Reactive Mobility by Failure

Alejandro Zunino

ISISTAN Research Institute - UNICEN University
Campus Universitario (B7001BBO), Tandil, Bs. As., Argentina
TEL/FAX: +54 (2293) 440363 - azunino@exa.unicen.edu.ar

Abstract

Mobile agent development is mainly supported by Java-based platforms. However, the weak mobility model they use, added to the inherent complexity of building location-aware software, impose strong limitations for developing mobile agents. In this article we present MoviLog, a platform for building Prolog-based mobile agents with a strong mobility model. The major contribution of MoviLog is its *Reactive Mobility by Failure* (RMF) mechanism. MRF is a mechanism that acts when an agent needs a resource that is not available at the current executing site. RMF uses a distributed multi-agent system to transparently transport the executing agent to the site where the resource is available, thus reducing the development effort with respect of traditional platforms, while maintaining its advantages.

1 Introduction

The wide diffusion of the Internet and the WWW are providing a global platform for the development and deployment of massively distributed systems. Despite recent advances on technologies and standards, the development of such a massively distributed applications remains expensive and difficult. Moreover, the near future appears much more complex. Just image the WWW with >1000 millions of hosts, many of these hosts with unreliable connections, several different operating systems and hardware. One of the main risks with such a complex scenery is to under exploit its resources, not taking advantage of its huge volumes of resources.

Mobile agents, software able to autonomously migrate between network hosts in order to achieve their users' goals, emerged as a novel paradigm offering advantages over traditional stationary software in distributed systems [4]. Mobile agents are different from stationary software in one very important aspect: mobile agents are *location aware.* As a consequence mobile agents are able to choose, for example, whether to remotely access to a non-local resource or to migrate to the place where the resource is located, depending both on the cost of remotely accessing to the resource and the cost of migrating.

Although the well known advantages of mobile agents [6] and the availability of many platforms that support mobile agents [5, 7, 8], their usage is still limited to small applications. We believe this is caused by the fact that mobile agents are inherently more complex than traditional stationary systems. Clearly, mobile agent developers have to provide mechanisms to decide when and where to migrate. Therefore, though agents' location awareness may be very beneficial, it also adds further complexity to the development of mobile agents, specially with respect of stationary applications.

In this paper we describe MoviLog [11], a platform for Prolog-based mobile agents that aims at reducing the development effort while fully exploiting available network resources. MoviLog is based on a novel concept called *Reactive Mobility by Failure* (RMF). The idea behind RMF is to use a stationary multi-agent system (MAS) distributed across the network to assist a mobile agent on taking decisions on *when* and *where* to move, based on resource needs. In order to retain agents' autonomy, RMF only interferes with the normal execution when it detects a *failure*. A failure occurs when an agent accesses to a specially declared resource that is not available at the current executing site.

The paper is organized as follows: the next section describes RMF, then, section 3 describes its implementation; finally, the conclusions are presented in section 4.

2 Reactive Mobility by Failure

MoviLog is a mobile agent platform that implements a new form of mobility called Reactive Mobility by Failure which aims at reducing the effort of developing mobile agents by automating some decisions about mobility. RMF is based on the assumption that mobility is orthogonal to the rest of attributes that an agent may possess (intelligence, agency, etc) [2]. Under this assumption it is possible to think of a separation between these concerns at the implementation level [3]. RFM exploits this separation by allowing the programmer to focus his efforts on the stationary functionality, and delegating mobility issues on a distributed MAS that is part of MoviLog.

Though some of the concepts of RMF are language independent, we will limit the scope of the paper to Prolog-based agents called *Brainlets*. Brainlets are written in JavaLog [1, 10], a multi-paradigm language that integrates Java and Prolog. A Brainlet is composed of Prolog clauses, Java objects and a number of *protocols*. A protocol is a declaration of the interface used to access a resource. By resource we mean, for example, data (Prolog clauses, databases, Web pages, Java objects, etc) or code (Prolog clauses, Web accessible programs, Java methods, etc).

When a predicate of a Brainlet declared as a protocol fails (it is not possible to evaluate the predicate at the current site, or there are no more choices left), RMF moves the Brainlet to another site having definitions for such a predicate and continues the normal execution to try to find a solution. The implementation of this mechanism requires the MoviLog inference engine, named MARlet, to know where to send the Brainlet, thus the MARlet has to know the protocols available at every site of the network. In order to maintain this information, some of the mobility agents named PNS (protocol name servers) discover MARlets and keep up to date the protocols offered by each site. The next section describes the main concepts of RMF and its differences with proactive mobility.

