Evolving Agent Architecture for Data Collection

Ali Esserhir
LACL
Université Pars Est Créteil (UPEC)
Créteil, France
ali.esserhir@lacl.fr

*Abstract***— The use of mobile agents has shown that agent migration is a solution for the change management. However, the use of such architecture is often quite complex, depending on the design approach. In this work, we propose to define a set of Rest (REpresentational State Transfer) interfaces for each element of our mobile agent architecture. Moreover, we have built a toolchain based on the small set of tools that allows the designer to easily update the facades of each element. Then, we apply our strategy for a validation of concept. A study is built about using mobile agents for data collection in a Rest architecture. We analyze our results and how we treat any changes in the design of our architecture.**

Keywords-software architecture; mobile agent; REST architecture; RESTful system.

I. Introduction

Nowadays, computer science projects have to deal with software architecture on one hand and agile methodology on the other side. By the end, we obtain a context where changes in architecture have a heavy impact on the lifecycle of the project. This topic is interesting since architecture and agile seem to have some conflicting forces at work. The definition of software architecture is a first challenge and the experts provide different visions about this concept. Our current definition is quite common, but explicit: software architecture is the collection of decisions affecting the system's quality attributes, which have global effects and are hardest to change.

As a precision, software architecture provides the frame within which the design is built by the developers. Because a component diagram often describes an architecture, the component definition is essential. In our work, it is an autonomous part of our software, which exposes a public interface and needs outside interface for working. B. Wallace introduces the idea that a component definition cannot exist without a framework selection [1]. For Rainer Niekamp, a component is first a reusable unit of software, which is able to communicate with other components via interfaces [2]. Raphael Gfeller considers a component as an administrable entity into a software project with dependencies and features [3]. All these definitions depict different facets of what is a component. In addition, we consider that, when a software architecture evolves, it is crucial that all its components are administrable and exchangeable through well-defined interfaces under the control of a selection of specifications or frameworks.

On the client side, the invocation way of a component must be as simple as a local method call. R. Fielding has worked on this problem and his REST philosophy is welcomed in many projects, regardless of the programming language is used. Such simplicity had already been applied which the SOA (Service-Oriented Architecture) architecture with the use of ESB (Enterprise Service Bus) framework like Apache ServiceMix [4]. This framework requires the use of VETO pattern for the treatment of the client requests.

Moreover, to manage the complexity of evolving a software-distributed system, its architecture description has to be linked through an accurate and traceable way to its implementation. Too often, only the software architect is able to maintain the software architecture and no one knows the keys for updating the architecture. The software architecture deals with multiple views of a system, including both its functional and nonfunctional facets. A structural approach looks at the system as a set of components that interact via interfaces. Complexity is mastered by means of hierarchical decomposition; a component can be composed of subcomponents with the hierarchy's leaf components representing coded functionality. As the Architecture Description Languages (ADL) group describes, this research community has proposed numerous ADL versions [5].

Any software-distributed system is constantly subject to software changes, usually driven by external constraints from the runtime environment over which the developers have no control at all. These constraints may be as diverse and unpredictable as technological changes, enhanced user organizational structures or business processes, new legislation, or changes in resources. To cope with any of these issues, all software artifacts produced and used by the software-distributed system have to evolve. Depending on the software artifacts' type, the impact and rate of change may differ. Evolving a software architecture by modifying its description to accommodate change requests faces numerous research challenges. In particular, the evolution of an architectural description should typically preserve its purpose and criticality concerns. However, often, the clients express the changes and it is not a set of bug fixes. In addition, the checks are on the preservation of the existing architecture.

Our work is about the lifecycle of the architectural changes and, more precisely, in the context of Resource Oriented Architecture (ROA) architecture. This acronym is created by the Django framework [6] and is now used for designing a software architecture based on REST concepts. It requires that all components are accessible through a REST API (Application Programming Interface).

This document is structured into several sections. Section 1 is around the topic of the evolving architecture for collecting data. Section 2 is about the related works, which are closed to our topic. Section 3 is on the use of mobile agent for the management of changes. In Section 4, we explain how our four-step strategy pilots the development process of the changes of a component. In Section 5, we apply this process for a case study on data collection over a network. In Section 6, we detail our results and analyze the reasons of the success of such approach. Finally, in Section 7, we summarize the main results of our contribution.

II. RELATED WORK

The software architecture field is often considered too abstract or too technical. Software architecture includes the global control structures, protocols for communication, synchronization, physical distribution, scaling and performance, and selection among design alternatives.

