of JAAF-S. For instance, the process used by agents to communicate, and the agents' identifiers are examples of JAAF-S hot-spots inherited from JADE.

The hot-spots specifically defined in JAAF-S are:

Agent (*AdaptationAgent* Class). By extending this class and implementing the *executedPlan* method, it is possible to define different algorithms to execute the plans of an agent.

Plan of self-adaptation (*PlanAdaptation* Class). It is possible to define new control-loops (or plans) and the sequence to execute the activities of the control loops. JAAF-S already provides a default control-loop implemented in the *ControlLoop* class.

Activities (*Behaviour* Class). It is possible to define new activities to be called by the control loops by extending the *Behaviour* class. JAAF-S already offers four activities (Collect, Analyze, Decision and Effector).

Intelligent Algorithm Module. JAAF-S offers three kinds of algorithms: rule-based reasoning (forward chaining, backward chaining and fuzzy logic), case-based reasoning and genetic algorithm. These types of algorithms can be used at any point of the system to help with the self-adaptation.

The JAAF-S already provides one matching algorithm based on (Srinivasan, Paolucci, Sycara, 2006) and two selection techniques one based on reputation (Koogan and Houaiss, 1995) and other in utility function (Petrucci and Loques, 2007). However, we use the strategy pattern (Gamma, Helm, Johnson and Vlissides, 1994) in order to enable the addition of others algorithms and techniques in the framework.

4 CASE STUDY: SELF-ADAPTIVE PERSONAL WEB PAGE

In this section we describe a service-oriented multi-agent system that applies self-adaptation in order to satisfy customers' needs. Such needs are represented by a user requirement ontology that is used to discover and select the most adequate service for a given situation.

4.1 Main Idea

The implemented system provides three kinds of services: hotel reservation, buy airline tickets and car rental. The application begins with a customer accessing a Web page in order to specify which service he/she desires. Depending on the chosen service, different information is requested. From the data provided by the customer the system will discover the most appropriate service to attend him/her.

The agent responsible for receiving the information is called Manager agent. It first identifies the context of the desired service (hotel reservation, buy airline tickets or car rental) and then tries to find a Service agent able to provide the desired service. Service agents are chosen according to their reputations. Reputations are defined based on feedback provided by the customers about the quality of the services they have used.

Note that different *Service* agents can be used to provide the same type of service and each agent is responsible for managing a set of semantic web services. Therefore, when a request is performed the chosen agent will select which web service is the most adequate in order to attend the customer. If the selected web service is offline, the agent adapts to meet another service. Otherwise, the *Manager* agent is the one that should adapt in order to choose another *Service* agent that can provide the requested service.

At the end, when a service is selected from a *Service* agent, the customer executes it. After having executed the service, the customer informs the *Manager* agent whether he/she liked or not the service provided. After the *Manager* agent receives the feedback, it updates the reputation of the executed service and forwards the feedback to the *Service* agent.

In order to clarify the tasks of the agents provided by the application, each one is explained in detail in following subsections.

4.2 Manager Agent

As mentioned previously, the *Manager* agent receives the data provided by the customer, identifies the context of the request and meets the *Service* agent that can provide the desired service. When a service fails while it is being provided, the *Manager* agent receives the feedback from the customer to search for another service. The *Manager* agent uses the default control-loops provided by the framework to perform these activities.

In the collect activity the agent receives the information provided by the user. Next, in the analyze activity the agent uses rule-based reasoning to identify the context and send it to the decision activity. This activity selects the *Service* agent that is able to provide a service in the context required and that has the service with best reputation. If a chosen *Service* agent had a previous failure while providing a service, its reputation is changed and compared with the reputation of the other *Service* agents that can attend the same request. Therefore, the *Manager* agent adapts itself to select another *Service* agent. Finally, in the effector activity the *Manager* agent forwards the data provided by the user to the chosen *Service* agent.

4.3 Service Agent

Each *Service* agent is responsible for selecting the web service that best satisfies the customer's necessities. If the service selected cannot be executed, the *Service* agent should perform a self-adaptation to select another service. In order to represent such self-adaptation, the agent applies a control-loop composed of five steps: Collect, Analyze, Decision, Test and Effector. Note that different from the default control loop, the framework JAAF was used to define another control-loop composed of a new activity, the Test activity.

