IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 6, November 2010
ISSN (Online): 1694-0814
www.IJCSI.org

Mobile agent driven by aspect

Youssef Hannad[1], Fabrice Mourlin[2]

[1] LACL, Laboratory Algorithm Complexity Logics
Computer Science department, Paris 12 University,
Creteil, 94010, France

[2] LACL, Laboratory Algorithm Complexity Logics
Computer Science department, Paris 12 University,
Creteil, 94010, France

Abstract

Domain application of mobile agents is quite large. They are used for network management and the monitoring of complex architecture. Mobile agent is also essential into specific software architecture such that adaptable grid architecture. Even if the concept of mobile agent seems to be obvious, the development is always complex because it needs to understand network features but also security features and negotiation algorithms.

We present a work about an application of aspects dedicated to mobile agent development over a local network. At this level, the underlying protocol is called jini and allows managing several essential concepts such that short transaction and permission management.

Three subsets of aspects are defined in this work. A part is for the description of agent host and its security level, accessible resource, etc. A second part is about mobile agent and their collaboration. This means how they can operate on an agent host with the respect of the execution context.

All the results are illustrated through a distributed monitoring application called DMA. Its main objective is the observation of component servers.

Keywords: *mobile agent, aspect programming, distributed application.*

1. Introduction

In a distributed context, where software shares resources, also a key concept is adaptability. Behind this word, several problems are hidden. Such as examples, numerical application need to access to computing resources over a network. These resources can be locked by another application, or the rights of the numerical application are not sufficient for the operation. In that case, this local anomaly can involve a global perturbation. It is essential to solve this problem locally. A central approach will involve a lot of message exchanges, every pieces of the distributed application will be touched by a local resource access violation. Therefore, a solution should be found locally. It means that a strategy has to be deployed for finding another resource for instance, or for acquiring new access permissions.

For network monitoring, similar problems occur. When an administrator wants to observe the hosts and the state of the applications which are deployed, a part of an application can be inaccessible, also a remote observation can not be computed. Idem, administrator can not reboot the whole application because of a local anomaly. A diagnostic can be found and a solution can be set. For instance, this solution could be to move the local activity to another host or to save the current state of the activity, then to restart local processes. Similar as before, the decision graph has to be efficient if we want to solve a problem without to many perturbations.

Previous examples highlight several features. First, adaptable behavior means that diagnostic and action have to be decided locally to the location where the problem happens. Secondly, action has to be effective: if an activity is not realized, it should be realized somewhere else. Often, problems are due to a service failure or a breakdown of material. Also, migration is a solution to replay an activity into another context.

When a set of actions is moved from one node of the network to another, a collection of properties have to be checked. We can divide these properties into three subsets. A first one is about network characteristics, this means material description, protocol configuration and details on message routes. A second subset describes security permissions; this is essential for the negotiation step. When an activity moves from a node to another one, this activity has to be accepted by a host node. Because each node has its own permission strategy, the host negotiates with the activity to know whether or not this activity can be imported. Of course, this depends on what the activity wish to do and the resource it needs to use. Last subset of properties is about administration of the migration. When

an activity changes its locality, this can impact other activities, especially for message exchanges and more generally for monitoring. We can easily sum up that implementation of such mobile code is a complex task and several technologies can be applied for that objective [1].

This document is divided into five parts with first an introduction about mobile agent programming and application. Secondly, we detail our choice about aspect programming and the reasons for their introduction. Thirdly, we present our design strategy through UML notation and then we present our monitoring application. By the end, we present results of data collection and also the expressive power of message format

1.1 Mobile Agent Programming

Our past experience on mobile agent programming allows us to compare frameworks and also to choose technology depending on its limits. For instance, JADE [2] needs to install services on host node; users cannot decide whether a service is mandatory or not. When runtime environment is restricted about CPU capacity of deployment ability, too many services involves side effect. Tryllian Mobile Agent Technology provides the necessary elements such as security [3], mobility, decision for activity management over a network, but administration is missing. Also, it becomes difficult to diagnostic precisely blocking into an agent application. The framework SOMA [4] has been designed to achieve two main objectives: security and interoperability. Application based on SOMA, can interoperate with different application components designed with different programming styles; it grants interoperability by closely considering compliance with CORBA and MASIF. But this brings a large amount of development rules based on design patterns. Also, software development costs much more with SOMA than other framework like JADE or JAVact [5]. Developers need to use tools for checking if design patterns are correctly applied, but such tools do not exist, also each developer has to assimilate a lot of knowledge to have programming level. Our main observation is this lack of help; it appears that a standard approach of agent mobile programming has to be defined to reduce cost of development. We present in the current document our contribution to the domain of mobile agent development based on the use of aspects approach. Our architecture constraints, described previously, are based on these three subsets and their controls imply particular properties of aspect programming.

1.2 Application domain of mobile agent system

Mobile agents are software abstractions that can migrate across the network. This property has a large spectrum of applications. As mentioned previously, network administration was the first domain where mobile agents are considered as a probe or a spy which provides details about activities of a remote computer or device. A first example is an agent which gauges the load on specific ports. When a threshold is achieved, then agent can export a messenger (or mobile agent) to server with data about the alert.

