Mobile Agent for Orchestrating Web Services

Charif Mahmoudi
LACL, Paris 12 university
Laboratory of Algorithmic, Complexity and Logic
Computer Science Department
Paris 12 University, France
cm@ramses.fr

Fabrice Mourlin
LACL, Paris 12 university
Laboratory of Algorithmic, Complexity and Logic
Computer Science Department
Paris 12 University, France
fabrice.mourlin@wanadoo.fr

***Abstract*— Mobile agent concept can be considered as a mediator between concepts. We use this principle for management of Web Services at runtime. A set of services are placed on distinct computers, and our business process need to coordinate all the services. Mobile agents allow us to navigate on the computers to access to their local services. The role of mobile agent is not only a trigger of service, but also a transformer and a memory. A transformer because, it applies transformation onto input data and output data to adapt business interfaces. Secondly, it is also a memory because; it manages the state of the business process during its scheduling. This has an essential impact for error handling. Diagnostics are created directly by observation of location of mobile agent but also its progression into mission realization. Depending on the runtime context, computation can be restarted when resource become available. We highlight our results on poll application for training evaluation.**

Keywords- Mobile agent; web service, computation

I. INTRODUCTION

In distributed system, resources are placed on computers and some of them are shared between software. Also, a resource can be available for an application and not for another. A key concept is adaptability. First of all, we consider resource as data exposed on network or anything which is accessible through a distributed protocol. Now the problem is how to manage a computation, if access to part of input data is not possible. Runtime context can become unstable if a computation service is unavailable. Reasons are multiple: data can be locked by another application or load performance does not allow executing another local service. In that case, this local anomaly can involve a global perturbation. It is essential to solve this problem locally, near the origin of problem itself.

Our objective is to build a solution for adapting a business process in case of anomaly at runtime. It means that a strategy has to be deployed for finding another resource for instance, or for waiting its availability. Some works already exist, which use replication of resources [1]. The idea is to manage a pool of resources like data sources and a priority list. When a resource is missing, its successor is selected. Global knowledge of configuration is possible only with toy project. Moreover, a clone resource is not always a solution; when a web service is not available, a solution could be to save messages into a message queue and to replay its content later. Also, it appears that decisions have to be done depending on local information.

This decision power should have the property to move close to the location where the problem occurs. The action to invoke depends on the local properties of the wished resource. Mobile agent can become a solution to export decision power near to required resources. Into next section, we introduce mobile agent concept, then the use of web service as a distributed exposure of a software part.

II. MOBILE AGENT AND MOBILE HOST

In our working context, mobile agent is considered as a piece of executable code which has the ability to move from one computer to another one. Its business action depends on the location where it is. For instance, it can wish to access to data set for preparing a computation. Or, a mobile agent can access to an authorization service. In both cases, mobile agents arrive on a computer where a specific service is available. It means that every computers which accept to belong to a computation, have to be first identified. Then, mobile agent can migrate to these nodes, if local services are useful for their mission.

Thereby, it appears that two software components are essential into our architecture: mobile agent and agent host. A deployment of a distributed application over a network means at least one agent host per computer and a local registry where all remote services are published. The role of agent host is richer than it appears, because it receives mobile agents and negotiates their arrival.

Negotiation step is also a key feature because two sets of constraints have to be resolve when a mobile agent arrives. On one hand, there is a set of requirements due to what the mobile agent need to use locally (to read a dataset, to extract pattern from result set, etc.). To sum up, a constraint is a couple based on a resource and an action. For instance a constraint might be a file of real data and a read action. On the other hand, there is another set of permissions, managed by agent host. Each local resource has its own constraints, for instance a local file is only readable by a particular kind of mobile agent or a specific origin or code base. For each incoming agent, agent host launches a negotiator component and its conclusion is eventually an agent authorization for continuing its local activity.