In the collect activity the agent receives the information provided by the *Manager* agent. Next, in the analyze activity the agent uses case-based reasoning to discover which services could be used. The reasoner takes into account similar situations where these services were used to select the service. The services are then provided to the decision activity by using the OWL-S ontology to describe the service, by creating the profile of the desired service

The decision activity applies matchmaking algorithms between the OWL-S profile provided by the analyze activity and the OWL-S profiles of the available services also provided by the *Service* agent. In the case of matching, the reputation of the services is used to select the one that is most reliable.

After selecting a service, it is tested in the tester activity to assure that it is online. In the case the service is offline, the information is stored in a "Service with Problem" database and the decision activity is executed again. However, when the decision activity meets an online service, the effector activity is executed. It is responsible for communicating the chosen service to the application.

5 CONCLUSIONS

This paper proposes a framework that provides support to discover, reason and select web services by the creation of self-adaptive agents, i.e., agents able to adapt their behavior due to problems that may occur while trying to access a service. Nonetheless, it also provides reasoning methods based on rules and cases algorithms that can be used by the agents.

The applicability of such a framework can be verified by the case study presented in Section 4. The two different agents (*Manager* and *Service* agent) illustrated in that section use different self-adaptation processes while manipulating services. One of them uses the adaptation process proposed as default by the framework while the other instantiates the framework by implementing another activity and defining a different self-adaptation process in order to test if the service is online before providing the service to the user. Working together these agents are able to keep the customer satisfied with the services provided by the system.

We are in the process of defining new self-adaptation control-loops and mechanisms able to handle the control-loops. It is also our intention to extend JAAF-S in order to provide a framework not only for self-adaptation but also for self-organization in a multi-agent environment. This framework would guide the development of organizations inspired by biological systems.

REFERENCES

Amodt, A. and Plaza, E., 1994, Case-based reasoning: Foundational issues, methodological variations, and system approaches. In AI Communications, volume 7:1, pages 39–59.

Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., 2007, Jade Programmer's Guide.

Bigus, J. P.; Schlosnagle, D. A., Pilgrim, J. R.; et. al. 2002, .ABLE: A toolkit for building multiagent autonomic systems. IBM Syst. J. 41, 3, 350–371.

Costa, A., Lucena, C. J. P.; Silva, V., Cowan, D.; Alencar, P., A Hybrid Diagnostic-Recommendation System for Agent Execution in Multi-Agent Systems, ICSOFT 2008 – 3rd International Conference on Software and Data Technologies, Porto, Portugal, July 2008.

Dobson, S., Denazis, S., Fernández, A., Gaiti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., and Zambonelli, F., 2006, A survey of autonomic communications. *ACM Transactions Autonomous Adaptive Systems (TAAS)*, 1(2):223{259}.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994, Design Patterns : Elements of Reusable Object-Oriented Software, Addison-Wesley Professional Computing Series.

Garlan D., Cheng, S., Huang, A., Schmerl B. and Steenkiste, P., 2004, Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure. In IEEE Computer, Vol. 37(10).

Huns, M., Singh, M., et. al., 2005. Research Directions for Service-Oriented Multiagent Systems. IEEE Internet Computing.

Koogan, A.,Houaiss, 1995, A.: Encyclopedia and Dictionary. Delta Publisher.

Martin, D., et. Al. ,OWL-S: Semantic Markup for Web Services, Last access at April 2009 , http://www.w3.org/Submission/OWL-S/.

McILraith, S., Son, T. and Zeng, H., 2001. Semantic Web Services, IEEE Intelligent System.

Mitchell, M., 1998,An Introduction to Genetic Algorithms (Complex Adaptive Systems), The MIT Press.

Petrucci, V. and Loques, O. 2007, Suporte a adaptação de aplicações usando funções de utilidade. In 1st Workshop on Pervasive and Ubiquitous Computing, WPUC 2007, SBAC-PAD 2007.

Poggi, A., Tomaiuolo, M. and Turci, P. 2007, An Agent-Based Service Oriented Architecture, WOA.

Srinivasan, N., Paolucci, M., Sycara, K., 2006, Semantic Web Service Discovery in the OWL-S IDE, Proccedings of the 39th Hawaii International Conference on System Sciences.