Mobile agents are also used into intrusion detection system. They have two roles. At the beginning, mobile agents are deployed from an agent server onto hosts where controls have to be done. For instance, an agent can observe protocol login and filters users which try to connect too many times. Regularly, agents notify server to ensure that they are always alive. This is essential when an attack occurs, because the invasion starts by killing agent which observes the protocol. When an agent is not alive, the server exports immediately another one. Thereby the safety service continues until the next attack. We have developed project based on AAFID (Autonomous Agents for Intrusion Detection) [6] strategy, this means a hierarchical architecture based on four kind of agent: monitor, decider, guardian, and filter. Some of them are mobile: decider and guardian, the others are static. Our prototype was used to observe activities on network local to teaching department. The results were surprising about the attack number even on computer without any strategic resource. The robustness of our approach has been enhanced and the concept of mobile agent has been linked for the first time to adaptable context. But a prototype is not enough to affirm general assertion and other examples are developed over the past decade.

To destroy an agent is similar to material failure. We encountered this problem with grid computing where a computation is distributed over the nodes of a grid. The input data are scattered on the nodes, but when a part of the whole computation has an exception, the global result is touched. This exception can come from material or software. To solve this problem of exception management, we defined a software architecture based on a set of mobile agents, called computing space [7]. A server manages not only all components of numerical code, but also all the input data. When a processor of the grid becomes free, a mobile agent is exported onto that node with corresponding input data. When this part of the computation is done, the state of this part is set into the space computing. When all the parts of computation are realized, the termination of the computing case is detected. We use this architecture for several case studies: Choleski computing [8], FDTD computing, Laplace resolution [9]. With these examples, we enhance the idea that mobile agent can adapt a code to its working context: exploitation of free computing resources, replay a piece of computation previously interrupted, etc. We highlight also a new facet of mobile agent called local negotiation. Before agent is exported on a node, a negotiation is established between

agent host and mobile agent to know whether or not the computation can take place on this specific node. Control can be about resource access, location constraints, or security permissions, etc.

This new facet of mobile agent programming add a layer of concepts. This increases development complexity and a project can become hard to maintain if its authors do not apply clear development rules and code convention. Also people who have enough experience in software development can accept and understand easily these constraints but it could be more useful to strive to identify a set of best practices. Then, software engineering tools could help developers to apply them. In order to prepare such tools, we have to isolate concept families.

1.3 Concept partitioning

Today, everyone knows aspect-oriented programming as a new approach to software design. But its advantages are not so used such as modularity, concept isolation, etc. Other approaches, including structured programming and object-oriented programming need user experience to obtain same results. Also we can consider aspect-oriented programming as complement to traditional approaches. It introduces the mechanism of cross cutting for expressing concerns like action migration and automatically incorporating the associated code into the whole system.

Thus, it enhances our ability to express the separation of concerns necessary for a well-designed, maintainable mobile agent system. Some concerns are appropriately expressed as encapsulated agents, or components. As an example, we can note the behavior of a mobile agent onto a node. Others are best expressed as cross-cutting concerns, for example, the precise route of a mobile agent through the client network. A difficulty remains: the identification of places in the code where we want to insert the route description. This is called defining join points. Where the aspects are used much depend one what they are used for. For instance, route definition of mobile agent is key information which can be defined at the declaration step. Again, security constraints are good candidates for becoming an aspect definition. This is functionality that is often used in agent but not a part of the normal business logic. This is also aspects that can be reused in many mobile agents and also be reused in different applications.

The main design improvement we get with aspect oriented programming is better modularization. Redundant code can be placed in an aspect instead of copied to all agent definitions that need it. Developer can concentrate on putting the business logic in the agent definitions and the other part can be handled with aspects. Maintenance of mobile agent application can be improved by an aspect approach programming. This makes the code easier to read and observe. This is particularly essential during debugging phase. A disadvantage is about the understanding of join points and aspect definition. The code can become harder to follow because this can specially be a problem if the design is changed later during the lifecycle of application and functionality can be added with aspects. In our case study, we can sum up with three groups of facets: mobility, security and administration. We consider this objective as a basis of our framework. Moreover, limits can be added in a first approach: dynamicity of migration, permission evolution or evolving observation can be considered as advanced concepts. Also, we are interested in their application but in a second development step. The first development step is about migration, negotiation and agent management. This involves technical choice about aspect definition tool.

2. Aspect definition tool

Aspect definition raises technical constraint depending on kind of aspect. Most of aspect compilers are based on a clear principle: code injection. But the strategy to apply this principle can vary from one implementation to another. A large set of aspect weaver works on source code or byte code. This means that they modify project by injection of technical code into source or byte code. This is useful for generation of XML descriptor into a J2EE project for instance. In our context of mobile agent application, when the migrations of agent are predefined, technical code can be generated from agent definition. But because the effect of the weaver occurs before the execution of the project, the limit of this approach happens when the travel of mobile agent changes during its work. The weaver is not able to change what it was previously generated.

Tools operate with that kind of mechanism: such that AspectJ (first version) [10], JAC [11] or Hyper/J [12]. To solve the lack of dynamicity, a tool like AspectWerkz [13] or JMangler [14] proposes to apply aspect weaver when the class is loading. In a distributed system, this approach is interesting because each agent host has its own class loader. Also, we obtain a new behavior where the effect of advices can be used not only once, but several times depending of the travel of mobile agent. It is dynamic in the sense that it is possible to add, remove and restructure advices as well as swapping the implementation of the introductions at runtime. Like before, the byte code is touched at the loading step but a new limit is achieved. Now, agent behavior is determined at its entrance on to an agent host. This means that its local activity and its next migration are fixed when it arrives on an agent host. In other words, this mechanism prohibits any change into local agent behavior and also with the decision to navigate over the network.