A. *Agent architecture*

One of the alternatives in the design of the software architecture is how to access remote resources or make calls to remote objects; or how to send the program code over the network. Four different paradigms have been identified: Client-Server, Remote Evaluation, Code on Demand, Mobile Agents. In this case, the code, including its execution state and some of its resources, is sent to a remote site where it executes. It can continue to another site if needed.

In order to make a mobile agent system work, it is not enough to build the agents themselves. A program at each site is also needed to handle the incoming agents and send out agents. This program is often called an agent factory. The agent factory can be built differently depending on which type of agent system is needed. The generic mobile agent system can have a range of varying components. It needs a communication module that handles incoming and outgoing agents, as well as the messaging between non-local agents. It has a repository that performs authentication, sets priorities and queues up agents for later execution.

One of the first ideas was to use mobile agents for searching through the Internet for the lowest prices of products and services. While the idea was good in theory, few companies wanted other people's agents in their computers, not only for security reasons, but probably for marketing reasons, too. They wanted people to come to their place and keep them there. Another domain is remote control, where applications are intended to control or reprogram remote computers, devices or unmanned vehicles by sending agents with new commands or program updates. These updates can be done very quickly, making it very good for applying security patches. Agents can also be used for monitoring devices and reporting back when status changes or problems occur, or can even be used for intrusion detection and active defense of computer systems [7]. A similar example of remote control is an abstraction called Mobile Streams [14]. Using that system, a distributed, event-driven application can be scripted from a single point of control and dynamically extended and reconfigured during execution.

Another application area for mobile agents or simply mobile code is to dynamically program the networks themselves in order to make them more flexible, customized and give them higher performance. At the lower level, the network devices like routers and switches can be remotely programmed [8] by sending mobile code which can change the topology and routing [9]. Instead of being passive, the networks become more active by taking a certain part in the computations or filtering the data [10].

Instead of doing all the processing and computations on a central computer, they can be distributed to several computers in a network. It is somewhat similar to process migration, but the difference is that processes usually migrate within a tightly coupled unit with several synchronized processors. The code is distributed to the remote computer to do the filtering and processing locally. This often reduces the network traffic and is a way to balance the load of computers with different capacities. It can also be more redundant when several computers do the same processing and the results can be compared. The agent migration allows the agent hosts for receiving updates of their business code without any service stop and the clients cannot observe any interruption of services. What stays difficult is the agent factory interface. Often, it is written in a specific programming language and its use from a given project needs to write new wrapper technical code. Also, developers and architects prefer the use of interoperable API over a standard protocol like http.

B. *REST API definition and restful system*

We have studied many architectures before choosing REST architecture. We have found that this architecture is the most suitable for data collecting by mobile agent. We will see below what is à REST architecture.

Roy Thomas Fielding defined REST in his 2000 PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures". REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

A web service must respect 6 constraints in order to be a Restful system:

- Client-Server: there must be a separation between server and client (separation of concerns)
- Stateless: Client does not conserve any client contest between two requests.
- Cacheable: Clients and intermediaries can cache responses
- Layered system: Requires that this middleware be inserted transparently, so that interaction between a given service and consumer is consistent, regardless of whether the consumer is communicating with a service residing in a middleware layer or a service that represents the ultimate receiver of a message.
- Code on demand (optional): Servers can temporarily extend or customize the functionality of a client by the transfer of executable code.

- Uniform interface: The uniform interface constraint is fundamental to the design of any REST service. The uniform interface simplifies and decouples the architecture, which enables each part to evolve independently. Uniform interface have four constraints:
  - Identification of resources
  - Manipulation of resources through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application

III. Evolving architecture based on mobile agents

We have already used mobile agents in our previous work. Because technologies evolves quickly, the way we implemented such this software concept changes also following the discovery of more suitable frameworks. The main concepts are unchanged like, an agent factory, an agent server, a registry of agents and so on, but the technical aspects are more or less hidden to the developers and the users.

A. What is an evolution in an agent architecture ?

When a mobile agent architecture is deployed, a mobile agent imports a behavior and data from an agent server to an agent host. In addition, this host is enriched at runtime with the incoming agents. In the context of network control, a mobile agent can reconfigure an agent host with the import of updated features such that the address of the mail server, the connection string to a database, the name of the persistence unit.

Another application domain is the data collection where a mobile agent or a set of mobile agents walk through the agent hosts and collect useful data such that the log files, the updates of a NoSQL database, etc. Back to the agent server, the collected data are parsed, and actions are planned depending on their semantics. This means that a main database is updated, or some piece of code is applied on the log file to detect the anomalies at runtime (post analysis).