2.1 Proactive Mobility vs. RMF

In order to clarify the concepts introduced up to now we will describe a distributed meeting scheduling system developed with MoviLog. In this system, each user is represented by a Brainlet that manages his calendar. A Brainlet stores and incrementally updates a profile of the user, including its preferred places, meetings, contacts, etc. When a Brainlet receives a proposal for a meeting, it begins a negotiation with its counterpart. Since the counterpart may reside in a different place of the network and the negotiation involves several interactions, it may be beneficial to migrate. By using traditional proactive mobility, each agent autonomously decides whether to migrate or not. The following code shows a fragment of a Brainlet that uses proactive mobility. The clause *proposalListener* is invoked when the Brainlet receives a message proposing a meeting. When this occurs the Brainlet analyzes whether to move to the site where the counterpart is located and then initiates a negotiation.

```
PROTOCOLS
CLAUSES
 proposalListener(User,P) :- in(role(User,OtherAgent,Role)),
   moveToOrStayHere(User,OtherAgent,Role,P), in(consensus(User,C)), eval(User,C).
 moveToOrStayHere(User,OtherAgent,'persuader',P) :-
   getBrainletsForRole(OtherAgent,id(_,H)), ( estTraffic(negotiation,H,Tn),
   estTraffic(move,H,Tm), Tn>Tm -> moveTo(H) ), negotiate(User,OtherAgent,
   'persuader',P), return.
 ...
```

In this example, the parts of the code in bold decide whether to move to the remote site or not based on the estimated network traffic generated by the negotiation and the estimated network traffic used to migrate the agent. Since MoviLog provides strong mobility, the execution of the agent is resumed at the remote site after the *move* predicate.

Though the support for strong mobility provided by MoviLog is easier from a programmer's point of view than the weak mobility model provided by most Java-based platforms [5, 8, 7], it is still

difficult to use and exploit. A simple fact will back our claim: non-mobile agent development it is well-know for being a challenging task [9], thus adding mobility to those already complex agents just make everything much harder. This is the problem that RMF addresses by combining proactive mobility (decided by the mobile agent) and reactive mobility (decided by PNSs and mobility agents).

RMF acts only when a predicate declared as a protocol fails, thus in order to modify the previous example to use RMF we have to define a number of protocols. Let us assume that an agent *AgentId* asserts the fact *role(User, AgentId, Role)* when it wants to negotiate a meeting with the role *Role* for a given user. By declaring *role/3* as a protocol, other agents may take advantage of RMF by simply calling *role* as a goal. If one or more facts *role/3* are present at the local MARlet, then they will be tried one by one. When no more choices are left, RMF will migrate the agent to a remote MARlet where some Brainlet asserted *role*, if considers appropriate to do so. In the code, the clause *moveEval* is used to tailor the decision mechanism of RMF to this specific application.

```
PROTOCOLS
 role/4.
CLAUSES
 init :- moveEval( role/4, estTraffic(negotiation,H,Tn),
 estTraffic(move,H,Tm), Tn>Tm ).
 proposalListener(User,P) :-
   in(role(User,OtherAgent,Role)), role(User,OtherAgent,Role),
   negotiate(User,OtherAgent,'persuader',P), in(consensus(User,C)),
   return, eval(User,C).
 ...
```

As can be seen, the code using RMF is cleaner and simpler than the code using proactive mobility. Note that a simpler code implies that less code is transferred through the network. As a consequence RMF potentially may be faster than proactive mobility [11].

3 The MoviLog Platform

MoviLog is an implementation of the concept of RMF. MoviLog is an extension of JavaLog [1, 10], a framework for building agent-oriented languages, to support mobile agents on the Web. MoviLog supports the execution of Prolog-based mobile agents named *Brainlets*. Those Brainlets run on top of *MARlets*, special Java servlets encapsulating the MoviLog inference engine and providing services to access it. A set of MARlets distributed across several sites where all the MARlets listen exactly the same port number is called a *logical network*. As a consequence, a host may execute more than one MARlet, however, all of them would belong to different logical networks. The rest of the section is devoted to describing MoviLog and its main features.

3.1 MARlets

In order to enable mobility across sites, each host belonging to a MoviLog network must be extended with a MARlet which provides the runtime support running on top of a Web server. In this way, MARlets enable Brainlets to interact with standard Web technology. Additionally, a MARlet is able to provide intelligent services under request, such as performing logic queries, adding and deleting logic modules, activating and deactivating logic modules, etc. In this sense, a MARlet can also be used to provide inferential services to legacy Web applications or lightweight agents.

3.2 Brainlets

A Brainlet is composed of *code*, *data*, *execution state* and *protocol clauses*. Code and data consist of Prolog clauses and Java objects. JavaLog, the framework on which MoviLog is based, provides a smooth integration between Prolog and Java that is useful when developing agents. An agent's

Table 1: A simple logical network

M_1	M_2	M_2
hd(#123,eide,wd,5400,40,72)	hd(#78,scsi,ibm,15000,36.7,187)	hd(#22,scsi,seagate,7200,36,210)
hd(#23,eide,maxtor,7200,40,79)	hd(#33,eide,quantum,7200,20,54)	hd(#44,eide,panasonic,7200,30,582)
hd(#45,eide,ibm,5200,30,114)		

execution state consists of one or more Prolog threads. Protocol clauses are used by RMF to automatically migrate a Brainlet when a failure occurs. A protocol has the syntax functor/arity, for example, *article/3* denotes all the clauses with the form article(_,_,_,_).