2.1 Dynamic weaving process

This approach is more powerful, aspect declaration has specific language in XML and tools are encapsulated into IDE plug-ins. The aspect oriented programming (AOP) instrumentation process modifies the Java byte code to add runtime hooks around point cuts into agent definition. Those hooks collect reflection information and invoke advices. But to be really dynamic, aspect weaver should be launched by program, this means when a specific event occurs or after a given request. Few aspect tools have this ability. The JBoss AOP [15] instruments takes point cuts definition from an XML file or the metadata file or the annotation tags already embedded in the byte code by the annotation compiler. Now there are three different modes to run our aspect oriented applications: precompiled, load time or hot swap. Hot swap weaving is the most dynamic use when we need to enable aspects in runtime and don't want that the flow control of our classes be changed before that. When using this mode, our classes are instrumented a minimum necessary before getting loaded, without affecting the flow control. If any join point is intercepted in runtime due to a dynamic aspect operation, the affected classes are weaved, so that the added interceptors and aspects can be invoked. As the previous mode, hot swap has some drawbacks that need to be considered such as performance perturbation. But this has a real impact into a reactive environment and for the current case study, this feature is not so essential. Also, our AOP choice has been to adopt JBoss AOP toolkit and associated IDE plug in.

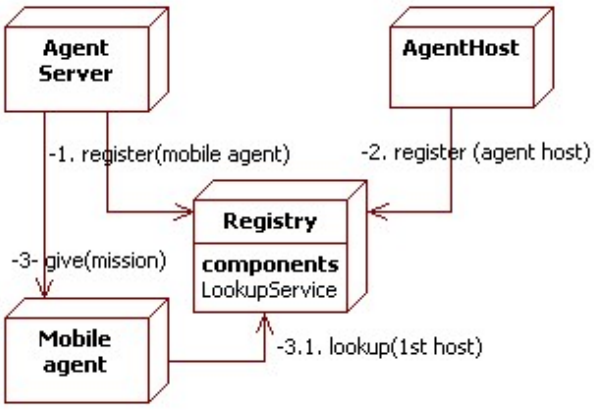

Figure 1: Interaction diagram between main parts of a mobile agent system

2.2 Mobile aspect declaration

The first step in creating a mobile aspect in JBoss AOP is to encapsulate the whole mobile feature in a Java class. The kernel of the mechanism is based on use of distributed services. The first one is a lookup service which is used to register not only mobile agents but also mobile agent hosts.

On our deployment diagram, there is one such registry per node of the network. An agent host is a candidate to a future reception of a mobile agent. It has to publish its reference into the registry of the node where it is. After that, it will be accessible by a mobile agent. Agent server is first a server which creates and manages mobile agents. After creation, mobile agents are published into local registry (1). When a mission is available, mobile agent can book it and then this agent can start to realize it. A mission consists in two parts: a route of nodes and a set of actions. The route is a sequence of computers which support agent host. The set of actions is written in an extern piece of code. Also to find out first node, mobile agent has to look up it into the lookup service. We can identify two aspect definitions: one for mobile agent, another for agent host.

The role of the first mobile agent aspect covers the initial part of the mobile agent life: from its creation until its first publication. When mobile agent starts its mission, its configuration is done by the application of aspect called MobileAspect. The business logic of mobile agent is similar to an automaton with six states: first reading the mission, preparing migration, migration, negotiation, application of the mission, updating it description into registry.

A second aspect is about authorized actions on an agent host. This one has also the role of a gate keeper. This means that it has to check what a mobile agent is going to do before operating its mission. Also, this aspect called HostAspect, is coupled with another one called SafetyAH, (figure 2). HostAspect is used to management of a set of mobile agents. An agent host knows what kinds of mobile agent it is waiting for, also this aspect implements this control before checking the rights of a permitted incoming agent.

It business logic is simply an automaton with four states: waiting for an incoming message, negotiation, observation of activity, sending output message. Because an agent host has to be published into lookup service, it supports also first aspect MAAspect.

When first part of its mission is ended, mobile agent updates own information in the local lookup service. This will be useful for administration of mobile agents. Next, it continues until the end of its mission.

2.3 Safety aspect declaration

When a mobile agent is imported by an agent host, it cannot start its local activity before checking by host whether or not the permissions of mobile agent are

enough. Because this control algorithm is similar for all distributed system. Permission management is intrinsically

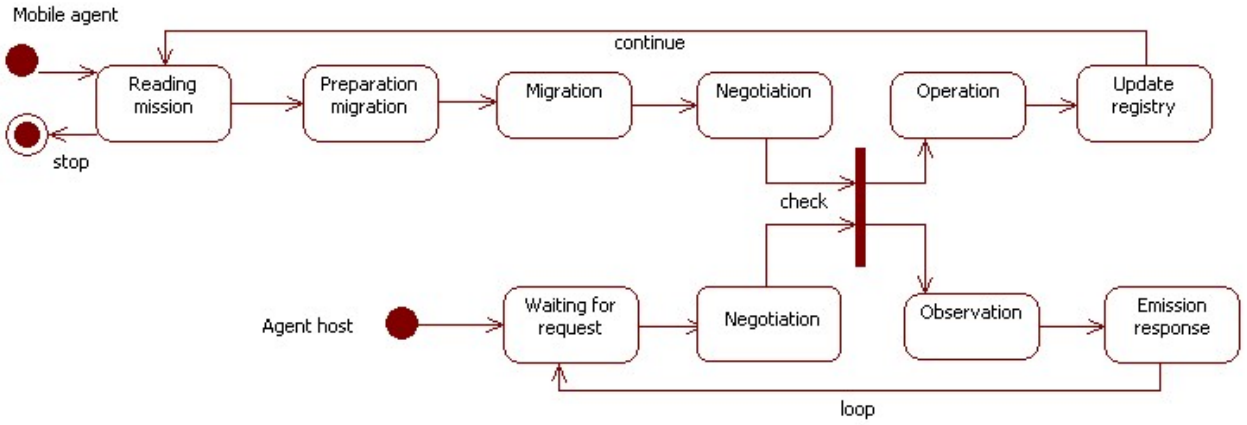

Figure 2: State chart of two main piece of mobile agent system

mobile agent importation, we defined a new aspect called SafetyMA for the creation of negotiation step. During this step, mobile agent has to send its requirements to agent host; it means all local resources used for by agent with action. Its requirements can not be computed at the compile time, but at its entrance. This corresponds to agent loading time on agent host. A dynamic weaver can respect this strategy.