Furthermore, when mobile agent is accepted by a host, its activities are under control of a supervisor, whether it transgresses the according permissions. For instance, mobile agent can try to add anything at the end of a file, or it can decide to access new local resource. Because, these permissions cannot be implied from the first ones, it should have asked these permissions before. That privileged action won't be evaluated and an exception occurred. We have already presented these works in previous conferences [2], [3] about two domains: monitoring applications and numerical computations. Mobile agents are often used as remote probes, which collect anomaly about a specific protocol, filter data and extract priority data and send them to server. For computation case, mobile agents manage all parts of a whole computation; in case of problem, we have the ability to replay since an event of the execution. They play role of distributed transaction manager. This feature is always important for the use of Web Services.

III. ADVANTAGE OF WEB SERVICE FOR COMPUTING

A. *Web Service*

Distributed service is a very simple mechanism for doing functional thing that has been created since the middle of 1970s with old technologies like Corba for instance [4]. More concretely, it is a new way to do something for which there is a need such a mathematical computation or data extraction. The difference this time is that it is based on open standards upon which the entire industry agrees: one of them is XML language.

We consider web service as a programmatic tool that allows developers to promote local resource onto a distributed protocol used by other pieces of software. As an example, matrix reduction (Choleski program on Figure 1) can be considered as a local program written in C language and a web service is built as an ideal client of that program. In that case, web service is just application of Proxy (CholeskiWS component on Figure 1) and Adapter design pattern [5]. But, web service is not only a wrapping of business code; it uses also standard kind of stream for requesting and eventually for the response (based on SOAP language). Moreover, XML language has rich properties which allow validation, transformation, enrichment and so on. Also dynamic statements can be realized at run time by the use of web service broker (see Figure 1). That one is a remote façade to for all requests about Web Services, which are managed locally (through WebService interface, see Figure 1).

Web Services standard also includes a technology for metadata information about the methods and parameters for a remote method called Web Services Description Language (WSDL) [6]. This has impact on problem recovery, when timeout is achieved or when negotiation fails. Then, an alternative has to be found (a clone of web service or a backup of current request).

Web Services are based on existing standard protocols like HTTP or FTP. This means that Web Services are stateless in nature. Statelessness is a good feature for scalability and is one of the reasons that HTTP is such a scalable protocol. However, this feature limits some of the types of code you can write as a web service. Generally, web service methods perform all the required work in a single invocation. It is not unusual to create web service methods that internally call many other methods to perform a task. For example, if you have transactional method calls, they could all execute within the same web service method call.

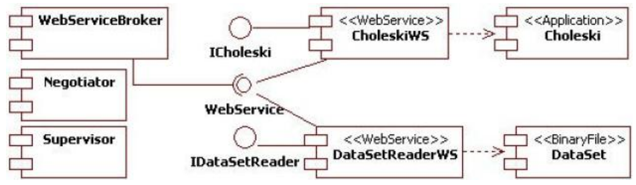

Figure 1. Components on an agent host computer (MAH)

But today, the difficulty is not to call dynamically a web service but to organize a whole algorithm based on a set of Web Services (CholeskiWS and DataSetReaderWS for instance). This means definition of business process, for instance for linear equation resolution, it starts by data extraction step, then a filters are applied, next computation can be done and finally results are consolidated by the use of previous computation. This whole algorithm should be interpreted as a main entity. Next section is about this step composition.

B. *Web Service Composition*

The composition of Web Services is more complex than a composite structure of objects. A working context has to be managed and the sequence of web service calls is not the only useful operator. Several languages were defined for the composition of web service such as WSFL/XLang [7] or BPEL [8]. Other definitions come from specification languages, such as BPMN [9] or polyadic pi calculus [8]. In a more pragmatic approach, a notation based on XML language allows easier management especially at run time. Business Process Elementary Language (BPEL) seems to have more dedicated frameworks and tools such as ODE, Orchestration Director Engine [10]. Enterprise Service Bus (ESB) are also BPEL consumers, they interpret a BPEL stream as a business process. These tools are based on a design pattern called VETRO standing for Validation, Enrichment, Transformation, Routing, and Operation. The BPEL process uses various services provided by other components. The BPEL process itself also provides a service to other components. Its main drawback is the routing step, which costs time especially when problems occur (such a failure). Programmers should develop their own strategy to detect lack of service and to recover it. This is the kernel of our current work.