To summarize, we consider a configured mobile agent as a concrete evolution into a mobile agent system. This evolution will be considered when this mobile agent would have browsed the interested agent hosts.

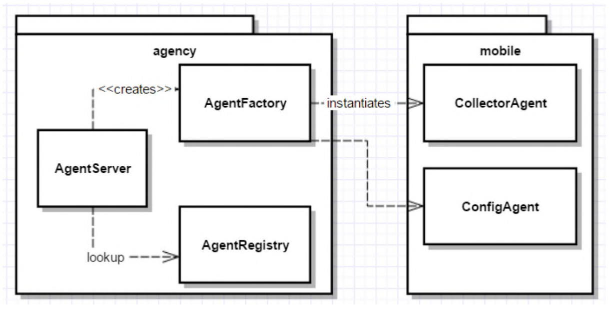


Figure 1. Overview of a mobile agent architecture.

Figure 1 shows the main elements of a mobile agent architecture. The agent server receives all the client requests and depending on the previous demands; it finds a mobile agent, which is already instantiated into the agent registry or it asks the agent factory for creating a new mobile agent. Then, it configures the mobile agent depending on the incoming request. This scenario describes a strategy called AOD. This is useful when the agent hosts know their need or when they know that a part of their knowledge is obsolete. Often, the hosts are not requestor and they do not know that they need of a refresh event.

This second scenario considers another mobile agent strategy called Proactive. It means that a mobile agent has to propagate the updates without any demand from the hosts. In addition, it knows a list of destination and sequentially it updates each host identified by its uniform resource location. Of course, the hosts should have been first configured to accept any mobile agents. We do not consider the security aspect in this document but it appears obvious that this has to be enforced in a professional context. When the mobile agent finishes its mission, then it comes back to the agent server and provides a report about its activities on each host. This involves new actions from the server. It can shut off a host or restart it with a minimal configuration, etc.

We have considered only one mobile agent in action, but we can improve the implementation of the evolution with the launch of several mobile agent concurrently. In that context, the mobile agent needs to exchange messages at runtime. The use of messages allows reducing the migration of agent. In some situation, it allows the validation of a mobile agent activity. For instance, when a mobile agent configures the REST interface of an agent host, then a remote agent can test whether the availability of this interface is ok. To sum up, we note that different types of mobile agent implement an evolution through the browsing of the hosts. Each of them contribute to the satisfaction of the whole mission, this include the validation of the activity and the generation of reports by the end. In both scenarios, the reuse of agent is done with the use of a registry.

B. Self adaptation system

Today, the adaptation of a system is a crucial property for the lifetime of an application. The adaptation is necessary not only because of the aging of a software, but also because the resources are limited in all contexts. In software, we have to consider that the memory is finite, the run time has to be under a threshold required by a product owner and the energy cost has to be also under a given consumption. When a limit is achieved, the system has to adapt its behavior for avoiding a global failure. Aldo, in that context we have already experimented that a mobile agent architecture is a solution. For instance, in an embedded context (mobile phone, tablet, etc.), it is thrifty to move the back end of a mobile application to an application server and keep only the front end with the graphical user interface on the mobile device.

The transfer of data is quite common but in case of mobile agent, the import of code is under security constraints. Therefore, an agent loader receives a byte code stream, converts it into a local agent, and then run it. We consider this operation as an access to a remote resource. We implement this network importation as a message exchange pattern based on request / response schema. The content of the message is the code of the mobile agent and its current state. Moreover, because of the security rules, the vehicle of our message is the http protocol. These technical choices are not too restrictive and allow several message formats depending on the frameworks.

This means that all the elements of our architecture (Figure 1) respect a Facade design pattern where all the remote operations are exposed to their future clients. Although, this involve that new operations could appear at runtime depending on where the mobile agents could be or depending on the scheduling of an evolution. We clearly separate the migration concern form the implementation of the evolution and this stresses the inner structure of a mobile agent. It is a composite based on several parts.