Another aspect is applied to agent host which contains a definition of permissions with signature and localization. During its negotiation step (figure 2), agent host receives safety request from mobile agent. Its business logic starts by the analysis of this demand (figure 3) and the comparison with the accepted operation on the host and also permission list. This aspect called SafetyAH, creates from these collections, an AccessController instance which realized the controls/ If the request is satisfied, mobile agent could operate its local activity on the host. If the demand is not satisfied, host rejects the request and raises an exception.

When negotiation step is satisfied, a security manager is created by agent host to control local activity. This last element is essential when mobile agent executes a local script. The actions which belong to that script have to respect initial contract of mobile agent with agent host.

The lifetime of an AccessController instance depends on the safety request, but a SecurityManager instance observes mobile agent until the end of its mission. Then it will notify agent host and built a report about its own activity. It could be used for a post mortem analysis of mobile agent system.

In previous section, two aspect were defined which can be used at compile time. But this second subset of aspects can not be used before deployment of all the dynamic, for instance, when a mobile agent has realized an operation on a main agent host, it can operate on all hosts which depend on the main one. At the opposite, if a local resource is not accessible on a specific kind of agent host, then this property can be kept to simplify negotiation step on next host.

But the scope of our safety aspects allows us to extract all safety property control from the source code of our project. Now, we can evolve separately the business code of distributed system (work of developer) and safety concerns which are managed by administrator or architect of a project.

2.4 Instrumentation aspect declaration

Instrumentation is a large spectrum of activities from deployment step to runtime observation (performance, security, transaction state, etc). At the first level of domain is use of log files. We decided to manage centralized log information. But, if the used of log file is a basic example in all AOP framework, it becomes more complex into a distributed system. We want not only log centralization but also log consolidation. Standard UNIX syslogd offers UDP-based log forwarding to a central log consolidator today. We need additional features that make it a powerful tool for log forwarding, log centralization and log consolidation. We decided to use a technical framework, called JMX (for Java Management eXtension [16]. JMX specification defines instrumentation of services as MXBeans, agent architecture and standard services. The contract for MXBeans is simple, easy to implement, and unobtrusive for managed resources. Furthermore, the architecture set in the specification decouples the management clients from the managed resources,

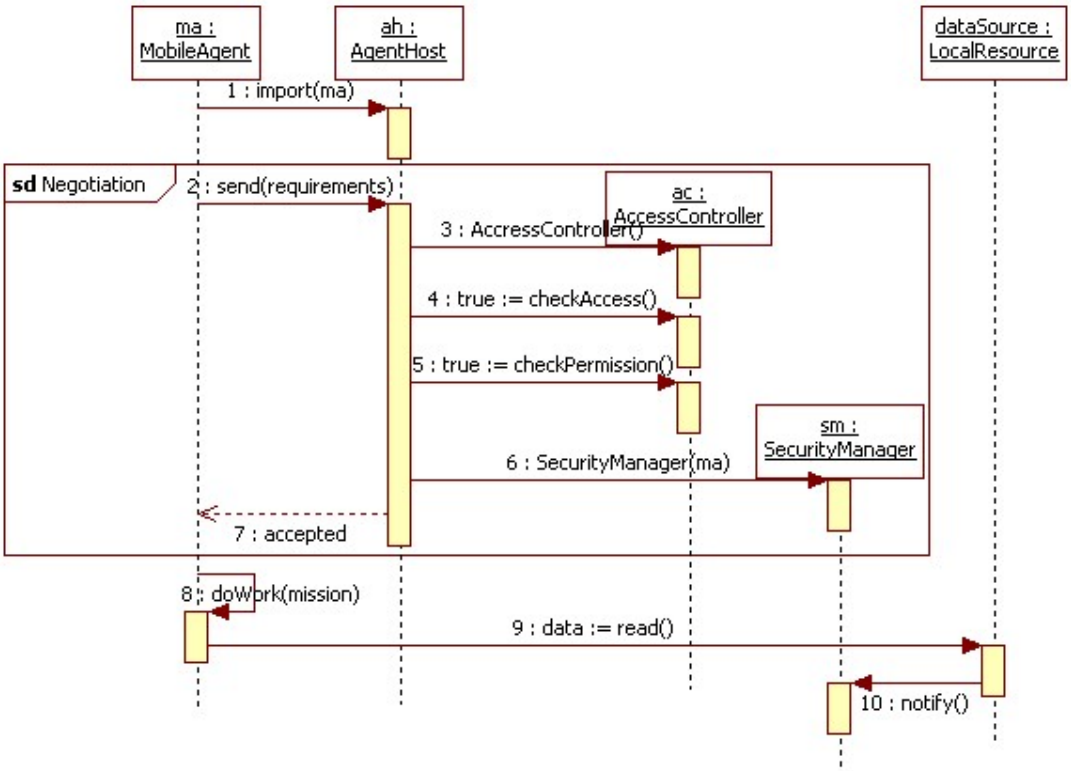

Figure 3: Sequence diagram of negotiation step.

increasing the reusability of JMX-based components. Also, its creation is candidate to an aspect definition.