In our computing context, we have several components with different size and technologies and we need to adapt them into a whole application. Also, web service is a right answer to that problem; this means interoperability of several programming languages, standardization of data exchange via XML language, and also lazy resource management. Moreover, it is natural to anticipate that the compositions are performed dynamically by a large number of end-users.

However, the current process technology based on central process engines implies the adoption of BPEL for this purpose. In fact, in BPEL, processes and composite services are synonymous. A business process is an activity, which has a hierarchical structure. It consists of a collection of sub-activities, either all to be executed in some prescribed order, such as in sequence or in parallel. When such a process is interpreted by a central engine, it involves a large set of XML messages over network. This traffic can be perturbed by data overloading on several web service engines.

When we work on the effects of a business process over the other, again the volume of messages increases always to a key limit and system test are necessary. Because a central architecture is a too strong constraint, we decided to assign the treatment of a business process to mobile agent. Its role is to interpret BPEL definition and to adapt its behavior to the current context. The main idea behind this adaptation is to move from agent server to the computer where the web service broker is. The first impact is the reduction of message number and changes of security control.

C. *Mobile agents as web service pilot*

Our mobile agent philosophy on BPEL is that it is a language for describing how to implement a collection of message-based communication capabilities in terms of state manipulation and messages exchanged with external services.

The core of mobile agent uses an integration layer implementation to receive and deliver messages to external parts and to get access to resources such as input dataset or web service. Also, we developed mobile agent as a BPEL engine with specific challenges such as the state management of executing process and the distribution of parallel statements. Our approach is based on our previous works about higher order pi calculus [12]. This formal specification language has its own operational semantics. We defined each operator of BPEL by the use of pi calculus language [11] (see Figure 2).

BPEL expression

```
<assign name="setMatrix">
  <copy>
    <from variable="BPELInput" part="payload"
           query="/tns:CholeskiProcessRequest/tns:Matrix"/>
    <to part="parameters" query="/mtrx:reduce/mtrx:Matrix"
         variable="webservice_matrix"/>
  </copy>
</assign>
<invoke name="CholeskiWS" partnerLink="CholeskiWS"
          portType="mtrx:CholeskiWS" operation=" reduce "
          inputVariable="webservice_matrix"
          outputVariable="webservice_matrixResults"/>
```

Higher order pi calculus specification

$$CholeskiWS(request, response) = \nu\, result$$

$$\begin{array}{ll} \textit{message reception} & \left(\begin{array}{l} request(matrix) \\ .\overline{reduce}\langle matrix, result\rangle \\ .\overline{response}\langle result\rangle \end{array} \right) \\ \textit{sub process call} & \\ \textit{message emission} & \end{array}$$

Figure 2. Transformation from BPEL language pi calculus

We have already implemented a pi calculus interpreter from a reference operational semantics [11]. Now, we have implemented mobile agent behavior from these formal rules as an application of Interpreter pattern [5]. An agent becomes a BPEL script evaluator. From BPEL expression (see Figure 2: a piece of a specification of Choleski process is displayed), we used an operational semantics of BPEL language based on pi calculus. Then, mobile agent can interpret script as its mission. During this evaluation, intermediate states are saved. This will be powerful information for future comparison or equivalence detection.

In next section, we present the structure of our application and its deployment over network. The location of service forces specific placement and impose roadmaps for mobile agents.

IV. ARCHITECTURE

We adopted a two level specification [12]. The upper one defines software architecture and how components are defined with stereotype. This description is useful for understanding technical frameworks. The lower level of specification describes how our components are placed onto the node of network and it gives a map of available services.

A. *Software architecture*

1) *Overview*

Main part of our platform is called mobile agent server. It first, receives requests from client and delegates given activity to a mobile agent. At the beginning, a mobile agent is enough for a treatment, but it could need help if the business process is too complex. This means that it is not a simple sequence of web service calls.

BPEL is based on Web Services in the sense that each of the business process involved is assumed to be implemented as a Web service. Also, as mentioned before, all elements of a computation have to be equipped for their future use. Also, when a request is received about a given business process, we built a new business space, called BS (see Figure 4). This distributed structure is useful at runtime, because it contains all parts of the execution context and it manages some aspects such as transaction management, state backup and part of safety control.