At a first level, a mobile agent implements a contract, which is used at the top abstraction of a proxy pattern (Figure 2). Since the agent proxy and agent behavior both implement AgentResource, it allows the client to treat the

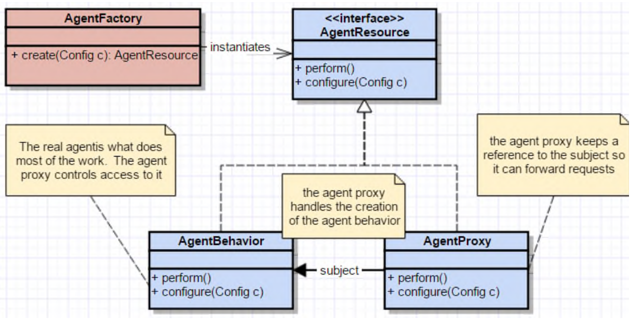

AgentFactory
+ create(Config c): AgentResource
instantiates
<<interface>>
AgentResource
+ perform()
+ configure(Config c)
The real agentis what does most of the work. The agent proxy controls access to it
the agent proxy keeps a reference to the subject so it can forward requests
the agent proxy handles the creation of the agent behavior
AgentBehavior
+ perform()
+ configure(Config c)
subject
AgentProxy
+ perform()
+ configure(Config c)

Figure 2. Mobile agent structure.

proxy like the AgentBehavior. An agent host loads such agent and depending on the implemented strategy, its activity is run by a call of perform method.

IV. Four step lifecycle

We apply an incremental project lifecycle divided into 4 steps: agent analysis, agent design, agent development and agent validation. This lifecycle unit is repeated until the end of a distributed project. In the following sections, we detail the definition of the REST interfaces of all the elements of our distributed system based on mobile agents.

A. *Agent system analysis*

The analysis system means that we define rigorously the facade of each element of our system. It starts with the naming of the elements and their public interface with a signature. A suitable approach consists in the construction of an interaction diagram that describes how a group of objects collaborates in some behavior. Typically, a single use-case has an execution, which is described by this sequence of interaction. The diagrams show a number of example objects and the messages that are passed between these objects within the use-case. It is difficult to write much about interaction diagrams because they are so simple. However, they have weaknesses; the main one is that although they are good at describing behavior: they do not define it. They typically do not show all the iteration and control that is needed to give a computationally complete description.

A first reading of such sequence diagram provides the naming of methods and parameters, with potential types. Jacobson uses pseudo-code in conjunction with sequence charts to provide a more executable model. Others have added various diagrammatic notations to increase the model's usability. Many of these are included in the UML notation. We adopted a standard open software for the construction of such diagrams like the online tool called web sequence diagram. Because a use case can have several distinct executions, we create more than one sequence diagrams. By the end of the analysis step, we create all the façade of the project.

B. *REST API design*

The design of the mobile agent interfaces consists in a deeper reading of the previous sequence diagrams and the selection of a more rigorous language for a more precise description. He has selected the Swagger language for such description. Its goal is to define a standard, language-agnostic interface to REST APIs. It allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation.

Swagger is a formal specification surrounded by a large ecosystem of tools, which includes everything from front-end user interfaces, low-level code libraries and commercial API management solutions. In addition, from a Swagger specification, it is possible to create the service implementations and the client parts.

First, a Swagger specification file allows us to describe an API including:

- General information about the API
- Available paths or resource naming (/resources)
- Available operations on each path http verbs (get /resources)
- Input/output for each operation and message format.

Once written, OpenAPI specification file contains the description of a data model and all the supported operations. It can also be used as:

- source material for documentation
- specification for developers
- partial or complete code generation
- and many other things such as analysis and diagram generation

C. *Skeleton generation and business delegate pattern*

Because code generation means a framework selection, our previous experiments lead us to choose Spring framework for the service implementation. The principle consists in the separation of concern and the use of the Business Delegate pattern. It means that the business code does not belong to the service but it is called from the service implementation. The swagger code generator project, allows the generation of API client libraries (SDK generation), server stubs and documentation automatically given a YAML specification.

Because of the code generation, the developer obtains a Maven project with a pom.xml file. This file descriptor contains all the knowledge of the project, its dependencies, the useful build plugins and the report plugins. Maven requires respecting its own lifecycle for the installation of the project. Some user modifications are necessary into this file descriptor to adapt the code generation to the user platform. Then a run of the project exposes the main uniform resource location to the public. When the application is launched locally to a server, then a possible URL is as follows:

http://localhost:8080/v1

The response is a web page where all the operations are proposed and a documentation page is displayed for all the users. Therefore, a user can browse all the operations and read the textual contract. This page allows him to create basic test request by the use of specific forms.

This step contains also the generation of clients. In that context, it means, the client for each element of our architecture. These clients support the invocation process described in Section 3 about the agent lookup into a registry and the agent creation from the agent factory interface. For the same reason, we selected Spring framework for the client part and based on this result we can build our own agent strategy.