Centralized log consolidation offers the following benefits: easier log file analysis, increased security, simplified archiving of logs. A centralized log provides a single location for the administrator to perform log file analysis. It offers a single view of events that impact mobile agent systems. A security breach might compromise the local logs but not the centralized copy. Moreover, it is usually simpler to archive a set of centralized logs rather than per-system logs.

Using the JMX technology, a given agent host is instrumented by one remote objects known as Managed Beans, or MBeans. These MBeans are registered in a core-managed object server, known as an MBean server. The MBean server acts as a management agent. The JMX technology provides scalable, dynamic management architecture. Every JMX agent service is an independent module that can be plugged into the management agent, depending on the requirements

We defined standard connectors (known as JMX connectors) that enable us to access JMX agents from remote management server. JMX connectors using different protocols provide the same management interface. Consequently, a management application can manage resources transparently.

We defined a third subset of aspects which contains a definition for the creation of the instrumentation classes for agent host. This one uses it as a local logger but the logger exposes its interface on Jini protocol. Also, it is now possible to observe it from the agent server. Now, administrators are able to know where mobile agents are and also where are current problems, for instance negotiation failure, resource access violation.

From agent host definition, aspect extracts location information and then creates and registers an instance of Instrumentation class. This instance is launched as a parallel flow of agent host. Its starter will be decided at run time. This means that the aspect weaver has to be call at run time. This last subset completes our application of aspect onto mobile agent system.

3. Aspect oriented design to aspect development

Software engineering of mobile agent systems involves a number of concerns, including migration, safety, instrumentation, but also error handling, and a lot

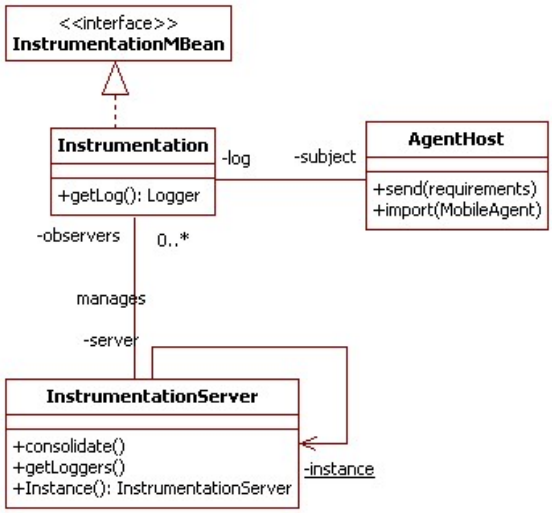

Figure 4: Class diagram for instrumentation of an agent host

of other facets. The modeling, design, and implementation of many of these concerns are essential because they are inherently crosscutting as the system complexity increases. There is a pressing need for specifying aspect approach and its relations towards object-oriented design. This projection is also interesting because, we have already projection from mobile agent model towards object oriented specification. Some works are already published about mapping between AO design and OO design [17], this work is based on a UML specification of all the concepts belonging to aspect oriented programming and evaluation. Main disadvantage is lack generality, a specification is also linked to a aspect framework such as AspectJ or Spring AOP. A specification language like UML is a language specification providing a common interface usable for defining semantics applicable toward arbitrary AOP framework binding

Suzuki and Yamamoto [18] propose a general approach to express aspects during design. They suggest an extension to UML to support aspects appropriately without violation to the meta model specification. For this purpose, they add to the meta model new elements for the aspect and the weaver and an existing element is reused to model the relationship class-aspect.

We emphasize the generation of technology independent models using the Unified Modeling Language (UML) at different points in the software development lifecycle, e.g. requirements modeling, system analysis modeling, and design modeling with use of annotation. It helps to minimize changes on logic specification.

3.1 Analysis modeling

We adopted UML notation with additional stereotype to place our aspects as sub classes of technical class belonging into JBoss AOP framework. Link between aspect and class is done by the use of a class association. For a couple aspects, class, this association represents the point cut for the weaving. Advice concept is described on Fig5 as a subclass of Interceptor class. This class belongs to JBoss, framework, this is why it is placed into its technical package. It is behavior that can be inserted between a caller and a callee, a method invoker and the actual method: for instance, between mobile agent and agent server. These aspect construction allow us to define cross-cutting behavior

Point cuts tell the JBoss AOP framework which interceptors to bind to which classes, what metadata to apply to which classes, for example MobilePointcut is the link between MobileAgent class and one of its facets called MobileAspect.

Aspect definitions of the three subsets are quite similar except for their business logic. The aspects MobileAspect and HostAspect are written with the use of Jini toolkit [19]. Jini is a simple set of Java Classes and services that allows node on a network (e.g., workstation) and agent (e.g., mobile collector) to access each other seamlessly, adapt to a continually changing environment,

The package aspect.safety contains aspects and point cuts about security control. The aspect definition (SafetyAM and SafetyAH) are based on JCE [20] for the cryptographic features (used for certificate management) and JAAS [21] for authentication and authorization strategy. It allows us to plug an external authentication mechanism into message queue.

The package aspect.instrumentation contains aspects definition for remote observation of mobile agent and agent hosts. These definitions are written by the use of JMX framework as remote protocol, and Java Platform Debugger Architecture (JPDA) [22]. It is a programming interface used by development and monitoring tools. It provides both a way to inspect the state and to control the execution of applications running in the Java virtual machine.