We have already defined BPEL scripts for some of our business process. They are all about statistics; and more precisely two kinds of statistics descriptive and inferential. They allow us to create graphs and charts, averages, dispersion of data, probability and its distributions etc on large data sets. Our experience about BPEL language allows us to define two different types of business processes: executable processes and abstract processes. Our models are based on actual process of statistics domain. Also, we declared only executable process which can be executed by an orchestration engine at least.

2) *BPEL mobile engine*

To define business processes, BPEL describes a variety of elements: the actors in a business transaction, the messages that need to be transmitted, the type of Web Services that are required and the kinds of Web Services connections that are required for operations. All these data pilot the behavior of our agent which can find relation between the BPEL process definition and the WSDL which defines the interface details of the Web Services.

For all of the local action our implementation of BPEL language is quite similar to official reference. But when it's about invoke statement or data collection; we replace a remote call with an agent migration and a local call.

Of course, our agent uses same standard languages (SOAP, WSDL and UDDI), but we reduce message volume and we consider as pre treatments some aspects which were evaluated before during a call. This takes into account a part of security controls, event tracking, etc. As example, the interpretation of invoke element is one of the main changes.

This XML block (see Figure 2) contains all the details to realize a service call. Also, the agent divides this operation into two steps: migration and call. If the url connection is not valid and exception is raised and the business space is notified on this problem, then a message is logged into anomaly report. This will be the trace that a business process failed and a robot could be collected them and a recover strategy could be implemnted.

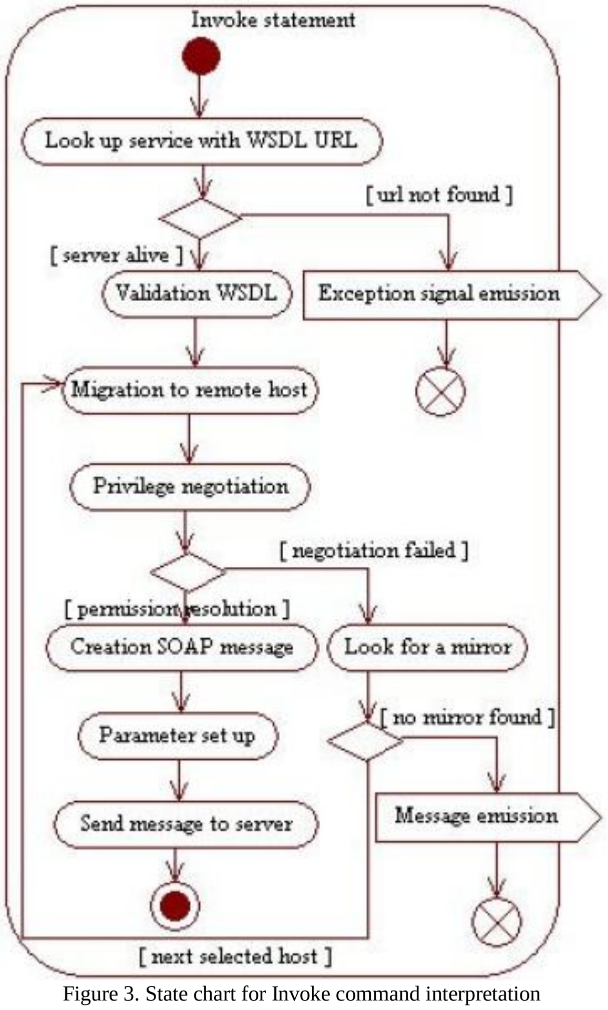

Figure 3. State chart for Invoke command interpretation

When the url is valid, it asks remote web server for receiving WSDL stream (see Figure 3). Then, after its reception, he uses the service tag (from WSDL stream) to find out its destination. It notifies its business space about its future migration and look up the acceptance service of its next destination. If this technical service is available, it will move from its current host to the host where the wished service is published.