D. *Test and validation based on Exchange pattern*

Validation is determining whether the system complies with the requirements and performs treatments for which it is intended. In addition, it meets the organization's goals and user needs. Of course, this step does not contain a large set of requests sent manually to the services. We use also a framework for the test build. SmartBear is the editor of SOAPUI application that is a standard in the validation domain. This means that from an interface definition, it is able to build template of requests and then it submits them. By the end, it compares an expected result with an actual result.

Instead of spinning up a new browser tab, typing into a slick user interface and clicking buttons, we reached for a tool and thought carefully about data and endpoint paths. When we test an API, we deal with the stuff under the covers and the framework called REST API Testing offers all these concepts. Therefore, we are able to test each REST interface of our architecture and we can create script where the interfaces are composed to validate our strategy of architecture adaptation. We used this approach for the demand of a new data collection to the agent server.

We use also the Swagger Test Templates (STT) module, through either the command line interface or programmatically. It generates a robust, end-to-end testing suite for all a developer's API endpoints defined in their Swagger specification. Then we are able to fill some parts of the code with specific assertions about the business data we

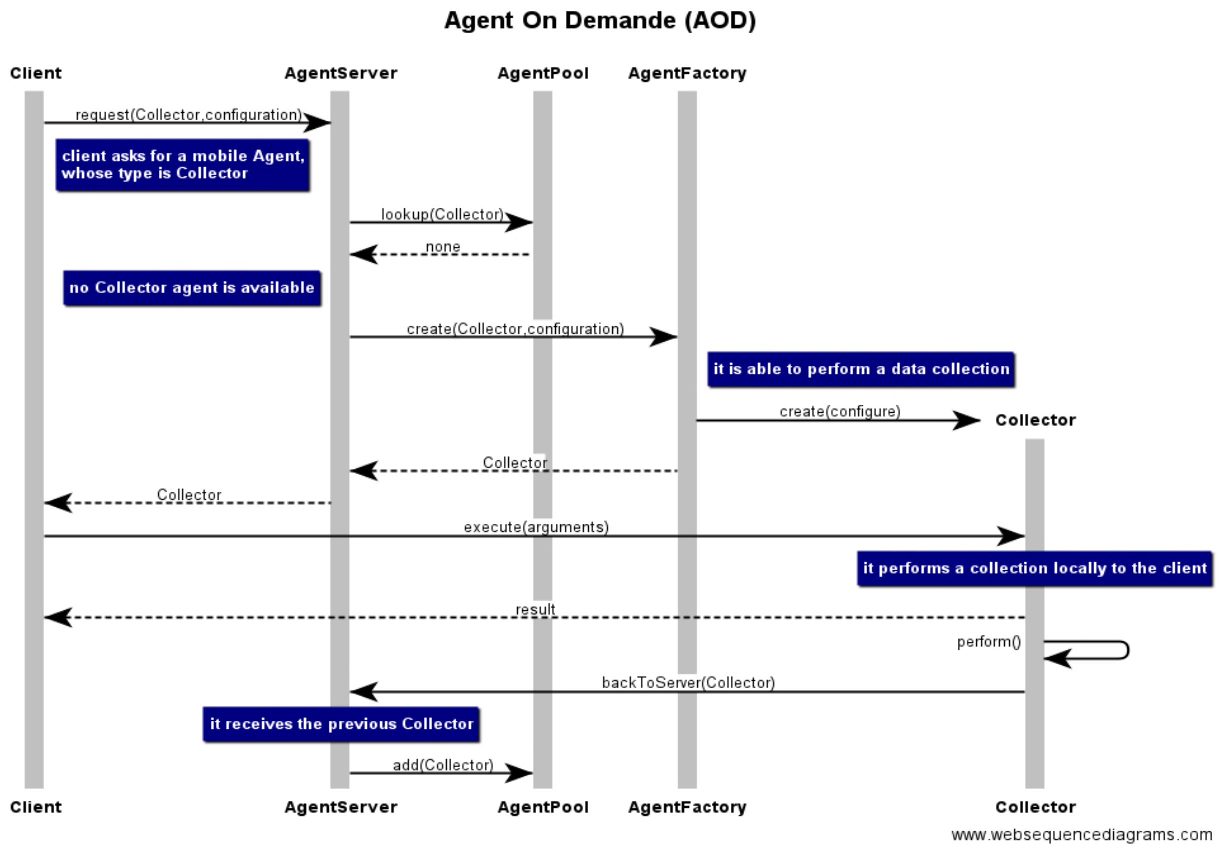
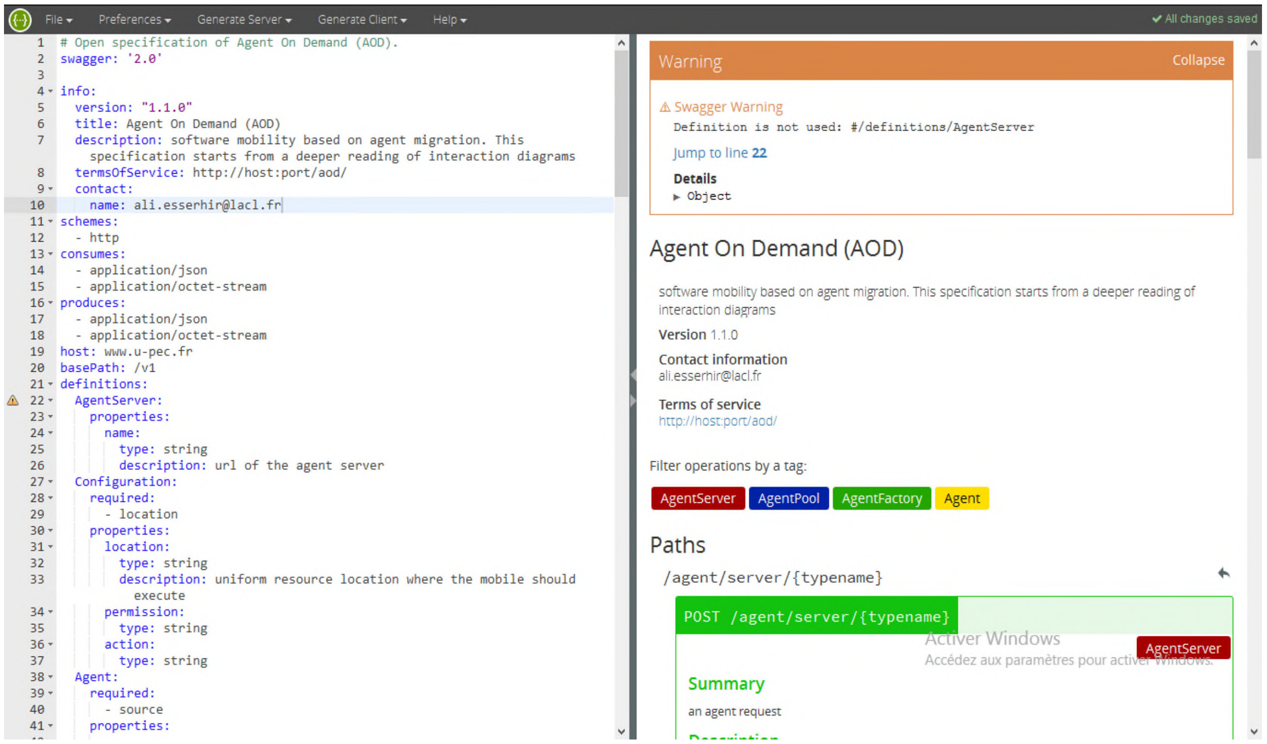