3.2. Design modeling

In previous section, declarative description of our approach is explained. Now, we focus on precise definition of one mobile aspect. This aspect encapsulates technical facet of agent: migration via a specific protocol. It allows us to layer, rather than embed, functionality so that code is more readable and easier to maintain. When our migration mechanism will change, modification will be clearly identified.

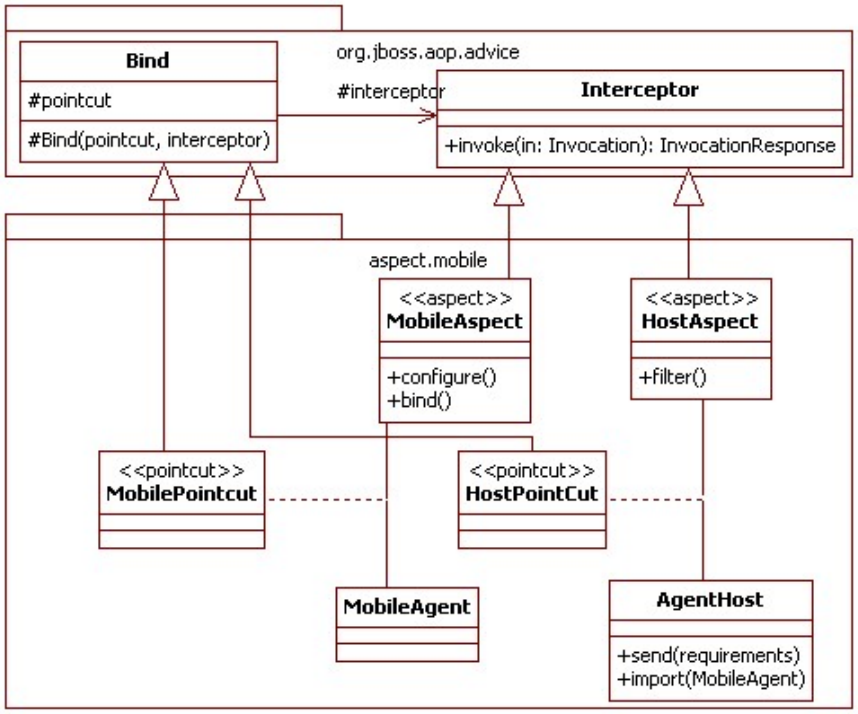

Figure 5: Class diagram of first subset of aspect definition

The point cut called MobilePointcut, defines an event linked to MobileAgent constructor call. So, at the construction, the central mechanism of a Jini system (the lookup service) is called and mobile agent is registered. Then this mobile agent is a mobile service available on the network. While registering, the mobile agent provides a callable interface to access its functionality and attributes those may be useful while querying the lookup from an agent host. Now, mobile agent is waiting until a mission is available. A new available mission is also an event which can be described by a point cut expression language. Also, it is a trigger for the preparation of migration.

A mobile agent is just a temporary worker and its behavior is intrinsically asynchronous. We use another skill of JBoss server: it is a message queue server. Also, we defined a queue per group of mobile agents. The server notifies a MissionAdapter instance (Fig6), then this instance assigns mission to mobile agent.

To add this filtering functionality, we modify MobileAspect and inject a pre statement; this is "Preparation migration" state (Fig2). The actions of mobile agent are basic: first, the access to agent host list. It involves discovering lookup, querying it for the specific agent host service (called acceptance) and invoking the callable interface of the service required. The callable interfaces are exposed and accessed

There is symmetry with aspect HostAspect. When an agent host is plugged into the network, it locates the lookup service (by multi cast discovery) and registers its acceptance service there.

Easily, Jini framework allows building up clusters of agents that know about one another and cooperate, creating a "federation" of agents.

4. Distributed audit

Based on previous aspect approach of mobile agents, we defined a distributed monitoring application called DMA. Its main objective is the observation of agent host and mobile agent traffic. Main concept is a centralized log about distributed observations.

4.1 Mobility as a principle

Data collection is a reference example for mobile agent system. In our context, we need not only to collect data but also to configure local collect algorithm. As mentioned previously, we use JMX framework for management of the distributed collection, but JPDA for local observation. The scheduling of our application has two main phases. First, there is a transient step where observer agents are deployed over network. Then, during a stable step, data are collected and consolidated on server;

Transient step consists in two waves. A first wave

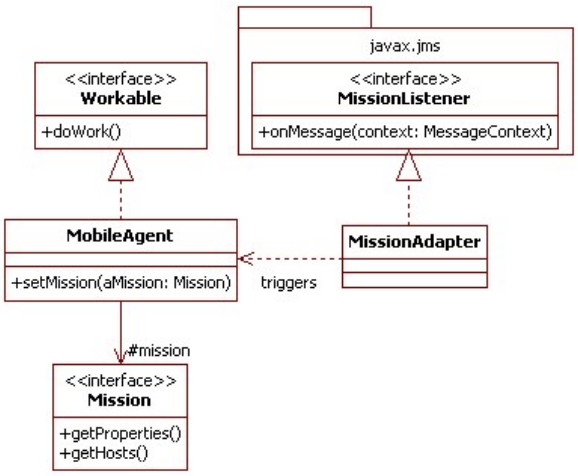

Figure6: class diagram for mission reception

exports a mobile agent, called Observer, on each agent host where data have to be collected. This kind of agent is just a mobile agent which executes an Observation mission. A second wave is configuration of all the observers. Several features can be set: data format, kind of observations (memory, cpu or network). Each value represents a set of events, for instance network is a filter for observing incoming agent and out coming agent.