For this migration, mobile agent needs a technical service, called acceptance service, which has to be defined on next host and published into local register. An acceptance service allows knowing if a mobile agent has enough rights to perform all the actions onto the current host. Our objective is to prevent anomaly before raising an exception, when mobile agent does not have enough permission to access to local resources. Controller exception can not be totally suppressed because, mobile agent can access to a local resource which changes its mission (for instance, a file contains a list of email addresses and they will be used to notify that the mission is ended). The negotiation between agent host and mobile agents looks like a frontier where all incoming agent are placed into a queue. This is a partial filter against access violation and misleading actions.

3) *Negotiation at entrance*

When an agent host receives a mobile agent, its role is to check the action list, the mobile agent wants to realize. And then it validates with the permissions, it has for this agent. If these permissions are not sufficient for the evaluation of the agent mission, the negotiation fails (see Figure 3) and mobile agent has to find another agent host where the service is declared.

For the permission assignment, mobile agent has to have certificates and details about its origin and its owner. When agent host accepts this new worker, it is launching in parallel a supervisor component to observe what mobile agent will really do during its mission. This is particularly crucial when mobile agent sends requests about importation of other agents or when it uses value of local resource.

Main events are recorded into local register: agent importation and exportation, but also negotiation success and failure, local resource access and extra data about current business process. This is used to trace behavior and also to look for a better execution time. Blockings occurs when resources are not shared, then object locks are declared. These enable multiple agents to independently work on shared data without interfering with each other. The mutual exclusion refers to the mutually exclusive execution of monitor regions by multiple mobile agents. At any one time, only one agent can be executing a monitor region of a particular monitor. If two mobile agents are not working with any common data or resource, they usually can't interfere with each other and needn't execute in a mutually exclusive way.

Agent host can apply an order among all current mobile agents. A higher priority agent that is never blocked will interfere with any lower priority agent, even if none of the agents share data. The higher priority agent will monopolize the CPU at the expense of the lower priority agents. Lower priority agents will never get any CPU time. In such a case, a monitor that protects no data may be used by agent host to orchestrate these agents to ensure all agents get some CPU time. The monitor strategy is based on quota distribution which depends on past analysis of traces of event. So, a host can privilege current agents compared to the new ones. Similarly, a host can fail negotiation because it guesses; there have already been enough agents. Idem, in case the negotiation fails, mobile agent is looking for a mirror site (see Figure 3).

B. *Material architecture*

Material architecture is to model the physical aspect of an object-oriented software system. It models the run-time configuration in a static view and visualizes the distribution of components in an application. In most cases, it involves modeling the material configurations together with the software components that lived on. In our working context, this involves the display of main services which are deployed on computer of the business space. The current work doesn't take into account new materials during execution.

1) *Basic services*

In a distributed system, our basic schema of services contains two kinds of lookup services on each used computer. Technical services are registered into Jini lookup service [13], while Web Services and business services are published into UDDI registers [14]. Both kinds of registers need front web server and sql database service as persistent layer. Of course, more technical services are present for transaction management and security control, but the size of the document does not allow us to detail more.

2) *New services*

The Figure 4 depicts three kinds of node: agent server (MAS), agent host computer (MAH) and UDDI register node (MAR). Of course a concrete computation net has more than one node per category, but this figure allows us to explain concrete scheduling of a business process from client request.

The agent server is a facade which receives all business requests. Depending on business process, a business space is created and equipped to receive the whole scenario. This means a transaction can be open if it is required by the business process. Then, evaluation of BPEL script can start with a first lookup into UDDI registers. By the end of its mission, mobile agent will come back to agent server and it will notify the business space about its stop.

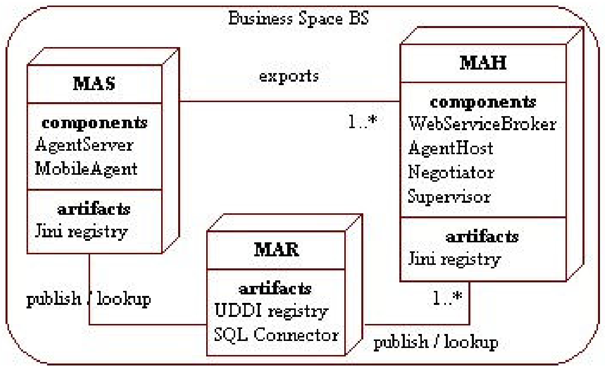

Figure 4: Deployment diagram of a mobile agent application

V. RESULTS

Our experiments show new interesting functionalities, main result is the ability to replay a business service following the same strategy as the first time. Performance studies need to replay a process or to choose between several strategies. Our second result is about anomaly tracking and more particularly how to resume from blocking states.