Agent On Demande (AOD)
Client
AgentServer
AgentPool
AgentFactory
request(Collector,configuration)
client asks for a mobile Agent, whose type is Collector
lookup(Collector)
none
no Collector agent is available
create(Collector,configuration)
it is able to perform a data collection
create(configure)
Collector
Collector
Collector
execute(arguments)
it performs a collection locally to the client
result
perform()
backToServer(Collector)
it receives the previous Collector
add(Collector)
Client
AgentServer
AgentPool
AgentFactory
Collector
www.websequencediagrams.com

Figure 3. Sequence Diagram : Agent On Demand



```yaml
# Open specification of Agent On Demand (AOD).
swagger: '2.0'

info:
  version: "1.1.0"
  title: Agent On Demand (AOD)
  description: software mobility based on agent migration. This
    specification starts from a deeper reading of interaction diagrams
  termsOfService: http://host:port/aod/
  contact:
    name: ali.esserhir@lacl.fr
schemes:
  - http
consumes:
  - application/json
  - application/octet-stream
produces:
  - application/json
  - application/octet-stream
host: www.u-pec.fr
basePath: /v1
definitions:
  AgentServer:
    properties:
      name:
        type: string
        description: url of the agent server
  Configuration:
    required:
      - location
    properties:
      location:
        type: string
        description: uniform resource location where the mobile should
          execute
      permission:
        type: string
      action:
        type: string
  Agent:
    required:
      - source
    properties:
```

Warning Collapse

⚠ Swagger Warning
Definition is not used: #/definitions/AgentServer
Jump to line 22
Details
▸ Object