By the end of this wave, the stable step starts. This means that data collection starts by the use of mobile agent also (called Messenger). This kind of agent is just a mobile agent which executes an Messenger mission. This step is also structure as a loop where local data are first recorded on the agent host (into XML file) and then exported by a Messenger agent towards AgentServer. Messenger agent is able to encode and decode data via a specific format. AgentServer receives all Messenger agents, reads their data and consolidate all information into a single log file. Because the size of such file is limited, log file is sampled; each part is identified by a timestamp.

4.2 Monitoring information

Because data format can be distinct, Messenger agent are essential to transform data into intermediate format of Agent Server. There are three kind of message depending on the previous configuration. If "network" is selected, migration information is sent. This is result of negotiation, duration of the presence of each mobile agent

When memory is chosen, heap memory information is saved; the format follows the generation structure of the memory. Information about garbage collector is also saved (algorithm, frequency, etc).

The option cpu is about local activities of mobile agents. This contains all resource access, but also the place where a security manager checks permissions during execution. Moreover, for each resource access, there is trace of mobile agent responsible and also local time stamp.

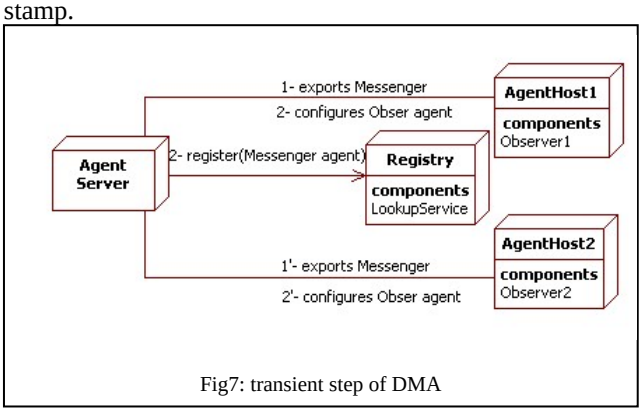

Fig7: transient step of DMA

5 Results

Our approach was validated with a small group of ten nodes. It allows us to gather data for improving administration. Format of exchanged data is XML, but each agent host lays its own format, expressed an XML schema descriptor (XSD). Because, server format is unique, it transforms all input message into its server through an XSL transformation.

5.1 Data collection

Each message is a single XML block of text with an associated namespace and a set of data. Main tag has a "facility" attribute. The facility can be thought of as a category that depends upon the mobile agent from which the message originates.

A short example is given below; this message is exported by the end of configuration of an Observer agent:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns:Message id="th1" facility="obs1"
mnemonic="cpu" severity="2" time="2010-08-
08T12:10:21">
  <ns:Text value="End configuration"/>
</ns:Message>
```

This message is a part of a larger discussion, called "th1" between observer and agent server. It marks the starter of data collection about local activities of agent host ("cpu" collection). During stable phase, a lot of packets can be received and it is necessary to filter input data and provide a time interval for collection. For instance, the given period is 20 ms and only trace of "allocate" events. These changes are done from the server to the observer by the use of JMX service. Then, the data format evolves to transmit only useful information.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns:Message id="th1" facility="obs1"
mnemonic="cpu" severity="2" time="2010-08-
08T14:15:56">
  <ns:Cpu rank="1" self="81.17"
accum="81.17" count="221010" trace="101152"
method="DataAccessReader"/>
</ns:Message>
```

In that case, the information is a snapshot of the current observation. The trace number seen above is related to stack traces in the file itself. This printout shows the rank, the amount of CPU consumed by that method call and total across the application execution, and then finally how many times that individual method was invoked. Then, it references back to a trace for that method invocation. Because this message is a member of a global thread of message, it is possible to rebuild its evolution over a period of time.

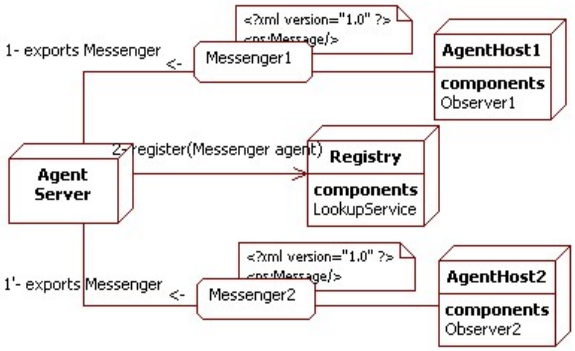

Fig8: data collection with mobile messengers.

5.2 Benefits

Upon initial setup of a new agent host, an observer is exported and messages are stored locally at the beginning. Then, automatically messages are routed via "Messenger" to a centralized location. The benefits are facilities to analyze what may have happened (normal behavior versus strange event) and simplification to archive collected logs off-line to removable media.

Because cross format is XML, it is easy to filter a part of its contain to extract anomalies for instance, or to filter event that cost much more time than the others (the top 10 for instance). One of the main benefits of XML is that it separates data from its presentation. However, because we combine XML data with an XSL Transformations (XSLT) style sheet, we then have a powerful way to dynamically transform and present information in any format we want. Furthermore, often the structure of our XML stream created by our application does not match the structure required by other application parts to process that XML data. To transform the existing XML data structure into one that can be processed, we need to use XSLT. Moreover, transformation can provide SVG or GraphML representations that highlight cpu ratio or call numbers.