Enterprise service bus help to debug by the use of save SOAP messages. This information is not complete enough to understand the reasons of a problem. Often, it is concretion of a sequence of events. Also, because agent host saves locally trace of events, it becomes easy to build prefix event sequence from a given event. We developed specific mobile agent for event collection. They allow developer and architect to build a map of distributed events.

Finally, our recovery process was of great value in case of anomaly. Mobile agent can decide to look for a clone of given service and then continue the evaluation of BPEL script. This new service depends on the agent host where mobile agent is.

VI. CONCLUSION

To sum up, we consider our experiments as a validation of new ideas about mobility of actions. We knew that adaptability is a key feature into distributed system on wide network. We establish that mobility of code by the use of agent is a right way for finding alternative when a problem occurs. It becomes crucial to manage set of agents in order to improve negotiation step and supervision.

REFERENCES

[1] Jamali, N. and Xinghui Zhao. 1993. “Self-Adaptive and Self-Organizing Systems”. *SASO '07. First International Conference on*, 9-11 July 2007, 311 - 314. DOI= 10.1109/SASO.2007. pp. 49.

[2] Cyril Dumont and Fabrice Mourlin: Space Based Architecture for Numerical Solving. CIMCA/IAWTIC/ISE 2008: pp. 309-314, 2008.

[3] Mâamoun Bernichi and Fabrice Mourlin: Software management based on mobile agents. ICSNC 2007: 64-74.

[4] Robert Orfali, Dan Harkey and Jeri Edwards, CORBA Fundamentals and Programming, edited by Jon Siegel, publishing by John Wiley & sons, NY, ISBN 0471-12148-7.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", ISBN 978-0201633610 , ISBN 0-201-63361-2, Addison Wesley Professional (Nov 10, 1994).

[6] Benslimane, Djamal; Schahram Dustdar, and Amit Sheth (2008). "Services Mashups: The New Generation of Web Applications". IEEE Internet Computing, vol. 12, no. 5. Institute of Electrical and Electronics Engineers. pp. 13–15..

[7] Frank Leymann, Web Services Flow Language (WSFL 1.0), IBM, May 2001. Web Services Flow Language (WSFL 1.0)"

Frank Leymann] URI:http://www.ibm.com/software/ solutions/webservices /pdf/WSFL.pdf.

[8] Yuli Vasiliev, "SOA and WS-BPEL: Composing Service-Oriented Architecture Solutions with PHP and Open-Source ActiveBPEL", Packt Publishing (September 10, 2007), pp. 316, ISBN-10: 184719270X, ISBN-13: 978-1847192707.

[9] Thomas Allweyer, "BPMN 2.0 Introduction to the Standard for Business Process Modeling", ISBN 978-3-8391-4985-0, Paperback, pp. 156.

[10] Petri Nets and Other Models of Concurrency – ICATPN 2007, Lecture Notes in Computer Science, 2007, Volume 4546/2007, 263-280, DOI: 10.1007/978-3-540-73094-1_17

[11] R. Milner, J. Parrow, and D. Walker (1992). "A calculus of mobile processes". Information and Computation 100 (100): 1--40. doi:10.1016/0890-5401(92)90008-4.

[12] Mâamoun Bernichi and Fabrice Mourlin, "Two Level Specification for Mobile Agent Application," icons, pp.54-59, 2010 Fifth International Conference on Systems, 2010

[13] Scott Oaks and Henry Wong, "Jini in a Nutshell", Publisher: O'Reilly March 2000 pp. 413 Print ISBN:978-1-56592-759-9, ISBN 10: 1-56592-759-1

[1] Tyler Jewell, David Chappell, "Java Web Services", March 2002, 0-596-00269-6, 276 pages, O'Reilly & Associate