Agent On Demand (AOD)

software mobility based on agent migration. This specification starts from a deeper reading of interaction diagrams

Version 1.1.0

Contact information
ali.esserhir@lacl.fr

Terms of service
http://host:port/aod/

Filter operations by a tag:

AgentServer AgentPool AgentFactory Agent

Paths

/agent/server/{typename}

POST /agent/server/{typename}

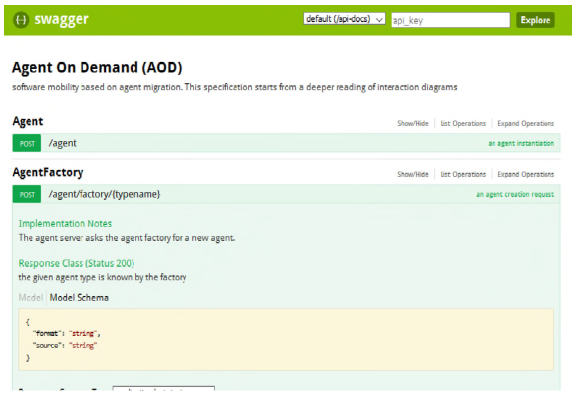
Activer Windows
Accédez aux paramètres pour activer Windows.

AgentServer

Summary
an agent request

Figure 4. Open specification for an agent server

want to send. This saves hours of test writing and enables us to quickly deploy on our server a suitable version of all the REST services.



Agent On Demand (AOD)
software mobility based on agent migration. This specification starts from a deeper reading of interaction diagrams

Agent
POST /agent an agent instantiation

AgentFactory
POST /agent/factory/{typename} an agent creation request

Implementation Notes
The agent server asks the agent factory for a new agent.

Response Class (Status 200)
the given agent type is known by the factory

Model | Model Schema

```
{
  "format": "string",
  "source": "string"
}
```

Figure 5. Agent interface generation

After finishing writing the specification of REST Web services, we generate a server and a client using the tool provided by Swagger. In this way, we can test our REST web services (in Figure 5).

V. Case Study

A. *Agent sequence diagram*

The sequence diagram In Figure 3 describes the way that a client asks an AgentServer for an agent able to collect data.

B. *Open specification for an agent system*

We have used Open specification, in order to specify our REST web services, Swagger tool offer an online "Swagger editor". We have used Yaml language for specifying our API. In Figure 4 we can see on the left side the Yaml code we have written, and on the right side, the documentation generated by Swagger.

C. *Agent interface generation and packaging*

VI. Analysis and Results

Our scenario is a brief narrative description of a system to capture relevant information for computer health and problem management monitoring. For example, practice management systems manage the business of the general practice by recording computer details, managing application servers and database system.

A. Scenario on data collection

The data collection covers a network of computers where log files are saved. They describe the health of application servers and database systems. This data collection starts at a given time and the process is repeated every day. The format of the files is text and the collection principle means to append the file of the same structure into a master file. Behind a file format, there is at least one mobile agent. The number of agents depends on the number of hosts to walk through.

In our test scenario, we consider per computer a JBoss Server as the application server, and a MySQL server. The application server contains many tools, which configured to work together. Several log files are touched by the data collections. The configuration of the database server set features for a verbose mode and the trace of SQL statements with all database events. The locations of the log files are known paths when the installation is done successfully. To sum up, this scenario groups 8 kind of file format and the size of the files depends on the activity of the servers.

Before starting the scenario, all the agent hosts install permissions for accepting the mobile agents of our factory and for executing a file reading operation. The scenario starts with a request to the agent server (Figure 1) and the build of 8 mobile agents. Their configurations come from .ini files, which are read, by the agent factory. Next, the factory registers the mobile agents and the agent server launches the mission of each agent sequentially. Each mobile agent contains the route of computers where it has to perform its data collection. Because each agent host has a remote interface for accepting such mobile agent, then a mobile agent invokes the first host of its list and if automatically installed into the virtual machine of the host. Then it reads the set of files, which corresponds to the file format it knows and leaves the agent host for the next host in its list. When all the items of the list are visited, the mobile agent comes back to the server where its data are consumed and parsed. The test contains 6 computers with two servers to manage.

The whole data collection is finished when all the mobile agents are back to the server and their data depose in the corresponding folder. Another part of the scenario starts with the analysis of text patterns and the recognition of abnormal events or durations that are greater than expected. The consequence could the restart of services, which belong to a computer or the final stop in case of high gravity.

B. Events and time

In addition to the business data collection, a mobile agent records its own activity. Each import or export event is recorded with time. The route followed by an agent can be observed during its course by JMX components (Java Management eXtension).