Since the mechanics of applying XSLT style sheets to XML in Java code are generally the same, the process can be refactored out of the business-specific code into something more reusable. The chain starts with a source XML stream (though not necessarily a file), and applies a series of XSLT style sheets to it until it produces the final stream used to observe local activities. For instance, an XML stream describes memory management. It is the result of data collection (done by a mobile agent) about garbage collector activity on agent host. The Java runtime uses a garbage collector that reclaims the memory occupied by an object once it determines that object is no longer accessible. This automatic process makes it safe to throw away unneeded object references because the garbage collector does not collect the object if it is still needed elsewhere. Therefore, in agent host, the act of letting go of unneeded references never runs the risk of deallocating memory prematurely. This event is serialized into XML stream when mobile agent is physically on this agent host.

At the end of its trip, a mobile agent contains a large data set and XSLT transformations are used to extract into specific order memory information. Transformations are about size of collected data or collection time or agent host address. It is also a strategy to compute memory amount used at a given time of a distributed computation.

4. Conclusions

Aspect-oriented programming is a powerful new tool for software development especially mobile agent system. With JBoss AOP, we can implement your own interceptors, metadata, to make our mobile agent development process more dynamic and fluid.

According to our experience, there was a number of crosscutting mobile agent system concerns which aspect-oriented abstractions succeeded to cope with their modularization. This was often the case for mobility. For these agent properties, the design and implementation have shown expressive improvements in terms of separation of concerns. By the end, our objective of mobile agent instrumentation is achieved, especially for information collection for monitoring.

References

1. Ivan Kiselev, Aspect-Oriented Programming with AspectJ, 2001, Sams, ISBN: 0-672-32410-5
2. Pavel Vrba, E.Cortese, F. Quarta, G. Vitaglione, Scalability and Performance of the JADE Message Transport System. Analysis of suitability for Holonic Manufacturing Systems.

this number of EXP. LNCS, vol. 4128, pp. 1148--1158. Springer, Heidelberg (2006)
3. Mobile agents, a.k.a distributed agents, according to Tryllian, doi:10.1049/ic:20010004, IEE Seminar Mobile Agents - Where Are They Going (2001/150) London, UK, 11 April 2001,
4. A. Corradi, R. Montanari, C. Stefanelli, Mobile Agents Protection in the Internet Environment, Proceedings of the COMPSAC'99, IEEE Computer Society Press, Phoenix, October '99.
5. Jean-Paul Arcangeli, Vincent Hennebert, Sébastien Leriche, Frédéric Migeon, Marc Pantel. JavAct 0.5.0 : principes, installation, utilisation et développement d'applications. Rapport de recherche, IRIT/2004-5-R, IRIT, février 2004
6. E. Spafford and D. Zamboni. A framework and prototype for a distributed intrusion detection system. Technical Report 98-06, COAST Laboratory, Purdue University, West Lafayette, IN 47907-1398, May 1998.
7. Cyril Dumont, Fabrice Mourlin: Space Based Architecture for Numerical Solving. CIMCA/IAWTIC/ISE 2008: 309-314
8. Cyril Dumont, Fabrice Mourlin: A Mobile Computing Architecture for Numerical Simulation CoRR abs/0711.1786: (2007)
9. Cyril Dumont, Fabrice Mourlin: Adaptive runtime for numerical code, 8th ENIM IFAC International Conference of Modeling and Simulation Evaluation and optimization of innovative production systems of goods and services (2010) pp310-320
10 Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhot´ak, Ond¡rej Lhot´ak, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. abc: An extensible AspectJ compiler. In AOSD, mar 2005, pages 87–98. ACM Press.
11. Chitchyan, R. et al. “Survey of Aspect-Oriented Analysis and Design”. AOSD-Europe Project Deliverable No: AOSD-Europe-ULANC-9. www.aosd-europe.net
12. Garcia, A., Chavez, C., Kulesza, U., Lucena, C. “The Role Aspect Pattern”. Proc. of the 10th European Conf. on Pattern Languages of Programs (EuroPLoP’05), July 2005, Irsee, Germany.
13. Griswold, W. et al, "Modular Software Design with Crosscutting Interfaces", IEEE Software, 2006
14. Filman, R. et al. “Aspect-Oriented Software Development”. Addison-Wesley, 2005.
15. Tom Marrs, Scott Davis, JBoss at Work: A Practical Guide, O'Reilly (2004)
16. Benjamin G. Sullins and Mark B. Whipple, JMX in Action, 2002 | 424 pages, ISBN: 1930110561
17. Kendall, E. "Role Model Designs and Implementations with Aspect-oriented Programming". OOPSLA 1999, pp. 353-369.
18. Extending UML with Aspects: Aspect Support in the Design Phase. 3er Aspect-Oriented Programming (AOP) Workshop at ECOOP ´99. Junichi Suzuki, Yoshikazu Yamamoto.
19. Scott Oaks, Henry Wong, Jini in a Nutshell, O'Reilly Media, March 2000
20. David Hook, Beginning Cryptography with Java, ISBN13: 978-0-7645-9633-9, ed. Wrox, 2005-08-19
21. Michael Cote, Java Authentication and Authorization Service (JAAS) in Action, ed Wiley, 2005
22. Harbourne-Thomas A., Bell J., Brown S., "online: Professional Java Servlets 2. 3", ISBN: 186100561X, 2003

Youssef Hannad is PhD student at Paris 12 University. His position is set by Team Up corporate. His subject is on aspect for mobile agent management.

Fabrice Mourlin. Is associated professor at Paris 12 University since 1992. He manages a working group on mobile agent and space computing. He obtained HDR habilitation in 2008 at Paris 12 university. Current projects are network monitoring and numerical computing.