3.3 Mobility

MoviLog supports strong mobility in its two forms: proactive (initiated by the agent itself) and reactive (initiated by an external event). With respect to proactive mobility, it is triggered by the execution of the Prolog predicate *moveTo(host)*. When a Brainlet executes *moveTo*, its threads are suspended. Then information on how to resume these threads is serialized, including the current program counter and execution stack (choice points, variables and network itinerary); and send through the network to the destination MARlet. At the destination, after verifying the integrity of the received information, the Brainlet is restored from its compressed serialized representation, and the threads are resumed.

RMF is a novel concept introduced by MoviLog [11]. RMF acts when the evaluation of a predicate declared in the protocols section fails. Let us show a simple example that will clarify this:

```
PROTOCOLS
 hd/4.
CLAUSES
 preferred(L,ID) :- ...
 run :- findall(ID+Pr,hd(ID,scsi,Brand,RPM,MB,Price),L), preferred(Data,ID), buy(ID).
```

Table 1 shows the databases of three MARlets M_1, M_2 and M_3. The goal of the Brainlet is to find a hard disk that satisfies several constraints. The predicate *findall(Term, Goal, Bag)* unifies *Bag* with a list of all the instances of *Term* for which *Goal* succeeds. *Terms* in *Bag* are in solution order, i.e. the first term corresponds to the first solution found, and so on. If the Brainlet starts its execution at M_1, *findall* will try with all the predicates *hd/3*. The failure will occur after trying with *hd(#45,...)*. At this point mobility agents will ask the local PNS agent for MARlets offering the protocol *hd/4*. In this particular case, the result will be $\{M_1, M_2, M_3\}$. Then, mobility agents will build an itinerary and reactively migrate the Brainlet to the next destination, for example M_2, by using the same strong migrating mechanism as *moveTo*. At M_2 and M_3 the process will be similar. As a result *findAll* will evaluate alternatively all the clauses *hd/4* offered by the MARlets. At this point it is worth noting that RMF can be adapted in a number of ways. For example, it is possible to specify how to build the itinerary, how to decide whether to migrate or not, and how to estimate costs.

4 Conclusion

Intelligent mobile agents represent one of the most challenging research areas due to the different factors and technologies involved in their development. Strong mobility and inference mechanisms are, undoubtedly, two important features that an effective platform should provide. MoviLog represents a step forward in that direction. The main contribution of our work is the *reactive mobility by failure* concept. It enables the development of agents using common Prolog programming style, making in it easier thus for Prolog programmers. This concept, combined with proactive mobility mechanisms, also provides a powerful tool for exploiting the Web.

References

[1] Analía Amandi, Alejandro Zunino, and Ramiro Iturregui. Multi-paradigm languages supporting multi-agent development. In Francisco J. Garijo and Magnus Boman, editors, *Multi-Agent System Engineering*, volume 1647 of *Lecture Notes in Artificial Intelligence*, pages 128–139, Valencia, Spain, June 1999. Springer-Verlag.

[2] Jeffrey M. Bradshaw. *Software Agents*. AAAI Press, Menlo Park, USA, 1997.

[3] A. Garcia, C. Chavez, O. Silva, V. Silva, and C. Lucena. Promoting Advanced Separation of Concerns in Intra-Agent and Inter-Agent Software Engineering. In *Workshop on Advanced Separation of Concerns in Object-Oriented Systems (ASoC) at OOPSLA'2001*, Tampa Bay, Florida, USA, 14 October 2001.

[4] Robert S. Gray, David Kotz, George Cybenko, and Daniela Rus. Mobile Agents: Motivations and State-of-the-Art Systems. Technical Report TR2000-365, Dartmouth College, Computer Science, Hanover, NH, April 2000.

[5] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Mobile Agents with Java Aglets*. Addison-Wesley, Reading, MA, USA, September 1998.

[6] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, March 1999.

[7] Gian Pietro Picco. μCode: A Lightweight and Flexible Mobile Code Toolkit. In K. Rothermel and F. Hohl, editors, *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 160–171. Springer-Verlag: Heidelberg, Germany, 1998.

[8] Alberto Silva, Miguel Mira da Silva, and José Delgado. An overview of AgentSpace: A next-generation mobile agent system. *Lecture Notes in Computer Science*, 1477:148–158, 1998.

[9] M. J. Wooldridge and N. R. Jennings. Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing*, 3(3):20–27, 1999.

[10] Alejandro Zunino, Luis Berdún, and Analía Amandi. Javalog: un lenguaje para la programación de agentes. *Revista Iberoamericana de Inteligencia Artificial*, (13):94–99, 2001.

[11] Alejandro Zunino, Marcelo Campo, and Cristian Mateos. Simplifying mobile agent development through reactive mobility by failure. In Guilherme Bittencourt and Geber Ramalho, editors, *Advances in Artificial Intelligente*, volume 2507 of *Lecture Notes in Computer Science*. Springer-Verlag, November 2002.