Our first results are about the collected data size. A mobile agent browses 6 nodes (or computers). This means that an agent appends six log files (about 200 kbytes) in the current test and the time cost is near 140 ms. When the data collection occurs too early, the time results are not meaningful, because the size of the data has consequences on the duration of the course.

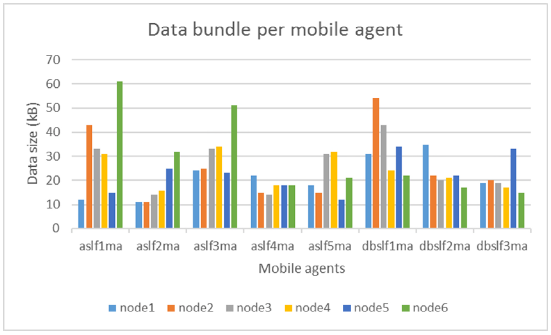


Figure 6. Data size per mobile agent.

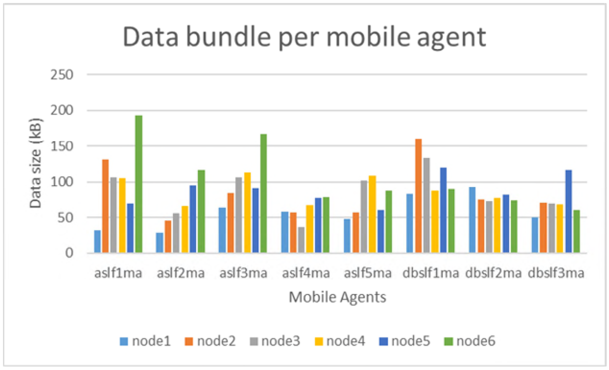


Figure 7. Data size per mobile agent.

So, we rebuild the same scenario another time but after an intensive activity period. We use Apache JMeter for increasing the number of requests and this involves more traffic on the database server.

During the second test, the activity has increased and the log files are bigger. We observe that the shape of the results is similar but the duration is bigger than the previous one (524ms). The ratio is more than three but the size of the data is a quadruple in comparison with the first scenario. We have pursued this benchmark (in Figure 7) with this metric and we confirm this observation.

VII. Conclusion and Future Works

There are many parameters to observe in such kind of application. We have shown that mobile agent is well suited for building a dynamic architecture. The automatize design and build of REST interfaces is another key result of our work. We consider that our project could help other designers to convince that the building of a REST layer can be done in a very predictive time.

We want to continue our measures and more precisely the cost of a couple of actions import/export. We would like to show that a mobile agent a good approach for the reactive systems, especially in the control of network and its administration.

REFERENCES

[1] R. B. Wallace, R. M. Dansereau, and R. A. Goubran: "Methods for the detection of ECG characteristic points". MeMeA 2012: pp. 1-6

[2] M. Krosche, R. Niekamp, and H. G. Matthies: "A Component Based Architecture for Coupling Optimization and Simulation Software in a Distributed Environment". SNPD 2003: pp. 20-23

[3] R. Gfeller and P. Hauser: "Rotated Lines - A Heatmap Representation Method for People Affected by any Kind of Color Blindness". Mensch & Computer 2010: pp. 235-240

[4] F. Amato and F. Moscato: "Exploiting Cloud and Workflow Patterns for the Analysis of Composite Cloud Services". Future Generation Comp. Syst. 67: pp. 255-265 2017

[5] E. S. de Almeida and F. Oquendo: "Software Components, Architectures and ReuseModeling, Customization and Evaluation". J. UCS 19(2): pp. 183-185 2013.

[6] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann: "What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings". IEEE CLOUD 2011: pp. 484-491

[7] S. Bayati, A. K. Tripathi: "Designing a Knowledge Base for OSS Project Recommender System: a Big Data Analytics Approach". ECIS 2016: Research-in-Progress Paper 37

[8] K. Ranganathan and S. Arora: "Enabling Grassroots Communication: A Memory-Aided Broadcast Mechanism for a Community Radio Service on an Ad hoc Device-to-Device Mobile Network". IEEE Trans. Communications 62(3): pp. 1138-1150 2014

[9] D. P. Bertsekas: "Robust Shortest Path Planning and Semicontractive Dynamic Programming". CoRR abs/1608.01670 2016.

[10] L. Wang, X. Wang, and T. S. T. Mak: "Adaptive Routing Algorithms for Lifetime Reliability Optimization in Network-on-Chip". IEEE Trans. Computers 65(9): pp. 2896-2902 2016.